2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

iTrustSO: An Intelligent System for Automatic Detection of Insecure Code Snippets in Stack Overflow

Lingwei Chen[1], Shifu Hou[2], Yanfang Ye*[2], Thirimachos Bourlai[1], Shouhuai Xu[3], Liang Zhao[4]
[1]Department of CSEE, West Virginia University, WV, USA
[2]Department of CDS, Case Western Reserve University, OH, USA
[3]Department of CS, University of Texas at San Antonio, TX, USA
[4]Department of IST, George Mason University, VA, USA
[1,2]yanfang.ye@case.edu, [3]shxu@cs.utsa.edu, [4]lzhao9@gmu.edu

***Abstract*—Despite the apparent benefits of modern social coding paradigm such as Stack Overflow, its potential security risks have been largely overlooked (e.g., insecure codes could be easily embedded and distributed). To address this imminent issue, in this paper, we bring a significant insight to leverage both social coding properties and code content for automatic detection of insecure code snippets in Stack Overflow. To determine if the given code snippets are insecure, we not only analyze the code content, but also utilize various kinds of relations among users, badges, questions, answers and code snippets in Stack Overflow. To model the rich semantic relationships, we first introduce a structured heterogeneous information network (HIN) for representation and then use meta-path based approach to incorporate higher-level semantics to build up relatedness over code snippets. Later, we propose a novel hierarchical attention-based sequence learning model named *CodeHin2Vec* to seamlessly integrate node (i.e., code snippet) content with HIN-based relations for representation learning. After that, a classifier is built for insecure code snippet detection. Integrating our proposed method, an intelligent system named *iTrustSO* is accordingly developed to address the code security issues in modern software coding platforms. Comprehensive experiments on the data collections from Stack Overflow are conducted to validate the effectiveness of our developed system *iTrustSO* by comparisons with alternative methods.**

I. Introduction

Software has played a vital role in modern society for entertainment, education, and social communication, etc. Unlike conventional approaches, modern software developers heavily engage in a social coding environment to reuse code snippets and projects during the process of software development [33]. In particular, Stack Overflow, as the largest online programming discussion platform, has attracted 9.9 million registered developers [29]. The active discussions and abundant code snippets make it one of the most important information sources to software developers [9]. Despite the apparent benefits of such social coding environment, its potential security risks have been largely overlooked [1], [15]. According to a recent study [3], collected question-answer samples from Stack Overflow contain various security-related issues such as encryption with insecure mode, and insecure Application Programming Interface (API) usage. Those innocent-looking yet insecure code snippets could cause severe damage or even a disaster if not properly handled and directly transplanted to production software. For example, as shown in Figure 1, attackers have injected malicious cryptocurrency mining code such as *Coinhive* into Stack Overflow; once developers reuse such code snippets to generate the production software, its users' devices could be compromised (e.g., processing power would be stolen to mine bits of cryptocurrency). To deal with insecure code snippets included in the questions/answers, Stack Overflow has no principled way other than labeling the moderator flag, downvoting those threads or warning in the comments [3]. Given the rich structure and information, there is imminent need to develop novel and sound solutions to address the code security issue in Stack Overflow.

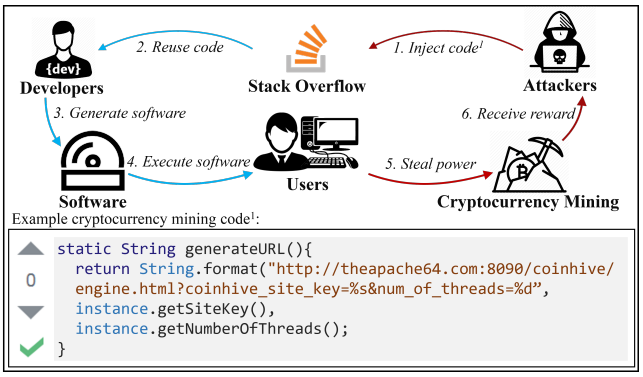

Figure 1: Example of code security attacks in Stack Overflow.

In this work, we propose to leverage both social coding properties and code content for insecure code snippet detection. As a social coding environment, Stack Overflow is characterized by user communication through questions and answers, i.e., a rich source of heterogeneous information are available including users, badges, questions, answers, code snippets, and their semantic relationships. To utilize such social coding properties, our previous work [39] proposed *ICSD* over heterogeneous information network (HIN) [30] for

ASONAM '19, August 27-30, 2019, Vancouver, Canada

ACM ISBN 978-1-4503-6868-1/19/08...$15.00
https://doi.org/10.1145/3341161.3343524

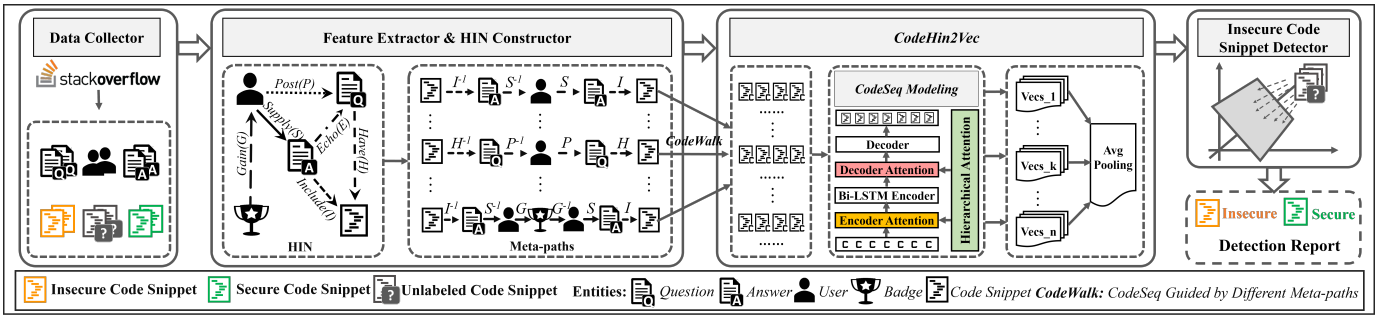

Figure 2: System architecture of *iTrustSO*.

insecure code snippet detection. *ICSD* encodes code content and social relations to construct the HIN, and then learns node representations from HIN to detect insecure code snippets. In this paper, we'd like to take a different tact to see if we can enhance the representation learning of code snippets in the detection of insecure ones. Different from *ICSD*, we utilize HIN to depict relatedness over code snippets to generate code-to-code sequences, based on which sequence to sequence (seq2seq) concept in machine translation is further leveraged to learn representations of code snippets. More specifically, we introduce HIN as an abstract representation, and then use meta-path [30] to incorporate higher-level semantic relations to build up relatedness over the code snippets. Afterwards, considering both code content and social coding properties, we propose a novel seq2seq learning model named *CodeHin2Vec* for representation learning of code snippets. Different from the traditional seq2seq model [31], [21], *CodeHin2Vec* extends the basic encoder-decoder architecture [8] by elaborately devising hierarchical attention mechanism to first learn the context between node embeddings in the input sequence, and then learn the alignments and relevances between hidden layer vectors for the output sequence generation. This allows a refined architecture to cope better with sequence modeling and thus fully exploit code content and HIN structure to learn better representations of code snippets. After that, a classifier is built for insecure code snippet detection. We develop a system called *iTrustSO* shown in Figure 2 integrating our proposed method, which has the following merits:

- It introduces HIN as an abstract representation of Stack Overflow data, and exploits a meta-path based approach to characterize the relatedness over code snippets. The proposed solution provides a natural way of expressing complex relationships in social coding platforms.
- It integrates HIN with seq2seq concept for representation learning. In *iTrustSO*, a new model *CodeHin2Vec* is proposed to seamlessly combine code content and HIN-based relations to learn representations of code snippets, in which code sequences are first generated based on the walk paths guided by different meta-paths; in each code sequence, its elements are represented by the code content feature vectors; then, LSTM using hierarchical attention mechanism is leveraged for code sequence modeling. *CodeHin2Vec* is a generic framework which can also be applicable for other representation learning task.
- Comprehensive experimental studies demonstrate the performance of our developed system *iTrustSO*, which is practical for automatic detection of insecure code snippets.

II. Proposed Method

In this section, we present the detailed approaches of how we represent and detect the code snippets in Stack Overflow.

A. Feature Extraction

Code snippets. Code snippets in Stack Overflow can be first separated from accompanying texts in question and answer threads through pairs of $\langle code\rangle$ $\langle/code\rangle$ tags; afterwards, each code snippet can be represented by a feature vector (i.e., $\mathbf{x}_c$) to denote its code content using *word2vec* [25], [24].

Social coding properties. To characterize a code snippet in Stack Overflow, we not only consider its code content, but also extract its social coding properties including: i) *R1*: *question-have-code* relation describes whether a question thread has a code snippet embedded; ii) *R2*: *answer-include-code* relation denotes that an answer thread includes a code snippet; iii) *R3*: *user-post-question* describes the relationship between a user and a question he/she posts; iv) *R4*: *user-supply-answer* relation represents if a user supplies an answer; v) *R5*: *answer-echo-question* relation denotes if an answer echoes a question; vi) *R6*: *user-gain-badge* relation means a user gains a badge, denoting his/her level (i.e., gold, silver, or bronze) over different contributions (e.g., question, answer, etc.).

B. HIN Construction

This section introduces how to use the extracted entities and social coding properties to represent code snippets in Stack Overflow. We first present the concepts related to HIN:

Definition 2.1: ***Heterogeneous information network (HIN)*** [30]. A HIN is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an entity type mapping ϕ: $\mathcal{V} \rightarrow \mathcal{A}$ and a relation type mapping ψ: $\mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{V}$ denotes the entity set and $\mathcal{E}$ is the relation set, $\mathcal{A}$ denotes the entity type set and $\mathcal{R}$ is the relation type set, and the number of entity types $|\mathcal{A}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The ***network schema*** $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$ for a HIN $\mathcal{G}$ is a graph with nodes as entity types from $\mathcal{A}$ and edges as relation types from $\mathcal{R}$.

For our case, we have five entity types and six types of relations among them; accordingly, the network schema for HIN in our application is shown in Figure 3.

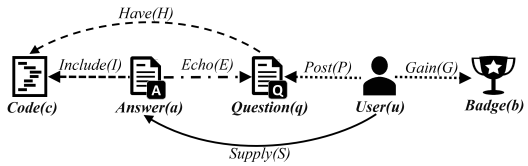

Figure 3: Network schema for HIN in our application.

The different types of entities and relations motivate us to use a machine-readable representation to enrich the semantics of relatedness among code snippets. To handle this, the concept of meta-path [30] to formulate the higher-order relationships among entities in HIN is extended to our application of insecure code snippet detection.

Definition 2.2: ***Meta-path*** [30]. A meta-path $\mathcal{P}$ is a path defined on the graph of network schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \ldots \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \ldots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is the length of $\mathcal{P}$.

Table I: Meta-paths built for insecure code snippet detection

ID	Meta-paths
PID1	$c \xrightarrow{I^{-1}} a \xrightarrow{S^{-1}} u \xrightarrow{S} a \xrightarrow{I} c$
PID2	$c \xrightarrow{H^{-1}} q \xrightarrow{P^{-1}} u \xrightarrow{P} q \xrightarrow{H} c$
PID3	$c \xrightarrow{I^{-1}} a \xrightarrow{E} q \xrightarrow{P^{-1}} u \xrightarrow{P} q \xrightarrow{E^{-1}} a \xrightarrow{I} c$
PID4	$c \xrightarrow{H^{-1}} q \xrightarrow{E^{-1}} a \xrightarrow{S^{-1}} u \xrightarrow{S} a \xrightarrow{E} q \xrightarrow{H} c$
PID5	$c \xrightarrow{I^{-1}} a \xrightarrow{S^{-1}} u \xrightarrow{G} b \xrightarrow{G^{-1}} u \xrightarrow{S} a \xrightarrow{I} c$
PID6	$c \xrightarrow{H^{-1}} q \xrightarrow{P^{-1}} u \xrightarrow{G} b \xrightarrow{G^{-1}} u \xrightarrow{P} q \xrightarrow{H} c$
PID7	$c \xrightarrow{I^{-1}} a \xrightarrow{E} q \xrightarrow{P^{-1}} u \xrightarrow{G} b \xrightarrow{G^{-1}} u \xrightarrow{P} q \xrightarrow{E^{-1}} a \xrightarrow{I} c$
PID8	$c \xrightarrow{H^{-1}} q \xrightarrow{E^{-1}} a \xrightarrow{S^{-1}} u \xrightarrow{G} b \xrightarrow{G^{-1}} u \xrightarrow{S} a \xrightarrow{E} q \xrightarrow{H} c$

Given a network schema with different types of entities and relations, we can enumerate a lot of meta-paths. In our application, based on the collected data, resting on the six different kinds of relationships, we design eight meaningful meta-paths for characterizing relatedness over code snippets, i.e., *PID1-PID8* shown in Table I (symbols are the abbreviations shown in Figure 3). Different meta-paths depict the relatedness between two code snippets at different views. For example, a typical one to formulate the relatedness over code snippets in Stack Overflow is *PID1*: $c \xrightarrow{I^{-1}} a \xrightarrow{S^{-1}} u \xrightarrow{S} a \xrightarrow{I} c$ which means that two code snippets can be connected as they are included in the answers supplied by the same user.

C. CodeHin2Vec

To devise a comprehensive solution to combine both node (i.e., code snippet) content and HIN-based relations for insecure code snippet detection, we observe from our previous work [39] that the HIN-based neighborhood relationships among code snippets can be represented by the code sequences (denoted as CodeSeq) based on different meta-paths. In this way, the generated CodeSeqs can preserve both semantic and structure information of HIN. To further couple CodeSeqs with code content, a straightforward yet novel way is to use the content feature vector $\mathbf{x}_c$ to represent each code snippet in the CodeSeq. To this end, the representation learning of code snippets can be viewed as a sequence modeling task. As LSTM has shown significant improvement in language modeling [4], we leverage its power to seamlessly integrate code content and HIN structure into hidden layer vectors that can be used as the representations of code snippets [21].

Although it is promising to comprehensively utilize LSTM to learn the mapping from the code content sequence to code identity sequence, it still faces the following two challenges: (1) word2vec assigns each code snippet a static embedding vector based on code content which is not context-aware to different sequences it interacts with. For example, as illustrated in Figure 4, guided by the designed meta-paths, we may generate *CodeSeq-A* and *CodeSeq-B*. With function *fileProcess* defined, *Code-1* in *CodeSeq-A* performs as file encryption for Ransomware while *Code-3* in *CodeSeq-B* implements the regular file reading and writing; in this respect, even though *Code-2* listed in both sequences calls the same function *fileProcess*, its embedding vector should be significantly different which may demonstrate insecure potential when interacting with *Code-1* and normal aspect when related to *Code-3*. LSTM is known to learn the sequential dependencies [27], but strict to align the positions of the input sequence; therefore, contextualized code content embeddings may help to refine the hidden-layer information in the early stage. (2) Since LSTM needs to read the whole input sequence to further generate the output sequence, its performance using a basic encoder-decoder architecture may degrade as the length of an input sequence increases [7], [4] which may in turn degenerate the representations learned from hidden layers, especially in our case that code sequences are much longer than the sentences.

Attention mechanism has shown remarkable effectiveness in various sequence modeling tasks, allowing models to learn alignments between different modalities [34], [4], [23], [19]. In this work, to address the challenges above, we propose *CodeHin2Vec* to elaborate a hierarchical attention mechanism into LSTM to fully exploit code content and HIN structure to learn effective representations of code snippets, which first generates CodeSeqs based on the walk paths guided by different meta-paths; and then leverages LSTM with hierarchical attention mechanism for CodeSeq modeling.

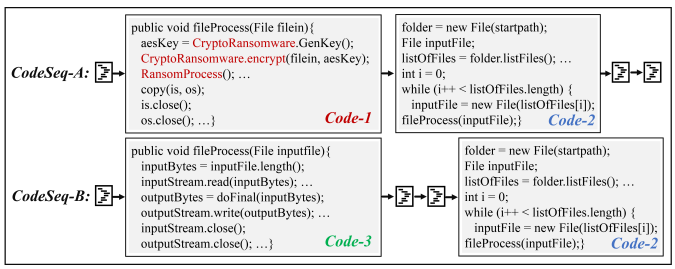

Figure 4: Different contexts among code snippets.

CodeSeq generation guided by different meta-paths. Given a source node v_j in a homogeneous network, the traditional random walk is a stochastic process with random variables $v_j^1, v_j^2, ..., v_j^k$ such that v_j^{k+1} is a node chosen at random from the neighbors of node v_k. The transition probability $p(v_j^{i+1}|v_j^i)$ at step i is the normalized probability distributed over the neighbors of v_j^i by ignoring their node types. However, this mechanism is unable to capture the semantic and structural correlations among different types of nodes in a HIN. In our application, given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, and a set of different meta-paths $\mathbf{P} = \{\mathcal{P}_j\}_{j=1}^n$, each of which is in the form of $A_1 \rightarrow ...A_t \rightarrow A_{t+1}... \rightarrow A_l$, we put a random walker to traverse the HIN. The random walker first randomly chooses a meta-path $\mathcal{P}_k$ from $\mathbf{P}$ and the transition probabilities at step i are defined as follows:

$$p(v^{i+1}|v_{A_t}^i, \mathbf{P}) = \begin{cases} \frac{\lambda}{|\mathbf{P}|}\frac{1}{|N_{A_{t+1}}(v_{A_t}^i)|} \\ \quad \text{if } (v^{i+1}, v_{A_t}^i) \in \mathcal{E}, \phi(v_{A_t}^i) = A_c, \phi(v^{i+1}) = A_{t+1} \\ \frac{1}{|N_{A_{t+1}}(v_{A_t}^i)|} \\ \quad \text{if } (v^{i+1}, v_{A_t}^i) \in \mathcal{E}, \phi(v_{A_t}^i) \neq A_c, \\ \quad \phi(v^{i+1}) = A_{t+1}, (A_t, A_{t+1}) \in \mathcal{P}_k \\ 0 \quad \text{otherwise} \end{cases} \tag{1}$$

where ϕ is the node type mapping function, $N_{A_{t+1}}(v_{A_t}^i)$ denotes the A_{t+1}-type neighborhood of node $v_{A_t}^i$, A_c is entity type of *Code*, and λ is the number of meta-paths starting with $A_c \rightarrow A_{t+1}$. For each walk path, the nodes whose entity types are not *Code* will be removed; then the remaining ones form a CodeSeq, whose element is represented by the content feature vector $\mathbf{x}_c$. In such way, given walk path length l, a CodeSeq is presented as $(\mathbf{x}_{c_1}, \mathbf{x}_{c_2}, ..., \mathbf{x}_{c_l})$.

CodeSeq modeling with LSTM. LSTM learns a mapping from an input sequence to an output sequence. As intermediate states, a hidden vector is generated for each timestep; we can extract it as the embedding vector for the input at that timestep. In our application, we employ an encoder-decoder LSTM architecture [8] for CodeSeq modeling in which two attention layers are elaborately added to improve the quality of representation learning (as illustrated in Figure 5).

Encoder attention: Resting on all the content vectors in the input sequence, the encoder attention layer computes the contextualized embedding for each code snippet as a weighted sum where the weight, also called context score, assigned to each content vector is computed by a dot product of the corresponding pair of content vectors [34]. Specifically, given an input CodeSeq $(\mathbf{x}_{c_1}, \mathbf{x}_{c_2}, ..., \mathbf{x}_{c_l})$, for any two code snippets c_t and c_i, the context score can be calculated as

$$\mathcal{S}(\mathbf{x}_{c_t}, \mathbf{x}_{c_i}) = \mathbf{x}_{c_t} \top \mathbf{x}_{c_i}, \tag{2}$$

where $\top$ denotes the dot product, and thus the contextualized embedding for code snippet c_t can be computed as

$$\tilde{\mathbf{x}}_{c_t} = \sum_{i=1}^{l} \frac{\exp(\mathcal{S}(\mathbf{x}_{c_t}, \mathbf{x}_{c_i}))}{\sum_{j=1}^{l} \exp(\mathcal{S}(\mathbf{x}_{c_t}, \mathbf{x}_{c_j}))} \mathbf{x}_{c_i}. \tag{3}$$

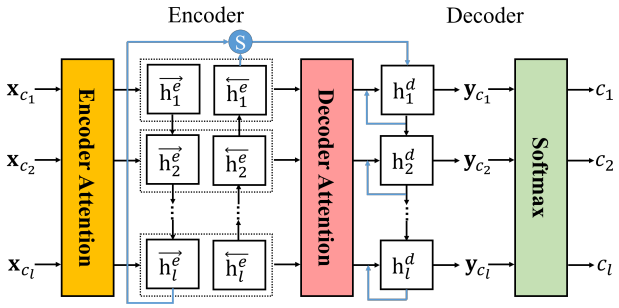

Figure 5: Architecture of LSTM using hierarchical attention.

In this sense, a CodeSeq can be refined as $(\tilde{\mathbf{x}}_{c_1}, \tilde{\mathbf{x}}_{c_2}, ..., \tilde{\mathbf{x}}_{c_l})$, which will be used as the actual input sequence.

Encoder: The encoder reads $(\tilde{\mathbf{x}}_{c_1}, \tilde{\mathbf{x}}_{c_2}, ..., \tilde{\mathbf{x}}_{c_l})$ through the hidden layer function $\mathcal{H}$ so that each hidden layer vector $\mathbf{h}_t^e$ at timestep t can be denoted as

$$\mathbf{h}_t^e = \mathcal{H}(\tilde{\mathbf{x}}_{c_t}, \mathbf{h}_{t-1}^e), \tag{4}$$

where $\mathcal{H}$ is implemented using memory cells to store information, which can be formulated as the following composite functions [16]:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\tilde{\mathbf{x}}_{c_t} + \mathbf{W}_{hi}\mathbf{h}_{t-1}^e + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \tag{5}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\tilde{\mathbf{x}}_{c_t} + \mathbf{W}_{hf}\mathbf{h}_{t-1}^e + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f) \tag{6}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tanh(\mathbf{W}_{xc}\tilde{\mathbf{x}}_{c_t} + \mathbf{W}_{hc}\mathbf{h}_{t-1}^e + \mathbf{b}_c) \tag{7}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\tilde{\mathbf{x}}_{c_t} + \mathbf{W}_{ho}\mathbf{h}_{t-1}^e + \mathbf{W}_{co}\mathbf{c}_{t-1} + \mathbf{b}_o) \tag{8}$$

$$\mathbf{h}_t^e = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \tag{9}$$

where σ is the logistic sigmoid function, $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$ are the input gate, forget gate, output gate, and cell activation vectors respectively, $\mathbf{W}$s are the weight matrices, $\mathbf{b}$s are the bias vectors, and $\circ$ is the point-wise product between two vectors. Since the input sequence has no direction, in order to learn both the forward and backward sequential dependency information, we utilize bidirectional encoder so that hidden layer vector $\mathbf{h}_t^e$ at timestep t can be concatenated as $\mathbf{h}_t^e = [\overrightarrow{\mathbf{h}_t^e}; \overleftarrow{\mathbf{h}_t^e}]$. After forward and backward reading CodeSeq, the concatenation of the last two hidden states $[\overrightarrow{\mathbf{h}_l^e}; \overleftarrow{\mathbf{h}_1^e}]$ is used as the summary vector $\mathbf{s}$ of the whole input sequence.

Decoder attention: The decoder attention layer exploits all the hidden states of the encoder to compute the aligned and joint information as the context vector [4], [23], which is integrated with the summary vector $\mathbf{s}$ to extract the target code identity. Similar to the encoder attention, the alignment scores need to be first defined to formulate such context vector as a weighted sum. Note that, unlike the dot product attention, decoder attention should allow the gradient of the cost function to be backpropagated through [4]. We accordingly use a simple feed-forward neural network to compute the alignment score

$$\boldsymbol{\alpha}_t = \mathbf{W}_\alpha^2 ReLU(\mathbf{W}_\alpha^1 \mathbf{h}_t^e + \mathbf{b}_\alpha^1) + \mathbf{b}_\alpha^2, \tag{10}$$

where $\mathbf{W}_\alpha$s and $\mathbf{b}_\alpha$s denote the weight matrices and the bias

vectors, and the alignment score vector $\boldsymbol{\alpha}_t$ trained by all the other hidden states of the encoder reflects the importance of $\mathbf{h}_t^e$ in generating $\mathbf{y}_t$. The context vector for $\mathbf{h}_t^e$ can thus be

$$\tilde{\mathbf{h}}_t^e = \sum_{i=1}^{l} \frac{\exp(\alpha_{t,i})}{\sum_{j=1}^{l} \exp(\alpha_{t,j})} \mathbf{h}_i^e. \quad (11)$$

Decoder: The decoder takes the summary vector $\mathbf{s}$ as input (i.e., $\mathbf{h}_0^d = \mathbf{s}$) and generates a sequence of target hidden states; each hidden state $\mathbf{h}_t^d$ at timestep t can be calculated as

$$\mathbf{h}_t^d = \mathcal{H}(\mathbf{0}, \mathbf{h}_{t-1}^d), \quad (12)$$

where $\mathbf{0}$ is an all-zero vector. Given the target hidden state $\mathbf{h}_t^d$ and the context vector $\tilde{\mathbf{h}}_t^e$, we concatenate them to formulate an attentional hidden state $\tilde{\mathbf{h}}_t^d = [\tilde{\mathbf{h}}_t^e;\ \mathbf{h}_t^d]$ [23]. Accordingly, the output vector $\mathbf{y}_t \in \mathbb{R}^{|\mathcal{V}|}$ can be generated as follows [16]

$$\mathbf{y}_t = \sigma(\mathbf{W}_{hy}\tilde{\mathbf{h}}_t^d + \mathbf{b}_y). \quad (13)$$

$\mathbf{y}_t$ is capable to predict the real code snippet c_t through a softmax layer. The sequence loss $\mathcal{L}$ is adopted to measure the correctness of decoding, which is computed as

$$\mathcal{L} = -\sum_{t=1}^{l} \log p(c_t|\mathbf{y}_t) = -\sum_{t=1}^{l} \log \frac{\exp(y_t^{c_t})}{\sum_{i=1}^{|\mathcal{V}|} \exp(y_t^{c_i})}. \quad (14)$$

The weights can be efficiently calculated with backpropagation through time [35], [16], and the LSTM model can then be trained using Adam optimization algorithm.

For the generated CodeSeqs guided by different meta-paths, each code snippet may appear in multiple CodeSeqs. Suppose that code snippet c_t exists in $|c_t|$ CodeSeqs, by doing avg pooling over all $\mathbf{h}_i^e$'s for code snippet c_t, $\forall i = 1, ..., |c_t|$, we obtain an embedding $\mathbf{h}$ for each code snippet

$$\mathbf{h} = avgPooling(\{\mathbf{h}_i^e : i = 1, ..., |c_t|\}). \quad (15)$$

Using *CodeHin2Vec*, the mapped feature vectors of code snippets, encoding the informaiton of code content and HIN-based relations, can be fed to a classifier to train the classification model, based on which the unlabeled code snippets can be predicted if they are insecure or not.

III. Experimental Results and Analysis

In this section, we fully evaluate the performance of *iTrustSO* in insecure code snippet detection. We consider Java programming language for Android app as a case study. Based on our prior work *ICSD* [39], in this paper, we further expand our data collection and annotation from Stack Overflow: (1) using our developed crawlers, we collect 505,548 question threads and 719,430 answer threads posted by 229,394 users including 821,792 code snippets, through March 2010 to October 2018; (2) we also expand our annotated data in [39] to finally obtain 21,989 labeled code snippets (10,013 are insecure while 11,976 are secure) as the ground truth to evaluate different detection methods. To quantitatively validate the effectiveness of different methods, we use accuracy (ACC) and F1 measure (F1) as the performance measures.

A. Evaluation of Different Meta-paths

In this set of experiments, given a specific meta-path scheme, we use a basic LSTM to learn the latent representations of code snippets in HIN, which is then fed to SVM for detection. Here we perform 10-fold cross validations for evaluation. The experimental results are shown in Table II, from which we can see that different meta-paths indeed show different performances: (1) *PID1*, *PID3*, *PID5*, and *PID7* perform better than *PID2*, *PID4*, *PID6*, and *PID8*; the reason behind this is that the code snippets posted in the answer threads are more likely to be reused by the developers than the ones posted in question threads, and thus they have closer connections. (2) *PID3* outperforms the others, which indicates that its semantics reflecting the insecure code snippet detection problem is better than the others. (3) *PID9* using different meta-paths is more expressive than individuals in depicting the code snippets and thus achieve better performance.

Table II: Detection Results of different meta-paths

ID	Meta-paths included	Recall	Precision	ACC	F1
PID1	–	0.8481	0.7956	0.8316	0.8210
PID2	–	0.8098	0.7491	0.7899	0.7783
PID3	–	0.8596	0.8119	0.8454	0.8351
PID4	–	0.8344	0.7769	0.8155	0.8046
PID5	–	0.8605	0.8086	0.8437	0.8337
PID6	–	0.8140	0.7588	0.7975	0.7854
PID7	–	0.8042	0.7444	0.7851	0.7731
PID8	–	0.7843	0.7203	0.7631	0.7509
PID9	**P** = (PID1,..., PID8)	0.8785	0.8415	0.8693	0.8596

B. Evaluation of Attentions

In this set of experiments, we'd like to assess whether the hierarchical attention mechanism devised in our model is meaningful for representation learning. To this end, we explore the performances of basic LSTM without attention (LSTM-b), LSTM with encoder attention (LSTM-e), LSTM with decoder attention (LSTM-d), and *CodeHin2Vec*. The better detection result implies that the learn representations take better advantage of the corresponding sequence learning architecture. From the results illustrated in Figure 6, we have the following observations: (1) LSTM-e and LSTM-d with single attention layer both outperform LSTM-b without attention; (2) *CodeHin2Vec* achieves the most promising performance for fully utilizing the contextualized input embeddings and the aligned information from the hidden states of the encoder. In other words, *CodeHin2Vec* has potential to let LSTM learn better sequential dependencies and code better with the sequence

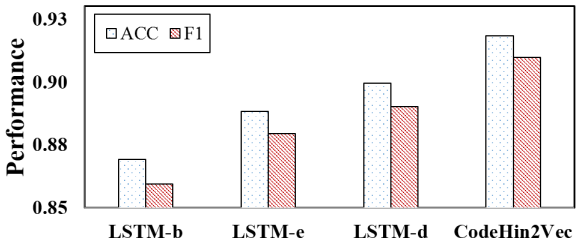

Figure 6: Attention evaluation.

Table III: Comparisons of *CodeHin2Vec* with other network representation learning methods in insecure code snippet detection

Metric	Method	Feature	10%	20%	30%	40%	50%	60%	70%	80%	90%
ACC	word2vec	Content	0.6554	0.6757	0.6989	0.7105	0.7379	0.7590	0.7725	0.7730	0.7753
	DeepWalk	Relation	0.6263	0.6632	0.6678	0.6809	0.7087	0.7132	0.7349	0.7403	0.7430
	metapath2vec	Relation	0.7241	0.7415	0.7562	0.7747	0.7898	0.8065	0.8035	0.8204	0.8312
	TADW	Content&Relation	0.7659	0.7761	0.7902	0.8029	0.8144	0.8312	0.8394	0.8478	0.8537
	ICSD	Content&Relation	0.8026	0.8234	0.8487	0.8588	0.8783	0.8884	0.8968	0.9035	0.9107
	CodeHin2Vec	Content&Relation	0.7983	0.8281	0.8630	0.8665	0.8752	0.8787	0.8975	0.9175	0.9223
F1	word2vec	Content	0.6292	0.6507	0.6756	0.6875	0.7166	0.7379	0.7519	0.7530	0.7560
	DeepWalk	Relation	0.6023	0.6415	0.6439	0.6551	0.6871	0.6911	0.7139	0.7203	0.7233
	metapath2vec	Relation	0.7005	0.7199	0.7356	0.7552	0.7711	0.7884	0.7853	0.8038	0.8147
	TADW	Content&Relation	0.7446	0.7565	0.7717	0.7855	0.7977	0.8147	0.8239	0.8330	0.8390
	ICSD	Content&Relation	0.7855	0.8083	0.8338	0.8454	0.8662	0.8769	0.8866	0.8933	0.9015
	CodeHin2Vec	Content&Relation	0.7831	0.8135	0.8502	0.8536	0.8624	0.8665	0.8873	0.9084	0.9160

extraction from the proper context information, which in turn generates better representations for code snippets.

C. Evaluation of CodeHin2Vec

Here, *CodeHin2Vec* is evaluated by comparisons with several representation learning methods: (1) word2vec [24] is a baseline using code content information; (2) DeepWalk [26] is a homogeneous network embedding method leveraging relation information; (3) metapath2vec [11] is a HIN embedding model utilizing HIN-based relations; (4) TADW [36] considers both content and relation information for homogeneous network representation learning; (5) ICSD [39] takes content and relation into account in HIN. For DeepWalk and TADW, we ignore the heterogeneous property of HIN and directly feed the HIN for embedding; in metapath2vec, a walk path is generated based on a single meta-path scheme; in ICSD, code content is extracted as keywords to be devised to HIN. The parameter settings used for *CodeHin2Vec* are in line with typical values used for the baselines: content dimension $c = 300$, vector dimension $d = 200$, walks per node $r = 10$, walk length $l = 80$ (TADW: walk steps are set to 2), and window size $w = 10$. To facilitate the comparisons, we randomly select a portion of labeled code snippets (ranging from 10% to 90%) for training and the remaining ones for testing. SVM is used as the classification model for all the methods. Table III illustrates the detection results: *CodeHin2Vec* outperforms all baselines in terms of *ACC* and *F1* in most cases. That is to say, *CodeHin2Vec* learns significantly better code snippet representation than current state-of-the-art methods. The success of *CodeHin2Vec* lies in the seamless integration of code content with HIN-based relations for representation learning, which leverages the advantage of (1) CodeSeq generation based on the different meta-paths and (2) the CodeSeq modeling power of LSTM using hierarchical attentions.

D. Evaluation of Parameters

In this set of experiments, we first conduct the **sensitivity** analysis of how different choices of parameters will affect the performance of *CodeHin2Vec*. From the results shown in Figure 7(a) and 7(b), we can observe that the balance between computational cost (number of walks per node r and walk length l in x-axis) and efficacy (F1 in y-axis) can be achieved when $r \geq 10$ and $l \geq 80$. As shown in Figure 7(c), we

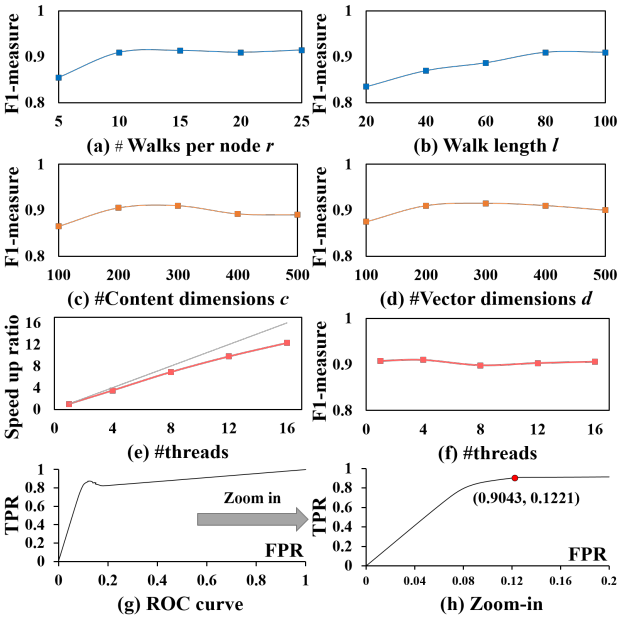

Figure 7: Parameter sensitivity evaluation.

can see that the performance tends to be stable once content vector dimension c reaches around 200 to 300; similarly, from Figure 7(d) we can find that the performance inclines to be stable when vector dimensions d increases to around 200 to 400. Overall, *CodeHin2Vec* is not strictly sensitive to these parameters, and is able to reach high performance under a cost-effective parameter choice. We then further evaluate the **scalability** of *CodeHin2Vec* which can be parallelized for optimization. We run the experiments using the default parameters with different number of threads (i.e., 1, 4, 8, 12, 16), each of which utilizes one CPU core. Figure 7(e) shows the speed-up of *CodeHin2Vec* deploying multiple threads over the single-threaded case, which reveals that the model achieves acceptable sub-linear speed-ups as the line is close to the optimal line; while Figure 7(f) shows that the performance remains stable when using multiple threads for model updating. Overall, the proposed system are efficient and scalable for large-scale HIN with large numbers of nodes. For **stability** evaluation, Figure 7(g) shows the ROC curves of *CodeHin2Vec* based on the 10-fold cross validations; it achieves an average 0.9043 TPR at the 0.1221 FPR for detection.

E. Comparisons with Traditional Machine Learning Methods

In this set of experiments, *iTrustSO* is compared with other traditional machine learning methods. For these methods, we construct three types of features: *f–1*: content-based features (i.e., $\mathbf{x}_c$); *f–2*: two original relation-based features (i.e., *R1* and *R2*); *f–3*: augmented features of content-based features and *R1–R2*. Based on these features, we consider two typical classification models, i.e., Naive Bayes (NB) and SVM. The experimental results shown in Table IV illustrates that feature engineering (*f-3*) helps the performance of machine learning, but *iTrustSO* leveraging the knowledge represented as HIN and the long-range influence among code snippets learned from LSTM with attentions significantly outperforms other baselines. This again demonstrates that, to detect the insecure code snippets, *iTrustSO* using *CodeHin2Vec* to seamlessly integrate node content with HIN relations is able to build the higher-level semantic and structural connection between code snippets with a more expressive and comprehensive view and thus achieves better detection performance.

Table IV: Comparisons of other machine learning methods

Metric	NB			SVM			*iTrustSO*
	f-1	***f-2***	***f-3***	***f-1***	***f-2***	***f-3***	
ACC	0.7493	0.6854	0.7952	0.7753	0.7034	0.8415	0.9184
F1	0.7284	0.6613	0.7834	0.7560	0.6793	0.8317	0.9098

F. Case Studies

To gain deeper insights into the security-related risks of modern social coding platform of Stack Overflow, in this section, based on our developed system *iTrustSO*, we further analyze 8,105 detected insecure code snippets in Stack Overflow. We categorize the security risks or vulnerabilities resulted from these insecure code snippets into six types: (1) Android Manifest configuration (28.43%), (2) WebView component (03.20%), (3) data security (22.62%), (4) file directory traversal (15.15%), (5) implicit intents (09.06%), and (6) security checking (21.55%). From these categories, we can observe that the most prevalent insecure code infiltration for Android apps in Stack Overflow is Android Manifest configuration (28.43%), which would pose serious threats to Android apps since Manifest retains all the components, and structure information for an app [6]. Such insecure code snippets include violation of least permission request, the component features being configured as exported, and data backup and debuggable setting being turned on, etc. For example, as shown in Figure 8, code (a) configures activity component as exported, and code (b) configures data backup and debuggable setting as turned on, both of which could be exploited by cyberattackers to perform the attacks on Android apps. Actually, these types of code snippets were provided by many experienced and inexperienced users, which can thus be easily copied and pasted by other users in their answer threads responding to different posted questions.

```
<activity
 android:name
  ="com.mp.app.MainActivity"
 android:exported="true"
 android:screenOrientation="portrait"
 android:permission=
 "com.mymundane.app.mypermission"/>
```

(a)

```
<application
 android:name=".App"
 android:allowBackup="true"
 android:supportsRtl="true"
 android:debuggable="true"
 android:label="@string/app_name"
 android:theme="@style/AppTheme" >
```

(b)

Figure 8: Insecure codes with manifest vulnerabilities.

The study based on the detected insecure code snippets in Stack Overflow using our developed system *iTrustSO* demonstrates that knowledge gained from social coding platform data mining could facilitate the understanding and thus enhance its code security in modern software programming ecosystem.

IV. Related Work

There have been many works on knowledge discovery from Stack Overflow data [10], [20], [22], [5], [2] - from gamification motivation for voluntary contributions [5], discussion interest trend [20] to developer interaction [2] and repair patterns from extracted code samples [22]. However, most of these works have focused in Stack Overflow semantics and users behavior but rarely addressed the issue of code security analysis. The only exceptions appear to be [1] and [15] which both exploited Android application (app) codes as a case study to evaluate the security of information source in Stack Overflow. Though those research results are promising, [1] only performed empirical studies while [15] merely analyzed the code snippet itself without considering any relationship to other Stack Overflow data. Different from the existing works, in this paper, to detect the insecure code snippets in Stack Overflow, we propose to utilize not only the code content, but also social coding properties, based on which, the code snippets are depicted by a structured HIN.

HIN has been intensively deployed to various applications, such as authorship identification [40], malware detection [18], [12], [38], and health intelligence [14], [13]. To reduce the high computation and space cost in network mining, many efficient network embedding methods have been proposed, including homogeneous network representation learning (e.g., DeepWalk [26], node2vec [17], LINE [32], and TADW [36]) and HIN representation learning (e.g., ESim [28], metapath2vec [11] and HIN2vec [37]). Unfortunately, these methods cannot be directly employed in our application, i.e., we aim to seamlessly combine both code content and HIN-based relations for insecure code snippet detection in Stack Overflow. To tackle this challenge, our previous work [39] proposed an automatic system *ICSD* over HIN which treats keywords extracted from code content as a type of entities that are further integrated with other social coding entities to facilitate HIN representation learning. In this paper, we take a different tact that represents code content as feature vector while extracts HIN as pure relations over code snippets, and then inspired by the effectiveness of attention mechanism in various learning tasks [34], [23], [4], we further propose *CodeHin2Vec* to embed HIN-based relations with code content through a hierarchical attention-based sequence learning model for the representation learning.

V. Conclusion

To enhance code security in modern social coding platforms, in this paper, we exploit social coding properties in addition to code content for automatic detection of insecure code snippets in Stack Overflow. To depict the code snippets, we not only analyze the code content, but also utilize various kinds of relations among users, badges, questions, answers and code snippets in Stack Overflow. To model the rich semantic relationships, we first introduce a structured HIN for representation and then use meta-path based approach to incorporate higher-level semantics to build up relatedness over code snippets. Later, we propose a novel hierarchical attention-based sequence learning model named *CodeHin2Vec* to seamlessly embed code content with HIN-based relations for representation learning. After that, a classifier is built for insecure code snippet detection. The experimental results based on the data collections from Stack Overflow demonstrate that *iTrustSO* integrating our proposed method outperforms alternative approaches in insecure code snippet detection.

Acknowledgement

This work is partially supported by the NSF under grants CNS-1618629, CNS-1814825, CNS-1845138 and OAC-1839909, the DoJ/NIJ under grant NIJ 2018-75-CX-0032, and the WV HEPC Grant (HEPC.dsr.18.5).

References

[1] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for the impact of information sources on code security," in *S&P*, 2016.

[2] T. Ahmed and A. Srivastava, "Understanding and evaluating the behavior of technical users. a study of developer interaction at stackoverflow," *Hum. Cent. Comput. Inf. Sci.*, vol. 7, no. 8, 2017.

[3] AttackFlow, "Watch out for insecure stackoverflow answers," in *https://www.attackflow.com/Blog/StackOverflow*, 2017.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[5] H. Cavusoglu, Z. Li, and K.-W. Huang, "Can gamification motivate voluntary contributions? the case of stackoverflow q&a community," in *CSCW*, 2015.

[6] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC)*, 2017, pp. 362–372.

[7] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *arXiv:1406.1078*, 2014.

[9] J. Coogle, J. Gajjar, and C. Greco, "Stackintheflow: Stackoverflow search engine," in *VCU Capstone Design Expo Posters*, 2017.

[10] D. Czyczyn-Egird and R. Wojszczyk, "Determining the popularity of design patterns used by programmers based on the analysis of questions and answers on stackoverflow.com social network," in *CCIS*, 2016.

[11] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017.

[12] Y. Fan, S. Hou, Y. Zhang, Y. Ye, and M. Abdulhayoglu, "Gotcha-sly malware! scorpion: a metagraph2vec based malware detection system," in *KDD*, 2018, pp. 253–262.

[13] Y. Fan, Y. Zhang, Y. Ye, and X. Li, "Automatic opioid user detection from twitter: Transductive ensemble built on different meta-graph based similarities over heterogeneous information network," in *IJCAI*, 2018.

[14] Y. Fan, Y. Zhang, Y. Ye, and W. Zheng, "Social media for opioid addiction epidemiology: Automatic detection of opioid addicts from twitter and case studies," in *CIKM*, 2017.

[15] F. Fischer, K. Bottinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy and paste on android application security," in *S&P*, 2017.

[16] A. Graves, "Generating sequences with recurrent neural networks," in *Arxiv preprint arXiv:1308.0850*, 2013.

[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

[18] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *KDD*, 2017.

[19] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," *arXiv preprint arXiv:1702.00887*, 2017.

[20] M. Linares-Vasquez, G. Bavota, M. D. Penta, and R. Oliveto, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *ICPC '14*, 2014, pp. 83–94.

[21] J. Liu, Z. He, L. Wei, and Y. Huang, "Content to node: Self-translation network embedding," in *KDD*, 2018, pp. 1794–1802.

[22] X. Liu and H. Zhong, "Mining stackoverflow for program repair," in *SANER 2018*, 2018, pp. 118–129.

[23] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

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

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.

[26] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD '14*, 2014, pp. 701–710.

[27] M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power, "Semi-supervised sequence tagging with bidirectional language models," *arXiv preprint arXiv:1705.00108*, 2017.

[28] J. Shang, M. Qu, J. Liu, L. M. Kaplan, J. Han, and J. Peng, "Meta-path guided embedding for similarity search in large-scale heterogeneous information networks," in *arXiv:1610.09769*, 2016.

[29] StackExchange, "Stackexchange statistics," in *https://stackexchange.com/sites#traffic*, 2018.

[30] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *PVLDB*, 2011.

[31] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[32] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015.

[33] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *SocialCom*, 2013.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[35] R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Back-propagation: Theory, Architectures and Applications*, 1995.

[36] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *IJCAI*, 2015.

[37] T. yang Fu, W.-C. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *CIKM*, 2017.

[38] Y. Ye, S. Hou, L. Chen, J. Lei, W. Wan, J. Wang, Q. Xiong, and F. Shao, "Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection," in *IJCAI*, 2019.

[39] Y. Ye, S. Hou, L. Chen, X. Li, L. Zhao, S. Xu, J. Wang, and Q. Xiong, "Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network," in *ACSAC*, 2018.

[40] Y. Zhang, Y. Fan, W. Song, S. Hou, Y. Ye, X. Li, L. Zhao, C. Shi, J. Wang, and Q. Xiong, "Your style your identity: Leveraging writing and photography styles for drug trafficker identification in darknet markets over attributed heterogeneous information network," in *WWW*, 2019.