applied sciences

MDPI

Article

Code Edit Recommendation Using a Recurrent Neural Network

Seonah Lee [1], Jaejun Lee [2], Sungwon Kang [3,*], Jongsun Ahn [3] and Heetae Cho [1]

[1] Department of AI Convergence Engineering (Graduate) and Aerospace and Software Engineering (Undergraduate), Gyeongsang National University, Jinju 52828, Korea; saleese@gnu.ac.kr (S.L.); cht3205@gnu.ac.kr (H.C.)
[2] BigPictureLabs Inc., Daejeon 34047, Korea; mono@kaist.ac.kr
[3] School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Korea; jsahn@kaist.ac.kr
* Correspondence: sungwon.kang@kaist.ac.kr; Tel.: +82-42-350-3512

Abstract: When performing software evolution tasks, developers spend a significant amount of time looking for files to modify. By recommending files to modify, a code edit recommendation system reduces the developer's navigation time when conducting software evolution tasks. In this paper, we propose a code edit recommendation method using a recurrent neural network (CERNN). CERNN forms contexts that maintain the sequence of developers' interactions to recommend files to edit and stops recommendations when the first recommendation becomes incorrect for the given evolution task. We evaluated our method by comparing it with the state-of-the-art method MI-EA that was developed based on the association rule mining technique. The result shows that our proposed method improves the average recommendation accuracy by approximately 5% over MI-EA (0.64 vs. 0.59 F-score).

Keywords: data-based software engineering; code edit recommendation; recurrent neural network; machine learning; interaction histories

Citation: Lee, S.; Lee, J.; Kang, S.; Ahn, J.; Cho, H. Code Edit Recommendation Using a Recurrent Neural Network. *Appl. Sci.* **2021**, *11*, 9286. https://doi.org/10.3390/app11199286

Received: 6 September 2021
Accepted: 30 September 2021
Published: 6 October 2021

1. Introduction

Developers spend a significant amount of time looking for files to modify when performing software evolution tasks [1]. By recommending files to modify [2], a code edit recommendation system allows developers to reduce search time during software evolution tasks.

To reduce the time spent by the developer on code navigation, Zimmermann et al. [3] developed a recommendation system for mining association rules between changed files by collecting the revision history stored in the version management system. The recommendation system ROSE recommends files to edit with the context of one changed file and yields a 0.33 F-score [3]. Subsequently, Lee et al. [1] developed a recommendation system MI-EA that works by mining association rules between viewed files and edited files from an interaction history, such as what was recorded by Mylyn [4] and stored in the Eclipse Bugzilla system. MI-EA recommends files to edit with the context of one edited file and three viewed files and yields a 0.58 F-score [1]. Lee et al. [1] viewed revision history as a subset of the interaction history in that revision history utilizes edit information for recommendations whereas interaction history utilizes both view information and edit information for recommendations. Lee et al. [1] also showed that Zimmerman's method applied to the interaction history obtains almost the same accuracy values as Zimmerman's [3]. According to Lee et al. [1], MI-EA yields a recommendation accuracy that is higher than that of ROSE because MI-EA uses a more elaborate context that includes viewed files.

Both of the recommendation systems of Lee et al. [1] and Zimmermann et al. [3] use association rule mining as the main technique. Although association rule mining has been successfully used for code recommendation, it has the disadvantage that it can consider only the co-occurrence of items. Since it does not consider the order of files navigated by

developers, we view that it misses a chance to use a more elaborate context that could contribute to accurate recommendation. We conducted a preliminary experiment by using an N-Gram model that maintains the order of files navigated by developers and found that the precision of the N-Gram model is higher than that of MI-EA, while the recall of the N-Gram is significantly lower [5].

In this paper, we propose a code edit recommendation method based on a recurrent neural network known as the multi-label model. We name our proposed approach the code edit recommendation method using a recurrent neural network (CERNN). CERNN uses a recurrent neural network model to learn sequential information and has the potential to surpass precisions of the previous methods while maintaining reasonable recalls. CERNN stops recommendations when the first recommendation becomes incorrect for the given evolution task. We compared CERNN with the state-of-the-art approach MI-EA [1]. In the comparison, our approach CERNN yielded a 64% F-score, while MI-EA yielded 59% F-score precision, which amounts to an improvement of 5% with our approach.

Our contributions are as follows. First, we propose elaborating the contexts of the code edit recommendation method based on the RNN model. Second, we implement the online-learning evaluation method to set-up the same experimental environment as previous studies did. Third, we show that the proposed approach CERNN yields higher recommendation accuracy than MI-EA in the same experimental environment.

This paper is organized as follows. Section 2 describes the related work on edit recommendation systems. Section 3 explains N-Gram and recurrent neural networks and describes our preliminary experimental results with those models. Section 4 presents our code edit recommendation method using a recurrent neural network (CERNN). Section 5 explains our evaluation setup, and Section 6 evaluates our method using the system that implements it. Section 7 discusses our experimental results and additional experiments. Section 8 discusses the threats to validity. Finally, Section 9 concludes this paper.

2. Related Work

A recommendation system for software development is "an application that provides valuable information for software engineering work in a given situation" [2]. A code edit recommendation system is an application that recommends files to edit to reduce the time developers spend on code navigation activities during software evolution tasks. Research related to this paper can be classified largely into four groups: research for code edit recommendation systems, tools for collecting developers' interaction histories, empirical studies on developers' interaction histories, and research using artificial neural networks for recommendations.

2.1. Research for Code Edit Recommendation Systems

Code edit recommendation systems can be divided into two types according to their input data: those based on software revision history and those based on the developers' interaction history. These two types of recommendation systems have been developed independently and early works on both types used association rule mining as the standard technique.

Using software revision history, Zimmermann et al. [3] developed a recommendation system ROSE that mines association rules between changed files. When a developer changes a particular file (or method), ROSE recommends a list of files (or methods) associated with that file (or method). For code navigation tasks, ROSE recommends files to edit with the context of one changed file and yields a 0.33 F-score. Likewise, Ying et al. [6] developed a recommendation system that collects association rules for frequently changed files from the revision history provided by Mozilla, a web browser. The tool uses the frequent pattern tree algorithm [6] to recommend additional files to change. The system showed a recommendation accuracy similar to that of ROSE [3].

At about the same time, DeLine et al. [7] proposed TeamTracks, which mines consecutive visits between two program elements in developers' interaction history. Likewise,

Singer et al. [8] proposed NavTracks, which mines association rules in the sequence of program elements that exist in a navigation loop. Then, Lee et al. [1] developed NavClus, which uses a clustering technique to recommend frequently visited program elements in a navigation loop. Unfortunately, the three works [1,7,8] yield a lower recommendation accuracy than ROSE [3]. TeamTracks yields a 0.073 F-score and NavClus yields a 0.144 F-score. In addition, Zou et al. [9] defined three logical associations among files that frequently arise when developers switch from a file to another: co-edited files, co-viewed files, and edited and viewed files. Likewise, Maalej et al. [10] mined association rules by using method-level action information. However, the additional research work did not show yet a higher recommendation accuracy than that of ROSE.

Lee et al. broke the status quo by developing MI-EA [1], which takes the same association rule mining approach as ROSE did but elaborates the context with one edited file and three viewed files. MI-EA is a recommendation system that extends ROSE [3] but uses the developer's interaction history. In particular, MI-EA uses viewed files (or methods) as additional information to construct a context to recommend files to edit. As a result, with the more elaborate context augmented with viewed files, MI-EA yielded a 0.58 F-score. However, the recommendation accuracy of a 0.58 F-score is still insufficient for a practical use.

With respect to mining sequences in developers' interaction history, two previous works exist [11,12]. The first work [11] explored developers' workflow by mining frequent sequential patterns in developers' interaction histories and found out several developers' usage patterns. The second work [12] proposed applying temporal LDA to the interaction history to recommend commands in IDE. Because the first work understands developers' work patterns based on the sequential patterns and the second work used the temporal information to recommend commands, neither of the works are relevant to the code edit recommendation systems that we focus on in this paper.

In this paper, we propose a code edit recommendation system based on a recurrent neural network that utilizes the sequential information of developers' interaction history. We take a different approach from that of the typical association rule mining, which considers only the co-occurrence of items and thus misses the chronological order of visits to program elements. Our approach confirms that the interaction order information can enhance recommendation accuracy.

2.2. Tools for Collecting the Developer's Interaction History

Several tools have been developed to collect detailed developers' interaction history. Instead of simply recording the edits of program elements, those tools record various pieces of information about the operations that developers perform during software evolution tasks.

Kersten and Murphy developed Mylyn, an Eclipse plugin that records the interaction history of developers using Eclipse [4]. Mylyn categorizes developers' interactions into four broad categories: file opening, directory browsing, file reading, and file modification. It also records the reading and editing of files at the method-level granularity and thus stores very detailed operation information. Mylyn has been adopted by the issue tracking system provided by Eclipse to record and store the evolution information of several open-source projects related to the Eclipse IDE.

Kobayashi et al. [13] developed a tool called PLOG that records information related to developers' operations, such as file reading and file modification. It also records the standard output (stdout) and standard error (stderr) output when developers debug or build their programs.

Gu et al. [14] developed IDE++, which records developer interactions in the finest detail. IDE++ records a total of 44 types of interactions, which account for almost all of the operational information that can be collected from the IDE.

Among the data used by these tools, our research work analyzes the interaction history recorded by Mylyn [4] and stored in the Eclipse Bugzilla system, because it was used in the

previous state-of-the-art research work [1]. Maintaining the same data would make it easy to compare our approach with the previous approach [1].

2.3. Empirical Studies on the Developer's Interaction Histories

Recently, empirical studies on the relationship between edit events in interaction histories and the change events in revision histories have been conducted.

Soh et al. [15] noticed that several previous studies considered edit events in interaction histories as change events in software revision histories. Therefore, Soh et al. conducted a controlled laboratory experiment by asking 19 developers to work on 4 tasks. As a result, Soh et al. found false edit events included in the Mylyn interaction traces. Soh et al. also reported that the Mylyn interaction traces contain 28% more edit events than the video-captured interaction traces and that edit events that took less than 24 s were false edit events. Subsequently, Soh et al. reported that only 15% of the edit events in the Mylyn interaction traces are related to actual changes based on analyzing the results of their previous study in more detail [16]. Interestingly, Soh et al. also showed that eliminating noise and false editing events can help improve the accuracy of the recommendations made by MI-EA (56% improvement in precision and 62% improvement in recall).

The evaluation by Soh et al., however, was not statistically significant because the total number of edit events collected in the experiment was small. Additionally, they did not actually map the edited elements in the interaction history to the changed elements in the revision history. Yamamori et al. [17] concatenated the Mylyn data with the revision history of the projects associated with the Mylyn data by using bug IDs, author names, and creation dates. Interestingly, they showed that the concatenated data yielded an F-measure value that is 1.5 times higher than that yielded by the revision history. However, their F-measure value of 14% is very low compared to those of e-ROSE (35%) and MI-EA (63%). Additionally, it is not guaranteed that their method correctly mapped the interaction history to the change history.

These studies have revealed that edit events in the interaction history can have more edit events than the edit events in the revision history and that removing noisy edit events can improve recommendation accuracy. However, as described in the qualitative evaluation by Soh et al. [16], Mylyn edit events are still important events for developers. Thus, our approach increases the recommendation accuracy by using a deep learning technique instead of manipulating the edit events. We believe that our approach and that of Soh et al. [16] can be complementary to each other.

2.4. Research on Using Artificial Neural Networks for Recommendations

Recommendation systems are currently applied in various fields, and studies to enhance the accuracy of recommendation systems are drawing much attention. Of these studies, the most notable are the studies using artificial neural networks.

Felden et al. [18] studied various machine learning techniques, including artificial neural networks, to recommend documents, such as product descriptions and test reports, in Internet shopping. Alvarez et al. [19] conducted artificial neural network-based research using movie contents and relationships as input in order to improve the accuracy of movie recommendation systems. Hidasi et al. [20] proposed converting session-based data into a mini-batch format for RNN. To deal with short session-based data, Hidasi et al. claimed that RNN, which can handle sequences, should be applied. Their approach has session-based data in common with ours, even if the two domains are different. Wu et al. [21] proposed a deep RNN-based recommendation using data from web history to provide a customized recommendation service to an e-commerce system. Pei et al. [22] proposed the interacting attention gated recurrent network, which learns users and item scores in an interactive way to learn the dependencies between the users and the items that form interactions. He et al. [23] proposed an RNN-based similarity model that replaces the cosine similarity and Pearson coefficients used in the existing collaboration filtering. Rakkappan et al. [24] proposed stacked RNN, which stacks multiple recurrent hidden layers so that different

timescales in each layer are captured. However, the dynamics of the input and contexts were not modeled in these existing methods.

Deep learning techniques have been applied to the software engineering field, too. Lee et al. [25] showed that a convolution neural network (CNN) can be used to recommend which developer should be responsible for an issue report. Wen et al. [26] utilized RNN for defect prediction and identified six different sequence patterns of changes to a file in software revision histories. Kurtukova et al. used a combination of CNN with BiGRU to identify an author of source code [26].

However, to the best of our knowledge, there is no prior research work that applies a recurrent neural network to developers' interaction histories other than our previous study [5], where we applied an RNN to developers' interaction history but could not compare it with MI-EA as the experimental setting in the latter was different. In this paper, we set up an almost similar experimental setting and compared the recommendation accuracy of CERNN with that of MI-EA.

3. Preliminary Experimental Results with N-Gram and RNN

We explored the methods that can be used to utilize the sequential information of developers' interactions with IDE to elaborate the context for recommendation and found two effective models: N-Gram and RNN. While N-Gram is a basic model for sequential data, RNN is an up-to-date order-based technique. In the following, Section 3.1 explains the concept of the N-Gram model, Section 3.2 explains the concept RNN model, and Section 3.3 explains the preliminary experimental results with N-Gram and RNN.

3.1. N-Gram Model

The N-Gram model uses the Markov assumption. The Markov assumption means that we can predict the probability of a word appearing in a sentence based on the previous words. N-Gram means a sequence of N words, where N can be replaced with a specific number, such as 2, 3, or 4. For example, "turn your homework" is a 3-g because it consists of three words. In this case, the probability of "homework" that will appear after "turn your" can be found [27]. The probability is calculated as follows:

$$\mathrm{P}(y/x) = \mathrm{C}(x \cap y)/\mathrm{C}(x) \tag{1}$$

That is, the probability of y that will appear after x is calculated as the number of times that x and y appear together compared to the number of times that x appears in a sentence.

3.2. Recurrent Neural Network

A recurrent neural network (RNN) is a type of neural network that specializes in processing sequential data [28,29]. Sequential data are ordered data that can be represented as $(x_1, x_2, \ldots, x_n)$. At time t, upon input x_t, the state of this network is changed from the previous state h_{t-1} to the current state h_t.

The recurrent neural network (RNN) is expressed as the formulae (2) and (3). RNN reads the previous state h_{t-1} and the current input feature x_t and makes a prediction that calculates the current state h_t. In (2) and (3), W_x, W_h, and W_o are weight matrices that are multiplied by the variables x_t, h_{-1}, and o_t, respectively. The values o_t and h_t are the same. Here, information on the previous input is maintained by entering the previous state h_{t-1}:

$$h_t = \tanh\left(W_x x_t + W_h h_{t-1}\right) \tag{2}$$

$$o_t = W_y h_t \tag{3}$$

There are several variations of the basic RNN. Among them, the most widely used are long short-term memory (LSTM) [30] and gated recurrent unit (GRU) [31]. Long short-term memory (LSTM) is capable of learning so-called long-term dependencies [31], but the problem with long-term dependencies is that a simple RNN does not learn from previous information if the information is far from the time point where it is needed. Gated recurrent

unit (GRU) is a simpler variant of LSTM [32] and it is known that the performances of GRU and LSTM are similar to each other but differ case by case.

3.3. Preliminary Experimental Setup and Results

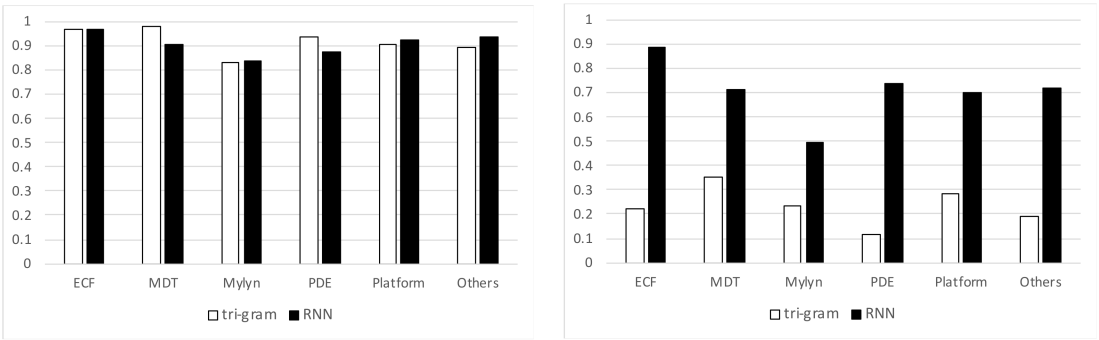
We had the N-Gram and RNN models predict the files to edit. MI-EA, which is our comparison target, formed its context with three viewed elements and one edit element in the previous experiment [1]. Therefore, we considered similar size contexts when we experimented with the N-gram model and RNN.

With the N-gram model, recommendations occur only when the same contextual input as the learned data is given. We varied N in our experiment and found that the bi-gram model yields significantly low precision, which is caused by indiscriminate recommendations, while the models that use more than four grams yield significantly low recall due to the unique contexts being used. We thus decided to use the tri-gram model that yields the best F-score value for our experiment.

The RNN model is expected to recommend files to edit even if developers' three actions in sequence do not exactly match past interaction histories. The RNN model is thus expected to improve both precision and recall values. We experimented with two specific RNN models, LSTM and GRU. Through our experiment, we found that GRU yields slightly higher accuracy than LSTM. We set up the hyperparameters of the GRU model as follows. We set the hidden size to 500 and the epoch, which is the number of repetitions of training, to 100. We also repeated five experiments for the RNN model per project and reported the average of the results of a total of five experiments.

In the experiment, our simulator read all of the interaction traces per project and listed the traces in chronological order. It then created pairs of two viewed files and one edited file. It randomly selected and used 70% of the pairs for training of the models and 30% for testing. We repeated five experiments for the N-gram model per project and reported the averages of the results from the five experiments.

Figure 1 shows the results from the preliminary experiments. As shown in Figure 1a, the precision values of the N-Gram model are high. However, as shown in Figure 1b, the recall values of the N-Gram model are significantly low. In Figure 1a, the precision values of the RNN model are comparable to those of the N-Gram model. In Figure 1b, the recall values of RNN are significantly higher than those of N-Gram.



(**a**) Precision (**b**) Recall

Figure 1. The preliminary experimental results with N-Gram and RNN, Reprinted with permission from ref. [5], Copyright 2018 Korea Information Processing Society.

From the preliminary results of our experiment, we found that the RNN model produces more accurate recommendation results than the N-Gram model. However, this experiment has some limitations in comparing the experimental results with the results of past work. First, task boundaries identified by interaction traces are not considered when extracting data from the interaction history. Simply three viewed elements and one edited element are extracted from a sequence of developers' interactions. In contrast, the previous

studies used task boundaries, where only the edited elements in each interaction trace are regarded as an answer set. Second, we divided the data into 70% and 30% of it and used 70% as the training data, and 30% as the testing data in the experiment. In contrast, the previous studies [1,3] used on-line learning evaluation methods in which each interaction trace is used as a testing set and then is added to the training data. Because interaction traces are accumulated over time, online learning is a practical evaluation method that considers the temporal characteristics of interaction traces. Due to these differences between the previous studies and ours, such as task boundaries and experimental methods, we could not directly compare our preliminary results with the recommendation accuracy of the previous studies [1,3].

In this paper, we extracted data by considering the task boundaries of interaction traces. We also set up our experiment by adopting an online learning evaluation method. We set up the evaluation environment close to the one that was used in the previous studies as much as possible. In the set-up, we found that LSTM yielded a higher accuracy than GRU. Therefore, we present our recommendation method with the adopted LSTM model.

4. The Code Edit Recommendation Method Based on a Recurrent Neural Network

In this section, we present our code edit recommendation method using a recurrent neural network (CERNN), which takes a context as an input and recommends multiple files to edit based on the context. This section consists of the following subsections. In the following, Section 4.1 describes the architecture of CERNN. Section 4.2 first explains how to preprocess the interaction traces. Section 4.3 explains how to form a context from the interaction traces in the recommendation system. Section 4.4 explains how to construct a model with training data. Finally, Section 4.5 explains how CERNN recommends code edits.

4.1. Architecture of CERNN

Figure 2 shows the architecture of the code edit recommendation system using a recurrent neural network (CERNN).

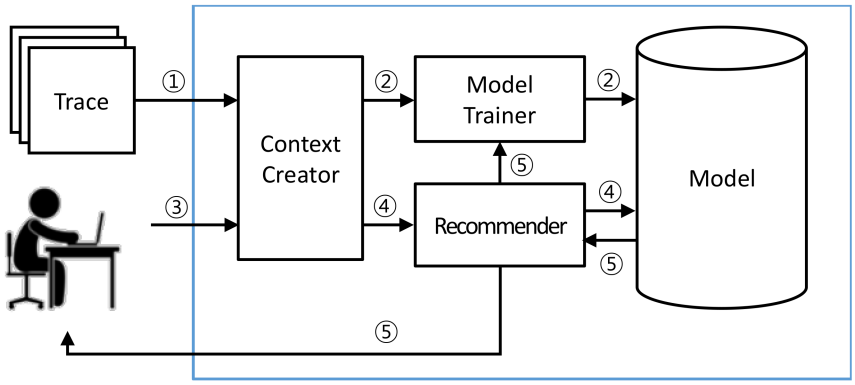


Figure 2. Architecture of the code edit recommendation system using a recurrent neural network (CERNN).

In Step 1 of Figure 2, *Context Creator* preprocesses the interaction traces and creates contexts in advance from the interaction traces by using a sliding window. These contexts become the input data for training the RNN. In Step 2, *Model Trainer* receives the input data and trains the RNN. Based on the training, *Model Trainer* creates a training model for a code edit recommendation and stores the model. In Step 3, when a developer performs operations during a software evolution task, *Context Creator* records the developer's actions and creates a context. In Step 4, *Recommender* receives the context from *Context Creator* and passes it to the model generated by *Model Trainer*. The model takes the context as an input and derives code edit recommendations at the file level. In addition, *Recommender* passes the context to *Model Trainer*, allowing *Model Trainer* to update the model. In Step 5,

Recommender displays the code edit recommendations to the developer who is working on a software evolution task. The following sections describe these steps in detail.

4.2. Preprocessing Interaction Traces

An interaction trace is a log file recorded while a developer interacts with a code base during a software evolution task. Figure 3 shows an example of the interaction data recorded from a user's actions, such as viewing or editing source files. In the example, one of the interaction traces recorded the history that a developer viewed Java files A and B, edited Java file A, and viewed files C.

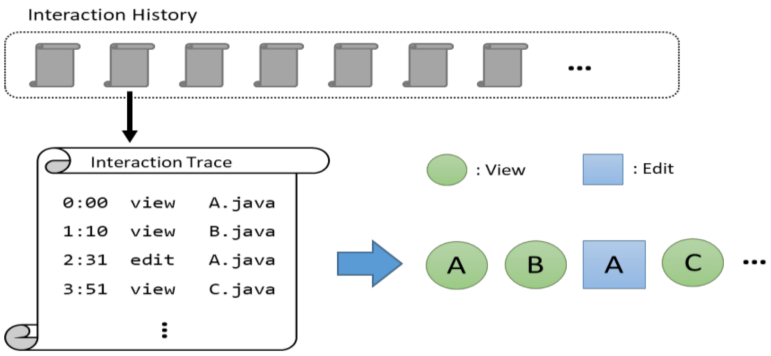


Figure 3. The process of generating an interaction history.

The interaction history can be defined as a set of interaction traces as follows:

$$\textit{Interaction History} = \{T_1, T_2, \ldots, T_n\}\ (T_i \text{ is sorted chronologically})$$

The interaction trace is defined as a set of records describing the actions performed by a developer in the corresponding software evolution task as follows:

$$T_i = \{a_1, a_2, \ldots, a_n\}\ (a_j \text{ is sorted chronologically})$$

The element a_j of T_i denotes the jth operation in interaction trace T_i. The element a_j can be defined as a pair as follows:

$$a_j = (kind, target),\ kind \in \{edit, view\}$$

The *kind* element of a_j is the type of operation. If a developer edits a program element, the *kind* element takes the value "*edit*". If a developer navigates a program element, the *kind* element takes the value "*view*". Another element *target* is the name of the program element that was edited or viewed as the target of the operation.

The sequence data collected by Mylyn may include duplicate events. So, as a preprocessing step, duplicate events are removed from the interaction traces. Figure 4 illustrates how duplicate events are removed. CERNN constructs a context using a sliding window that maintains the temporal order of elements and eliminating duplicate events leaves only essential information in the context, thereby facilitating the learning process of the training model.

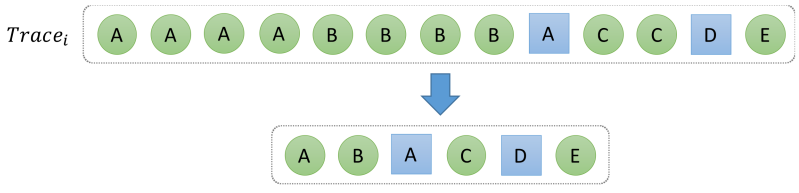


Figure 4. Elimination of duplicate events.

4.3. Creating Contexts

The dictionary definition of the term *context* is "the information used to capture the current situation". In a recommendation system, the term *context* refers to "a query that searches the data to recommend information that will be useful in the user's current situation" [2]. In a code edit recommendation method, a context is formed by the files most recently viewed or edited while a developer is performing an operation (edit, view) on several program elements during a software evolution task. The context plays a role of a query to find the interaction traces that have edited files in the similar contexts.

Context Creator plays a role in Steps 1 and 3 of Figure 2. Context Creator preprocesses the interaction traces and creates contexts in advance in Step 1 and creates a context from a developer's actions in Step 3. By matching the two contexts generated respectively from Step 1 and Step 3, CERNN makes a recommendation. Section 4.3.1 explains how a context is created in MI-EA and Section 4.3.2 explains how a context is created in CERNN.

4.3.1. Context Formation in MI-EA

Lee et al. proposed a recommendation system called MI-EA [1]. The system mines the association rules of viewed and edited files in programmer interaction histories and recommends the files to edit based on the viewed files. In the recommendation system, MI-EA is one of the methods that forms a context by combining viewed and edited files at the time point when a programmer edits a file. A context formed in MI-EA [1] is the set of the last n files that a developer viewed together with the set of the last m files that the developer edited. A context is expressed as follows:

$$C = (Vc, Ec) \tag{4}$$

where $Vc = \{v_1, v_2, \ldots, v_n\}$ denotes the set of n files most recently viewed by the developer, and $Ec = \{e_1, e_2, \ldots, e_m\}$ represents the set of m files most recently edited by the developer.

Figure 5 shows an example of context creation from an interaction trace in the MI-EA process. If MI-EA creates a context consisting of three viewed files and one edited file from the operations in an interaction trace, it uses the sliding window of size (3-1) to fill up the three viewed files and the one changed file. In the example, the set of the three viewed files is $\{A, B, C\}$ and the set of the changed file is $\{A\}$.

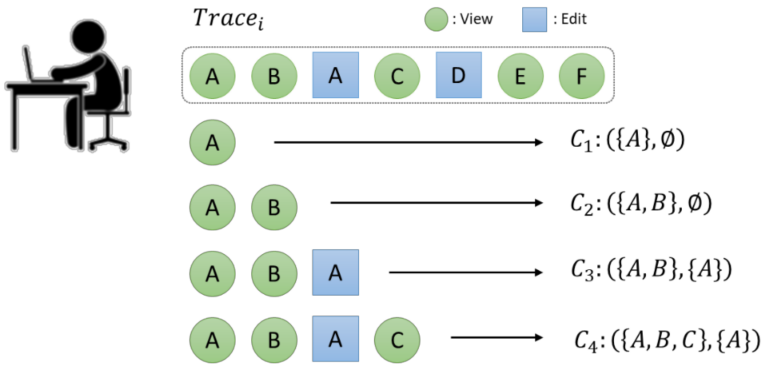


Figure 5. An example of the MI-EA process for creating a context of 3 views and 1 edit.

The recommendation system recommends files to edit based on the context formed by MI-EA. In the example of Figure 5, the context formed is C_4 = ({*A*,*B*,*C*}, {*A*}). In the case, the recommendation system recommends files edited in the interaction traces that include the same viewed and edited files as C4 maintains.

4.3.2. Context Formation in CERNN

The context formation of CERNN is one of the main differences from MI-EA [1] in whether the chronological order of the operations is maintained in the context. Because

the context of MI-EA is a set of operations, the context does not maintain the order of the viewed or edited elements. Meanwhile, the context of CERNN is a sequence of operations that can maintain the order of viewed or edited elements.

Figure 6 shows an example of context creation from an interaction trace with a CERNN process. First, a sequence of four events is selected from the beginning of the interaction trace, which is [A, B, $\underline{A}$, C], to create a context that maintains the sequence. Next, by moving the sliding window one step to the right, another sequence of four events [B, $\underline{A}$, C, $\underline{D}$] is selected to create another context. In this way, the sliding window can be moved up to the end of the interaction trace, creating all the contexts.

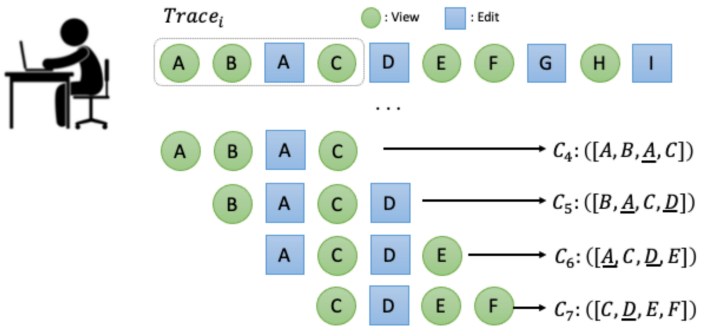


Figure 6. Example contexts created by setting the size of the sliding window to 4 based on a CERNN process.

We designed CERNN so that it utilizes the order of elements and thus creates more elaborate contexts for recommendation. In addition, by this context formation the context of CERNN can contain more than two edited elements. Typically, an edited element is a stronger indicator for driving correct edit recommendations than a viewed element. Even if the case where a context has two edited elements may be rare, such a case could impact recommendation accuracy. Finally, a context is formed only when an edited element is included in it, which will also affect recommendation accuracy. However, this is not a difference between CERNN and MI-EA, because, in MI-EA, a context must also contain an edited element.

4.4. Training a Model

The training steps of CERNN consist of generating learning data and training a model with the data. Section 4.4.1 describes how to generate training data and Section 4.4.2 describes how to train a model.

4.4.1. Generating Training Data

CERNN takes interaction traces as training data to train a model. To generate training data from interaction traces, CERNN uses a sliding window to create a context and pairs up the context with edit recommendations. To create a context, CERNN uses the context creation method described in Section 4.3.2. CERNN extracts the contexts as well as the files edited after the context. The files to be recommended to a developer for editing are the files edited after the context but not those included in the context.

Figure 7 shows the process of generating training data. The training data consist of pairs of contexts and edit recommendation files corresponding to the context. In the example of Figure 7, the first training pair becomes ({A, B, $\underline{A}$, C}, {D, G, I}). However, to train a model, each file should be expressed in number. Therefore, CERNN converts the training data to numbers and indexes them by using the one-hot-encoding technique in which each location in a vector represents each file.

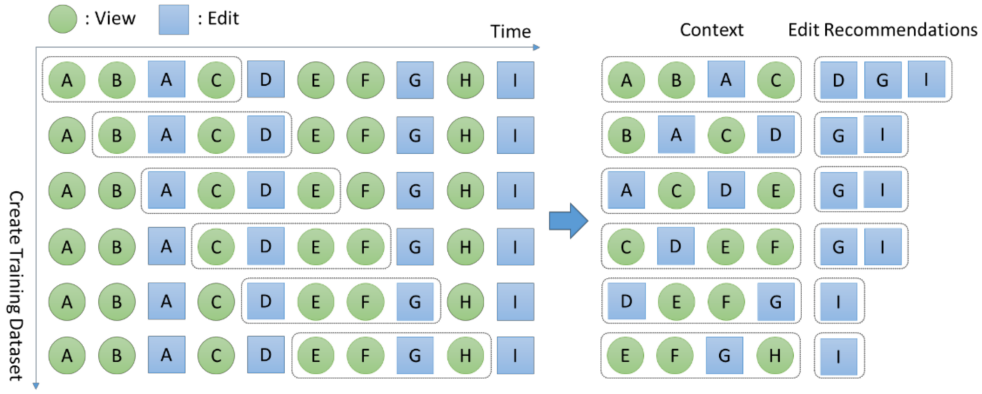


Figure 7. Sliding window method for generating training data.

4.4.2. Applying an LSTM model

To train a model with the training data, CERNN uses the model shown in Figure 8. The model has two layers: the embedding layer and the LSTM layer. In Figure 8, the create_model function obtains the number of files and the number of categories (i.e., the number of edited files) as an input and returns the created model.

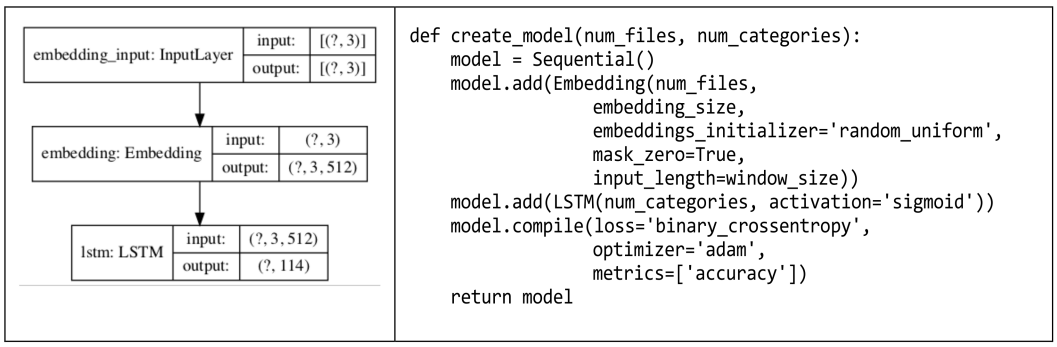


Figure 8. CERNN model for the learning process.

In the embedding layer, CERNN reduces the dimensions of the indexed training data set. The embedding layer transforms a sparse vector into a dense vector. If CERNN directly indexes files in various open-source projects, the training time would be very long. Therefore, using the embedding layer to reduce the dimensions while preserving the characteristics of the data is instrumental to reducing the training time.

Then, CERNN passes the training data reduced through the embedding layer to the LSTM layer. In the LSTM layer, CERNN creates an LSTM model. As shown in Figure 7, CERNN should select multiple output values for the recommendation of several files to be edited for a single context. To construct and utilize such a multi-label model, CERNN uses the sigmoid function as an activation function to derive the output values from the LSTM layer. (An LSTM can be one of the four different models. The first one is a binary category model in which the output value is one of two options. The second one is a multi-category model that selects one of several output values. The third one is a multilabel model in which multiple output values are selected. The final one is a sequence generation model, which shows a continuous output result. To implement a different model, a different activation function should be chosen. For example, to implement a multi-category (or multi-class) model that selects one of several output values, the softmax function should be used as an activation function. In our case, we intend to implement a multilabel model and so we chose the sigmoid function)

Finally, CERNN constructs the settings for the training process. Regarding the loss function, we chose 'binary_corssentropy' because we classified a context into edited files.

For the optimizer, we chose 'adam.' We tried other optimizers (e.g., 'sgd' and 'adagrad') to check whether they could improve the recommendation accuracy or reduce the training time while maintaining the same accuracy. However, in our experiment, the 'adam' optimizer showed the best accuracy. Therefore, we decided to use it.

4.5. Making a Recommendation Based on the Trained Model

Once a model is constructed with the training data, CERNN can recommend files to edit by getting a developer's actions and forming a context. Sections 4.2 and 4.3 already described how to record developers' actions and form a context. Once a context is formed, the context becomes an input to recommend files to edit. With the input and the trained model, CERNN recommends files to edit to the developer.

In addition, we suggest considering user interactions for recommendations. If a developer marks a given recommendation as incorrect, s/he can make the recommender stop recommendations for the task. Therefore, we add an option of stopping recommendations when the first edit is found to be false in a given recommendation. The option is based on our observation in an experiment where we observed that the recommendations in the same task (i.e., the same interaction trace) yield similar accuracy (low recommendation accuracy or high recommendation accuracy). We will show that recommendation accuracy is higher in the case of stopping recommendations when the first recommendation is found to be incorrect for a given evolution task in the following sections.

5. Evaluation Setup

This section is organized as follows: Section 5.1 defines the research questions; Section 5.2 describes the target projects for which the interaction histories were simulated; Section 5.3 defines the experimental procedure; and Section 5.4 defines the evaluation criteria.

5.1. Research Questions

We evaluated whether our proposed approach improves recommendation accuracy over existing approaches. We select two research questions.

RQ1: What is the recommendation accuracy of CERNN, compared with the state-of-the-art approach MI-EA, in the case that it keeps recommending files to edit even when the first recommendation is incorrect?

RQ2: What is the recommendation accuracy of CERNN, compared with the state-of-the-art approach MI-EA, in the case that it stops recommending files to edit when the first recommendation is found to be incorrect?

5.2. Experimental Targets

The Eclipse Bugzilla system collects interaction traces by using a Mylyn plugin tool, and the system stores the interaction traces as an XML file. Figure 9 shows an example of an interaction trace where each interaction event record specifies its kind, "selection" or "edit", and identifies the *StructureHandle* attribute, which contains the name of the target file.

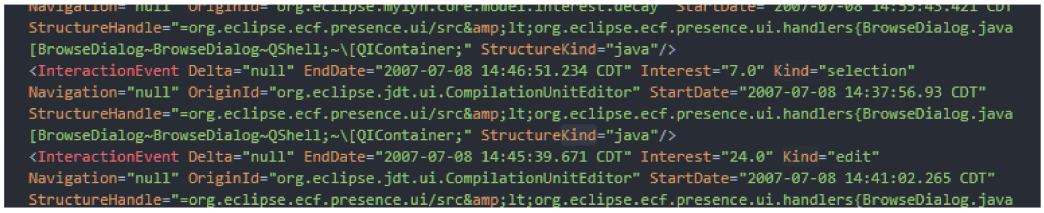


Figure 9. An example of Mylyn data.

The Eclipse Bugzilla system has collected more than 1600 interaction traces from multiple open-source projects over several years, and MI-EA used the interaction traces

for its evaluation. We used the same interaction traces (the collected interaction traces can be downloaded from http://salab.kaist.ac.kr/tse2015/AllProjects.zip (accessed on 30 September 2021)) in which more than 6000 files were recorded. Table 1 shows the project-specific statistics. For example, there are 2726 interaction traces for the Mylyn project. The 2726 interaction traces contain 46,732 interaction events.

Table 1. Target projects for experiments.

Project	Description	Duration	#Traces	#Events
Mylyn	A task management tool for programmers	18 May 2016–7 July 2011	2726	46,732
Platform	A set of frameworks for Eclipse	18 October 2007–25 May 2011	582	15,662
PDE	Eclipse's plugin development environment	11 November 2007–25 February 2011	536	3991
ECF	Eclipse communication framework	6 April 2007–18 June 2011	308	3091
MDT	The model development tools	12 February 2009–23 June 2011	262	13,523

Mylyn data have the kind attribute, indicating the type of operation performed by the developer in a software evolution task. The kind attribute has several different values; however, we constructed our data by using only the "view" and "edit" values of the kind attribute.

5.3. Experimental Procedure

For our experiment, we implemented a simulator. The source code of the simulator is available at https://github.com/saleese/cernn (accessed on 30 September 2021) (We ran our experiment with Tensorflow version 2.1). Section 5.3.1 explains the online learning simulation and Section 5.3.2 explains the set-up of hyperparameters. Sections 5.3.3 and 5.3.4 explain the experimental procedure for research questions RQ1 and RQ2, respectively.

5.3.1. Simulation

To evaluate CERNN, we chose the online learning evaluation method that our previous work [1] used to evaluate MI-EA. The online learning evaluation method simulates training and testing in a manner similar to actual software project evolution processes. Figure 10 shows how the online learning evaluation method forms the training and test sets. In the first iteration, the simulator trains a model with the data from the first interaction trace (i.e., $Trace_1$). Based on this trained model, the simulator evaluates the recommendations by using the second interaction trace (i.e., $Trace_2$) as test data. In the second iteration, the simulator trains the model with the data from the first and second interaction traces (i.e., $Trace_1$ and $Trace_2$), and evaluates the recommendations in the third interaction trace (i.e., $Trace_3$). In this way, the simulator creates a model using the traces from $Trace_1$ to $Trace_{i-1}$ and tests the model with $Trace_i$ by gradually iterating through the interaction traces from $Trace_1$ to $Trace_n$.

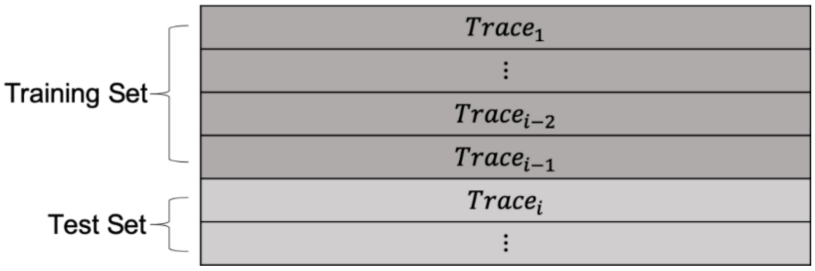


Figure 10. Training set and test set.

Since previous researchers evaluated their approaches with online learning [1,3], we set up our experimental evaluation in the same way as the previous work [1,3] did. However, it was difficult to find a way to evaluate a deep learning technique by using an online learning evaluation method, although, for an ever-increasing size of data, online learning is a more practical way of learning from data than the traditional machine learning evaluation method, which is the 10-fold (or full) cross validation method.

5.3.2. Hyperparameter Set-Up

In CERNN, we have five main hyperparameters as shown in Table 2. Regarding the sliding window described in Section 4.3.2, we set up the window size to 3 (W = 3). Additionally, there is the embedding size that determines the size of the vectors that are used to convert program elements into several numeric values. We set up the embedding size to 512. Next, the batch size represents the size of the training data to be used at once. Regarding the batch size, it is generally known that 32 is approximately the most effective value. Therefore, we set up the batch size to 32. Then, the epoch is the number of training repetitions. We repeatedly changed the epoch value from 100 to 600, increasing it by 50. From the experiment, we selected one of the optimal values, 500, for the epoch value. Finally, a threshold is used to determine which program elements should be recommended. We changed the threshold value from 84 to 95, increasing it by 1, and selected one of the optimal values, 91, for the threshold.

Table 2. Values of hyperparameters.

Hyperparameter	Default Value
window_size	3
embedding_size	512
batch_size	32
epoch	500
threshold	0.91

The reasons why we set up the values of hyperparameters as in Table 2 are as follows. In order to compare our approach CERNN with MI-EA [1], we set the representative window size to 3, which is the same size as Lee et al.'s [1]. Regarding the epochs and the threshold, we varied their values and observed the trends of the recommendation accuracy. Figure 11 shows the recommendation accuracy obtained from the five projects mentioned earlier, by varying the epochs value changes from 100 to 600, increasing it by 50. The curve in Figure 11 shows that the recommendation accuracy is high when the number of epochs ranges from 400 to 600. We also checked the weighted average recommendation accuracy of the five projects and found that the curve is close to flat from 100 to 600. Therefore, we concluded that our set-up of the epochs value to 500 is reasonable.

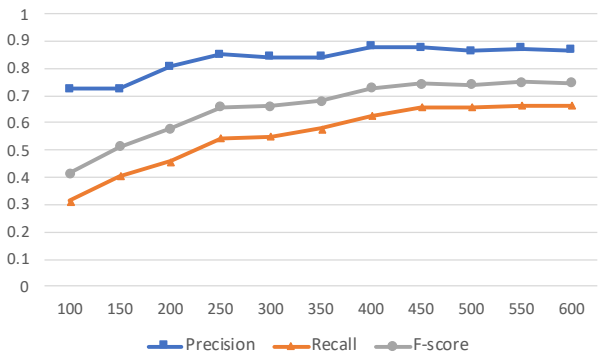


Figure 11. Average recommendation accuracy from varying epochs value with five projects.

Figure 12 shows the average recommendation accuracy of the five projects, by varying the threshold value from 84% to 95%, increasing it by 1%. The curve in Figure 12 shows that the recommendation accuracy (F-score) is highest from 0.91 to 0.94. We also checked the weighted average recommendation accuracy of the five projects, and we found that the highest points are at 91 and at 93. Therefore, we found that our set-up of the threshold value to 0.91 is reasonable.

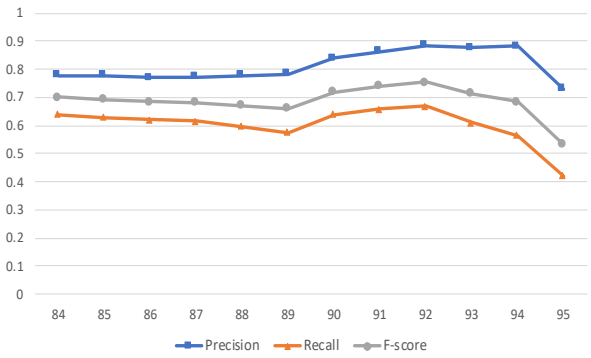


Figure 12. Average recommendation accuracy from the varying threshold value with five projects.

5.3.3. Evaluation for RQ1

To evaluate RQ1, we compared the recommendation accuracy of CERNN with that of MI-EA by performing the on-line learning simulation described in Section 5.3.1. Because MI-EA showed the highest performance among the previously described approaches, we chose MI-EA for this comparison. The MI-EA approaches can be divided into two types based on the time at which the recommendations are made. The first type (i.e., MI-EA) makes recommendations when a developer edits a file. The second type (i.e., MI-VA, MI-VO, and MI-VOA) makes recommendations when a developer browses or edits files. We compared CERNN with the MI-EA model in which the recommendations are made when a developer edits a file. To compare CERNN with MI-EA, CERNN makes recommendations with a context that includes an edited file at the end of a sequence of elements in the context.

In addition, we made some modifications in our experiment. When a name is split up by "-", the last part of the number is related to the time when the interaction trace was registered in the Bugzilla system. Therefore, we sorted the interaction traces by the last part of the number before evaluation. For example, "ECF-192778-73274.xml" is the name of the interaction trace used in this experiment. Therefore, the results of MI-EA in this paper are slightly different from those of the previous paper [1].

5.3.4. Evaluation for RQ2

To evaluate RQ2, we compared the recommendation accuracy of CERNN with that of MI-EA, as we did for RQ1. The difference from RQ1 is that we made CERNN stop recommendations when the first recommendation is found to be incorrect in the simulation. It is from our observation that the recommendation results in each task (i.e., interaction trace in our simulation) show similar recommendation accuracy. Once a recommendation yields low accuracy, the remaining recommendations in the same task (i.e., interaction trace) yield low accuracy. Thus, we conjecture that a user could make CERNN stop recommendations when the user found that the given recommendation is not helpful. We applied our conjecture to our experiment for RQ2.

5.4. Evaluation Metrics

To compare the recommendation accuracy of the CERNN model with that of MI, we used the metrics of precision, recall, and F1 score. These metrics are widely used to evaluate the accuracy of recommendation approaches and were also used in a previous MI-EA accuracy measurement experiment [1].

These metrics are defined as follows:

$$\text{Precision}: \ P = \frac{|E \cap R|}{|R|} \ \text{Recall}: \ R = \frac{|E \cap R|}{|E|} \ \text{F1 score}: \ F1 = \frac{2PR}{P+R}$$

where E is the set of files actually edited in one interaction trace and thus regarded as an answer set and R is the set of files recommended by the recommendation system.

The above metrics measure the accuracy of a recommendation that occurs within one interaction trace of a project. Recommendations can occur multiple times within an interaction trace. According to the procedure described in Section 5.3, recommendations also occur multiple times from the second trace (T_2) through the last interaction (T_n). We measured precision, recall, and F1 score for all the recommendations and then averaged the results to quantify the accuracy of all recommendations from a project.

6. Evaluation Results

This section explains the experimental results for RQ1 and RQ2.

6.1. RQ1: Recommendation Accuracies of CERNN and MI-EA

We compared the recommendation accuracy of our CERNN approach with that of the state-of-the-art approach MI-EA. The main difference between CERNN and MI-EA is that CERNN maintains sequential information related to developers' operations and uses a deep learning technique, LSTM.

To compare the accuracies of CERNN and MI-EA under similar conditions, we chose 3 as the size of the sliding window. The size of 3 in CERNN means that the sliding window maintains three sequentially viewed or edited elements. Because recommendation occurs at an edit event, one edited element is necessary in the context. The sliding window used in MI-EA maintains three viewed elements and one edited element. Therefore, the configuration that sets the size of the window to three in CERNN is similar to the configuration of the sliding window in MI-EA.

Table 3 shows the recommendation results made by CERNN and MI-EA. The last two rows of Table 3 show the averaged precision, recall, and F1 score of the projects, Mylyn, Platform, PDE, ECF, and MDT. The second last row shows simple averages, where CERNN yielded 0.63 precision, 0.48 recall, and 0.54 F1-score with the five projects. The last row shows weighted averages by the numbers of recommendations, where CERNN yielded 0.57 precision, 0.38 recall, and 0.45 F1-score with 8307 recommendations.

Table 3. Comparison of the recommendation accuracies of CERNN (W = 3) and MI-EA.

Project	CERNN (W = 3)				MI-EA			
	P	**R**	**F1**	**#R**	**P**	**R**	**F1**	**#R**
Mylyn	0.53	0.35	0.41	6312	0.79	0.62	0.69	1096
Platform	0.71	0.47	0.57	1647	0.85	0.34	0.48	977
PDE	0.46	0.35	0.40	167	0.52	0.59	0.55	144
ECF	0.48	0.41	0.44	78	0.39	0.58	0.46	45
MDT	1.00	0.81	0.90	103	1.00	0.41	0.58	19
Avg./Total	0.63	0.48	**0.54**	8307	0.71	0.51	**0.55**	2281
Weighted Avg./Total	0.57	0.38	**0.45**	8307	0.79	0.50	**0.59**	2281

* P denotes Precision, R Recall, F1 F-score, and #R the number of recommendations.

In contrast to our expectation, CERNN did not yield higher recommendation accuracy than MI-EA. While the averaged precision of MI-EA is 0.71, that of CERNN is 0.63. While the averaged recall of MI-EA is 0.51, that of CERNN is 0.48. Therefore, while the averaged F1 score of MI-EA is 0.55, that of CERNN is 0.54. The F1-score of CERNN is at least 1% lower than that of MI-EA.

When we considered the number of recommendations per project and calculated the weighted averaged precision, recall, and F1-score, the gap between CERNN and MI-EA becomes larger. As shown in the last row of Table 3, the weighted average precision of CERNN is 0.57, while that of MI-EA is 0.79. The weighted average recall of CERNN is 0.38, while that of MI-EA is 0.50. The weighted average F1 score of CERNN is 0.45, while that of MI-EA is 0.59. In this case, the F1-score of CERNN is 14% lower than that of MI-EA. This is mainly because a large number of recommendations occur in the Mylyn project.

6.2. RQ2: Recommendation Accuracies of CERNN and MI-EA When Stopping Recommendations if the First Edit Is Found to Be False

As we explained in Section 5.3.4, we observed that the recommendations occurring in the same interaction traces maintain similar recommendation accuracy. Therefore, we decided to stop recommendations when the first edit in an interaction trace was found to be false in our simulation. Now, the main difference between CERNN and MI-EA is that CERNN maintains sequential information related to developers' operations, uses a deep learning technique, and uses the result of the first edit at each interaction trace.

Table 4 shows the recommendation results made by CERNN and MI-EA. In Table 4, the second last row shows simple averages, where CERNN yielded 0.80 precision, 0.60 recall, and 0.69 F1-score with five projects. The last row shows weighted averages by the numbers of recommendations, where CERNN yielded 0.79 precision, 0.54 recall, and 0.64 F1-score with 4987 recommendations.

Table 4. Comparison of the recommendation accuracies of CERNN (W = 3) and MI-EA when stopping recommendations if the first edit is found to be false.

Project	CERNN (W = 3)				MI-EA			
	P	**R**	**F1**	**#R**	**P**	**R**	**F1**	**#R**
Mylyn	0.77	0.52	0.62	3728	0.79	0.62	0.69	1096
Platform	0.86	0.59	0.70	995	0.85	0.34	0.48	977
PDE	0.65	0.49	0.56	114	0.52	0.59	0.55	144
ECF	0.73	0.58	0.65	47	0.39	0.58	0.46	45
MDT	1.00	0.82	0.90	103	1.00	0.41	0.58	19
Avg./Total	0.80	0.60	**0.69**	4987	0.71	0.51	**0.55**	2281
Weighted Avg./Total	0.79	0.54	**0.64**	4987	0.79	0.50	**0.59**	2281

* P denotes Precision, R Recall, F1 F-score, and #R the number of recommendations.

Reviewing the precision results first, as shown in the second last row of Table 4, the averaged precision of CERNN is 0.80, while that of MI-EA is 0.71. The precision of CERNN is 9% higher than that of MI-EA. However, as shown in the last row of Table 4, the weighted average precision of CERNN is 0.79, and that of MI-EA is the same. Therefore, we conclude that the precisions of the two models are not much different.

Concerning the recall results, the average recall of CERNN is 0.60, while that of MI-EA is 0.51. The recall of CERNN is 9% higher than that of MI-EA. The weighted average recall of CERNN is 0.54, while that of MI-EA is 0.50. The weighted recall of CERNN is 4% higher than that of MI-EA. In both cases, the recall values of CERNN are higher than those of MI-EA. Due to the recall values, the average F1 score of CERNN is 0.69, while that of MI-EA is 0.55. Additionally, the average F1 score of CERNN is 0.64, while that of MI-EA is 0.59. Thus, the F1-score of CERNN is at least 5% higher than that of MI-EA.

7. Discussion

In this section, we discuss why CERNN works, and then explain the additional experiments to evaluate the time performance of CERNN.

7.1. Why CERNN Works

The result of our experiments in Section 6.2 (for RQ2) shows that our CERNN model (called $CERNN_{v2}$) yields a 64% F-score, while MI-EA yields 59%, whereas the experimental result in Section 6.1 (for RQ1) shows that our CERNN model (called $CERNN_{v1}$) yields a 45% F-score. The main difference between the two cases is that $CERNN_{v2}$ stops recommendations when the first recommendation is found to be incorrect. We built $CERNN_{v2}$ with several differences. First, $CERNN_{v1}$ uses ordered contextual information. We expected that ordered contextual information can lead to more precise recommendations than unordered contextual information that is typically used in existing association rule mining methods. Second, CERNN $_{v2}$ uses a deep learning technique. Third, when we created data for $CERNN_{v2}$, we used the data that contained edit elements. Because the context around edit elements could be the information that might lead to the edit elements, using only the data that contain edit elements could positively affect the recommendation accuracy. However, the combination of these factors does not yield higher recommendation accuracy than MI-EA. Finally, we added one more difference to $CERNN_{v2}$ by making it stop recommending when the first recommendation in a task (i.e., interaction trace) is found to be false.

We also note that the answer set for the CERNN is slightly different from the answer set for the MI-EA. For CERNN, which keeps the order of records in interaction traces, only the files edited after the recommendation timepoint in the answer set are included. For MI-EA, on the other hand, the files edited over the entire interaction trace in the answer set are included. We conjecture that this decision would work negatively on the performance of CERNN, that is, the number of edited files in an answer set for CERNN would be smaller than that for MI-EA, because CERNN has fewer candidates than MI-EA in the answer set. However, we adopted this decision in our evaluation, because it is more reasonable to predict the next element to be edited. Because of this decision, we also trained CERNN by regarding the edited elements after a recommendation timepoint as an answer set. When generating the learning model, CERNN assigns only the edited files that appear after the last event of the context as files to recommend whereas MI-EA assigns all of the edited files existing in the interaction trace as the files to recommend. In the case of a model that emphasizes the sequence of interactions, it is reasonable that files edited before a context are not regarded as correct. However, because CERNN considers fewer edited files than MI-EA, it could negatively affect the recommendation accuracy of CERNN, compared to MI-EA. That is, if the number of edited files is smaller, the number of recommendation candidates will be smaller for the recommendations made in a similar context.

7.2. Time Performance of CERNN

In the experiment, we measured the execution time of CERNN spent on training and testing on-line learning evaluation methods. To measure the time, we used a computer with a 2.2-GHz processor, 128 GB of memory, and 2 TB of storage. Table 5 presents the execution time of CERNN for each project.

Table 5. The execution time of CERNN (dd:hh:mm:ss).

Project	Mylyn	Platform	PDE	ECF	MDT
Time	45:50:50:37	22:25:30	5:35:47	33:46	15:48

In our online learning evaluation, we made a simulator create a training model at each increment. Therefore, the time complexity is exponential. It is why MDT took 15 min 48 s, while Mylyn took 45 days 50 h 50 min and 37 s. We think the time performance of CERNN could be improved in three ways. First, we revised the CERNN to incrementally learn from data. We tried this but have not found the same high recommendation accuracy with the incremental learning, so did not report the result here. Second, we could use a cloud service to simulate the on-line learning for a project, especially by using distributed

systems or multi-GPUs. We tried it as well, but found we need to revise our model to fit to the distributed systems. It was not easy because our LSTM model relies on the previous status, so we found revising our model to the distributed version. Third, we could train the model, separated from the recommendation system. The time spent on the training model will not deteriorate the performance of a recommendation system.

8. Threats to Validity

The threats to internal validity are as follows. First, even if we experimented with hyperparameters, we could not guarantee that we have optimized hyperparameters. Second, we found that a deep learning technique that initially depends on random variables could yield different results whenever a simulator runs. However, we also found that the variations of the recommendation accuracy are within 0.02.

The threats to external validity are as follows. First, we only used the five projects that early studies used. Therefore, the experimental results may not be generalizable. Second, we also found that the recommendation accuracy differs from project to project. We found that the F1-score of recommendation accuracy was between 0.56 and 0.90. Third, the size of interaction traces also appears to be related to the recommendation accuracy, but we did not control it in our experiments; thus, the weighted average of the recommendation accuracy is somewhat favorably biased to the Mylyn project, which has the largest number of interaction traces. To mitigate this issue, we also calculated the average of the recommendation accuracy over five projects.

9. Conclusions

In this paper, we proposed a method, CERNN, which recommends files to edit based on a deep learning technique. We implemented our method as an edit recommendation system and evaluated our method with it. Our evaluation showed that CERNN yields a 64% F1-score whereas the previous state-of-the-art method (MI-EA) yields 59%, providing an improvement of the recommendation accuracy of 5%. This improvement has been achieved by utilizing sequential information for context, but sequential information is not the only factor that contributed to the improvement. For our preliminary experiment, an N-gram recommendation model that uses sequential information, which was explained in Section 3, yielded significantly low recall values. Adopting a deep learning technique (i.e., LSTM) in our method was a crucial factor that helped improve the recall values. We explained the additional factors that contribute to recommendation accuracy in Section 7.1.

To improve recommendation accuracy even further, we could consider ways of better reflecting the characteristics of data. For example, we could replace the default embedding layer with other embedding techniques, such as Word2Vec or AutoEncoder. Additionally, we could utilize weighting methods or oversampling techniques. Since the results of these weighting methods or oversampling techniques will vary depending on the characteristics of a training data set, we need to investigate the characteristics of the training data set more deeply. Besides, we could consider, in addition to sequential information in interaction records, also the dependency relationships, such as call relationships between methods and passing relationships of variables between method parameters. As yet another way to improve recommendation accuracy, we could consider changes in recommending files to edit. For CERNN, we used a multi-label model. In the future, we consider replacing it with a sequence generation model where a recommended file to edit becomes the next input for another recommendation of the next file to edit. Additionally, we could use a multi-category model, where a softmax function is used to recommend only the file with the highest probability. By creating variations of our deep learning model and comparing them, we would find a more effective deep learning model for recommending files to edit.

Supplementary Materials: The source code of the simulator is available at https://github.com/saleese/cernn (accessed on 4 October 2021). The collected interaction traces can be downloaded from http://salab.kaist.ac.kr/tse2015/AllProjects.zip (accessed on 4 October 2021).

Author Contributions: Conceptualization, J.L. and S.K.; methodology, J.L. and S.L.; software, J.L. and S.L.; validation, H.C., J.A., S.L. and S.K.; investigation, S.L. and J.A; resources, J.L. and S.L.; data curation, S.L.; writing—original draft preparation, J.L.; writing—revision, review and editing, S.L.; visualization, J.L. and S.L.; supervision, S.K.; project administration, S.K.; funding acquisition, S.K. and S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A1A02085551). This work was also supported by the Human Resources Development of the Korea Institute of Energy Technology Evaluation and Planning (KETEP) grant funded by the Ministry of Trade, Industry and Energy (No. 20194030202430).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is specified in the article or Supplementary Material.

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

References

1. Lee, S.; Kang, S.; Kim, S.; Staats, M. The Impact of View Histories on Edit Recommendations. *IEEE Trans. Softw. Eng.* **2014**, *41*, 314–330. [CrossRef]
2. Robillard, M.; Walker, R.; Zimmermann, T. Recommendation Systems for Software Engineering. *IEEE Softw.* **2010**, *27*, 80–86. [CrossRef]
3. Zimmermann, T.; Weisgerber, P.; Diehl, S.; Zeller, A. Mining version histories to guide software edits. *IEEE Trans. Softw. Eng.* **2005**, *31*, 429–445. [CrossRef]
4. Kersten, M.; Murphy, G.C. Mylar: A degree-of-interest model for IDEs. In Proceedings of the 4th International Conference on Aspect-oriented Software Development, Chicago, IL, USA, 14–18 March 2005.
5. Cho, H.; Seonah, L.; Sungwon, K. A Code Recommendation Method Using RNN Based on Interaction History. *KIPS Trans. Softw. Data Eng.* **2018**, *7*, 461–468.
6. Ying, A.T.T.; Gail, C.; Murphy, R.N.; Mark, C. Predicting source code edits by mining edit history. *IEEE Trans. Softw. Eng.* **2004**, *30*, 574–586. [CrossRef]
7. Deline, R.; Czerwinski, M.; Robertson, G. Easing Program Comprehension by Sharing Navigation Data. In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), Dallas, TX, USA, 14–20 September 2005; pp. 241–248.
8. Singer, J.; Elves, R.; Storey, M.-A. NavTracks: Supporting navigation in software maintenance. In Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), Washington, WA, USA, 25–30 September 2005; pp. 325–334.
9. Zou, L.; Godfrey, M.W.; Hassan, A.E. Detecting Interaction Coupling from Task Interaction Histories. In Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC'07), Banff, AB, Canada, 26–29 June 2007; pp. 135–144. [CrossRef]
10. Maalej, W.; Fritz, T.; Robbes, R. Collecting and processing interaction data for recommendation systems. In *Recommendation Systems in Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 173–197.
11. Damevski, K.; Shepherd, D.C.; Schneider, J.; Pollock, L. Mining Sequences of Developer Interactions in Visual Studio for Usage Smells. *IEEE Trans. Softw. Eng.* **2017**, *43*, 359–371. [CrossRef]
12. Damevski, K.; Chen, H.; Shepherd, D.C.; Kraft, N.A.; Pollock, L. Predicting Future Developer Behavior in the IDE Using Topic Models. *IEEE Trans. Softw. Eng.* **2018**, *44*, 1100–1111. [CrossRef]
13. Kobayashi, T.; Nozomu, K.; Kiyoshi, A. Interaction histories mining for software edit guide. In Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering, Zurich, Switzerland, 4 June 2012.
14. Gu, Z.; Schleck, D.; Barr, E.T.; Su, Z. Capturing and Exploiting IDE Interactions. In Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Portland, OR, USA, 14–20 October 2014; pp. 83–94.
15. Soh, Z.; Drioul, T.; Rappe, P.-A.; Khomh, F.; Guéhéneuc, Y.-G.; Habra, N. Noises in Interaction Traces Data and Their Impact on Previous Research Studies. In Proceedings of the 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Beijing, China, 22–23 October 2015; pp. 1–10.
16. Soh, Z.; Foutse, K.; Yann-Gaël, G.; Giuliano, A. Noise in Mylyn interaction traces and its impact on developers and recommendation systems. *Empir. Softw. Eng.* **2018**, *23*, 645–692. [CrossRef]
17. Yamamori, A.; Hagward, A.M.; Kobayashi, T. Can Developers' Interaction Data Improve Change Recommendation? In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017; Volume 1, pp. 128–137.

18. Felden, C.; Chamoni, P. Recommender Systems Based on an Active Data Warehouse with Text Documents. In Proceedings of the 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Waikoloa, HI, USA, 3–6 January 2007; p. 168.
19. Alvarez, S.A.; Ruiz, C.; Kawato, T.; Kogel, W. Neural expert networks for faster combined collaborative and con-tent-based recommendation. *J. Comput. Methods Sci. Eng.* **2011**, *11*, 161–172.
20. Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; Tikk, D. Session-based recommendations with recurrent neural networks. In Proceedings of the 4th International Conference on Learning Representations, San Diego, CA, USA, 2–4 May 2016.
21. Wu, S.; Ren, W.; Yu, C.; Chen, G.; Zhang, D.; Zhu, J. Personal recommendation using deep recurrent neural networks in NetEase. In Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 16–20 May 2016; pp. 1218–1229.
22. Pei, W.; Yang, J.; Sun, Z.; Zhang, J.; Bozzon, A.; Tax, D.M. Interacting Attention-gated Recurrent Networks for Recommendation. Proceedings of 2017 ACM on Conference on Information and Knowledge Management, Singapore, Singapore, 6–10 November 2017; pp. 1459–1468.
23. He, X.; He, Z.; Song, J.; Liu, Z.; Jiang, Y.G.; Chua, T.S. Nais: Neural attentive item similarity model for recommendation. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 2354–2366. [CrossRef]
24. Rakkappan, L.; Rajan, V. Context-Aware Sequential Recommendations with Stacked Recurrent Neural Networks. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13 May 2019; pp. 3172–3178.
25. Lee, S.-R.; Heo, M.-J.; Lee, C.-G.; Kim, M.; Jeong, G. Applying deep learning based automatic bug triager to industrial projects. In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), Paderborn, Germany, 4–8 2017; ACM: New York, NY, USA, 2017; pp. 926–931.
26. Wen, M.; Wu, R.; Cheung, S.-C. How Well Do Change Sequences Predict Defects? Sequence Learning from Software Changes. *IEEE Trans. Softw. Eng.* **2020**, *46*, 1155–1175. [CrossRef]
27. Kurtukova, A.; Romanov, A.; Shelupanov, A. Source Code Authorship Identification Using Deep Neural Networks. *Symmetry* **2020**, *12*, 2044. [CrossRef]
28. Jurafsky, D.; James, H.M. Chapter 3. N-gram Language Models" Speech and Language Processing. Draft of 30 December 2020. Available online: https://web.stanford.edu/~{}jurafsky/slp3/3.pdf (accessed on 30 September 2021).
29. Olah, C. Understanding LSTM Networks. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 27 August 2015).
30. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, UK, 2016; Volume 1.
31. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
32. Chung, J.; Chung, Y.; Gulcehre, C.; Cho, K.H.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, arXiv:1412.3555.