Multilevel Semantic Embedding of Software Patches: A Fine-to-Coarse Grained Approach Towards Security Patch Detection

Xunzhu Tang[1]*, Zhenghan Chen[2†], Saad Ezzini[3], Haoye Tian[1], Yewei Song[1], Jacques KLEIN[1], Tegawendé F. Bissyandé[1]

[1]University of Luxembourg, Luxembourg
[2]Peking University, Beijing, China
[3]Lancaster University, Lancaster, UK
xunzhu.tang@uni.lu, 1979282882@pku.edu.cn, s.ezzini@lancaster.ac.uk, haoye.tian@uni.lu, yewei.song@uni.lu, jacques.klein@uni.lu, tegawende.bissyande@uni.lu

Abstract

The growth of open-source software has increased the risk of hidden vulnerabilities that can affect downstream software applications. This concern is further exacerbated by software vendors' practice of silently releasing security patches without explicit warnings or common vulnerability and exposure (CVE) notifications. This lack of transparency leaves users unaware of potential security threats, giving attackers an opportunity to take advantage of these vulnerabilities. In the complex landscape of software patches, grasping the nuanced semantics of a patch is vital for ensuring secure software maintenance. To address this challenge, we introduce a multilevel Semantic Embedder for security patch detection, termed `MultiSEM`. This model harnesses word-centric vectors at a fine-grained level, emphasizing the significance of individual words, while the coarse-grained layer adopts entire code lines for vector representation, capturing the essence and interrelation of added or removed lines. We further enrich this representation by assimilating patch descriptions to obtain a holistic semantic portrait. This combination of multi-layered embeddings offers a robust representation, balancing word complexity, understanding code-line insights, and patch descriptions. Evaluating `MultiSEM` for detecting patch security, our results demonstrate its superiority, outperforming state-of-the-art models with promising margins: a 22.46% improvement on PatchDB and a 9.21% on SPI-DB in terms of the F1 metric.

Introduction

The rapid growth of the open source software (OSS) ecosystem has led to unparalleled advancements in computer software development. However, as with every silver lining, there's a cloud; the increasing reliance on OSS has been accompanied by a dramatic surge in the number of vulnerabilities. According to the 2021 OSSRA report (Synopsys 2023), while 98% of codebases are now composed of open source components, a large 84% of these codebases contain at least one open-source vulnerability. More concerning is the fact that 60% of them face high-risk vulnerability threats. Exploiting these chinks in the armor, attackers have launched "N-day" attacks against unpatched software systems. One glaring example of this is the remote command execution vulnerability (CVE-2021-22205) disclosed in April 2021 (NVD). Seven months after its release, more than 30,000 unpatched GitLab servers that weren't fixed were hacked, making them sources for DDoS attacks. This scenario underscores the significance of timely software patching—a tried and tested countermeasure against "N-day" attacks. However, in reality, the sheer volume of patches, which range from feature additions and performance bug resolutions to security vulnerability fixes, can be overwhelming. As a consequence, software updates are frequently deferred due to the intricate workflow of collating, testing, validating, and scheduling patches (Vaniea and Rashidi 2016). Given this backdrop, there's an urgent need to enable users and administrators to differentiate security patches from the myriad of other updates. Yet, the task isn't as straightforward as it appears. Not all security patches make it to the NVD or are clearly marked in the changelog. Some software vendors, owing to the subjective nature of patch management, opt to release security patches on the down low (Wang et al. 2020a). Such "silent" security patches pose a conundrum for users and system administrators, leaving them in the dark about the true security implications and consequently, the urgency to apply them.

The academic and industry communities have explored multiple avenues to identify security patches. Traditional methods have revolved around machine learning (ML) models that primarily utilize syntax features (Wang et al. 2020a; Tian, Lawall, and Lo 2012). A more contemporary approach involves deploying recurrent neural networks (RNNs) to treat the patch code as sequential data (Wang et al. 2021b; Zhou et al. 2021b). However, these methods fall short in two key aspects: they overlook program semantics and suffer from high false-positive rates. While ML-based solutions often miss out on capturing the intricate dependencies between code statements, RNN-based models, although inspired by natural language processing (NLP) techniques, neglect the unique attributes of programming languages. The end result? Unacceptably high false-positive rates, as evidenced by two RNN-based models that registered rates of 11.6% and 33.2%, respectively. Given that a mere 6-10% of all patches are security-centric (Wang et al. 2021a), the impetus to re-

*Corresponding author
†These authors contributed equally.

duce these false positives becomes even more pronounced.

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.

Approach

We present MultiSEM, a comprehensive framework comprising six distinct components: Preprocessing, Multilevel Compressed CNN, Semantic Alignment, Feature Refinement, Hybrid Feature Aggregation, and Prediction. Our model, designed to capture the fine-to-coarse-grained multilevel semantic embedding of software patches, takes the source code as its primary input and processes it through these components to yield a security patch detection output. The entire structure and flow of our approach can be visualized in Figure 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.

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.

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 submatrices 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.

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.

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:

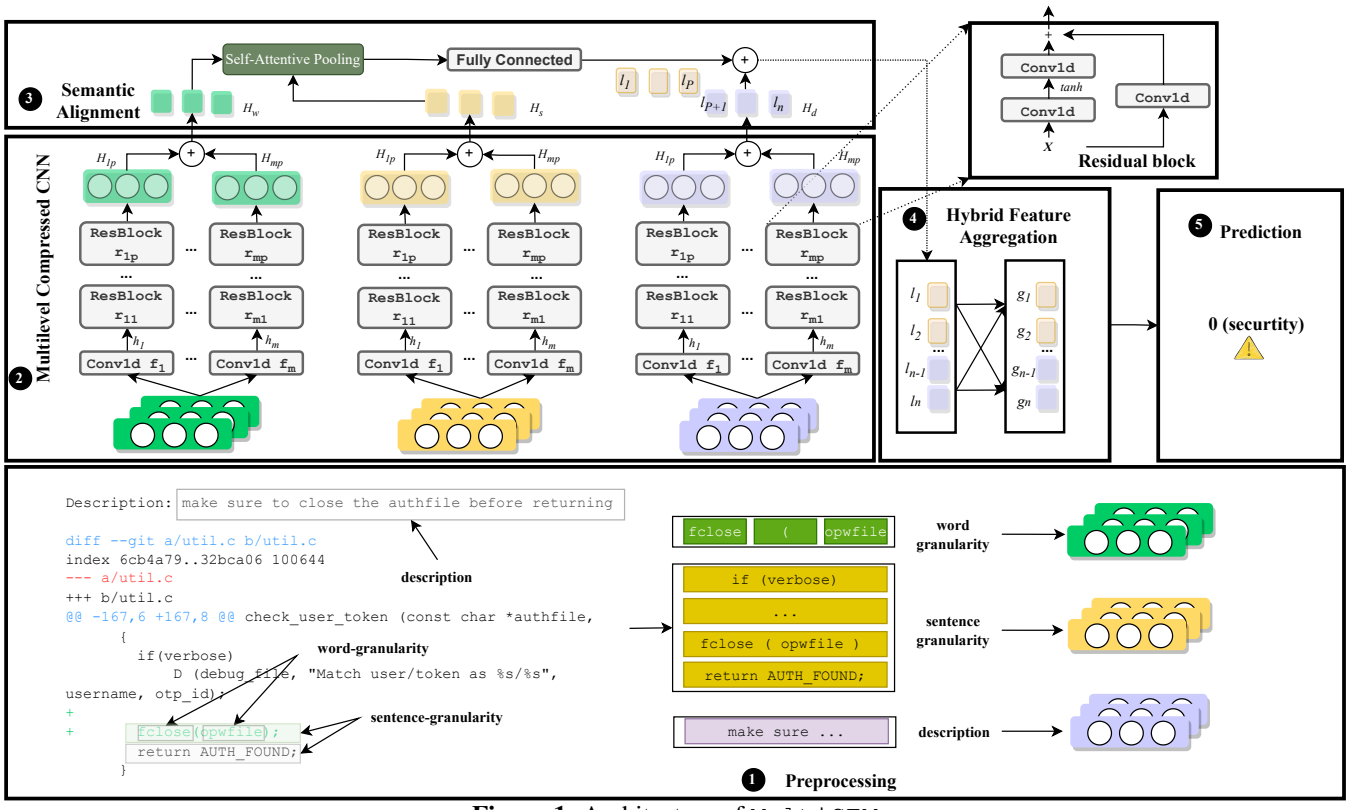

Figure 1: Architecture of `MultiSEM`

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

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.

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.

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:

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

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 $\left\{\mathbf{o}_1^i, \mathbf{o}_2^i, \cdots, \mathbf{o}_P^i\right\}$.

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:

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

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:

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

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.

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.

Experiment Design

In this section, we present our experimental dataset, metrics, state-of-the-art, and research questions.

Dataset

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

PatchDB offers a diverse collection of patches in C/C++ with 12K security-specific and 24K general patches. This dataset is an amalgamation of patches sourced from NVD reference links and direct GitHub commits from 311 renowned open-source projects, such as Linux kernel, MySQL, and OpenSSL. Such diversity allows us to comprehensively evaluate the robustness of security patch detection across various projects.

SPI-DB focuses on patches from projects like Linux, FFmpeg, Wireshark, and QEMU. However, only data concerning FFmpeg and QEMU, totaling 25,790 patches (10K security and 15K non-security), has been made public. Together, these datasets not only grant us a wide spectrum of patch types for cross-project and within-project evaluation but also ensure a balanced representation, catering to both real-world applicability and optimal model training.

Metric

+Recall and **-Recall** as presented in (Tian et al. 2022), serve as specific metrics for evaluating patch correctness. The +Recall metric gauges the ability to predict correct patches, while -Recall assesses the effectiveness in filtering out incorrect ones.

Area Under Curve (AUC) and F1 Score. For the purpose of determining patch correctness, we developed a deep learning-based NLP classifier. To assess the effectiveness of our methodology, we employed the widely-recognized metrics: AUC and the F1 score. The latter is the harmonic mean of precision and recall, specifically applied for identifying correct patches (Hossin and Sulaiman 2015).

True-Positive Rate (TPR). Also known as sensitivity, TPR quantifies how well a classifier identifies positive instances. In the context of security patches, it represents the percentage of valid patches correctly recognized. A high TPR indicates fewer overlooked genuine patches, reducing potential vulnerabilities (Wang et al. 2023).

State-of-the-art

TwinRNN: TwinRNN is metioned in (Wang et al. 2023), emerging from the insights presented in (Zhou et al. 2021b) and (Wang et al. 2021b), the TwinRNN model leverages a unique architecture anchored on RNN-based solutions for detecting security patches. The model's moniker, "twin", originates from its dual RNN module setup, wherein each module processes pre-patch and post-patch code sequences, respectively.

GraphSPD: While TwinRNN offers a commendable benchmark in the realm of patch detection, GraphSPD, presents an alternative perspective and methodology.

Research Questions

RQ-1 How effective is *MultiSEM* in security patch detection?

RQ-2 How does *MultiSEM* fare across various patch categories?

RQ-3 What is the impact of key design choices on the performance of *MultiSEM*?

Experiment Results

[RQ-1:] Overall Performance

Overall Results As depicted in Table 1, our proposed method **MultiSEM** yields the highest performance on both PatchDB and SPI-DB datasets. On PatchDB, MultiSEM achieves an impressive AUC of 83.15% with a commendable F1-score of 77.19. For the SPI-DB dataset, MultiSEM attains an AUC of 68.45% with an F1-score of 57.63. It's pertinent to note that the performances on PatchDB and SPI-DB shouldn't be directly compared due to their differing data distributions. Both datasets serve as our baseline to contrast our solution with existing methodologies.

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

Comparison with Security Patch Detection Approaches We assess the efficiency of MultiSEM against both GraphSPD and TwinRNN by consistently applying the same training and test set divisions, as summarized in Table 1.

Effectiveness. On the PatchDB dataset, MultiSEM surpasses TwinRNN by a significant increase in AUC and an impressive increase in F1-score. When matched against GraphSPD, MultiSEM remains superior in both AUC and F1-score. Similarly, for the SPI-DB dataset, our method MultiSEM outperforms both TwinRNN and GraphSPD in AUC and F1-score. An enhanced AUC indicates that MultiSEM has improved in distinguishing between positive and negative classes. The increased F1-score suggests that MultiSEM provides a more balanced classification, successfully improving both precision and recall. The improvements in Recall+ and TPR indicate that our model is becoming more adept at correctly identifying positive instances, while the advancement in Recall- denotes better classification of negative instances.

Practicality. Precision and the false positive rate are pivotal metrics for ensuring reduced update frequencies and heightened labor productivity. As demonstrated in Table 1, MultiSEM reveals that a significant percentage of the predicted security patches are genuinely security-related. Moreover, for the SPI-DB dataset, MultiSEM consistently excels in precision and has a reduced false positive rate, rendering it an optimal and pragmatic choice for real-world applications.

✍ **Answer to RQ-1:** *Compared to previous approaches, the **MultiSEM** method shows significant advancements: an improvement of 22.46% in F1 over GraphSPD on PatchDB and 9.21% on SPI-DB. This marked enhancement solidifies **MultiSEM** as the optimal choice for real-world applications.*

[RQ-2:] Assessing the Efficacy of *MultiSEM* across Diverse Patch Categories

Table 2: Data statistics of PatchDB based on vulnerability type.

Severity	Vulnerability Type of Patch	Number	Proportion
1	Buffer overflow	1211	10.03%
2	Improper authentication	38	0.31%
3	Resource leakage	197	1.63%
4	Double free/use after free	1162	9.62%
5	Integer overflow	602	4.99%
6	NULL pointer dereference	8484	70.27%
7	Improper input validation	155	1.28%
8	Uncontrolled resource consumption	9	0.07%
9	Race condition	25	0.21%
10	Uninitialized use	62	0.51%
11	Other vulnerabilities	128	1.06%

PatchDB is a comprehensive dataset detailing various vulnerability types observed in software patches. We manually label all patches according to the types of resolved vulnerabilities. As shown in Table 2, the most prevalent vulnerability in the dataset is the "NULL pointer dereference," constituting a significant 70.27% of the entries. Other notable vulnerabilities include "buffer overflow" at 10.03% and "double free/use after free" making up 9.62%. The dataset also captures more nuanced vulnerabilities, such as "improper authentication" and "uncontrolled resource consumption," representing 0.31% and 0.07%, respectively. Lesser observed vulnerabilities like "race condition" and "uninitialized use" are also cataloged, making PatchDB a diverse repository for analyzing and understanding software vulnerabilities. More details about labelling will be shown in Appendix.

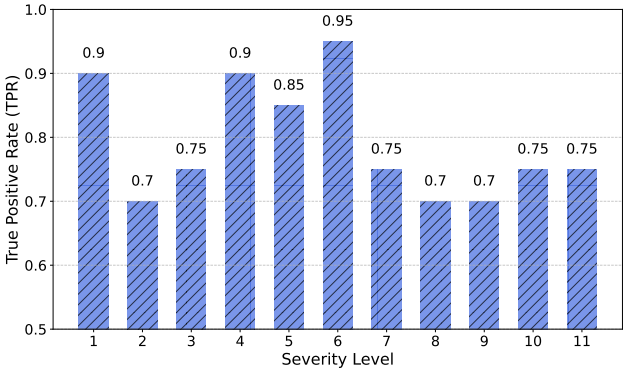

Figure 2: True Positive Rate (TPR) by Vulnerability Type in PatchDB Dataset.

As shown in Figure 2, through an in-depth performance analysis of MultiSEM across various types of security patches, we unearthed two pivotal insights.

Firstly, security patch types with a commendable TPR by our tool, such as 'NULL pointer dereference' (with a TPR of 95%), 'Buffer overflow', and 'Double free/use after free' (both around 90%), indicate that these patches possess distinct features from their non-security counterparts. This distinction aids in designing more effective detection systems. For example, patches addressing 'Resource leakage' typically involve memory reinitialization and file operations, pointing towards memory API interactions. Similarly, patches for 'Race conditions' predominantly leverage lock/unlock operations to synchronize processes or threads, making them intimately linked with lock APIs.

Secondly, some security patch types have a less impressive TPR under MultiSEM, usually because they involve subtler security check modifications. Patches for issues like 'Improper input validation', 'Buffer overflow', and 'Improper authentication' often rely on conditional statements to delineate operational boundaries. While such checks are typical security patch patterns, they can easily be confounded with non-security patterns. Developers frequently employ conditional constructs to introduce new functionalities for specific scenarios. Hence, gleaning insights from the broader context is quintessential for accurate detection by MultiSEM. Additionally, the effect of data imbalance is palpable. For instance, patches for 'Uncontrolled resource consumption' constitute only about 0.07% of the dataset, furnishing sparse patterns for effective deep learning. We believe the performance for such categories would see a considerable enhancement with richer data availability.

Tail Problem As shown in Table 3, in order to evaluate the effectiveness in solving tail problem, we compare our MultiSEM and GraphSPD on severity #2,#8,#9, and #10.

Table 3: Comparison of MultiSEM and GraphSPD on TPR for specific severities.

Severity	MultiSEM TPR (%)	GraphSPD TPR (%)
2 (Improper authentication)	70	68
8 (Uncontrolled resource consumption)	70	65
9 (Race condition)	70	67
10 (Uninitialized use)	75	69

The so-called "tail problem" in machine learning is a challenge that arises when certain classes or types within a dataset are under-represented, resulting in suboptimal performance for models trained on such data. Typically, these rare categories, or the 'tail' classes, may get overshadowed by the more dominant or frequently occurring classes during the model's learning process.

To gauge the prowess of our solution, MultiSEM, in effectively addressing the tail problem, we carried out an in-depth comparison between MultiSEM and its counterpart, GraphSPD. Our comparative study primarily focused on the specific severities of #2, #8, #9, and #10—categories that are notably challenging due to their sparse occurrence in the dataset.

The results of our study, as highlighted in Table 3, indicate a noticeable edge that MultiSEM boasts over GraphSPD. For severity 2 (Improper authentication), our model registers a TPR of 70%, surpassing GraphSPD's 68%. Similar superiority is observed for severity 8 (Uncontrolled resource consumption) and severity 9 (Race condition), where MultiSEM's TPRs are 70% and 70% respectively, in contrast to GraphSPD's 65% and 67%. The most prominent difference is discerned in severity 10 (Uninitialized use), where MultiSEM leads with a 75% TPR, a notable 6% ahead of GraphSPD.

This comparative analysis underlines the adeptness of MultiSEM in handling the tail problem. By achieving higher True Positive Rates (TPRs) across these challenging severities, MultiSEM proves its efficiency in identifying and correctly classifying under-represented vulnerability types. This is crucial, as addressing the tail problem ensures

that even the rarest of vulnerabilities do not slip under the radar, bolstering the overall security robustness.

✍ **Answer to RQ-2:** *PatchDB highlights a variety of software vulnerabilities, with "NULL pointer dereference" being the most prevalent at 70.27%.* ***MultiSEM*** *performs exceptionally well in detecting major vulnerabilities but faces challenges with subtler ones due to data imbalances. In addressing the "tail problem" of under-represented classes,* ***MultiSEM*** *consistently outperforms GraphSPD, exemplifying its capability to effectively detect even rare vulnerabilities.*

[RQ-3:] Ablation Study

To discern the relative importance of the different levels of context used in our approach — specifically token-level (**TL**), sentence-level (**SL**), and description-level (**DL**) — we embarked on an ablation study. By systematically omitting one of these levels at a time, we generated three variants of MultiSEM: MultiSEM $TL-$ (without token-level context), MultiSEM $SL-$ (sans sentence-level context), and MultiSEM $_{DL-}$ (devoid of description-level context). The goal of this study was to shed light on how each contextual level contributes to the overall performance in security patch detection.

Table 4: Performance evaluation of MultiSEM variants on security patch detection (%).

Method	Dataset	AUC	F1	Recall+	Recall-	TPR
MultiSEM $_{TL-}$	PatchDB	80.50	73.12	75.35	82.37	75.42
	SPI-DB	65.00	54.25	66.00	75.10	68.60
MultiSEM $_{SL-}$	PatchDB	81.50	74.00	76.20	83.40	76.50
	SPI-DB	66.50	55.30	67.20	76.20	69.60
MultiSEM $_{DL-}$	PatchDB	78.00	70.50	72.00	80.00	73.00
	SPI-DB	62.50	52.00	63.00	72.00	66.50
MultiSEM	PatchDB	83.15	77.19	79.52	86.78	79.52
	SPI-DB	68.45	57.63	70.24	80.12	73.25

As revealed by the results in Table 4: The removal of token-level information (MultiSEM $_{TL-}$) significantly hampered the performance across both datasets, but it was especially pronounced in the PatchDB dataset. This indicates that token-level insights are vital, providing a granularity of detail that's essential for detecting nuances in security patches. Without the sentence-level context (MultiSEM $SL-$), there was a noticeable drop in performance, although not as drastic as with the MultiSEM $TL-$ variant. This reveals the utility of understanding the broader semantics of the code within the scope of a sentence or statement. This context helps in capturing relations between various tokens and offers a more comprehensive view than tokens in isolation. Most significantly, the exclusion of the description-level context (MultiSEM $_{DL-}$) led to the most substantial degradation in performance. This was particularly evident in the AUC, F1, and Recall+ metrics across both datasets. Such a marked decline underscores the criticality of understanding the overarching narrative or intention behind a patch. The descriptive context often contains rich semantic information that can offer valuable hints or differentiate between patches, more so than local contexts like tokens or sentences.

In summation, while all contextual levels contribute positively to the performance, the description-level context emerged as the most influential. It proves that an understanding of the holistic description or rationale behind a patch is paramount in security patch detection tasks. This ablation study, thus, underscores the multi-faceted nature of our approach and reaffirms the necessity of a multi-level contextual understanding for high-precision security patch detection.

✍ **Answer to RQ-3:** *The ablation study on MultiSEM reveals the significant role each contextual level plays in security patch detection. The removal of the description-level context (MultiSEM $_{DL-}$) resulted in the most pronounced performance drop, emphasizing its paramount importance in understanding patches. While all contexts are beneficial, the holistic understanding provided by the description-level is crucial for precise security patch detection.*

Related Work

Advancements and Techniques in Security Patch Analysis

In the realm of patch analysis, Li et al. (Li and Paxson 2017) undertook an empirical study of security patches, unearthing key behaviors. Soto et al. (Soto et al. 2016) provided insights into Java patches, advancing automated code repairs. VCCFinder (Perl et al. 2015) utilized SVM to detect suspicious patches, while Tian et al. (Tian, Lawall, and Lo 2012) targeted bug corrections within Linux. SPIDER (Machiry et al. 2020) highlighted secure patches that maintain normal program function. Rule-driven approaches for discerning security patches were proposed by Wu et al. and Huang et al. (Wu et al. 2020; Huang et al. 2019). Vulmet (Xu et al. 2020) offers automatic urgent patching for Android, and Wang's group (Wang et al. 2020b) combined random forests with patch features to classify vulnerabilities. The rise in machine learning applications for patch analysis, especially deep learning, is evident in recent works (Hoang et al. 2019a; Tian et al. 2020; Hoang et al. 2019b). PatchRNN (Wang et al. 2021b) and SPI (Zhou et al. 2021b) employed RNNs for security patch identification, and GraphSPD (Wang et al. 2023) tapped into graph structures to improve detection accuracy. The broader field of binary patch analysis includes binary differentiation (Ming et al. 2017; Duan et al. 2020; Zhao et al. 2020), verification (Dai et al. 2020; Zhang et al. 2021), recognition (Xu et al. 2017), and automation (Duan et al. 2019; Tian et al. 2023; Niesler, Surminski, and Davi 2021).

Progress in Sequential Data Methods

The deep learning field has seen innovations in optimizing sequential data representations. Key strategies focus on multi-level methodologies (Tang et al. 2021a) capturing complexities in data from both NLP and CV. For example, Mototang et al. (Tang et al. 2021b) applied sequence embeddings across sentence facets. Niu et al. (Niu et al. 2020) introduced the MIA model for person re-identification. Du et al. (Du et al. 2020) proposed a method for fine-grained visual classification. Other notable works include (Jin, Wang, and Wan 2020; Ling et al. 2023; Li et al. 2021; Zhou et al.

2021a). Additionally, compression techniques, like Luo et al.'s approach (Luo et al. 2021) with multi-filter CNNs and Resnet, have emerged for encoding complex sequences.

Conclusion

In the backdrop of an increasing reliance on open source software and the subsequent surge in vulnerabilities, there's an urgent need for accurate security patch detection. Our study introduces a groundbreaking method, employing a fine-to-coarse grained approach combined with multilevel semantic embedding techniques. This novel approach has proven to be highly effective, as evidenced by our experimental results, demonstrating significant improvements in accuracy and a notable reduction in false positives. As the software landscape continues to evolve, our methodology stands out, offering a beacon of hope for addressing the pressing challenges of security patch detection.

Acknowledgments

This work is supported by the NATURAL project, which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant No. 949014).

References

Dai, J.; Zhang, Y.; Jiang, Z.; Zhou, Y.; Chen, J.; Xing, X.; Zhang, X.; Tan, X.; Yang, M.; and Yang, Z. 2020. {BScout}: Direct Whole Patch Presence Test for Java Executables. In *29th USENIX Security Symposium (USENIX Security 20)*, 1147–1164.

Du, R.; Chang, D.; Bhunia, A. K.; Xie, J.; Ma, Z.; Song, Y.-Z.; and Guo, J. 2020. Fine-grained visual classification via progressive multi-granularity training of jigsaw patches. In *European Conference on Computer Vision*, 153–168. Springer.

Duan, R.; Bijlani, A.; Ji, Y.; Alrawi, O.; Xiong, Y.; Ike, M.; Saltaformaggio, B.; and Lee, W. 2019. Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries. In *NDSS*.

Duan, Y.; Li, X.; Wang, J.; and Yin, H. 2020. Deepbindiff: Learning program-wide code representations for binary diffing. In *Network and distributed system security symposium*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hoang, T.; Lawall, J.; Oentaryo, R. J.; Tian, Y.; and Lo, D. 2019a. PatchNet: a tool for deep patch classification. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 83–86. IEEE.

Hoang, T.; Lawall, J.; Tian, Y.; Oentaryo, R. J.; and Lo, D. 2019b. Patchnet: Hierarchical deep learning-based stable patch identification for the linux kernel. *IEEE Transactions on Software Engineering*, 47(11): 2471–2486.

Hossin, M.; and Sulaiman, M. N. 2015. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2): 1.

Huang, Z.; Lie, D.; Tan, G.; and Jaeger, T. 2019. Using safety properties to generate vulnerability patches. In *2019 IEEE Symposium on Security and Privacy (SP)*, 539–554. IEEE.

Jin, H.; Wang, T.; and Wan, X. 2020. Multi-granularity interaction network for extractive and abstractive multi-document summarization. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, 6244–6254.

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

Li, F.; and Paxson, V. 2017. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2201–2215.

Li, Y.; Qian, Y.; Yu, Y.; Qin, X.; Zhang, C.; Liu, Y.; Yao, K.; Han, J.; Liu, J.; and Ding, E. 2021. Structext: Structured text understanding with multi-modal transformers. In *Proceedings of the 29th ACM International Conference on Multimedia*, 1912–1920.

Ling, Y.; Zhong, Z.; Luo, Z.; Yang, F.; Cao, D.; Lin, Y.; Li, S.; and Sebe, N. 2023. Cross-modality earth mover's distance for visible thermal person re-identification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 1631–1639.

Luo, J.; Xiao, C.; Glass, L.; Sun, J.; and Ma, F. 2021. Fusion: towards automated ICD coding via feature compression. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2096–2101.

Machiry, A.; Redini, N.; Camellini, E.; Kruegel, C.; and Vigna, G. 2020. Spider: Enabling fast patch propagation in related software repositories. In *2020 IEEE Symposium on Security and Privacy (SP)*, 1562–1579. IEEE.

Ming, J.; Xu, D.; Jiang, Y.; and Wu, D. 2017. {BinSim}: Trace-based Semantic Binary Diffing via System Call Sliced Segment Equivalence Checking. In *26th USENIX Security Symposium (USENIX Security 17)*, 253–270.

Niesler, C.; Surminski, S.; and Davi, L. 2021. HERA: Hotpatching of Embedded Real-time Applications. In *NDSS*.

Niu, K.; Huang, Y.; Ouyang, W.; and Wang, L. 2020. Improving description-based person re-identification by multi-granularity image-text alignments. *IEEE Transactions on Image Processing*, 29: 5542–5556.

(NVD), N. V. D. 2021. CVE-2021-22205 Detail.

Perl, H.; Dechand, S.; Smith, M.; Arp, D.; Yamaguchi, F.; Rieck, K.; Fahl, S.; and Acar, Y. 2015. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 426–437.

Soto, M.; Thung, F.; Wong, C.-P.; Le Goues, C.; and Lo, D. 2016. A deeper look into bug fixes: patterns, replacements, deletions, and additions. In *Proceedings of the 13th International Conference on Mining Software Repositories*, 512–515.

Synopsys. 2023. Open Source Security Risk Analysis.

Tang, X.; Zhu, R.; Sun, T.; and Wang, S. 2021a. Moto: Enhancing embedding with multiple joint factors for chinese text classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 2882–2888. IEEE.

Tang, X.; Zhu, R.; Sun, T.; and Wang, S. 2021b. Moto: Enhancing Embedding with Multiple Joint Factors for Chinese Text Classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 2882–2888.

Tian, H.; Liu, K.; Kaboré, A. K.; Koyuncu, A.; Li, L.; Klein, J.; and Bissyandé, T. F. 2020. Evaluating representation learning of code changes for predicting patch correctness in program repair. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 981–992.

Tian, H.; Tang, X.; Habib, A.; Wang, S.; Liu, K.; Xia, X.; Klein, J.; and Bissyandé, T. F. 2022. Is this Change the Answer to that Problem? Correlating Descriptions of Bug and Code Changes for Evaluating Patch Correctness. *arXiv preprint arXiv:2208.04125*.

Tian, H.; Tang, X.; Habib, A.; Wang, S.; Liu, K.; Xia, X.; Klein, J.; and BissyandÉ, T. F. 2023. Is This Change the Answer to That Problem? Correlating Descriptions of Bug and Code Changes for Evaluating Patch Correctness. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22. New York, NY, USA: Association for Computing Machinery. ISBN 9781450394758.

Tian, Y.; Lawall, J.; and Lo, D. 2012. Identifying linux bug fixing patches. In *2012 34th international conference on software engineering (ICSE)*, 386–396. IEEE.

Vaniea, K.; and Rashidi, Y. 2016. Tales of Software Updates: The Process of Updating Software. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, 3215–3226. New York, NY, USA: Association for Computing Machinery. ISBN 9781450333627.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems, NeurIPS 2017, December 4-9, 2017, Long Beach, CA, USA*, 5998–6008.

Wang, S.; Wang, X.; Sun, K.; Jajodia, S.; Wang, H.; and Li, Q. 2023. GraphSPD: Graph-based security patch detection with enriched code semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2409–2426. IEEE.

Wang, X.; Sun, K.; Batcheller, A.; and Jajodia, S. 2020a. An empirical study of secret security patch in open source software. *Adaptive Autonomous Secure Cyber Systems*, 269–289.

Wang, X.; Wang, S.; Feng, P.; Sun, K.; and Jajodia, S. 2021a. Patchdb: A large-scale security patch dataset. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 149–160. IEEE.

Wang, X.; Wang, S.; Feng, P.; Sun, K.; Jajodia, S.; Benchaaboun, S.; and Geck, F. 2021b. Patchrnn: A deep learning-based system for security patch identification. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, 595–600. IEEE.

Wang, X.; Wang, S.; Sun, K.; Batcheller, A.; and Jajodia, S. 2020b. A machine learning approach to classify security patches into vulnerability types. In *2020 IEEE Conference on Communications and Network Security (CNS)*, 1–9. IEEE.

Wu, Q.; He, Y.; McCamant, S.; and Lu, K. 2020. Precisely characterizing security impact in a flood of patches via symbolic rule comparison. In *The 2020 Annual Network and Distributed System Security Symposium (NDSS'20)*.

Xu, Z.; Chen, B.; Chandramohan, M.; Liu, Y.; and Song, F. 2017. Spain: security patch analysis for binaries towards understanding the pain and pills. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 462–472. IEEE.

Xu, Z.; Zhang, Y.; Zheng, L.; Xia, L.; Bao, C.; Wang, Z.; and Liu, Y. 2020. Automatic hot patch generation for android kernels. In *29th USENIX Security Symposium (USENIX Security 20)*, 2397–2414.

Zhang, Z.; Zhang, H.; Qian, Z.; and Lau, B. 2021. An investigation of the android kernel patch ecosystem. In *30th USENIX Security Symposium (USENIX Security 21)*, 3649–3666.

Zhao, L.; Zhu, Y.; Ming, J.; Zhang, Y.; Zhang, H.; and Yin, H. 2020. Patchscope: Memory object centric patch diffing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 149–165.

Zhou, T.; Wang, W.; Liu, S.; Yang, Y.; and Van Gool, L. 2021a. Differentiable multi-granularity human representation learning for instance-aware human semantic parsing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1622–1631.

Zhou, Y.; Siow, J. K.; Wang, C.; Liu, S.; and Liu, Y. 2021b. Spi: Automated identification of security patches via commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1): 1–27.

Zügner, D.; Kirschstein, T.; Catasta, M.; Leskovec, J.; and Günnemann, S. 2021. 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*.