2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE) | 978-1-6654-2587-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISSRE52982.2021.00024

Contextual-Semantic-Aware Linkable Knowledge Prediction in Stack Overflow via Self-Attention

Zhaolin Luo[1,2] Ling Xu[1,2*], Zhou Xu[1,2], Meng Yan[1,2], Yan Lei[1,2], Can Li[1,2],
[1]Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[2]School of Big Data and Software Engineering, Chongqing University, Chongqing, China
Email:{luozhaolin1999, xuling, zhouxullx, mengy, yanlei, 20151611}@cqu.edu.cn

***Abstract*—In Stack Overflow, a question and its answers are defined as a knowledge unit. These knowledge units can be linked together for different purposes, which typically subdivided into four classes: Duplicate, Directly linkable, Indirectly linkable, and Isolated. Developers usually use these linkable knowledge units to search for more targeted information. Prior studies have found that deep learning or SVM technique can effectively predict the class of linkable knowledge units. However, they focus on short-distance semantic relationship but fail to capture global information (semantic relationship between a word and all the words in the same knowledge unit) and ignore joint semantics (semantic relationship between a word with all the words in different knowledge units).**

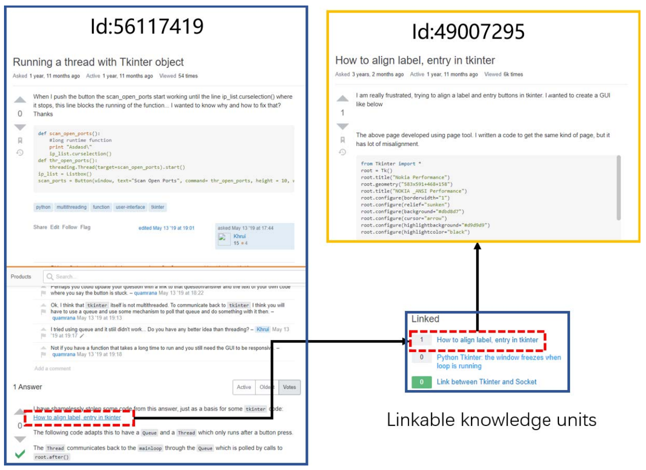
To address the issues, we propose a Self-Attention-based contextual semantic aware Linkable Knowledge prediction model (SALKU). SALKU leverages self-attention to pay attention to all the words in a knowledge unit and fully capture the global information needed for each word, then utilizes a variant of self-attention to extract joint semantics between two knowledge units. Experiment results on an existing dataset show that SALKU outperforms the state-of-the-art approaches CNN, Tuning SVM, and Soft-cos SVM in terms of three metrics, respectively. Additionally, SALKU is faster than the three baseline approaches.

***Index Terms*—self-attention, link prediction, joint semantic, Stack Overflow**

I. Introduction

Programming-specific question and answer site, such as Stack Overflow, accumulates a tremendous amount of knowledge over time. Following prior studies [1], [2], we refer to a question along with all its answers in Stack Overflow as a *knowledge unit*. Like Wikipedia knowledge network [3] and the general Web [4], a large number of knowledge units containing semantically relevant knowledge form a complex knowledge network, which provides more insights about their problems and possible solutions [1]. As shown in Figure 1, URL sharing activities in Stack Overflow are created as linkable knowledge units. The knowledge unit (id 56117419) is linked to another knowledge unit (id 49007295) to obtain some basic code about Tkinter. Developers can conveniently click on the link to get more related information for different purposes, such as solving technical problems, finding other solutions, and taking some inspiration.

In Stack Overflow, all linkable knowledge units are listed without distinction. However, for developers, linkable knowl-


Id:56117419
Running a thread with Tkinter object
Asked 1 year, 11 months ago Active 1 year, 11 months ago Viewed 54 times
0
python multithreading function user-interface tkinter
Share Edit Follow Flag
edited May 13 '19 at 19:01
asked May 13 '19 at 17:44
Khrui
15 4
Products
Search...
Add a comment
1 Answer
Active Oldest Votes
0
Id:49007295
How to align label, entry in tkinter
Asked 3 years, 2 months ago Active 1 year, 11 months ago Viewed 6k times
1
Linked
1 How to align label, entry in tkinter
0 Python Tkinter: the window freezes when loop is running
0 Link between Tkinter and Socket
Linkable knowledge units

When I push the button the scan_open_ports start working until the line ip_list.curselection() where it stops, this line blocks the running of the function... I wanted to know why and how to fix that? Thanks

```
def scan_open_ports():
    #long runtime function
    print "Asdasd\"
    ip_list.curselection()
def thr_open_ports():
    threading.Thread(target=scan_open_ports).start()
ip_list = Listbox()
scan_ports = Button(window, text="Scan Open Ports", command= thr_open_ports, height = 10,
```

where you say the button is stuck. – quamrana May 13 '19 at 18:22

Ok, I think that `tkinter` itself is not multithreaded. To communicate back to `tkinter` I think you will have to use a queue and use some mechanism to poll that queue and do something with it then. – quamrana May 13 '19 at 19:13

I tried using queue and it still didn't work... Do you have any better idea than threading? – Khrui May 13 '19 at 19:17

Not if you have a function that takes a long time to run and you still need the GUI to be responsiv. – quamrana May 13 '19 at 19:18

I have shamelessly stolen some code from this answer, just as a basis for some `tkinter` code: How to align label, entry in tkinter

The following code adapts this to have a `Queue` and a `Thread` which only runs after a button press.

The `Thread` communicates back to the `mainloop` through the `Queue` which is polled by calls to `root.after()`

I am really frustrated, trying to align a label and entry buttons in tkinter. I wanted to create a GUI like below

The above page developed using page tool. I written a code to get the same kind of page, but it has lot of misalignment.

```
from Tkinter import *
root = Tk()
root.title("Nokia Performance")
root.geometry("583x591+468+158")
root.title("NOKIA _ANSI Performance")
root.configure(borderwidth="1")
root.configure(relief="sunken")
root.configure(background="#dbd8d7")
root.configure(cursor="arrow")
root.configure(highlightbackground="#d9d9d9")
root.configure(highlightcolor="black")
```

Fig. 1. Linkable knowledge unit by URL sharing.

edge units are used for different purposes and achieve different relatedness. As shown in Table I, Xu et al. [2] divided these linkable knowledge units into four classes based on their relatedness: Duplicate, Directly linkable, Indirectly linkable, and Isolated. Table II shows some examples of different classes. The knowledge unit (id 42882632) is marked as Duplicate to the original knowledge unit (id 22646463) because they can be solved by the same answer. However, rare overlapping words exist between these two knowledge units. From table II we observe that the knowledge unit (id 471546) can help the original knowledge unit indicate that "python cannot overload logical operations", which is considered Directly linkable. While the knowledge unit (id 34598898) only discusses related knowledge (boolean operation) with the original knowledge unit. Therefore, they are regarded as Indirectly linkable.

Different classes of linkable knowledge units can recommend more targeted information for developers. For example, by searching duplicate knowledge units, developers can analyze problems from many aspects. Directly linkable knowledge units can provide helpful content, while indirectly linkable knowledge units expand developers' searching domain.

There exist several studies on predicting linkable knowledge units. Xu et al. [2] proposed a CNN model with various window sizes to capture word-level and document-level semantics of knowledge units. Fu et al. [5] proposed a Tuning SVM to calculate the mean vector of each word vector based on

*Corresponding author.

TABLE I
CLASSES OF TWO KNOWLEDGE UNITS

Class	Description
Duplicate	Two knowledge units describe the same question by different expressions
Directly linkable	One knowledge unit can help to solve the question in the other knowledge unit
Indirectly linkable	One knowledge provides related information for the other knowledge unit to solve it, but not a direct answer.
Isolated	Two knowledge units are unrelated.

word2vec [6], which is taken as their feature vector for each knowledge unit. Then, SVM after parameter tuning is used to classify the relatedness. In recent work, Xu et al. [7] proposed Soft-cos SVM, which computes the soft cosine similarity to measure the distance between two knowledge units according to Simbow [8]. SVM is used as the final classifier.

Unfortunately, the aforementioned models still suffer from three limitations: (1) When representing each word in a knowledge unit, CNN, Tuning SVM, and Soft-cos SVM based on word2vec or convolution operation can only pay attention to nearby words (local information). It is inadequate for understanding the complete semantics of a word in a knowledge unit. For each word's better representation, the semantic relationship between a word and all the words in the same knowledge unit (global information) should be focused on. (2) Both local information and global information are only concerned with internal words in a knowledge unit. To aware of extra semantics from words in another knowledge unit (joint semantic) is also significant for representing a word, which is ignored by the above approaches. (3) The time cost of the above models is expensive, such as the CNN model need 14 hours to train reported by [2]. Tuning SVM and Soft-cos SVM require several hours for feature extraction. In addition, Tuning SVM has a long-time tuning process.

To address these issues, we propose a **S**elf-**A**ttention based **L**inkable **K**nowledge **U**nits prediction model SALKU, which can make better use of contextual semantics for representing knowledge units. To capture the global information in a knowledge unit, we perform a self-attention network [9] as san-encoder, encoding all words and getting the attention weight to enhance each word's semantic representation. To extract the joint semantics, we build a variant of self-attention—joint-encoder. When representing a knowledge unit, the joint-encoder allocates weights on another knowledge unit to enrich the information for the former knowledge unit. Moreover, we set a callback to save a lot of training time, and the parallel computation of self-attention can also reduce the time costs of SALKU.

To evaluate the performance, we apply SALKU to an existing dataset released by Xu et al. [7], which contains 40000 pairs of knowledge units. We compare SALKU with three baseline models CNN [2], Tuning SVM [5], and Soft-cos SVM [7]. Experiment results show that SALKU achieves an Overall F1-score (the mean F1-score of four classes) of 0.58, outperforming CNN, Tuning SVM, and Soft-cos SVM by 34.88%, 16.94%, and 13.73%, respectively. Meanwhile,

TABLE II
EXAMPLES OF DIFFERENT CLASSES OF LINKABLE KNOWLEDGE UNIT

Class	Title	Body
Original knowledge unit Id 22646463	"and" (boolean) vs "&" (bitwise) - Why difference in behavior with lists vs numpy arrays?	What explains the difference in behavior of boolean and bitwise operations on lists vs NumPy arrays? I'm confused about the appropriate use of & vs and in Python, illustrated in the following examples ...
Duplicate knowlegde unit Id 42882632	Get count of an item in a column when values in other column are equal to some value.	I need counts into a list else groupby without second condition works. if row =[1,1,1,1,1,1,1,1] then the count_equals list should be [2,4,4,2,4,1,3,4] ...
Directly linkable knowledge unit Id 471546	Any way to override the and operator in Python?	I tried overriding __and__, but that is for the & operator, not and - the one that I want. Can I override and?
Indirectly linkable knowledge unit Id 34598898 (through 3845018)	Pandas boolean algebra: True if True in both columns.	I would like to make a boolean vector that is created by the comparison of two input boolean vectors. I can use a for loop, but is there a better way to do this? My ideal solution would look like this: ...

SALKU running on CPU is 7.1x, 102.7x, and 4.8x faster than CNN, Tuning SVM, and Soft-cos SVM, respectively. Additionally, SALKU running on GPU is more efficient.

The main contributions of this paper are as follows:

- We propose a linkable knowledge units prediction model SALKU, which extracts global information and the joint semantics in knowledge units based on self-attention networks.
- We evaluate the effectiveness and efficiency of SALKU on a public dataset in terms of three metrics. Experiment results show SALKU outperforms the state-of-the-art models CNN, Tuning SVM, and Soft-cos SVM.
- We open source our replication package [1], including the source code and dataset for reproducing our experiment.

The remaining of this paper includes the following parts. Section II introduces the background of linkable knowledge units prediction. Section III presents our proposed model SALKU. Section IV and Section V describes the experiment setup and results, respectively. Section VI discusses the advantage of SALKU and explains the reasons. Section VII presents the related works. Finally, we conclude the paper and present future work in Section VIII.

II. BACKGROUND

In this section, we introduce the background of linkable knowledge unis prediction and the general process.

For the state-of-the-art predicting approaches of linkable knowledge units (i.e., machine learning-based method, Tuning SVM or Soft-cos SVM, and deep learning-based method, CNN), the first step is to embed the texts in knowledge units (i.e., ku_1, ku_2). The pre-trained word2vec model, such as

[1]https://github.com/cqu-isse/SALKU

Google word2vec [6] and specialized word2vec model based on Stack overflow data, are typically used for $embedding$. e_1 and e_2 represent the knowledge unit matrices.

$$e_1, e_2 = embedding(ku_1, ku_2) \tag{1}$$

The next step is to extract features from knowledge unit matrices, such as the deep learning model using neural networks and machine learning model computing recessive features. The purpose of this step is to get the feature vector (i.e., $\boldsymbol{v}_1, \boldsymbol{v}_2$) of each knowledge unit. $feature$ is the feature extraction operation.

$$\boldsymbol{v}_1, \boldsymbol{v}_2 = feature(e_1, e_2) \tag{2}$$

Finally, researchers often use a simple classifier (i.e., fully connected network for deep learning and SVM for machine learning) to classify the relatedness into Duplicate, Directly linkable, Indirectly linkable, and Isolated. $classifier$ is the simple classifier. The whole classification process is as Equation (3):

$$distribution = classifier(\boldsymbol{v}_1, \boldsymbol{v}_2) \tag{3}$$

where $distribution$ is the classification distribution. Prediction result is the class with the highest probability in $distribution$.

III. The Approach

This section presents the overall structure and details of our proposed model SALKU.

A. Overview

Figure 2 shows the overall framework of SALKU which consists of three phases: san-encoder, joint-encoder and fully-connected-decoder. First of all, the inputs are texts in two knowledge units, represented as two sentences. These two sentences are converted into word vector representations by looking up the dictionary of word embeddings, then encoded into the corresponding feature matrices by san-encoder. Afterwards, joint-encoder learns the joint semantics between two feature matrices, getting feature vectors. Finally, the fully-connected-decoder calculates the probability distribution of the classification results.

The following subsections present the details. Specifically, Section III-B, Section III-C and Section III-D describe the san-encoder, joint-encoder and fully-connected-decoder in SALKU, respectively. Section III-E describes training settings, and section III-F describes the prediction process of linkable knowledge units.

B. San-encoder

Before san-encoder, we randomly extract 100000 knowledge units tagged with "java" from Stack Overflow to preprocess and build the vocabulary dictionary following Xu et al.'s method [2], [7]. To get a specialized dictionary of word embeddings for technical Q&A sites, we use word2vec [6] to train each knowledge unit. After looking up two sentences in the dictionary of word embeddings individually, we calculate the position vectors and adds them to embedding vectors. Then, instead of CNN [2], which can only get local information from nearby words limited by kernel size, we use the scaled dot-product attention model [9] to encode every word by paying attention to all the words in the sentence. In this way, each word can fully capture the global information needed. To enhance the feature representation, we build a residual structure [10], named Feed Forward Network [9], which contains a layer normalization, two linear transformations, and a non-linear activation with $ReLu$ [11]. Since our word embedding is trained by Stack Overflow data, SALKU will be overfitting soon if we use an encoder stack to loop encoding multiple times like other self-attention-based models [9]. So, the number of encoder layers is 1 in this case.

In detail, given a knowledge unit with N words, we perform lookup and position vector addition operation on each word, getting a matrix U. In scaled dot-product attention model, we use three linear transformations on U, respectively, to get the query matrix $Q_s \in \mathbb{R}^{N\times d}$, key matrix $K_s \in \mathbb{R}^{N\times d}$, value matrix $V_s \in \mathbb{R}^{N\times d}$:

$$Q_s = U \cdot W_{qs}^T \tag{4}$$

$$K_s = U \cdot W_{ks}^T \tag{5}$$

$$V_s = U \cdot W_{vs}^T \tag{6}$$

where $W_{qs} \in \mathbb{R}^{d\times d}, W_{ks} \in \mathbb{R}^{d\times d}, W_{vs} \in \mathbb{R}^{d\times d}$ are trainable parameters, and d is the dimensionality of word embedding.

Next, we combine Q_s and K_s to get the weights, and allocate them for corresponding value in V_s, getting $C_s \in \mathbb{R}^{N\times d}$. Each row in C_s represents a feature vector of a word, which assimilating part of the semantic information of each word in U:

$$C_s = Softmax(\frac{Q_s \cdot K_s^T}{\sqrt{d}}) \cdot V_s \tag{7}$$

where $\sqrt{d}$ can counteract the problem of gradient disappearance when training. $Softmax$ is the normalization function.

Afterwards, we obtain the output of san-encoder $S \in \mathbb{R}^{N\times d}$ through Feed Forward Network:

$$S = FFN(C_s) \tag{8}$$

where FFN is defined by Equation (9, 10, 11). C_s is normalized at layer-level, which is to speed up training, getting $residual$. R contains the high-dimensional feature for each word by three transformations over $residual$. The addition operation between $residual$ and R is to preserve the original semantic features in $residual$ and get extra high-dimensional features from R.

$$FFN(C_s) = residual + R \tag{9}$$

$$residual = LayerNorm(C_s) \tag{10}$$

$$R = Relu(residual \cdot W_1 + \boldsymbol{b}_1) \cdot W_2 + \boldsymbol{b}_2 \tag{11}$$

where $LayerNorm$ is layer normalization operation [12], $W_1 \in \mathbb{R}^{d\times 2d}$, $W_2 \in \mathbb{R}^{2d\times d}$, $b_1 \in \mathbb{R}^{2d}$ and $b_2 \in \mathbb{R}^{2d}$ are to be learned in training phase. $ReLu$ is a activation function.

We apply the above san-encoder to two different knowledge units, getting $S_1 \in \mathbb{R}^{N\times d}$ and $S_2 \in \mathbb{R}^{N\times d}$.

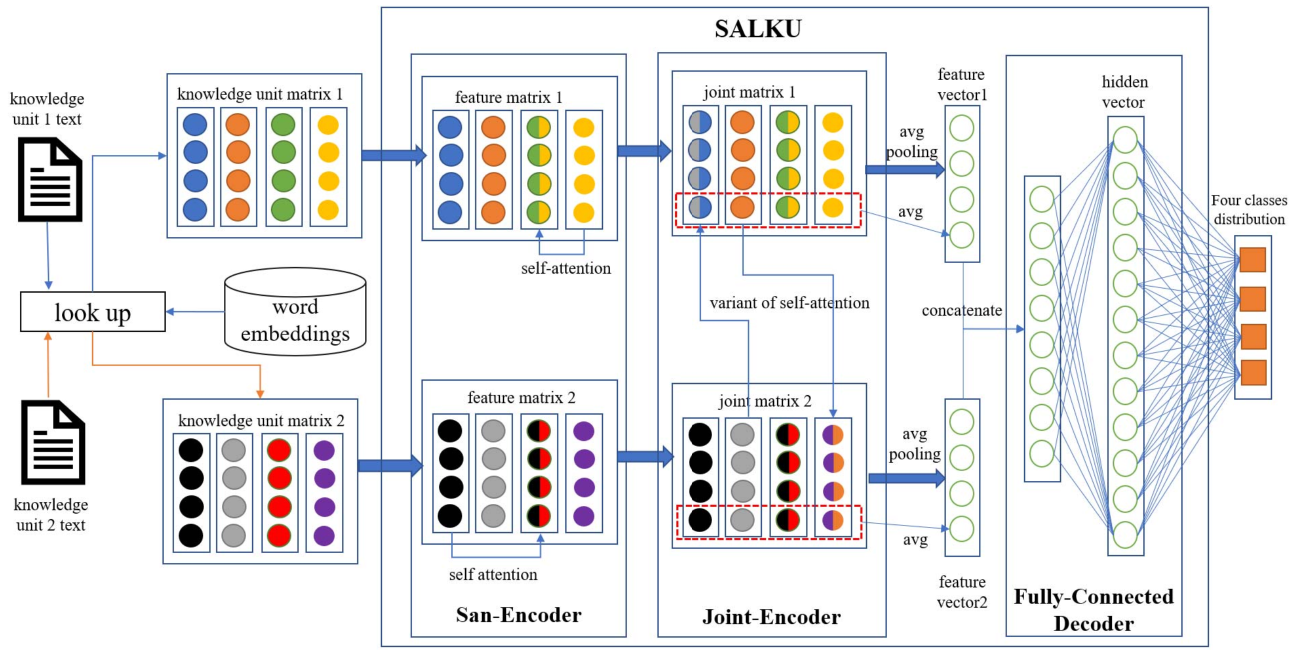


Fig. 2. Overall Framework of SALKU.

C. Joint-encoder

Many studies have shown that joint semantics are essential [13]–[17]. For example, Fang et al. [17] utilize the self-attention network to conduct an extra joint representation network to build semantic relationships between code snippets and their descriptions. Inspired by this, we build a joint-encoder using a variant of the self-attention network to learn joint semantics between two knowledge units to enhance the representation of each knowledge unit. That is, the representation of the knowledge unit S_1 is enhanced by the knowledge unit S_2, and the representation of S_2 is enhanced by S_1. Similar to Equation (4, 5, 6), we compute the query, key, value matrix for S_1 and S_2, respectively. $Q_{j1}, K_{j1}, V_{j1}, Q_{j2}, K_{j2}, V_{j2} \in \mathbb{R}^{N\times d}$ are calculated as follows:

$$Q_{j1} = S_1 \cdot W_{qj}^T \tag{12}$$

$$K_{j1} = S_1 \cdot W_{kj}^T \tag{13}$$

$$V_{j1} = S_1 \cdot W_{vj}^T \tag{14}$$

$$Q_{j2} = S_2 \cdot W_{qj}^T \tag{15}$$

$$K_{j2} = S_2 \cdot W_{kj}^T \tag{16}$$

$$V_{j2} = S_2 \cdot W_{vj}^T \tag{17}$$

where $W_{qj}^T \in \mathbb{R}^{d\times d}$, $W_{kj}^T \in \mathbb{R}^{d\times d}$, $W_{vj}^T \in \mathbb{R}^{d\times d}$ are trainable parameters.

After completing the above steps, we combine Q_{j2} and K_{j1} to get the joint weights, and allocate them for corresponding value in V_{j1}, getting $C_{j1} \in \mathbb{R}^{N\times d}$. Likewise, we can get $C_{j2} \in \mathbb{R}^{N\times d}$ by combining Q_{j1} and K_{j2} over V_{j2}. C_{j1} and C_{j2} are calculated as follows:

$$C_{j1} = Softmax(\frac{Q_{j2} \cdot K_{j1}^T}{\sqrt{d}}) \cdot V_{j1} \tag{18}$$

$$C_{j2} = Softmax(\frac{Q_{j1} \cdot K_{j2}^T}{\sqrt{d}}) \cdot V_{j2} \tag{19}$$

Then, similar to san-encoder, we also employ FFN to merge the original semantic feature and high-dimensional feature as Equation (8), obtaining $J_1 \in \mathbb{R}^{N\times d}$ and $J_2 \in \mathbb{R}^{N\times d}$:

$$J_1 = FFN(C_{j1}) \tag{20}$$

$$J_2 = FFN(C_{j2}) \tag{21}$$

Finally, we set a linear transformation and layer normalization on J_1 and J_2, respectively. The final feature matrices $F_1 \in \mathbb{R}^{N\times d}$, $F_2 \in \mathbb{R}^{N\times d}$ are defined as follows:

$$F_1 = LayerNorm(J_1 \cdot W_j + \boldsymbol{b}_j) \tag{22}$$

$$F_2 = LayerNorm(J_2 \cdot W_j + \boldsymbol{b}_j) \tag{23}$$

where $W_j \in \mathbb{R}^{d\times d}$, $b_j \in \mathbb{R}^d$ are trainable parameters.

To obtain feature vectors $\boldsymbol{v}_1 \in \mathbb{R}^d$ and $\boldsymbol{v}_2 \in \mathbb{R}^d$ from feature matrices, we conduct a pooling operation. Prior study shows that max-pooling may hurt the semantic relationship between each word [17], so we utilize avg-pooling to keep more semantic information. $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are defined as follows:

$$\boldsymbol{v}_1 = Avgpooling(F_1) \tag{24}$$

$$\boldsymbol{v}_2 = Avgpooling(F_2) \tag{25}$$

D. Fully-connected-decoder

Xu et al. [2] compute the cosine similarity between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ as the criterion to recognize different classes. They manually define Duplicate $\in [0.75, 1]$, Directly linkable $\in [0.5, 0.75)$, Indirectly linkable $\in [0.25, 0.5)$, Isolated $\in [0, 0.25)$. This manual setting uncertainty may lead to critical points being misclassified. To get a more reasonable probability distribution, we build a decoder to learn the possibility of each class. Specifically, we concatenate $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and perform a non-linear transformation with $Relu$ on it, getting a vector ***hidden*** $\in R^h$, which potentially contains relatedness between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Then, we calculate the probability distribution of four classes with the help of $Softmax$. The concrete definition is as follows:

$$\boldsymbol{v} = \boldsymbol{v}_1 \oplus \boldsymbol{v}_2 \tag{26}$$

$$\boldsymbol{hidden} = Relu(\boldsymbol{v} \cdot W_H + \boldsymbol{b}_H) \tag{27}$$

$$\boldsymbol{distribution} = Softmax(\boldsymbol{hidden} \cdot W_D + \boldsymbol{b}_D) \tag{28}$$

where $\oplus$ is concatenate operation, $W_H \in \mathbb{R}^{2d\times h}$, $W_D \in \mathbb{R}^{h\times 4}$, $\boldsymbol{b}_H \in \mathbb{R}^h$ and $\boldsymbol{b}_D \in \mathbb{R}^4$ are trainable parameters, $Relu$ and $Softmax$ are activation functions. ***distribution*** $\in \mathbb{R}^4$ is the final output of SALKU, in which each element represents the probability of a class.

E. Training

Like other classification models, we choose the categorical cross-entropy loss function, which can reduce the distance between the actual probability and the predicted probability. The loss function of SALKU is as following:

$$Loss = -\frac{1}{n}\sum_{i=0}^{n-1} y_i log(\hat{y}_i) \tag{29}$$

To minimize the loss value, we use Adam update algorithm [18]. Moreover, SALKU set an early stopping callback, i.e., if no updates are made to the best valid loss on the validation set for the last five epochs, we will stop training.

F. Predicting

After completing the training phase, SALKU has learned a large number of knowledge unit pairs. Given a new pair of preprocessed knowledge units, we feed them into SALKU, which captures the global information by san-encoder, then extracts the joint semantics between them in joint-encoder. The classification probability distribution is given by fully-connected-decoder. We select the class with the highest probability as the prediction result.

IV. Experiment Setup

In this section, we describe the datasets used, the evaluation metrics, and five investigated research questions (RQs). We evaluate the effectiveness of SALKU and compare its performance with three state-of-the-art approaches.

A. Dataset

For comparison, we use the same collection of datasets provided by Xu et al. [7]. It contains a total of 40,000 pairs of knowledge units. The dataset has been divided into training and testing data, containing 32,000 and 8000 pairs, respectively, stored as .csv. And each class of knowledge unit pairs accounts for 1/4 in both training and testing data. There are some error rows when reading the CSV files. To ensure correctness, we manually delete these rows. As shown in Table III, we get 31984 pairs for training and 7999 pairs for testing finally. To eliminate the influence of this data change on the experiments, we replicate all the baselines conducting on our dataset.

TABLE III
STATISTICS OF OUR DATASETS

Class	Train-Test	Actual Train-Test
Duplicate	8000-2000	8000-2000
Direct link	8000-2000	8000-2000
Indirect link	8000-2000	8000-2000
Isolated	8000-2000	7984-1999
Total	32000-8000	31984-7999

B. Baselines

This study compares the following three baselines:

CNN [2]. Xu et al. proposed the task of predicting semantically linkable knowledge in Stack Overflow, conducting a CNN model based on deep learning. As their implementation only supports CPU without any deep learning framework, we use Tensorflow to carefully build the same CNN model as outlined in [2] so that we can training the CNN model on not only CPU but also GPU.

Tuning SVM [5]. Fu et al. addressed the efficiency problem on CNN and proposed a machine learning method, Tuning SVM, which is based on differential evolution to fine-tune SVM parameters. Their method is 84 times faster hours than CNN. We reuse the source code opened by Fu et al. [5], and conduct it on our dataset.

Soft-cos SVM [7]. Xu et al. proposed Soft-cos SVM based on feature extraction and SVM without tuning parameters. In total four different features are extracted from the knowledge units: Cosine similarity based on Stack Overflow data, soft cosine similarity based on Stack Overflow data's word2vec, pre-trained Google word2vec, and Levenshtein distance, respectively. We use the source code released for Soft-cos SVM [7] and apply it to our dataset.

C. Evaluation Metrics

To evaluate the performance of predicting linkable knowledge units, we choose the same metrics as previous works [2], [5], [7], i.e., Precision, Recall, and F1-score, which are label-based metrics [19]. For the i-th class label y_i, p_i is the total number of samples belonging to class i. x_j is the j-th sample, Y_j is corresponding ground truth and $h(x_j)$ is prediction result for x_j. Four basic quantities characterizing the binary classification performance on this label can be defined based on $h(\cdot)$: $TP_i = |\{x_j | y_i \in Y_j \wedge y_i \in h(x_j), 1 \leq$

$j \leq p_i\}|, FP_i = |\{x_j | y_i \notin Y_j \wedge y_i \in h(x_j), 1 \leq j \leq p_i\}|, TN_i = |\{x_j | y_i \notin Y_j \wedge y_i \notin h(x_j), 1 \leq j \leq p_i\}|, FN_i = |\{x_j | y_i \in Y_j \wedge y_i \notin h(x_j), 1 \leq j \leq p_i\}|$.

Based on above definition, Precision, Recall and F1-score for class i are as below:

$Precision_i$ is the probability of knowledge unit pairs correctly classified as i over the number of knowledge unit pairs classified as class i.

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \tag{30}$$

$Recall_i$ is defined as the percentage of all knowledge unit pairs classified as i correctly classified.

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{31}$$

$F1\text{-}score_i$ is harmonic mean of $Precision_i$ and $Recall_i$.

$$F1\text{-}score_i = \frac{2 \times Precision_i \times Recall_i}{Precision_i + Recall_i} \tag{32}$$

D. Parameter Settings

For our model, 200-dimensional word embeddings are used according to experience. Since the length of 90% of knowledge units is less than 100, we fix the length of each sentence as 100 by post-truncating operation. As described in III-B, we do not repeat encoding the knowledge units, so the number of encoder layers in san-encoder is 1. In detail, we adopt a multi-head strategy on scaled dot-product attention to expand the attention domain, and the number of heads in multi-head attention is 2. In addition, the dimensionality of hidden vectors h in our fully-connected-decoder is set to 1024 after the experiment in IV-E5. The training parameters, including epochs, batch size, and learning rate, are 100, 128, and 0.0001, respectively.

For the CNN model, according to the parameter settings in [2], we use the same word embedding dictionary in our model. The length of each sentence is fixed as 200 by post-truncating operation. We build five convolution neural networks using 128 filters with different kernel sizes, capturing 1, 3, 5, 7, and 9-grams information, and we add the l2 norm regularization on parameters in convolution neural networks. Additionally, The dimensionality of the final hidden vector is 50.

For Tuning SVM, we use differential evolution to fine-tune parameters based on our dataset and our word embedding dictionary. For Soft-cos SVM, we use the same corpus to train a new word embedding dictionary with a dimensionality of 300 to make it consistent with pre-trained Google word2vec and calculate four features to classify the relatedness between two knowledge units.

E. Research Questions

In this subsection, we present five research questions:

1) Model Performance:

RQ1. Compared with the state-of-the-art approaches, how effective is SALKU?

Motivation. To obtain a more effective model for predicting linkable knowledge units is our key work. Baseline approaches only use simple local information to embed each word. They have difficulty connecting two words that are far apart in a knowledge unit. Besides, they cannot be aware of the semantic relationship between two words in different knowledge units. Therefore, we use SALKU to overcome these shortcomings. Additionally, to illustrate the impact of data change described in IV-A, we compare the performance of our replicated model CNN, Tuning SVM, and Soft-SVM with the results reported in [7].

2) Model Efficiency:

RQ2. Does SALKU run faster than the state-of-the-art approaches on CPU and GPU respectively?

Motivation. Fu et al. [5] indicate that long training time limits the ability of other researchers to repeat, improve, or even refute the work for deep learning-based models, such as CNN [2]. Although Xu et al. [7] instruction that Tuning SVM's efficiency is diminished as the number of instances for tuning increases, the efficiency of deep learning-based model is still unacceptable for predicting linkable knowledge units. Therefore, we apply Early Stopping on our proposed model to improve efficiency while ensuring similar effectiveness. To show more rigorous time costs, we conduct deep learning-based models on GPU and CPU, respectively. All the experiments using GPU are implemented on a server with one Nvidia Titan V GPU with 12 GB memory, while experiments using CPU are implemented on Intel(R) Core(TM) i5-1035G4 CPU with 16 GB memory.

3) Ablation Analysis:

RQ3. What is the contribution of san-encoder and joint-encoder to SALKU?

Motivation. In SALKU, san-encoder and joint-encoder play significant roles, representing global information and joint semantics, respectively. To research the importance about these two modules, we design two variants of SALKU, named SALKU-san and SALKU-joint. The former replaces the san-encoder with a simple embedding layer, the latter drops joint-encoder and concatenates the san-encoder's outputs to classify.

4) Prediction Method Analysis:

RQ4. Which method better predicts linkable knowledge units, classification or regression?

Motivation. We find that both Tuning SVM and Soft-cos SVM use the SVC classifier rather than the SVR classifier. They outperform CNN [2] which uses the regression method with mean square error loss function. Therefore, we analyze regression and classification methods in predicting linkable knowledge units by this research question. To clearly explain the prediction method of each model, we rename the SALKU and CNN to SALKU-classification and CNN-regression in this research question. Additionally, we build SALKU with regression and CNN with classification, named SALKU-regression

TABLE IV
EFFECTIVENESS COMPARISON OF MODELS CNN,TUNING SVM, SOFT-COS SVM, AND SALKU IN TERMS OF PRECISION, RECALL AND F1-SCORE AT LABEL LEVEL. FOR THE OVERALL COLUMN, THE VALUES OUTSIDE THE BRACKETS REPRESENTS THE AVERAGE VALUE OF FOUR CLASSES FOR THREE METRICS RESPECTIVELY, AND THE VALUES IN THE BRACKETS ARE AVERAGE VALUES REPORTED BY XU ET AL. [7]. THE HIGHEST SCORE FOR EACH CLASS IS MARKED IN BOLD

	Model	Duplicate	Directly linkable	Indirectly linkable	Isolated	Overall
Precision	CNN	0.55	0.34	0.33	0.73	0.48(0.50)
	Tuning SVM	0.50	0.35	0.47	0.65	0.49(0.49)
	Soft-cos SVM	0.48	**0.44**	0.42	0.71	0.51(0.53)
	SALKU	**0.56**	0.43	**0.51**	**0.83**	**0.58**
Recall	CNN	0.32	**0.54**	0.39	0.44	0.42(0.41)
	Tuning SVM	0.51	0.24	**0.56**	0.73	0.50(0.51)
	Soft-cos SVM	0.53	0.21	0.49	**0.90**	0.53(0.54)
	SALKU	**0.66**	0.40	0.44	0.85	**0.59**
F1-score	CNN	0.40	**0.42**	0.36	0.55	0.43(0.41)
	Tuning SVM	0.50	0.28	**0.51**	0.69	0.50(0.50)
	Soft-cos SVM	0.50	0.29	0.45	0.79	0.51(0.52)
	SALKU	**0.60**	**0.42**	0.47	**0.84**	**0.58**

and CNN-classification respectively. More specifically, model based on classification is using fully-connected-decoder described in III-D, while model based on regression computes the cosine similarity between two feature vectors, and bins similarity value as four classes: $[0, 0.25)$ as Isolated, $[0.25, 0.5)$ as Indirectly linkable, $[0.5, 0.75)$ as Directly linkable, and $[0.75, 1]$ as Duplicate.

5) Parameters Tuning:

RQ5. How do the key parameters affect the effectiveness?

Motivation. We have three key parameters: the number of encoder layers (num-layers), the number of heads in multi-head strategy (num-heads), and the dimensionality of hidden vector (h). To explore the influence of these parameters on SALKU's effectiveness, we design three parameter lists: $[1, 2, 3, 4, 5, 6]$, $[1, 2, 4, 8]$ and $[128, 256, 512, 1024]$ for num-layers, num-heads and h, respectively. The parameters tuning process was varying one value while fixing the other two values. The default value of the three parameters is 1, 2, and 1024, respectively.

V. EXPERIMENT RESULTS

This section investigates the five research questions (RQs) described in Section IV-E.

A. RQ1: Compared with the state-of-the-art approaches, how effective is SALKU?

We compare linkable knowledge prediction effectiveness with the state-of-the-art models CNN, Tuning SVM, and Soft-SVM under three metrics. For comparison, we replicated the baselines models. Experiment Results in the Overall column of Table IV show that our replication are similar to their results, which also show that the impact of the data change is subtle and negligible.

Table IV shows that SALKU achieves best performance i.e., 0.58, 0.59, 0.58 Overall Precision, Recall and F1-score, respectively. SALKU outperforms the models CNN, Tuning SVM, Soft-cos SVM by 21.83%, 17.89%, 13.73% in terms of Precision, by 40.48%, 18%, 11.32% in terms of Recall, by 34.88%, 16.94%, 13.73% in terms of F1-score. For individual classes, it is evident that SALKU shows better performance on Duplicate class. SALKU outperforms Tuning SVM, which is better than the other two baseline models for Duplicate class, by 19.05% in terms of F1-score. In the other three classes, the F1-score of SALKU is at least in the top two. For example, for Indirectly linkable class, SALKU achieves 0.04 F1-score worse than Soft-cos SVM, but 0.11, 0.02 better than CNN and Tuning SVM, respectively.

Result 1: *SALKU outperforms the baselines CNN, Tuning SVM, and Soft-cos SVM substantially on overall and each class's effectiveness.*

B. RQ2: Does SALKU run faster than the state-of-the-art approaches on CPU and GPU respectively?

As shown in table V, CNN on CPU need more time than Soft SVM, which is consistent with that reported by Xu et al. [7]. Compared with other techniques, SALKU on CPU is 7.1x, 102.7x, and 4.8x faster than CNN on CPU, Tuning SVM, and Soft-cos SVM respectively. Our proposed SALKU on GPU is around 2.4x faster than CNN on GPU.

TABLE V
TIME COST OF SALKU ON CPU, GPU RESPECTIVELY, CNN ON ON CPU, GPU RESPECTIVELY, TUNING SVM ON CPU AND SOFT-COS SVM ON CPU

Deep learning Model	Train time	Test time	Total time
SALKU(cpu)	1:05:38	0:00:29	1:06:07
SALKU(gpu)	0:11:10	0:00:07	0:11:19
CNN(cpu)	7:50:52	0:00:24	7:51:16
CNN(gpu)	0:26:26	0:00:27	0:26:53
Machine learning Model	**feature extraction**	**Train-Test**	**Total time**
Tuning SVM(cpu)	-	-	112:56:31
Soft-cos SVM(cpu)	5:15:22	0:00:32	5:15:54

The reason why SALKU is so efficient is that our model contains Early Stopping. When we use the same method on CNN, as shown in table VI, both CNN on CPU and GPU can reduce the time cost by nearly half. Moreover, they get better performance to some degree on account of avoiding overfitting.

Result 2: *SALKU outperforms the baselines CNN, Tuning SVM, and Soft-cos SVM substantially on efficiency.*

TABLE VI
THE EFFECT OF EARLY STOPPING FOR THE CNN MODEL ON CPU AND GPU RESPECTIVELY IN TERMS OF OVERALL PRECISION, RECALL, F1-SCORE AND TOTAL TIME

Model	Precision	Recall	F1-score	Total time
CNN(cpu-earlystop)	0.49	0.42	0.43	3:49:31
CNN(gpu-earlystop)	0.50	0.41	0.42	0:14:51
CNN(cpu)	0.47	0.41	0.41	7:51:16
CNN(gpu)	0.48	0.42	0.43	0:26:53

C. RQ3: What is the contribution of san-encoder and joint-encoder to SALKU??

As shown in table VII, for SALKU, by removing san-encoder, the Overall Precision, Recall, and F1-score decreases by 5.17%, 6.78%, and 6.9%; by removing joint-encoder, the Overall Precision, Recall, and F1-score decreases by 6.9%, 8.47%, and 8.62%. These results show that self-attention has a powerful representation ability to capture global information, and joint semantics is beneficial to build the relatedness between two knowledge units.

TABLE VII
EFFECTIVENESS COMPARISON OF MODELS WITHOUT SAN-ENCODER AND JOINT-ENCODER RESPECTIVELY IN TERMS OF OVERALL PRECISION, RECALL AND F1-SCORE

Model	Precision	Recall	F1-score
SALKU	0.58	0.59	0.58
SALKU-san	0.55	0.55	0.54
SALKU-joint	0.54	0.54	0.53

Result 3: *Both san-encoder and joint-encoder have a contribution on the effectiveness of SALKU.*

D. RQ4: Which method better predicts linkable knowledge units, classification or regression?

As shown in table VIII, model based on classification outperforms than that based on regression for not only SALKU but also CNN. SALKU-classification outperforms SALKU-regression by 5.45%, 20.4%, and 16% in terms of Overall Precision, Recall, and F1-score. CNN-classification outperforms CNN-regression by 6.25%, 21.43%, and 13.95% in terms of Overall Precision, Recall, and F1-score. These results show that classification is more suitable than regression for predicting linkable knowledge units.

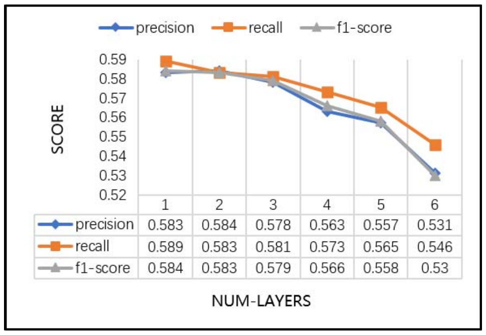
TABLE VIII
EFFECTIVENESS COMPARISON OF SALKU AND CNN WITH CLASSIFICATION AND REGRESSION PREDICTION MODULE RESPECTIVELY IN TERMS OF OVERALL PRECISION, RECALL AND F1-SCORE

Model	Precision	Recall	F1-score
SALKU-classification	0.58	0.59	0.58
SALKU-regression	0.55	0.49	0.50
CNN-classification	0.51	0.51	0.49
CNN-regression	0.48	0.42	0.43

Result 4: *Classification outperforms regression for predicting linkable knowledge units.*

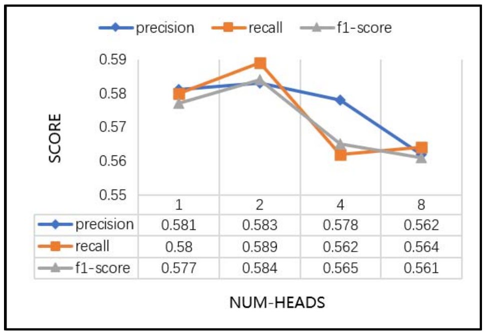
E. RQ5: How do the key parameters affect the effectiveness?

Figure 3 shows that the overall Precision, Recall, and F1-score are in a downward trend with the increase of num-layers. The best choice of num-layers is 1 for predicting linkable knowledge units. In Figure 4, we observe that these metrics reach optimum when the num-heads is set to 2. Finally, with the increase of h, as shown in Figure 5, SALKU obtains better effectiveness. Because the longer hidden vector contains more semantic information. So, we choose 1024 as the dimensionality of the hidden vector.



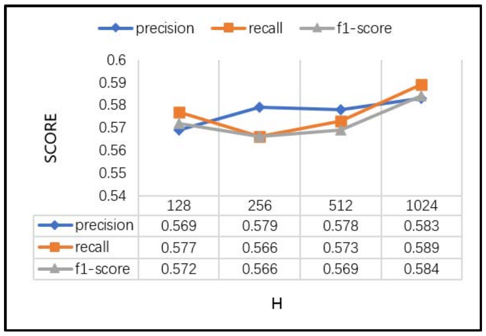
	1	2	3	4	5	6
precision	0.583	0.584	0.578	0.563	0.557	0.531
recall	0.589	0.583	0.581	0.573	0.565	0.546
f1-score	0.584	0.583	0.579	0.566	0.558	0.53

Fig. 3. The effectiveness of SALKU with various num-layers.



	1	2	4	8
precision	0.581	0.583	0.578	0.562
recall	0.58	0.589	0.562	0.564
f1-score	0.577	0.584	0.565	0.561

Fig. 4. The effectiveness of SALKU with various num-heads.



	128	256	512	1024
precision	0.569	0.579	0.578	0.583
recall	0.577	0.566	0.573	0.589
f1-score	0.572	0.566	0.569	0.584

Fig. 5. The effectiveness of SALKU with various h.

Result 5: *For SALKU, the best choice for num-layers is 1, and for num-heads is 2, while setting the dimensionality of hidden vector to 1024 is beneficial to predict linkable knowledge units.*

VI. DISCUSSION

This section first discusses the advantages of the proposed model SALKU in Section VI-A-VI-C. Then, in Section VI-D, we make an error Analysis. Finally, we discuss the threats to validity in Section VI-E.

A. Why does SALKU perform better?

The above experiment results indicate that the proposed model shows substantial advantages over the baseline models, which is due to two reasons—global information and joint semantics.

Global information. As shown in table IX, the first example is a pair of indirectly linkable knowledge units, which are talking about "File Upload". Many common words (i.e., "file", "upload", and "spring") may cause incorrect classification result—Duplicate. While a large number of irrelevant words (i.e., "MVC", "tutorial", and "JQery") may cause incorrect result—Isolated. As the global information extracted by self-attention, SALKU can understand that both knowledge units are related to "File Upload", but the knowledge unit (id 3577942) focuses on "application" and the knowledge unit (id 21985856) focuses on "javascript". So, SALKU correctly classify the two knowledge units as Directly linkable, rather than Duplicate or Isolated. Therefore, global information is beneficial for predicting linkable knowledge.

Joint semantics. Table IX show there are few common words between the knowledge unit (id 35245857) and the knowledge unit (id 1038308). If only considering each knowledge unit individually, the model only understands the knowledge unit (id 35245857) is about "scan null values", and the knowledge unit (id 1038308) is about "get all attributes and values". The meanings of the two knowledge units are far apart. The weakly related information "get values" leads them to be classified as indirectly linkable by baseline models. However, through joint semantics, SALKU understand that "for any/all null values" and "not know what are all the attributes in it at run time" have the same meanings (unknown values). So, SALKU correctly classify the two knowledge units as Duplicate. The joint semantics can well connect two knowledge units because both knowledge units have fully absorbed each other's information. So, each knowledge unit can be better represented. Therefore, joint semantics can improve the performance of SALKU.

B. Why does SALKU perform faster?

Unlike Tuning SVM and Soft-cos SVM, SALKU only needs to train and update model parameters by backpropagation, decreasing many feature extraction time and parameter tuning time. In addition, SALKU's key mechanism is self-attention, which can make parallel computation like CNN [20], [21]. Moreover, we adopt an early stopping on SALKU to save nearly half of time, as described in IV-E2. Thus, the time cost is more minor than baseline models.

C. Why does classification surpass regression?

For predicting linkable knowledge units, the difference between classification and regression is that classification is based on non-linear transformation, while regression is based on computing cosine similarity. The non-linear transformation captures all the elements in two feature vectors and maps feature vectors to classification space. The cosine similarity only computes the distance between two feature vectors, and it does not explicitly refer to classification results. So, regression-based models need to manually limit the range of values for each class, such as Duplicate $\in [0.75, 1]$, which is likely to cause misclassification of the critical points. The classification-based methods are driven by data and learn the probability of each class, which is equivalent to learning the range of values for each class. Compared with manual settings, automatic learning is more reasonable. Experiment results in V-D also indicate that classification outperforms regression for predicting linkable knowledge units.

D. Error Analysis

According to the experimental results, we find the number of cases that all models predict incorrectly is 1049. By sampling and analyzing 50 cases that all models predict incorrectly, we find that a large part of them requires code snippets or images to explain the information, and a small part is due to the noise of the text. Follow by Xu et al. [7], we remove code snippets and images during preprocessing. In the future, we could incorporate code snippets and images into our approach.

E. Threats To Validity

The dataset released by Xu et al. [7] only contains 40000 samples tagged with "java", which are far less than real data. And in this dataset, each class has the same number of 10000. The real condition is that the amount of data is extremely huge, and the imbalance problem is serious, such as duplicate knowledge units are much less than isolated knowledge units. In addition, some parameters like the length of a knowledge unit and word embedding dimensions are set by experience in our proposed model. Therefore, SALKU with such parameters may not perform well in other large and complex datasets.

VII. Related Work

This section provides the related works in three aspects: Link prediction in complex networks, Duplicate Question Detection, and Transformer-based model.

A. Link prediction in complex networks

Link prediction can be applied to many fields, such as physics, biochemical, and computer science communities, promoting related researches [22]–[27]. A link prediction algorithm was proposed using the modularity measure reflecting the community structure information of the network [28]. Specifically, Mart ́ınez et al. [29] applied link prediction technology in biology to predict previously unknown interactions between proteins. Wang et al. [30] proposed SHINE to predict the sign of sentiment links in online social networks. Shao et al. [31] treated Github as a repository-repository network and treated academic papers database such as ACM as a paper-paper network. They used GCN to predict the link path and recommend repositories for responding papers. Liu et al. [32] built the trust/distrust relationship on a signed social service network through link prediction.

B. Duplicate Question Detection

With the rampant growth in the number of duplicated questions, user experience is declining [33] in Stack Overflow. To alleviate this phenomenon, Duplicate Questions Detection becomes a hot issue. Our predicting linkable knowledge units

TABLE IX
Prediction results of SALKU and baselines

	Knowledge unit1	Knowledge unit2	Prediction results
Directly linkable	Id:3577942 Title: **Spring** MVC **File Upload** Help Body: I have been integrating **spring** into an *application*, and have to redo a file upload from forms. I am aware of what **Spring** MVC has to offer and what I need to do to configure my controllers to be able to upload files. I have read enough tutorials to be able to do this, but what none of these tutorials explain is correct/best practice methods on how/what is to be done to actually handle the **file** once you have it. Below is some code similar to code found on the **Spring** MVC Docs on handling **file uploads** which can be found at **Spring** MVC **File Upload** ...	Id: 21985856 Title: How to send **file** using *javascript* Body: I am trying to make an **upload** page which takes **file** to **upload**. I am using **Spring** framework here my query is on **Upload** button I am calling a JavaScript method which should send my file to controller using jQuery AJAX. Is there any way to pass this through JavaScript? Following is the code which I am trying.	SALKU: Directly linkable CNN: Isolated Tuning SVM: Isolated Soft-cos SVM: Duplicate
duplicate	Id:35245857 Title: Scan Objects for Null Properites Body: In Java, Is there a way to scan an object *for any/all null* **values**, and set them to a default **value**, no matter their Object class? (Longs, Booleans, **Strings**, etc). Syntactically I'd imagine it would be similar to this, or simpler ...	Id: 1038308 Title: How to get the list of all attributes of a Java object using BeanUtils introspection? Body: I have method which gets a POJO as it's parameter. Now I want to programmatically get all the attributes of the POJO (because my code may *not know what are all the attributes in it at run time*) and need to get the **values** for the attributes also. Finally I'll form a **string** representation of the POJO.	SALKU: Duplicate CNN: Indirectly linkable Tuning SVM: Indirectly linkable Soft-cos SVM: Indirectly linkable

is the further development of Duplicate Questions Detection, which is a binary classification problem, duplicate or not. Zhang et al. [34] proposed DupPredictor, an approach to predict whether a question is a duplicate. DupPredictor was based on retrieval, using title similarity, description similarity, topic similarity, and tag similarity to weighted sum. The final score was to judge whether a question is duplicated. Ahasanuzzaman et al. [35] proposed Dupe, which also extracted features from the question corpus to build a question pair binary classifier. Wang et al. [36] used CNN, RNN, and LSTM to predict duplicate questions, respectively. Furthermore, Liang et al. [37] took full advantage of the semantic information in the paired answers while alleviating the noise problem caused by adding the answers to enhance the performance of duplicate question detection. Zhang et al. [38] modeled this problem as a two-stage "ranking-classification" problem. First, they ranked the history data and collected the top-ranked questions as candidates. Second, to judge whether a question is a duplicate, they combined the deep learning and information retrieval techniques to capture both textual similarity and latent semantics.

C. Transformer-based model

Since self-attention [9] was proposed, transformer-based models were widely used in many NLP fields [39]–[41], even computer vision fields [42]–[45], and obtained significant successes. Hettiarachchi et al. [46] utilized a pretrained transformer to achieve compatible results without any language-specific processing and resources. Yang et al. [47] proposed DeepPseudo utilizing both transformer encoder and code feature extractor to perform encoding for source code. Then it used a pseudo-code generator to perform decoding, which can generate the corresponding pseudo-code. Ahmad et al. [48] explored the transformer model to learn the code representation for summarization. Pilault et al. [49] presented a method to produce abstractive summaries of long documents that exceed several thousand words via neural abstractive summarization based on the transformer language model. Han et al. [50] proposed a novel Transformer-iN-Transformer (TNT) model for modeling both patch-level and pixel-level representation for computer vision-related tasks.

VIII. Conclusion and Future Work

In this paper, we propose a self-attention-based model named SALKU for predicting linkable knowledge units. Compared with three state-of-the-art approaches (i.e., CNN, Tuning SVM, and Soft-cos SVM), we overcome the shortcomings of hardly capturing global information and lack of joint semantics. Experiment results on an existing dataset shows that SALKU outperforms the above three approaches by 34.88%, 16.94%, and 13.73% in terms of Overall F1-score, respectively. Moreover, running SALKU on CPU is 7.1x, 102.7x, and 4.8x faster than them, respectively. Therefore, the global information and joint semantics extracted by SALKU are beneficial for predicting linkable knowledge units.

In the future, we have two mainly working directions. First, we plan to take the link direction into account, which can clearly show the directed relatedness between two knowledge units, especially for directly and indirectly linkable knowledge units. Second, we plan to study large code snippets representations that were previously ignored. We aim to extract additional semantic features from them to enhance the representation of each knowledge unit.

Acknowledgment

This work was supported in part by the National Key Research and Development Project (No.2018YFB2101200), the Fundamental Research Funds for the Central Universities (No.2020CDCGRJ037), the National Nature Science Foundation of China (No.62002034) and the National Defense Basic Scientific Research Program (No.WDZC20205500308).

References

[1] D. Ye, Z. Xing, and N. Kapre, "The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow," *Empirical Software Engineering (ESE)*, vol. 22, no. 1, pp. 375–406, 2017.

[2] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 51–62.

[3] J. Preusse, J. Kunegis, M. Thimm, S. Staab, and T. Gottron, "Structural dynamics of knowledge networks," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 7, no. 1, 2013.

[4] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," in *The Structure and Dynamics of Networks*. Princeton University Press, 2011, pp. 183–194.

[5] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2017, pp. 49–60.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[7] B. Xu, A. Shirani, D. Lo, and M. A. Alipour, "Prediction of relatedness in stack overflow: deep learning vs. svm: a reproducibility study," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2018, pp. 1–10.

[8] D. Charlet and G. Damnati, "Simbow at semeval-2017 task 3: Soft-cosine semantic similarity between questions for community question answering," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 315–319.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[13] M. Raghothaman, Y. Wei, and Y. Hamadi, "Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 357–367.

[14] H. Nam, J.-W. Ha, and J. Kim, "Dual attention networks for multimodal reasoning and matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[15] S. Su, Z. Zhong, and C. Zhang, "Deep joint-semantics reconstructing hashing for large-scale unsupervised cross-modal retrieval," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[16] J. Shuai, L. Xu, C. Liu, M. Yan, X. Xia, and Y. Lei, "Improving code search with co-attentive representation learning," in *Proceedings of the 28th International Conference on Program Comprehension (ICPC)*, 2020, pp. 196–207.

[17] S. Fang, Y.-S. Tan, T. Zhang, and Y. Liu, "Self-attention networks for code search," *Information and Software Technology (IST)*, vol. 134, p. 106542, 2021.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 26, no. 8, pp. 1819–1837, 2013.

[20] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 1243–1252.

[21] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *arXiv preprint arXiv:1611.02344*, 2016.

[22] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*. Springer, 2011, pp. 243–275.

[23] A. Clauset, C. Moore, and M. E. Newman, "Hierarchical structure and the prediction of missing links in networks," *Nature*, vol. 453, no. 7191, pp. 98–101, 2008.

[24] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, pp. 1–27, 2011.

[25] E. Gilbert and K. Karahalios, "Predicting tie strength with social media," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 211–220.

[26] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[27] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[28] D. Caiyan, L. Chen, and B. Li, "Link prediction in complex network based on modularity," *Soft Computing*, vol. 21, no. 15, pp. 4197–4214, 2017.

[29] V. Martínez, C. Cano, and A. Blanco, "Prophnet: a generic prioritization method through propagation of information," *BMC bioinformatics*, vol. 15, no. 1, pp. 1–13, 2014.

[30] H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu, "Shine: Signed heterogeneous information network embedding for sentiment link prediction," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM)*, 2018, pp. 592–600.

[31] H. Shao, D. Sun, J. Wu, Z. Zhang, A. Zhang, S. Yao, S. Liu, T. Wang, C. Zhang, and T. Abdelzaher, "paper2repo: Github repository recommendation for academic papers," in *Proceedings of The Web Conference 2020 (WWW)*, 2020, pp. 629–639.

[32] H. Kou, H. Liu, Y. Duan, W. Gong, Y. Xu, X. Xu, and L. Qi, "Building trust/distrust relationships on signed social service network through privacy-aware link prediction process," *Applied Soft Computing*, vol. 100, p. 106942, 2021.

[33] I. Srba and M. Bielikova, "Why is stack overflow failing? preserving sustainability in community question answering," *IEEE Software*, vol. 33, no. 4, pp. 80–89, 2016.

[34] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in stack overflow," *Journal of Computer Science and Technology (JCST)*, vol. 30, no. 5, pp. 981–997, 2015.

[35] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining duplicate questions of stack overflow," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 402–412.

[36] L. Wang, L. Zhang, and J. Jiang, "Detecting duplicate questions in stack overflow via deep learning approaches," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2019, pp. 506–513.

[37] D. Liang, F. Zhang, W. Zhang, Q. Zhang, J. Fu, M. Peng, T. Gui, and X. Huang, "Adaptive multi-attention network incorporating answer information for duplicate question detection," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 95–104.

[38] W. E. Zhang, Q. Z. Sheng, J. H. Lau, E. Abebe, and W. Ruan, "Duplicate detection in programming question answering communities," *ACM Transactions on Internet Technology (TOIT)*, vol. 18, no. 3, pp. 1–21, 2018.

[39] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," *arXiv preprint arXiv:1804.09541*, 2018.

[40] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 7354–7363.

[41] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[42] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 213–229.

[43] M. Chen and A. Radford, "Rewon child, jeff wu, heewoo jun, prafulla dhariwal, david luan, and ilya sutskever. generative pretraining from pixels," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, vol. 1, 2020, p. 2.

[44] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[45] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao, “Pre-trained image processing transformer,” *arXiv preprint arXiv:2012.00364*, 2020.

[46] H. Hettiarachchi and T. Ranasinghe, “Transwic at semeval-2021 task 2: Transformer-based multilingual and cross-lingual word-in-context disambiguation,” *arXiv preprint arXiv:2104.04632*, 2021.

[47] G. Yang, X. Chen, K. Liu, and C. Yu, “Deeppseudo: Deep pseudo-code generation via transformer and code feature extraction,” *arXiv preprint arXiv:2102.06360*, 2021.

[48] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, “A transformer-based approach for source code summarization,” *arXiv preprint arXiv:2005.00653*, 2020.

[49] J. Pilault, R. Li, S. Subramanian, and C. Pal, “On extractive and abstractive neural document summarization with transformer language models,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 9308–9319.

[50] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, “Transformer in transformer,” *arXiv preprint arXiv:2103.00112*, 2021.