SEMANTIC PATCH EMBEDDING FOR SECURITY DETECTION: A FINE-TO-COARSE GRAINED APPROACH

Xunzhu Tang[1], Yewei Song[1], Zhenghan Chen[2], Haoye Tian[1], Tegawendé F. Bissyandé[1], and Jacques Klein[1]

[1]University of Luxembourg
[2]Peking University

ABSTRACT

The surge in open-source software usage has heightened the risk of concealed vulnerabilities impacting downstream applications. Compounded by vendors silently releasing security patches without explicit notifications, users remain unaware of potential threats, providing attackers with opportunities. In navigating the intricate realm of software patches, understanding patch semantics becomes crucial for secure maintenance. Addressing this, we present MultiSEM, a multi-level Semantic Embedder for security patch detection. It employs fine-grained word-centric vectors and coarse-grained code-line representations, integrating patch descriptions for a comprehensive semantic view. MultiSEM excels in security patch detection, outperforming state-of-the-art models by 22.46% on PatchDB and 9.21% on SPI-DB in F1 metric evaluations.

1 INTRODUCTION AND RELATED WORKS

The rapid growth of the open source software (OSS) ecosystem has led to significant advancements in computer software development. According to the 2021 OSSRA report Synopsys (2023), 98% of codebases are now composed of open source components, with 84% containing at least one open-source vulnerability. Moreover, 60% of them face high-risk vulnerability threats. The academic and industry communities have explored multiple avenues to identify security patches. Traditional methods have focused on machine learning (ML) models using syntax features Wang et al. (2020); Tian et al. (2012). More recently, recurrent neural networks (RNNs) have been used to treat patch code as sequential data Wang et al. (2021b); Zhou et al. (2021). However, these methods overlook program semantics and suffer from high false-positive rates. ML-based solutions often miss intricate dependencies between code statements, while RNN models, despite their NLP inspiration, neglect the unique attributes of programming languages. As a result, these methods yield high false-positive rates, with two RNN-based models showing rates of 11.6% and 33.2%, respectively. Given that only 6-10% of all patches are security-centric Wang et al. (2021a), reducing these false positives is crucial.

2 METHODS

In this project, we aim to address these challenges. Through a fine-to-coarse grained approach, we present a novel method (MultiSEM) that not only detects security patches with a high degree of accuracy but also reduces false positive rates. Drawing upon multilevel semantic embedding techniques and capitalizing on the content-rich information contained within software patches, our solution represents a quantum leap in the realm of security patch detection.

Our contributions are as follows:
Multilevel Semantic Embedding: We introduce a novel multilevel semantic embedding technique tailored for software patches. This method captures both high-level and granular details of the patches, ensuring a more nuanced and accurate representation.
Fine-to-Coarse Grained Approach: Our approach not only focuses on individual lines of code but

also looks at the broader structure and flow of the patch. The multi-scale perspective objective is to enhance the accuracy of security patch detection.

Experimental Results: We conduct exhaustive evaluations of our proposed method against state-of-the-art solutions. Our results demonstrate superior performance, particularly in reducing false positive rates, which has been a longstanding challenge in the field.

3 Approach

We introduce MultiSEM, a holistic framework with six components: Preprocessing, Multilevel Compressed CNN (MCC), Semantic Alignment, Feature Refinement, Hybrid Feature Aggregation, and Prediction. This model captures fine-to-coarse-grained multilevel semantic embedding of software patches, using source code as input and processing it through these components for security patch detection. The overall approach is visualized in Figure 1.

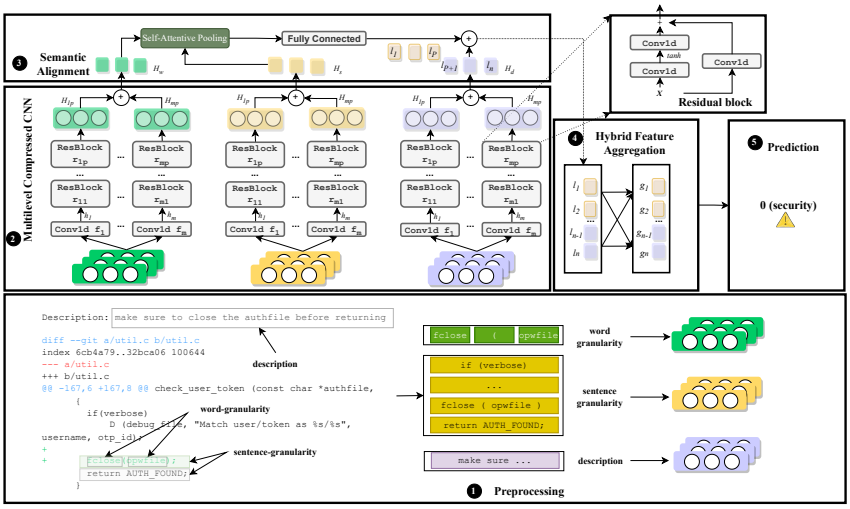

Figure 1: Architecture of MultiSEM

4 Example and Experiment

We evaluate MultiSEM on two popular datasets on security patch detection: PatchDB Wang et al. (2021a) and SPI-DB Zhou et al. (2021).

Table 1: Comparison of TwinRNN, GraphSPD, and MultiSEM on PatchDB and SPI-DB with various metrics (%).

Method	Dataset	AUC	F1	Recall+	Recall-	TPR
TwinRNN Wang et al. (2021b)	PatchDB	66.50	45.12	46.35	54.37	50.67
	SPI-DB	55.10	47.25	48.00	52.10	50.60
GraphSPD Wang et al. (2023)	PatchDB	78.29	54.73	75.17	79.67	70.82
	SPI-DB	63.04	48.42	60.29	65.33	65.93
MultiSEM	PatchDB	83.15	77.19	79.52	86.78	79.52
	SPI-DB	68.45	57.63	70.24	80.12	73.25

5 Conlusion

MultiSEM surpasses GraphSPD with a 22.46% F1 improvement on PatchDB and 9.21% on SPI-DB, establishing it as the optimal choice. While excelling in detecting major vulnerabilities, **MultiSEM** faces challenges with subtler ones due to data imbalances, outperforming GraphSPD in addressing the "tail problem" of under-represented classes. In an ablation study, we assessed the impact of different context levels (**TL**, **SL**, **DL**) on security patch detection by systematically omitting each. Results guide our understanding of contextual contributions.

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

REFERENCES

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Synopsys. Open source security risk analysis, 2023. URL `https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html`.

Yuan Tian, Julia Lawall, and David Lo. Identifying linux bug fixing patches. In *2012 34th international conference on software engineering (ICSE)*, pp. 386–396. IEEE, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems, NeurIPS 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Shu Wang, Xinda Wang, Kun Sun, Sushil Jajodia, Haining Wang, and Qi Li. Graphspd: Graph-based security patch detection with enriched code semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 2409–2426. IEEE, 2023.

Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. An empirical study of secret security patch in open source software. *Adaptive Autonomous Secure Cyber Systems*, pp. 269–289, 2020.

Xinda Wang, Shu Wang, Pengbin Feng, Kun Sun, and Sushil Jajodia. Patchdb: A large-scale security patch dataset. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 149–160. IEEE, 2021a.

Xinda Wang, Shu Wang, Pengbin Feng, Kun Sun, Sushil Jajodia, Sanae Benchaaboun, and Frank Geck. Patchrnn: A deep learning-based system for security patch identification. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, pp. 595–600. IEEE, 2021b.

Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. Spi: Automated identification of security patches via commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1):1–27, 2021.

Daniel Zügner, Tobias Kirschstein, Michele Catasta, Jure Leskovec, and Stephan Günnemann. Language-agnostic representation learning of source code from structure and context. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

A APPENDIX OF APPROACHES EXPLANATION

A.1 PREPROCESSING

As depicted in Figure 1(1), our first step is to decompose the input patches into three distinct segments: tokens, lines, and descriptions. At the word granularity, we define a word sequence $w = \{w_1, w_2, \ldots, w_{nw}\}$, where nw represents the sequence length. We utilize the random embedding function from PyTorch to transform this sequence into a numerical vector. Hence, a word w_i translates to the vector $\mathbf{ew}_i$. The cumulative word embedding of the patch is thus articulated as $\mathbf{EW} = \{\mathbf{ew}_1, \mathbf{ew}_2, \ldots, \mathbf{ew}_{nw}\}$. Analogously, for sequence granularity, we represent sequence vectors as $\mathbf{ES} = \{\mathbf{es}_1, \mathbf{es}_2, \ldots, \mathbf{es}_{ns}\}$, where ns indicates the number of lines, and the description vector is denoted by $\mathbf{ED} = \{\mathbf{ed}_1, \mathbf{ed}_2, \ldots, \mathbf{ed}_{nd}\}$, with nd marking the sequence length of the description.

A.2 MULTILEVEL COMPRESSED CNN (MCC)

Within the domain of our multilevel approach, feature compression across all representational levels is pivotal. To this end, this section is bifurcated into two integral components:

- **Multi-Channel Convolutional Block**: Designed to harness contextual semantics inherent within various level representations.
- **Compressed Residual Block**: A remedy to the vanishing or exploding gradient dilemma often encountered in deep networks, especially given the elongated nature of patches. The adoption of a residual structure enables gradients to directly traverse through residual connections, thereby substantially curtailing the associated risks.

A.2.1 MULTI CHANNEL CONVOLUTIONAL BLOCK

To adeptly grasp patterns of varying lengths from our input, we adopt the multi-channel convolutional neural network as proposed in Kim (2014). This approach utilizes filters, each characterized by distinct kernel sizes, which in essence, define the word window size. For an array of m channels, given as $f_1, f_2, \ldots, f_m$, we correspondingly assign kernel sizes $k_1, k_2, \ldots, k_m$. With these configurations, m 1-dimensional convolutions are executed on the input matrix E. This convolutional operation can be mathematically described as:

$$h_i = \bigwedge_{j=1}^{n} \tanh\left(W_i^T E[j : j + k_i - 1]\right), \tag{1}$$

where the symbol $\bigwedge_{j=1}^{n}$ demarcates the convolutional operations performed in a word sequence. Crucially, the design choice ensures that the output word count n of h_i remains invariant with the input E. This intention preserves the sequence length post-convolution. The term d^f signifies the out-channel size of a filter, with uniformity across filters in output dimensions. Delving deeper into the matrix details, $E[j : j + k_1 - 1]$ = and $E[j : j + k_m - 1]$ represent sub-matrices of E.

Therefore, after multi-channel convolutional block, we otain $h_1, \ldots, h_m$ for word-level vectors, sentence-level vectors, and description-level vectors.

A.2.2 COMPRESSED RESIDUAL BLOCK

To refine the multi-channel convolved word embeddings, we incorporate a series of optimized residual blocks. These blocks offer not only a compact representation of features but also address potential challenges related to gradient dynamics, which is especially crucial for long word sequences.

A.3 RESIDUAL LAYER OVERVIEW

The field of neural networks has witnessed significant advancements in recent years. One pivotal element that has consistently proven crucial in this evolution is the convolutional layer, responsible for primary feature extraction from input data. However, the challenge arises when we delve deeper: how can we maintain the hierarchical representation of features without the risk of information loss? An elegant solution, inspired by the work of He et al. He et al. (2016), introduces the concept of Residual Blocks, visualized in Figure 1.

Architecture of a Residual Block For the residual block r_{mi}, its architecture comprises three convolutional filters: r_{mi_1}, r_{mi_2}, and r_{mi_3}. The computational procedure for these filters on the input can be articulated as:

$$
\begin{aligned}
X_1 &= r_{mi_1}(X) = \bigwedge_{j=1}^{n} \tanh\left(W_{mi_1}^T X[j:j+k_m-1]\right), \\
X_2 &= r_{mi_2}(X_1) = \bigwedge_{j=1}^{n} W_{mi_2}^T X_1[j:j+k_m-1], \\
X_3 &= r_{mi_3}(X) = \bigwedge_{j=1}^{n} W_{mi_3}^T X[j:j+k_m-1], \\
H_{mi} &= \tanh\left(X_2 + X_3\right),
\end{aligned} \tag{2}
$$

In this context, X signifies the initial input to the block. Segments of this input, starting from the j-th row and concluding at the $j + k_m - 1$-th row, undergo transformations facilitated by the aforementioned filters.

Dimensionality and Spatial Relationships The matrix H_{mi} represents the output of each block and conforms to dimensions $\mathbb{R}^{n \times d^i}$. The parameters d^{i-1} and d^i are crucial as they depict the input and output channel sizes respectively. Consequently, the in-channel dimension for the initial block is identified as d^f, whereas the concluding block corresponds to d^p.

The convolutional filters, discerned by the weight matrices W_{mi_1}, W_{mi_2}, and W_{mi_3}, exhibit differential properties in terms of kernel sizes. Notably, while the first two filters align in kernel size with their counterpart in the multi-filter convolutional layer, the third filter distinguishes itself with a singular kernel size.

Summarizing the architecture, the output matrix H_{mp} stands as a testament to the intricate relationship between the convolutional layer's m-th filter and its series of residual blocks. With a total of m filters, the ultimate output is a composite of individual outputs, mathematically represented as $H = H_{1p} \oplus H_{2p} \oplus \cdots \oplus H_{mp}$).

Thus, after the residual block, we obtain Hw, Hs, Hd for word-level, sequence-level, and description-level vectors, respectively.

A.4 Semantic Alignment (SA)

In the vast realm of neural representations, it's often the harmonious interplay between different granularities of data that yields the most insightful results. The extraction of both coarse and fine embeddings from patch code is no exception. In this section, we explore the strategic fusion of the semantic nuances encapsulated in Hw (word-level) and Hs (sequence-level) vectors. By aligning and juxtaposing these embeddings, we aim to achieve a holistic understanding, allowing the model to seamlessly traverse between detailed token-level insights and broader sequence contexts. To facilitate this synthesis, our methodology is bifurcated into two main strategies: *Self-Attentive Pooling* and *Feature Refinement Layer*. The former hones in on the weighted importance of various components, while the latter serves to amalgamate and further process the pooled outputs, ensuring that the final representation is both compact and informative.

A.4.1 Self-Attentive Pooling

In our pursuit of an effective semantic fusion, we first amalgamate the vectors Hw and Hd to formulate the composite vector Hwd. Leveraging this combined vector, we introduce an attention-influenced soft-pooling technique to adeptly harmonize and integrate the underlying semantics of Hw and Hd.

For an exemplar vector, represented as hwd_j, and its contingent neighboring vectors $\{hwd_{j+1}, \cdots, hwd_{j+g-1}\}$, our approach begins by deducing the localized attention scores. This is articulated by:

$$
\alpha_j^i = \mathbf{Hwd}_\alpha^i \mathbf{x}_j^i + b \tag{3}
$$

Subsequent to this, the softmax function is employed to yield:

$$\begin{aligned} \left[\beta_j^i, \cdots, \beta_{j+g-1}^i\right] &= \operatorname{softmax}\left(\left[\alpha_j^i, \cdots, \alpha_{j+g-1}^i\right]\right) \\ \textit{s.t.} \quad \alpha_j^i &= \frac{(\boldsymbol{l}_j^i \boldsymbol{W}^Q)(\boldsymbol{l}_{j+g-1}^i \boldsymbol{W}^K)^{\mathrm{T}}}{\sqrt{d}} \end{aligned} \tag{4}$$

where d is the dimension of the hidden state, and $\boldsymbol{W}^Q \in \mathbb{R}^{d \times d_q}$, $\boldsymbol{W}^K \in \mathbb{R}^{d \times d_k}$, $\boldsymbol{W}^V \in \mathbb{R}^{d \times d_v}$ are the learnable parameters matrices of the self-attention component. Here, we follow the previous works (Vaswani et al., 2017; Zügner et al., 2021) and set $d_q = d_k = d_v = d$.

In this context, both $\mathbf{Hwd}_\alpha^{\mathbf{i}}$ and b are discerned as modifiable parameters. Post this determination, we engage in a soft-pooling mechanism on the g embeddings, which gives rise to the succinct representation:

$$\mathbf{o}_p^i = \sum_{q=j}^{j+g-1} \beta_q^i \mathbf{x}_q^i \tag{5}$$

By adopting this structured approach, the entirety of $nw + ns$ representations undergoes a metamorphosis, culminating in $P = \left\lceil \frac{nw+ns}{g} \right\rceil$ refreshed representations, succinctly denoted as $\left\{\mathbf{o}_1^i, \mathbf{o}_2^i, \cdots, \mathbf{o}_P^i\right\}$.

A.5 Feature Refinement and Embedding Synthesis

To extract high-level features from the transformed representations $\left\{\mathbf{o}_1^i, \mathbf{o}_2^i, \cdots, \mathbf{o}_P^i\right\}$, they are passed through a dense neural layer, serving as a bridge to the final output.

For each $\mathbf{o}_k^i$, where $k \in [1, P]$, the transformation is:

$$\begin{aligned} \mathbf{y}_k^i &= \mathbf{W}_{fc} \mathbf{o}_k^i + \mathbf{b}_{fc} \\ \textit{s.t.} \quad l_k &= \operatorname{ReLU}(\mathbf{y}_k^i) \end{aligned} \tag{6}$$

Here, $\mathbf{W}_{fc}$ is the weight matrix and $\mathbf{b}_{fc}$ is the bias vector of the fully connected layer. This results in the output vectors $l_1, l_2, \ldots, l_P$, which contain the distilled semantic information. The final embedding is a concatenation:

$$l_1, \ldots, l_n = (l_1, \ldots, l_P) \oplus Hd.$$

A.6 Hybrid Feature Aggregation through Advanced Attention Mechanism (HFA)

In modern natural language processing and code understanding tasks, representing data with feature vectors plays a pivotal role. Among the concatenations derived from our model, the vector $(l_1, l_2, \ldots, l_n)$ stands out. This particular vector emerges from the fusion of two distinct embeddings, serving as a primary source of local feature representations.

Given the nuanced interplay of these embeddings, a simple aggregation might not suffice. Herein, the key-query attention mechanism presents itself as an optimal solution. Not only does it weigh the importance of each feature in the local context, but it also juxtaposes it against a broader, global context, resulting in a more balanced and informative feature representation.

Mathematically, using the attention mechanism, the global features are articulated as:

$$\begin{aligned} \boldsymbol{g}_i &= \sum_{j=1}^{n} \frac{\exp(\beta_{ij})}{\sum_{k=1}^{n} \exp(\beta_{ik})} (\boldsymbol{x}_j \boldsymbol{W}^V) \\ \textit{s.t.} \quad \beta_{ij} &= \frac{(\boldsymbol{l}_i \boldsymbol{W}^Q)(\boldsymbol{l}_j \boldsymbol{W}^K)^{\mathrm{T}}}{\sqrt{d}} \end{aligned} \tag{7}$$

where $\boldsymbol{G} = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_P]$. $\boldsymbol{x}_i \in \mathbb{R}^d$.

Through this approach, each compressed word window in our dataset not only retains its inherent local features but also gets enriched by the global context, thereby enhancing the overall representational power of our model.

A.7 BINARY PREDICTION LAYER

For the global vector $\mathbf{D}_g$, a binary prediction is made with the sigmoid function:

$$\begin{aligned} & \tilde{y} = \sigma(\mathbf{w}^\top \mathbf{D}_g + b) \\ \textit{s.t.} \quad & \tilde{y} = \left(1 + \exp\left(-\mathbf{w}^\top \mathbf{D}_g - b\right)\right)^{-1}, \end{aligned} \tag{8}$$

Here, $\mathbf{w}$ represents the learnable weight vector, and b denotes the bias term. The cross-entropy loss for binary prediction is:

$$\mathcal{L} = -y \log(\tilde{y}) - (1 - y) \log(1 - \tilde{y}), \tag{9}$$

Where y is the true label, and $\tilde{y}$ is the predicted probability. This loss guides the optimization of the model's parameters.