

TITLE

MALICIOUS CODE INJECTION DETECTOR

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of and priority under 35 U.S.C. § 119(e) to U.S. Provisional Application Serial No. 63/511,746, filed July 3, 2023, entitled "MALICIOUS CODE INJECTION DETECTOR," the contents of which is hereby incorporated by reference in its entirety herein.

TECHNICAL FIELD

[0002] At least some aspects of the present disclosure relate to Programming Language Processing (PLP), such as, for example, detecting a feature of interest in source code using a neural network model.

BACKGROUND

[0003] Malicious actors may target e-commerce websites as a means for stealing consumers' personal information. For example, credit card fraudsters may attempt to inject malicious code into the source code of an e-commerce website's checkout webpage. This malicious code injection can be designed to collect credit card credentials provided by consumers while conducting a transaction via the website's checkout webpage. The fraudsters may then attempt to conduct fraudulent transactions with the stolen credit card credentials. These types of attacks can be mitigated by detecting and removing the malicious code injections.

[0004] Detecting malicious code injections can be a complex and time-consuming task. For example, a transaction service provider may have an interest in detecting and addressing malicious code injections that may exist in the source code of thousands of different e-commerce websites. Current approaches to detecting malicious code injections can therefore rely on comparing the source code for each webpage against a list of known malware code signatures. The list of known malware signatures is often manually created and may need to be manually updated as additional known malware signatures are identified. Thus, current approaches to detecting malicious code injections may not be able to detect new and/or unknown malware signatures that have yet to be identified and added to the list known malware signatures.

[0005] Accordingly, there exists a need for alternate systems, methods, and devices for detecting features of interest in source code, such as, for example, malicious code injections in source code. The present disclosure provides various solutions that employ a neural

network model for detecting features of interest in source code.

SUMMARY

[0006] In various aspects, the present disclosure provides computer-implemented method for detecting a feature of interest in source code. The computer-implemented method can include receiving, by a text preprocessing module, a candidate source code and segmenting, by the text preprocessing module, the candidate source code into snippets. The computer-implemented method can further include transforming, by an encoder model, the snippets to vectors. Each of the vectors may correspond to a different one of the snippets. The computer implemented-method can further include generating, by an embedding processing module, a tensor based on the vectors and applying, by the embedding processing module, the tensor to a decoder model. The computer-implemented method can further include determining, by the decoder model, a probability of a feature of interest being in the candidate source code.

[0007] In various aspects, the present disclosure provides a source code classification system. The source code classification system can include an online model inference configured to determine a probability that a candidate source code includes a feature of interest. The online model inference can include a text preprocessing module, an encoder model, an embedding processing module, and a decoder model. The text preprocessing module can be configured to segment the candidate source code into snippets. The encoder model can be configured to transform the snippets to vectors. Each of the vectors may correspond to a different one of the snippets. The embedding processing module can be configured to generate a tensor based on the vectors. The decoder model can be configured to determine the probability that the candidate source code includes the feature of interest based on the tensor.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] In the description, for purposes of explanation and not limitation, specific details are set forth, such as particular aspects, procedures, techniques, etc. to provide a thorough understanding of the present technology. However, it will be apparent to one skilled in the art that the present technology may be practiced in other aspects that depart from these specific details.

[0009] The accompanying drawings, together with the detailed description below, are incorporated in and form part of the specification, and serve to further illustrate aspects of concepts that include the claimed disclosure and explain various principles and advantages

of those aspects.

[0010] The apparatuses and methods disclosed herein have been represented where appropriate by conventional symbols in the drawings, showing only those specific details that are pertinent to understanding the various aspects of the present disclosure so as not to obscure the disclosure with details that will be readily apparent to those of ordinary skill in the art having the benefit of the description herein.

[0011] FIG. 1 illustrates a diagram of a source code classification system, according to at least one aspect of the present disclosure.

[0012] FIG. 2 illustrates a block diagram of a module configured to classify source code, according to at least one aspect of the present disclosure.

[0013] FIG. 3 illustrates a block diagram of a Bi-directional long short term memory (Bi-LSTM) model, according to at least one aspect of the present disclosure.

[0014] FIG. 4 is a flow diagram of a method for classifying source code, which may be carried out using the system of FIG. 1, according to at least one aspect of the present disclosure.

[0015] FIG. 5 is a flow diagram of a method for classifying source code, which may be carried out using the module of FIG. 2, according to at least one aspect of the present disclosure.

[0016] FIG. 6 is a block diagram of a computer apparatus with data processing subsystems or components, according to at least one aspect of the present disclosure.

[0017] FIG. 7 is a diagrammatic representation of an example system that includes a host machine, according to at least one aspect of the present disclosure.

[0018] Corresponding reference characters indicate corresponding parts throughout the several views. The exemplifications set out herein illustrate various aspects of the present disclosure, in one form, and such exemplifications are not to be construed as limiting the scope of the disclosure in any manner.

DESCRIPTION

[0019] Before explaining various forms of the vicinity use card, it should be noted that the illustrative forms disclosed herein are not limited in application or use to the details of construction and arrangement of components illustrated in the accompanying drawings and description. The illustrative forms may be implemented or incorporated in other forms, variations and modifications, and may be practiced or carried out in various ways. Further,

unless otherwise indicated, the terms and expressions utilized herein have been chosen for the purpose of describing the illustrative forms for the convenience of the reader and are not for the purpose of limitation thereof. Also in the following description, it is to be understood that terms such as “forward,” “rearward,” “left,” “right,” “above,” “below,” “upwardly,” “downwardly,” and the like are words of convenience and are not to be construed as limiting terms.

[0020] As described above, malicious actors may attempt to inject malicious code into the source code of an e-commerce website, such as the website’s checkout webpage. The malicious code injection may be designed to collect various consumer credentials. Current approaches to detecting malicious code injections can rely on comparing the webpage source code against a list of known malware code signatures. However, the list of known malware signatures is often manually created and may need to be manually updated as additional known malware signatures are identified. Thus, current approaches to detecting malicious code injections may not be able to detect new and/or unknown malware signatures that have yet to be discovered and added to the list. Accordingly, there exists a need for alternate systems, methods, and devices for detecting features of interest, such as malicious code injections, in source code.

[0021] The present disclosure provides systems, methods, and devices for detecting features of interest in source code. For example, in various aspects, the present disclosure provides a source code classification system. The source code classification system can comprise an online model inference and an offline training model. The online model inference determines a probability that candidate source code includes a feature of interest. The feature of interest can be, for example, malicious code injected into the candidate source code. The offline training model is trained based on labeled data and can be implemented as the online model inference. In some aspects, the offline training model is automatically retrained based on candidate source code classified by the online model inference, treating the classified candidate source code as labeled data. The online model inference can be automatically updated based on the retrained offline training model. Thus, the source code classification system can employ a feedback process for automated retraining.

[0022] In some aspects, the online model inference can include a text preprocessing module, an encoder model, an embedding processing module, and a decoder model. The text preprocessing module can segment the candidate source code into snippets. The encoder model can transform the snippets to vector embeddings. The embedding processing module can generate a tensor embedding based on the vector embeddings. The decoder model can determine the probability that the candidate source code includes the

feature of interest based on the tensor embedding. In one aspect, the decoder model comprises a bidirectional long-short term memory (Bi-LSTM) neural network. Thus, the online model inference can employ deep learning techniques to identify both known (e.g., previously identified) features of interest and unknown (e.g., not previously identified) features of interest in candidate source code.

[0023] The devices, systems, and methods disclosed herein can provide numerous technological benefits. For example, unlike conventional methods that may rely on the collection and comparison of candidate source code to a list of known features of interest (e.g., a list of known malicious code signatures), various devices, systems, and methods disclosed herein employ deep learning techniques to identify features of interest in candidate source code that may have not yet been identified, thereby providing an improvement in malware detection technology. Thus, various devices, systems, and methods of the present disclosure are used to identify malicious code injected into the source code of an e-commerce checkout webpage even in cases where the fraudsters employ a new type of malicious code and/or a new method of attack that has not yet been discovered.

[0024] As another example, various devices, systems, and methods disclosed herein provide an improvement in technology by employing a feedback process that automatically retrains the online model inference based on classified candidate source code. This automated retraining can enable the online model inference to be automatically updated as new code signatures are identified. Thus, for example, as the type of malicious code and/or methods of attack employed by fraudsters evolve, so can the ability of the online model inference to identify the corresponding code signatures.

[0025] Moreover, various devices, systems, and methods disclosed herein can be scaled to investigate candidate source code from thousands of different webpages, thereby identifying features of interest in candidate source code at a scale not practically performed by the human mind. Furthermore, the deep learning techniques that may be employed by the devices, systems, and methods disclosed herein can identify patterns in code signatures to identify previously unknown features of interest in a way that is not practically performed by the human mind.

[0026] FIG. 1 is a diagram of a source code classification system 100, according to at least one aspect of the present disclosure. The source code classification system 100 includes a source code classification server 102 that is in communication with one or more website servers 140₁, 140₂, . . . 140_n via a network 160. Each of the one or more website servers 140₁, 140₂, . . . 140_n hosts one or more webpages 141₁, 142₂, . . . 142_n. Each of the one or more webpages 141₁, 142₂, . . . 142_n can be generated based on source code stored

by the memory of a corresponding one of the website servers 140₁, 140₂, . . . 140_n. The source code for any of the one or more webpages 141₁, 142₂, . . . 142_n may comprise, for example, JavaScript source code, HTML (HyperText Markup Language) source code, CCS (Cascading Style Sheets), or any other type of source code suitable for generating webpages, or a combination thereof. Any one or more of the website servers 140₁, 140₂, . . . 140_n may be generally referred to herein as “a website server 140.” Any one or more of the webpages 141₁, 142₂, . . . 142_n may be generally referred to herein as “a webpage 142.”

[0027] The network 160 includes one or more wired and/or wireless networks. For example, the network 160 may include a cellular network (e.g., a long-term evolution (LTE) network, a fourth generation (4G) network, a fifth generation (5G) network, a code division multiple access (CDMA) network, etc.), a public land mobile network (PLMN), a local area network (LAN), a wide area network (WAN), a metropolitan area network (MAN), a telephone network (e.g., the public switched telephone network (PSTN)), a private network, an ad hoc network, an intranet, the Internet, a fiber optic-based network, a cloud computing network, and/or the like, and/or a combination of these or other types of networks.

[0028] The source code classification server 102 includes an online model inference 104 and an offline training model 106. In some aspects, the online model inference 104 and/or the offline training model 106 are generated based on instructions that are stored by a memory of the source code classification server 102 and are executable by a processor of the source code classification server 102. In other aspects, the online model inference 104 and/or the offline training model 106 are specially configured hardware units. In some aspects, the online model inference 104 and/or the offline training model 106 can be implemented as or otherwise comprise the module 210 described herein with respect to FIG. 2. The source code classification server 102, including the online model inference 104 and the offline training model 106, can classify source code from a webpage 142, as described herein with respect to the method 400 of FIG. 4.

[0029] Referring to FIG. 1, the online model inference 104 can receive (e.g., retrieve) and classify source code of a webpage 142 from a corresponding website server 140. The online model inference 104 can comprise an encoder model and a decoder model to perform the classification. The encoder model can generate vector embeddings corresponding to portions (e.g., snippets) of the source code. In some aspects, the encoder model can comprise a Bidirectional Encoder Representations from Transformers (BERT) model that is trained for processing natural language and/or programming language. The decoder model can classify the source code based on the vector embeddings generated by the encoder model. For example, the decoder model can classify the source code by determining a probability that the source code includes a feature of interest. In one aspect, the feature of

interest may be a malicious code injection, such as, for example malicious code injected into the source code of the webpage 142 by a fraudster and designed to steal a consumer's personal credentials entered via the webpage 142. In other aspects, the feature of interest can be another type of code signature that may be desirable to identify in the source code of the webpage 142. In some aspects, the decoder model can comprise a recurrent neural network, such as, for example, a bidirectional long-short term memory (Bi-LSTM) neural network.

[0030] Referring still to FIG. 1, the offline training model 106 can be an offline version of the online model inference 104. The offline training model 106 can be trained and implemented as the online model inference 104. For example, the offline training model 106 can include the above-mentioned encoder and decoder model. To initially train the offline training model 106, a labeled dataset can be applied to offline training model 106. The labeled dataset may include, for example, positive samples of source code that include a feature of interest (e.g., positive samples that respectively include various different malicious code injections) and/or negative samples of source code that do not include the feature of interest. Training with the labeled dataset can cause the offline training model 106 to update various parameters of the encoder model and/or the decoder model (e.g., fitting the encoder model and/or the decoder model based on the labeled dataset). The "trained" offline model 106 can then be implemented as the online model inference 104.

[0031] In some aspects, after the initial training, the offline training model 106 can be retrained and the resulting "retrained" offline model 106 can be implemented as an updated version of the online model inference 104. In one aspect, the source code classification server 102 can be configured to retrain the offline training model 106 based on a user-provided labeled data set. For example, after initially training the offline training model 106, additional labeled data sets including positive and/or negative samples of source code can be applied to the offline training model 106 for retraining.

[0032] In another aspect, the source code classification server 102 can be configured to automatically retrain the offline training model 106 based on the source code of a webpage 142 that has been classified by the online model inference 104. For example, the offline training model 106 can automatically receive the source code of a webpage 142 that has been classified by the online model inference 104. The offline training model 106 can automatically retrain itself by treating the received source code as a labeled dataset (e.g., treating the source code as a positive sample when the online model inference 104 has classified the source code as including the feature of interest and/or treating the source code as a negative sample when the online model inference 104 has classified the source code as not including the feature of interest). The online model inference 104 can be automatically

updated based on the resulting “retrained” offline model 106.

[0033] Although FIG. 1 depicts the source code classification server 102 in communication with one or more website servers $140_1, 140_2, \dots, 140_n$ for classifying the source code of one or more webpages $141_1, 142_2, \dots, 142_n$, in various aspects, the source code classification server 102 may be employed to classify source code from other sources. For example, the source code classification server 102 can be configured to communicate with any type of device and/or server to receive and classify source code stored thereon. Similarly, the other devices, systems, and methods described herein as classifying source code from a webpage may be configured to classify other types of source code and/or source code from other sources. The source code classification server 102 can be configured to classify source code written in various different programming languages, such as, for example, Python, Java, JavaScript, PHP, Ruby, Go, HTML, and/or CCS.

[0034] The number and arrangement of devices, systems, and networks shown in the source code classification system 100 of FIG. 1 are provided as an example. There may be additional devices, systems, and/or networks, fewer devices, systems, and/or networks, different devices, systems, and/or networks, or differently arranged devices, systems, and/or networks than those shown in FIG. 1. Furthermore, two or more devices shown in FIG. 1 may be implemented within a single device, or a single device shown in FIG. 1 may be implemented as multiple, distributed devices.

[0035] FIG. 2 illustrates a block diagram a module 210 for classifying source code, according to at least one aspect of the present disclosure. The online model inference 104 and/or the offline training model 106 described above with respect to FIG. 1 may be implemented as or may otherwise comprise the module 210. The module 210 can include a text preprocessing module 212, an encoder model 214, an embedding processing module 216, and a decoder model 218. In some aspects, the module 210, including the text preprocessing module 212, the encoder model 214, the embedding processing module 216, and/or the decoder model 218 classifies source code from a webpage 142, as described herein with respect to the method 500 of FIG. 5.

[0036] Referring primarily to FIG 2, and also to FIG. 1, the text preprocessing module 212 can preprocess the source code of a webpage 142 such that the source code is in a form processable by the encode model 214. For example, the text preprocessing module 212 can segment the source code of the webpage 142 into snippets of source code having a maximum string length. The maximum string length of the snippets can be selected to optimize the efficiency the encoder model 214. For example, as explained below, the encoder model 214 can generate vector embeddings from the snippets of the source code.

Selecting a longer maximum string length for the snippets may retain more contextual information in the vector embeddings generated from the snippets. However, applying a snippet with a longer maximum string length to the encoder model 214 may also require more time and/or processing resources.

[0037] Still referring primarily to FIG 2, and also to FIG. 1, as noted above, the encoder model 214 can transform each of the snippets of the source code into vector embeddings that are processable by the decoder model 218. The encoder model 214 can comprise a deep learning model that is pre-trained for natural language and/or programming language (e.g., programming languages such as any one or more of Python, Java, JavaScript, PHP, Ruby, Go, HTML, CCS, etc.). In some aspects, the encoder model 214 includes a Bidirectional Encoder Representations from Transformers (BERT) model. The encoder model 214 is operable to transform the snippets into vector embeddings of a specific length, such as, for example vector embeddings having a length of 768 units. In other aspects, the encoder model 214 is operable to transform the snippets into vector embeddings having a length other than 768, such as, for example, 128, 256, 512, or 1024 units.

[0038] The embedding processing module 216 can generate a tensor based on the vector embeddings output by the encoder model 214 and apply the tensor to the decoder model 218. The embedding processing module 216 can generate a tensor that is in a form processable by the decoder model 218. For example, as noted above, each of the vector embeddings output by the encoder model 214 may have a length of 768 units. In this aspect, the decoder model 218 may be configured to accept a tensor having dimensions of 10 x 768 units. Thus, based on the vector embeddings output by the encoder model 214, the embedding processing module 216 can generate a tensor having dimensions of 10 x 768 units. The embedding processing module 216 can pad the tensor with zeros as needed to generate a tensor of the desired dimension.

[0039] The decoder model 218 classifies the source code of the webpage 142 based on the tensor generated by the embedding processing module 216. For example, the decoder model 218 can comprise a neural network that classifies the source code by determining a probability that the source code includes a feature of interest. In one aspect, the feature of interest may be a malicious code injection, such as, for example, malicious code injected into the source code of a checkout webpage designed to steal a consumer's personal credentials. In other aspects, the feature of interest can be another type of code signature that may be desirable to identify in the source code of the webpage 142. In some aspects, the decoder model 218 can comprise a recurrent neural network, such as, for example, a bidirectional long-short term memory (Bi-LSTM) neural network. In some aspects, the decoder model 218 may be implemented as or otherwise comprise the Bi-LSTM model 320

described herein with respect to FIG. 3.

[0040] The number and arrangement of models and modules shown in the module 210 of FIG. 2 are provided as an example. The module 210 can include additional models/modules, fewer models/modules, and/or differently arranged models/modules than those shown in FIG. 2. Furthermore, two or more of the models/modules shown in FIG. 2 may be implemented within a single model/module, or a single model/module shown in FIG. 2 may be implemented as multiple separate models/modules.

[0041] FIG. 3 illustrates a block diagram of a Bi-directional long short term memory (Bi-LSTM) model 320, according to at least one aspect of the present disclosure. The Bi-LSTM model 320 can determine a probability of whether source code includes a feature of interest based on receiving a tensor embedding generated based on the source code. The decoder model 218 described above with respect to FIG. 2 may be implemented as or otherwise comprise the Bi-LSTM model 320.

[0042] Referring to FIG. 3, the Bi-LSTM model 320 can include any one or more of a Bi-LSTM layer 322, and average pooling layer 324, a max pooling layer 326, a first linear layer 328, a rectified linear unit (ReLU) layer 330, a dropout layer 332, a second linear layer 334, and a sigmoid layer 336. The Bi-LSTM layer 322 can comprise two long short term memory (LSTM) layers, with a first LSTM layer processing input flows in a first direction and a second LSTM layer processing input flows in a second direction that is reverse to the first direction. The bidirectional architecture of the Bi-LSTM layer 322 can enable the Bi-LSTM model 320 account for contextual information comprised in the input tensor embeddings, which can result in a more accurate classification of the source code.

[0043] Still referring to FIG. 3, the average pooling layer 324 and/or the max pooling layer 326 can be configured to reduce noise and amplify various features in the outputs from the Bi-LSTM layer 322 by respectively calculating the average and maximum values of collections of outputs from the Bi-LSTM layer 322. The outputs of the average pooling layer 324 and the max pooling layer 326 can be concentrated (e.g., recombined) to serve as an input for the first linear layer 328. The first linear layer 328 can be configured to perform a linear transformation of the concentrated output from the average pooling layer 324 and the max pooling layer 326. The ReLU layer 330 can be configured to apply a nonlinear activation function to the output of first linear layer 328. The dropout layer 332 can be configured to set various outputs from the ReLU layer 330 to zero at a specified dropout rate to reduce overfitting of the Bi-LSTM model 320 during training. A second linear layer 334 can be configured to perform a linear transformation of the output from the dropout layer 332 to produce a single output value. The sigmoid layer 336 can be configured to scale the output

value from the second linear layer 334 to a value in a range of 0 to 1. The output value of the sigmoid layer 336 can therefore represent the probability that the source code used to generate the tensor embedding input includes the feature of interest.

[0044] Still referring to FIG. 3, in one aspect, the Bi-LSTM model 320 is configured to accept a tensor embedding input with a 10 x 786 unit dimension and can be configured to accept a batch size of 64. In another aspect, each LSTM layer of the Bi-LSTM layer 322 may comprise 64 units. In yet another aspect, the first linear layer may comprise 16 hidden units. In yet another aspect, the dropout layer 332 may comprise a dropout rate of 0.1.

[0045] FIG. 4 is a flow diagram of a method 400 for classifying candidate source code, which may be carried out using the system of FIG. 1, according to at least one aspect of the present disclosure. Referring primarily to FIG 4, and also to FIG. 1, according to the method 400, a labeled dataset is provided 402 to the offline training model 106 for training and/or retraining. The labeled dataset provided 402 to the offline training module 106 can include positive samples of training source code that include a feature of interest and/or negative samples of training source code that do not include the feature of interest. The feature of interest can be, for example, a malicious code injected into the training source code. The training source code can include JavaScript source code, HTML (HyperText Markup Language) source code, CCS (Cascading Style Sheets), Python source code, PHP (PHP: Hypertext Preprocessor) source code, Ruby source code, Go source code, any other type of source code, or a combination thereof.

[0046] Still referring primarily to FIG 4, and also to FIG. 1, according to the method 400, the online model inference 104 is implemented 404 (*e.g.*, updated) based on the trained offline training model 106. As noted above, the offline training model 106 can be an offline version of the online model inference 104 that is trained and implemented as the online model inference 104. According to the method 400, the online model inference 104 can receive 406 candidate source code from a webpage 142 and classify 408 the candidate source code. For example, the online model inference 104 can classify 408 the candidate source code by determining a probability (*e.g.*, a value in a range of 0 to 1) that the candidate source code includes a feature of interest, such as for example, a malicious code injection (*e.g.*, a malicious code designed to steal consumer credentials). In some aspects of the method 400, if the probability is greater than or equal to a classification threshold value, then the candidate source code may be classified as including the feature of interest. If the probability is not greater than or equal to the classification threshold value, then the candidate source code may be classified as not including the feature of interest. In some aspects, the classification threshold value may be a value that is greater than or equal to 0.2, such as, for example, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, or 0.999,

[0047] Still referring primarily to FIG 4, and also to FIG. 1, the online model inference 104 may automatically provide 410 the classified 408 candidate source code to the offline training model 106 (e.g., and the offline training model 106 may automatically receive the classified 408 candidate source code) for retraining. In some aspects, the offline training model 106 may automatically provide 410 the candidate source code to the offline training model 106 for retraining based the probability that the candidate source code includes the feature of interest determined by the online model inference 104. For example, if the probability is greater than or equal to a retraining threshold value, then the online model inference may automatically provide 410 the candidate source code to the offline training model 106 as a positive sample for retraining. If the probability is less than the training threshold value, then the online model inference 104 may automatically provide 410 the candidate source code to the offline training model 106 as a negative sample for retraining. Alternately, if the probability is less than the training threshold value, then the online model inference 104 may not automatically provide 410 the candidate source code to the offline training model 106 for retraining. The offline training model 106 can be automatically retrained based on being provided 410 the candidate source code. Further, the retrained offline training model 106 can automatically be implemented 404 as the online model inference 104, thereby automatically generating an updated version of the online model inference 104.

[0048] In some aspects, the classification threshold value and the training threshold value may be different. For example, the training threshold value may greater than the classification threshold value. This may ensure that only candidate source code determined to have a high probably of including the feature of interest is used to retrain the offline training model 106, which can thereby ensure that updated versions of the online model inference 104 maintain accuracy when subsequently classifying candidate source code. In some aspects, the training threshold value may greater than or equal to 0.2, such as, for example, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, or 0.999.

[0049] In some aspects, the according to the method 400, the candidate source code that is to be automatically provided 410 to the offline training model 106 for retraining may optionally be sampled 412 by a user. In this aspect, a user may periodically and/or randomly review the candidate source code that is to be automatically provided 410 to the offline training model 106 for retraining. The user may review the candidate source code to ensure that it meets certain criteria before allowing the offline training model 106 to be retrained based on the candidate source code. For example, the user may verify the classification of the candidate source code for accuracy (e.g., to ensure that candidate source code classified 408 as including a feature of interest does in fact include the feature of interest

before allowing the candidate source code to be provided as a positive sample for retraining the offline training model 106). In one aspect, the optional sampling 412 can be user-initiated. In other aspects, the source code classification server 102 can be configured to periodically and/or randomly provide a notification related to the optional sampling 412 requesting user review. The optional sampling 412 may help to ensure that only candidate source code that is correctly classified as including the feature of interest is used to retrain the offline training model 106, which can thereby ensure that updated versions of the online model inference 104 maintain accuracy when subsequently classifying candidate source code.

[0050] FIG. 5 is a flow diagram of a method for classifying source code, which may be carried out using the module 210 of FIG. 2, according to at least one aspect of the present disclosure. Referring primarily to FIG 5, and also to FIG. 2, according to the method 500 the text preprocessing module 212 receives a candidate source code 502. In one aspect of the method 500, the candidate source code 502 can comprise JavaScript source code corresponding to a webpage (e.g., a webpage 142 of FIG. 1). In other aspects of the method 500, the candidate source code 502 can comprise source code written in other programming languages (e.g., Python, Java, JavaScript, PHP, Ruby, Go, HTML, CCS, etc.) and may be received from sources other than a webpage (e.g., any type of software source code).

[0051] Referring still primarily to FIG. 5, and also to FIG 2, according to the method 500, the text preprocessing module 212 segments the candidate source code 502 into snippets 504 of the candidate source code 502. Further, the encoder model 214 transforms the snippets 504 to vectors 506 (e.g., vector embeddings). Each of the vectors 506 can correspond to a different one of the snippets 504.

[0052] Referring still primarily to FIG. 5, and also to FIG 2, according to the method 500, the embedding processing module 216 generates a tensor 508 based on the vectors 506 and applies the tensor 508 to the decoder model 218. The decoder model 218 generates a classification output 510 based on the tensor 508. For example, according to various aspects of the method 500, the classification output 510 comprises a probability that the candidate source code 502 includes a feature of interest. In one aspect, the feature of interest can comprise a malicious code injected into the candidate source code 502.

[0053] In various aspects of the method 500, the decoder model 218 generates a classification output 510 based on the tensor 508 by applying the tensor 508 to a bidirectional long-short term memory (Bi-LSTM) layer to generate a Bi-LSTM layer output. In one aspect, the decoder model 218 applies the Bi-LSTM layer output to an average pooling layer to generate an average pooling layer output and applies the Bi-LSTM layer output to a

max pooling layer to generate a max pooling layer output. Further, the decoder model 218 combines the average pooling layer output and the max pooling layer output to generate a concentrated output. In another aspect, the decoder model 218 applies the concentrated output to a first linear layer to generate a first linear layer output, applies the first linear layer output to a rectified linear unit (ReLU) layer to generate a ReLU layer output, applies the ReLU layer output to a dropout layer to generate a dropout layer output, applies the dropout layer output to a second linear layer to generate a second linear layer output, and applies the second linear layer output to a sigmoid layer to generate the classification output 510.

[0054] In various aspects of the method 500, the decoder model 218 is trained by an offline training module based on a labeled dataset. The labeled dataset can include positive samples of training source code comprising the feature of interest. In various aspects of the method 500, the candidate source code 502 can be labeled by the decoder model 218 as an additional positive sample based on the classification output 510. Further, the training module can automatically retrain the decoder model based on the additional positive sample.

[0055] FIG. 6 is a block diagram of a computer apparatus 3000 comprising data processing subsystems or components, according to at least one aspect of the present disclosure. The subsystems shown in FIG. 6 are interconnected via a system bus 3010. Additional subsystems such as a printer 3018, keyboard 3026, fixed disk 3028 (or other memory comprising computer readable media), monitor 3022, which is coupled to a display adapter 3020, and others are shown. Peripherals and input/output (I/O) devices, which couple to an I/O controller 3012 (which can be a processor or other suitable controller), can be connected to the computer system by any number of means known in the art, such as a serial port 3024. For example, the serial port 3024 or external interface 3030 can be used to connect the computer apparatus to a wide area network such as the Internet, a mouse input device, or a scanner. The interconnection via system bus allows the central processor 3016 to communicate with each subsystem and to control the execution of instructions from system memory 3014 or the fixed disk 3028, as well as the exchange of information between subsystems. The system memory 3014 and/or the fixed disk 3028 may embody a computer readable medium.

[0056] FIG. 7 is a diagrammatic representation of an example computing system 4000 that includes a host machine 4002 within which a set of instructions to generate any one or more of the systems, models, and modules described herein and/or to perform any one or more of the methodologies described herein, according to at least one aspect of the present disclosure. In various aspects, the host machine 4002 operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the host machine 4002 may operate in the capacity of a server or a client machine in a server-client

network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The host machine 4002 may be a computer or computing device, a personal computer (PC), a tablet PC, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, a portable music player (e.g., a portable hard drive audio device such as an Moving Picture Experts Group Audio Layer 3 (MP3) player), a web appliance, a network router, switch or bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein.

[0057] The example system 4000 includes the host machine 4002, running a host operating system (OS) 4004 on a processor or multiple processor(s)/processor core(s) 4006 (e.g., a central processing unit (CPU), a graphics processing unit (GPU), or both), and various memory nodes 4008. The host OS 4004 may include a hypervisor 4010 which is able to control the functions and/or communicate with a virtual machine (“VM”) 4012 running on machine readable media. The VM 4012 also may include a virtual CPU or vCPU 4014. The memory nodes 4008 may be linked or pinned to virtual memory nodes or vNodes 4016. When the memory node 4008 is linked or pinned to a corresponding vNode 4016, then data may be mapped directly from the memory nodes 4008 to the corresponding vNode 4016.

[0058] All the various components shown in host machine 4002 may be connected with and to each other, or communicate to each other via a bus (not shown) or via other coupling or communication channels or mechanisms. The host machine 4002 may further include a video display, audio device or other peripherals 4018 (e.g., a liquid crystal display (LCD), alpha-numeric input device(s) including, e.g., a keyboard, a cursor control device, e.g., a mouse, a voice recognition or biometric verification unit, an external drive, a signal generation device, e.g., a speaker,) a persistent storage device 4020 (also referred to as disk drive unit), and a network interface device 4022. The host machine 4002 may further include a data encryption module (not shown) to encrypt data. The components provided in the host machine 4002 are those typically found in computer systems that may be suitable for use with aspects of the present disclosure and are intended to represent a broad category of such computer components that are known in the art. Thus, the system 4000 can be a server, minicomputer, mainframe computer, or any other computer system. The computer may also include different bus configurations, networked platforms, multi-processor platforms, and the like. Various operating systems may be used including UNIX, LINUX, WINDOWS, QNX ANDROID, IOS, CHROME, TIZEN, and other suitable operating systems.

[0059] The disk drive unit 4024 also may be a Solid-state Drive (SSD), a hard disk drive (HDD) or other includes a computer or machine-readable medium on which is stored one or more sets of instructions and data structures (e.g., data/instructions 4026) embodying or utilizing any one or more of the methodologies or functions described herein. The data/instructions 4026 also may reside, completely or at least partially, within the main memory node 4008 and/or within the processor(s) 4006 during execution thereof by the host machine 4002. The data/instructions 4026 may further be transmitted or received over a network 4028 via the network interface device 4022 utilizing any one of several well-known transfer protocols (e.g., Hyper Text Transfer Protocol (HTTP)).

[0060] The processor(s) 4006 and memory nodes 4008 also may comprise machine-readable media. The term "computer-readable medium" or "machine-readable medium" should be taken to include a single medium or multiple medium (e.g., a centralized or distributed database and/or associated caches and servers) that store the one or more sets of instructions. The term "computer-readable medium" shall also be taken to include any medium that is capable of storing, encoding, or carrying a set of instructions for execution by the host machine 4002 and that causes the host machine 4002 to perform any one or more of the methodologies of the present application, or that is capable of storing, encoding, or carrying data structures utilized by or associated with such a set of instructions. The term "computer-readable medium" shall accordingly be taken to include, but not be limited to, solid-state memories, optical and magnetic media, and carrier wave signals. Such media may also include, without limitation, hard disks, floppy disks, flash memory cards, digital video disks, random access memory (RAM), read only memory (ROM), and the like. The example aspects described herein may be implemented in an operating environment comprising software installed on a computer, in hardware, or in a combination of software and hardware.

[0061] One skilled in the art will recognize that Internet service may be configured to provide Internet access to one or more computing devices that are coupled to the Internet service, and that the computing devices may include one or more processors, buses, memory devices, display devices, input/output devices, and the like. Furthermore, those skilled in the art may appreciate that the Internet service may be coupled to one or more databases, repositories, servers, and the like, which may be utilized to implement any of the various aspects of the disclosure as described herein.

[0062] The computer program instructions also may be loaded onto a computer, a server, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which

execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0063] Suitable networks may include or interface with any one or more of, for instance, a local intranet, a PAN (Personal Area Network), a LAN (Local Area Network), a WAN (Wide Area Network), a MAN (Metropolitan Area Network), a virtual private network (VPN), a storage area network (SAN), a frame relay connection, an Advanced Intelligent Network (AIN) connection, a synchronous optical network (SONET) connection, a digital T1, T3, E1 or E3 line, Digital Data Service (DDS) connection, DSL (Digital Subscriber Line) connection, an Ethernet connection, an ISDN (Integrated Services Digital Network) line, a dial-up port such as a V.90, V.34 or V.34bis analog modem connection, a cable modem, an ATM (Asynchronous Transfer Mode) connection, or an FDDI (Fiber Distributed Data Interface) or CDDI (Copper Distributed Data Interface) connection. Furthermore, communications may also include links to any of a variety of wireless networks, including WAP (Wireless Application Protocol), GPRS (General Packet Radio Service), GSM (Global System for Mobile Communication), CDMA (Code Division Multiple Access) or TDMA (Time Division Multiple Access), cellular phone networks, GPS (Global Positioning System), CDPD (cellular digital packet data), RIM (Research in Motion, Limited) duplex paging network, Bluetooth radio, or an IEEE 802.11-based radio frequency network. The network 4028 can further include or interface with any one or more of an RS-232 serial connection, an IEEE-1394 (Firewire) connection, a Fiber Channel connection, an IrDA (infrared) port, a SCSI (Small Computer Systems Interface) connection, a USB (Universal Serial Bus) connection or other wired or wireless, digital or analog interface or connection, mesh or Digi® networking.

[0064] In general, a cloud-based computing environment is a resource that typically combines the computational power of a large grouping of processors (such as within web servers) and/or that combines the storage capacity of a large grouping of computer memories or storage devices. Systems that provide cloud-based resources may be utilized exclusively by their owners or such systems may be accessible to outside users who deploy applications within the computing infrastructure to obtain the benefit of large computational or storage resources.

[0065] The cloud is formed, for example, by a network of web servers that comprise a plurality of computing devices, such as the host machine 4002, with each server 4030 (or at least a plurality thereof) providing processor and/or storage resources. These servers manage workloads provided by multiple users (e.g., cloud resource customers or other users). Typically, each user places workload demands upon the cloud that vary in real-time, sometimes dramatically. The nature and extent of these variations typically depends on the

type of business associated with the user.

[0066] It is noteworthy that any hardware platform suitable for performing the processing described herein is suitable for use with the technology. The terms "computer-readable storage medium" and "computer-readable storage media" as used herein refer to any medium or media that participate in providing instructions to a CPU for execution. Such media can take many forms, including, but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media include, for example, optical or magnetic disks, such as a fixed disk. Volatile media include dynamic memory, such as system RAM. Transmission media include coaxial cables, copper wire and fiber optics, among others, including the wires that comprise one aspect of a bus. Transmission media can also take the form of acoustic or light waves, such as those generated during radio frequency (RF) and infrared (IR) data communications. Common forms of computer-readable media include, for example, a flexible disk, a hard disk, magnetic tape, any other magnetic medium, a CD-ROM disk, digital video disk (DVD), any other optical medium, any other physical medium with patterns of marks or holes, a RAM, a PROM, an EPROM, an EEPROM, a FLASH EPROM, any other memory chip or data exchange adapter, a carrier wave, or any other medium from which a computer can read.

[0067] Various forms of computer-readable media may be involved in carrying one or more sequences of one or more instructions to a CPU for execution. A bus carries the data to system RAM, from which a CPU retrieves and executes the instructions. The instructions received by system RAM can optionally be stored on a fixed disk either before or after execution by a CPU.

[0068] Computer program code for carrying out operations for aspects of the present technology may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++, or the like and conventional procedural programming languages, such as the "C" programming language, Go, Python, or other programming languages, including assembly languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

[0069] Examples of the devices, systems, and methods according to various aspects of the present disclosure are provided below in the following numbered clauses. An aspect of

any of the devices(s), method(s) and/or system(s) may include any one or more than one, and any combination of, the numbered clauses described below.

[0070] Clause 1: A computer-implemented method, comprising: receiving, by a text preprocessing module, a candidate source code; segmenting, by the text preprocessing module, the candidate source code into snippets; transforming, by an encoder model, the snippets to vectors, wherein each of the vectors corresponds to a different one of the snippets; generating, by an embedding processing module, a tensor based on the vectors; applying, by the embedding processing module, the tensor to a decoder model; and determining, by the decoder model, a probability of a feature of interest being in the candidate source code.

[0071] Clause 2: The computer-implemented method of Clause 1, wherein receiving the candidate source code comprises receiving a JavaScript source code corresponding to a webpage.

[0072] Clause 3: The computer-implemented method of any of Clauses 1-2, wherein the feature of interest comprises a malicious code injected into the candidate source code.

[0073] Clause 4: The computer-implemented method of any of Clauses 1-3, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code comprises: applying the tensor to a bidirectional long-short term memory (Bi-LSTM) layer to generate a Bi-LSTM layer output.

[0074] Clause 5: The computer-implemented method of Clause 4, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code further comprises: applying the Bi-LSTM layer output to an average pooling layer to generate an average pooling layer output; applying the Bi-LSTM layer output to a max pooling layer to generate a max pooling layer output; and combining the average pooling layer output and the max pooling layer output to generate a concentrated output.

[0075] Clause 6: The computer-implemented method of Clause 5, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code further comprises: applying the concentrated output to a first linear layer to generate a first linear layer output; applying the first linear layer output to a rectified linear unit (ReLU) layer to generate a ReLU layer output; applying the ReLU layer output to a dropout layer to generate a dropout layer output; applying the dropout layer output to a second linear layer to generate a second linear layer output; and applying the second linear layer output to a sigmoid layer to generate the probability of a feature of interest being in the candidate source code.

[0076] Clause 7: The computer-implemented method of any of Clauses 1-6, wherein transforming, by the encoder model, the snippets to the vectors comprises: applying the

snippets to a pre-trained model for programming language processing.

[0077] Clause 8: The computer-implemented method of any of Clauses 1-7, further comprising: training, by a training module, the decoder model based on a labeled dataset, wherein the labeled dataset comprises positive samples of training source code comprising the feature of interest.

[0078] Clause 9: The computer-implemented method of Clause 8, further comprising: labeling, by the training module, the candidate source code as an additional positive sample based on the probability satisfying a predetermined threshold; and retraining, by the training module, the decoder model based on the candidate source code.

[0079] Clause 10: The computer-implemented method of any of Clauses 1-9, further comprising classifying, by a classification module, the candidate source code as comprising the feature of interest based on the probability exceeding a predetermined threshold.

[0080] Clause 11: A source code classification system, comprising: an online model inference configured to determine a probability that a candidate source code includes a feature of interest, wherein the online model inference comprises: a text preprocessing module configured to segment the candidate source code into snippets; an encoder model configured to transform the snippets to vectors, wherein each of the vectors corresponds to a different one of the snippets; an embedding processing module configured to generate a tensor based on the vectors; and a decoder model configured to determine the probability that the candidate source code includes the feature of interest based on the tensor.

[0081] Clause 12: The source code classification system of Clause 11, further comprising: an offline training model for retraining the online model inference, wherein the offline training model is configured to update the online model inference based on a labeled dataset.

[0082] Clause 13: The source code classification system of Clause 12, wherein the labeled dataset comprises positive samples of source code that each include the feature of interest.

[0083] Clause 14: The source code classification system of any of Clauses 11-13, wherein the feature of interest comprises a malicious code injection.

[0084] Clause 15: The source code classification system of any of Clauses 11-14, wherein the candidate source code comprises JavaScript source code.

[0085] Clause 16: The source code classification system of any of Clauses 12-15, wherein the online model inference is configured to receive the candidate source code from an active webpage and determine the probability that the candidate source code from the active webpage includes the feature of interest.

[0086] Clause 17: The source code classification system of Clause 16, wherein the offline training model is configured to automatically apply the candidate source code from the

active webpage as a labeled dataset for updating the online model based on the probability that the candidate source code from the active webpage includes the feature of interest satisfying a probability threshold.

[0087] Clause 18: The source code classification system of any of Clauses 11-17, wherein the decoder model comprises a bidirectional long-short term memory (Bi-LSTM) neural network.

[0088] Clause 19: The source code classification system of Clause 18, wherein the Bi-LSTM neural network comprises: an average pooling layer; a max pooling layer, wherein outputs of the average pooling layer and the max pooling layer are combined to generate a concentrated output; a first linear layer; a rectified linear unit (ReLU) layer; a dropout layer; a second linear layer; and a sigmoid layer.

[0089] Clause 20: The source code classification system of any of Clauses 11-19, wherein the encoder model comprises a Bidirectional Encoder Representations from Transformers (BERT) model.

[0090] Further, it is understood that any one or more of the following-described forms, expressions of forms, examples, can be combined with any one or more of the other following-described forms, expressions of forms, and examples.

[0091] While several forms have been illustrated and described, it is not the intention of Applicant to restrict or limit the scope of the appended claims to such detail. Numerous modifications, variations, changes, substitutions, combinations, and equivalents to those forms may be implemented and will occur to those skilled in the art without departing from the scope of the present disclosure. Moreover, the structure of each element associated with the described forms can be alternatively described as a means for providing the function performed by the element. Also, where materials are disclosed for certain components, other materials may be used. It is therefore to be understood that the foregoing description and the appended claims are intended to cover all such modifications, combinations, and variations as falling within the scope of the disclosed forms. The appended claims are intended to cover all such modifications, variations, changes, substitutions, modifications, and equivalents.

[0092] As used herein, a “server” may include one or more computing devices which can be individual, stand-alone machines located at the same or different locations, may be owned or operated by the same or different entities, and may further be one or more clusters of distributed computers or “virtual” machines housed within a datacenter. It should be understood and appreciated by a person of skill in the art that functions performed by one “server” can be spread across multiple disparate computing devices for various reasons. As used herein, a “server” is intended to refer to all such scenarios and should not be construed

or limited to one specific configuration. Further, a server as described herein may, but need not, reside at (or be operated by) a merchant, a payment network, a financial institution, a healthcare provider, a social media provider, a government agency, or agents of any of the aforementioned entities. The term “server” may also refer to or include one or more processors or computers, storage devices, or similar computer arrangements that are operated by or facilitate communication and processing for multiple parties in a network environment, such as the Internet, although it will be appreciated that communication may be facilitated over one or more public or private network environments and that various other arrangements are possible. Further, multiple computers, *e.g.*, servers, or other computerized devices, *e.g.*, point-of-sale devices, directly or indirectly communicating in the network environment may constitute a “system,” such as a merchant's point-of-sale system. Reference to “a server” or “a processor,” as used herein, may refer to a previously recited server and/or processor that is recited as performing a previous step or function, a different server and/or processor, and/or a combination of servers and/or processors. For example, as used in the specification and the claims, a first server and/or a first processor that is recited as performing a first step or function may refer to the same or different server and/or a processor recited as performing a second step or function.

[0093] The term “system” may refer to one or more computing devices or combinations of computing devices (*e.g.*, processors, servers, client devices, software applications, modules, components of such, and/or the like). For example, a system may include a plurality of computing devices that include software applications, where the plurality of computing devices are connected via a network.

[0094] As used herein, a “server computer” may describe a powerful computer or cluster of computers. For example, the server computer can be a large mainframe, a minicomputer cluster, or a group of servers functioning as a unit. The server computer may be associated with an entity such as a payment processing network, a wallet provider, a merchant, an authentication cloud, an acquirer or an issuer. In one example, the server computer may be a database server coupled to a Web server. The server computer may be coupled to a database and may include any hardware, software, other logic, or combination of the preceding for servicing the requests from one or more client computers. The server computer may comprise one or more computational apparatuses and may use any of a variety of computing structures, arrangements, and compilations for servicing the requests from one or more client computers. In some embodiments or aspects, the server computer may provide and/or support payment network cloud service.

[0095] Reference to “a device,” “a server,” “a processor,” and/or the like, as used herein, may refer to a previously recited device, server, or processor that is recited as performing a

previous step or function, a different server or processor, and/or a combination of servers and/or processors. For example, as used in the specification and the claims, a first server or a first processor that is recited as performing a first step or a first function may refer to the same or different server or the same or different processor recited as performing a second step or a second function.

[0096] One or more components may be referred to herein as “configured to,” “configurable to,” “operable/operative to,” “adapted/adaptable,” “able to,” “conformable/conformed to,” etc. Those skilled in the art will recognize that “configured to” can generally encompass active-state components and/or inactive-state components and/or standby-state components, unless context requires otherwise.

[0097] Those skilled in the art will recognize that, in general, terms used herein, and especially in the appended claims (e.g., bodies of the appended claims) are generally intended as “open” terms (e.g., the term “including” should be interpreted as “including but not limited to,” the term “having” should be interpreted as “having at least,” the term “includes” should be interpreted as “includes but is not limited to,” etc.). It will be further understood by those within the art that if a specific number of an introduced claim recitation is intended, such an intent will be explicitly recited in the claim, and in the absence of such recitation no such intent is present. For example, as an aid to understanding, the following appended claims may contain usage of the introductory phrases “at least one” and “one or more” to introduce claim recitations. However, the use of such phrases should not be construed to imply that the introduction of a claim recitation by the indefinite articles “a” or “an” limits any particular claim containing such introduced claim recitation to claims containing only one such recitation, even when the same claim includes the introductory phrases “one or more” or “at least one” and indefinite articles such as “a” or “an” (e.g., “a” and/or “an” should typically be interpreted to mean “at least one” or “one or more”); the same holds true for the use of definite articles used to introduce claim recitations.

[0098] The term “substantially,” “about,” or “approximately” as used in the present disclosure, unless otherwise specified, means an acceptable error for a particular value as determined by one of ordinary skill in the art, which depends in part on how the value is measured or determined. In certain aspects, the term “substantially,” “about,” or “approximately” means within 1, 2, 3, or 4 standard deviations. In certain aspects, the term “substantially,” “about,” or “approximately” means within 50%, 20%, 15%, 10%, 9%, 8%, 7%, 6%, 5%, 4%, 3%, 2%, 1%, 0.5%, or 0.05% of a given value or range.

[0099] In addition, even if a specific number of an introduced claim recitation is explicitly recited, those skilled in the art will recognize that such recitation should typically be

interpreted to mean at least the recited number (e.g., the bare recitation of “two recitations,” without other modifiers, typically means at least two recitations, or two or more recitations). Furthermore, in those instances where a convention analogous to “at least one of A, B, and C, etc.” is used, in general such a construction is intended in the sense one having skill in the art would understand the convention (e.g., “a system having at least one of A, B, and C” would include but not be limited to systems that have A alone, B alone, C alone, A and B together, A and C together, B and C together, and/or A, B, and C together, etc.). In those instances where a convention analogous to “at least one of A, B, or C, etc.” is used, in general such a construction is intended in the sense one having skill in the art would understand the convention (e.g., “a system having at least one of A, B, or C” would include but not be limited to systems that have A alone, B alone, C alone, A and B together, A and C together, B and C together, and/or A, B, and C together, etc.). It will be further understood by those within the art that typically a disjunctive word and/or phrase presenting two or more alternative terms, whether in the description, claims, or drawings, should be understood to contemplate the possibilities of including one of the terms, either of the terms, or both terms unless context dictates otherwise. For example, the phrase “A or B” will be typically understood to include the possibilities of “A” or “B” or “A and B.”

[0100] With respect to the appended claims, those skilled in the art will appreciate that recited operations therein may generally be performed in any order. Also, although various operational flow diagrams are presented in a sequence(s), it should be understood that the various operations may be performed in other orders than those which are illustrated, or may be performed concurrently. Examples of such alternate orderings may include overlapping, interleaved, interrupted, reordered, incremental, preparatory, supplemental, simultaneous, reverse, or other variant orderings, unless context dictates otherwise. Furthermore, terms like “responsive to,” “related to,” or other past-tense adjectives are generally not intended to exclude such variants, unless context dictates otherwise.

[0101] It is worthy to note that any reference to “one aspect,” “an aspect,” “an exemplification,” “one exemplification,” and the like means that a particular feature, structure, or characteristic described in connection with the aspect is included in at least one aspect. Thus, appearances of the phrases “in one aspect,” “in an aspect,” “in an exemplification,” and “in one exemplification” in various places throughout the specification are not necessarily all referring to the same aspect. Furthermore, the particular features, structures or characteristics may be combined in any suitable manner in one or more aspects.

[0102] As used herein, the singular form of “a,” “an,” and “the” include the plural references unless the context clearly dictates otherwise.

[0103] Any patent application, patent, non-patent publication, or other disclosure material referred to in this specification and/or listed in any Application Data Sheet is incorporated by reference herein, to the extent that the incorporated materials is not inconsistent herewith. As such, and to the extent necessary, the disclosure as explicitly set forth herein supersedes any conflicting material incorporated herein by reference. Any material, or portion thereof, that is said to be incorporated by reference herein, but which conflicts with existing definitions, statements, or other disclosure material set forth herein will only be incorporated to the extent that no conflict arises between that incorporated material and the existing disclosure material.

[0104] In summary, numerous benefits have been described which result from employing the concepts described herein. The foregoing description of the one or more forms has been presented for purposes of illustration and description. It is not intended to be exhaustive or limiting to the precise form disclosed. Modifications or variations are possible in light of the above teachings. The one or more forms were chosen and described in order to illustrate principles and practical application to thereby enable one of ordinary skill in the art to utilize the various forms and with various modifications as are suited to the particular use contemplated. It is intended that the claims submitted herewith define the overall scope.

CLAIMS

What is claimed is:

1. A computer-implemented method, comprising:
 - receiving, by a text preprocessing module, a candidate source code;
 - segmenting, by the text preprocessing module, the candidate source code into snippets;
 - transforming, by an encoder model, the snippets to vectors, wherein each of the vectors corresponds to a different one of the snippets;
 - generating, by an embedding processing module, a tensor based on the vectors;
 - applying, by the embedding processing module, the tensor to a decoder model; and
 - determining, by the decoder model, a probability of a feature of interest being in the candidate source code.
2. The computer-implemented method of Claim 1, wherein receiving the candidate source code comprises receiving a JavaScript source code corresponding to a webpage.
3. The computer-implemented method of Claim 2, wherein the feature of interest comprises a malicious code injected into the candidate source code.
4. The computer-implemented method of Claim 1, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code comprises:
 - applying the tensor to a bidirectional long-short term memory (Bi-LSTM) layer to generate a Bi-LSTM layer output.
5. The computer-implemented method of Claim 4, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code further comprises:
 - applying the Bi-LSTM layer output to an average pooling layer to generate an average pooling layer output;
 - applying the Bi-LSTM layer output to a max pooling layer to generate a max pooling layer output; and
 - combining the average pooling layer output and the max pooling layer output to generate a concentrated output.
6. The computer-implemented method of Claim 5, wherein generating, by the decoder model, the probability of a feature of interest being in the candidate source code further

comprises:

- applying the concentrated output to a first linear layer to generate a first linear layer output;
- applying the first linear layer output to a rectified linear unit (ReLU) layer to generate a ReLU layer output;
- applying the ReLU layer output to a dropout layer to generate a dropout layer output;
- applying the dropout layer output to a second linear layer to generate a second linear layer output; and
- applying the second linear layer output to a sigmoid layer to generate the probability of a feature of interest being in the candidate source code.

7. The computer-implemented method of Claim 4, wherein transforming, by the encoder model, the snippets to the vectors comprises:

- applying the snippets to a pre-trained model for programming language processing.

8. The computer-implemented method of Claim 1, further comprising:

- training, by a training module, the decoder model based on a labeled dataset, wherein the labeled dataset comprises positive samples of training source code comprising the feature of interest.

9. The computer-implemented method of Claim 8, further comprising:

- labeling, by the training module, the candidate source code as an additional positive sample based on the probability satisfying a predetermined threshold; and
- retraining, by the training module, the decoder model based on the candidate source code.

10. The computer-implemented method of Claim 1, further comprising classifying, by a classification module, the candidate source code as comprising the feature of interest based on the probability exceeding a predetermined threshold.

11. A source code classification system, comprising:
 - an online model inference configured to determine a probability that a candidate source code includes a feature of interest, wherein the online model inference comprises:
 - a text preprocessing module configured to segment the candidate source code into snippets;
 - an encoder model configured to transform the snippets to vectors, wherein each of the vectors corresponds to a different one of the snippets;
 - an embedding processing module configured to generate a tensor based on the vectors; and
 - a decoder model configured to determine the probability that the candidate source code includes the feature of interest based on the tensor.
12. The source code classification system of Claim 11, further comprising:
 - an offline training model for retraining the online model inference, wherein the offline training model is configured to update the online model inference based on a labeled dataset.
13. The source code classification system of Claim 12, wherein the labeled dataset comprises positive samples of source code that each include the feature of interest.
14. The source code classification system of Claim 13, wherein the feature of interest comprises a malicious code injection.
15. The source code classification system of Claim 14, wherein the candidate source code comprises JavaScript source code.
16. The source code classification system of Claim 15, wherein the online model inference is configured to receive the candidate source code from an active webpage and determine the probability that the candidate source code from the active webpage includes the feature of interest.
17. The source code classification system of Claim 16, wherein the offline training model is configured to automatically apply the candidate source code from the active webpage as a labeled dataset for updating the online model based on the probability that the candidate source code from the active webpage includes the feature of interest satisfying a probability threshold.

18. The source code classification system of Claim 12, wherein the decoder model comprises a bidirectional long-short term memory (Bi-LSTM) neural network.
19. The source code classification system of Claim 18, wherein the Bi-LSTM neural network comprises:
- an average pooling layer;
 - a max pooling layer, wherein outputs of the average pooling layer and the max pooling layer are combined to generate a concentrated output;
 - a first linear layer;
 - a rectified linear unit (ReLU) layer;
 - a dropout layer;
 - a second linear layer; and
 - a sigmoid layer.
20. The source code classification system of Claim 18, wherein the encoder model comprises a Bidirectional Encoder Representations from Transformers (BERT) model.