

# Detection of Vulnerabilities in Tensorflow with LSTM and BERT <sup>†</sup>

Sergio Muñoz Martín , Luis Alberto Martinez Hernandez , Ana Lucila Sandoval Orozco   
and Luis Javier García Villalba \* 

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), Faculty of Computer Science and Engineering, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases, 9, Ciudad Universitaria, 28040 Madrid, Spain; sergmu03@ucm.es (S.M.M.); luisal11@ucm.es (L.A.M.H.); asandoval@fdi.ucm.es (A.L.S.O.)

\* Correspondence: javiergv@fdi.ucm.es; Tel.: +34-91-394-7638

<sup>†</sup> Presented at the First Summer School on Artificial Intelligence in Cybersecurity, Cancun, Mexico, 3–7 November 2025.

## Abstract

This work has developed a Deep Learning model that analyses the semantics of the Python code used when working with TensorFlow and detects vulnerabilities to improve data security and bug recognition. This research not only seeks to improve the security of TensorFlow, but also aims to be a solution for other deep learning frameworks in the future and help developers find existing vulnerabilities to facilitate secure code writing.

**Keywords:** static analysis; deep learning; cybersecurity; BERT; LSTM; TensorFlow; vulnerabilities

## 1. Introduction

Technology has become an essential pillar of people's daily lives and the functioning of businesses, and with it, the security of computer systems has taken on a crucial role in protecting information, especially in software security, where vulnerabilities often originate in the code, posing a potential threat to the integrity of systems. Artificial Intelligence (AI) algorithms have become very popular and are beginning to be implemented in most work environments, working with sensitive data such as a customer's financial data, so this research aims to improve the security that currently exists when working with AI algorithms.

In 2023, OWASP [1] conducted a study that lists the main vulnerabilities and security risks that appear in Large Language Models (LLMs) and derivative technologies according to their criteria. This research focuses on one of these vulnerabilities, the supply chain attacks. These attacks target the packages and frameworks used to develop models, which already include the code necessary for data pre-processing and neural network training.

Among those used in the supply chain in the development of AI applications, this study has chosen Tensorflow, one of the most widely used AI frameworks in Python, which includes in its package wide range of networks used today.

With all this in mind, the objective of this work is to develop a Deep Learning model [2] that analyses the semantics of the Python code used when working with TensorFlow and detects vulnerabilities to improve data security and bug recognition. This research not only seeks to improve the security of TensorFlow but also aims to be a solution for other deep learning frameworks in the future and help developers find existing vulnerabilities to facilitate secure code writing.



Academic Editors: Héctor Manuel Pérez Meana and Gabriel Sánchez Pérez

Published: 4 February 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

The rest of the work is organized as follows: Section 2 presents the state of the art. Section 3 presents the methodology followed for the development of the work and the materials that were used. Section 4 shows the experiments and results, and finally, Section 5 shows the conclusions of the work.

## 2. State of the Art

Most state-of-the-art proposals perform data pre-processing with the aim of helping the model find the relationships that exist in the code that make it vulnerable or not. The Table 1 provides a summary of the proposals that make up the state of the art. One data pre-processing methodology is to use GNN networks capable of directly processing graph-structured data and learning from the relationships between nodes. Using these networks, models such as FUNDED [3], Mvulpreter [4], and Vulccn [5] seek to generate a graph in which code variables, objects, and functions are represented by vertices, and links or branches are used to represent their relationships and dependencies. The use of this methodology allows for better collection of the relationships that appear in the code, and the task of finding which points are positive or negative becomes easier for the network [3].

The rest of the proposals divide the code into parts or sequences according to different criteria. The aim of this idea is to provide the network with important pieces of code and let the network perform all the natural language analysis. Its main function is to go through the entire code, generating small blocks of code using a window of  $m$  characters that contains the context to be analysed.

**Table 1.** State of art.

| Model              | Context    | Accuracy     | Precision | F1     |
|--------------------|------------|--------------|-----------|--------|
| FUNDED             | Grafos     | C, Java, PHP | GNN       | 94%    |
| mVulPreter         | Slices     | C y C++      | GNN       | 78.8%  |
| VulDeePecker       | C.Gadgets  | C y C++      | BLSTM     | 85.5%  |
| $\mu$ vuldeepecker | C.Gadgets  | C y C++      | BLSTM     | 94.22% |
| Vudenc             | Secuencial | Python       | LSTM      | 80 %   |
| BBVD               | Secuencial | C y C++      | RoBERTa   | 89.37  |

## 3. Materials and Methods

### 3.1. Data Recollection

TensorFlow is a framework developed mainly in C++, but it also has code in Python so that it can be used in this programming language. When searching for datasets or studies on vulnerabilities collected in TensorFlow, it can be observed that most articles [6,7] focus on C++ code and collect classic vulnerabilities such as integer overflow or out-of-bounds writing.

Therefore, to obtain a dataset with vulnerabilities in Python, 1500 commits were extracted from the TensorFlow Github repository [8] through a search that extracted commits related to vulnerabilities or bug fixes by identifying commits in which the message contained words such as: vulnerability, security, danger, bug, exploit, cve, etc.

### 3.2. Data Preprocessing

This stage involves the following steps: code cleaning, comment removal, code division into blocks, and block labelling.

To obtain a model capable of identifying which area of the code is vulnerable, each line of code will be analysed sequentially. Since the objective of the model is not only to identify whether the code is vulnerable, but also the area where the vulnerability is located, it is necessary to train the code in parts, as seen in the State of the Art. In this case, the code

will be divided sequentially into blocks. To determine the size of the code blocks, different reference measures can be implemented, such as characters, tokens, or lines of code. In this case, lines of code have been chosen, as they coincide with the lines removed from the GitHub commit. Each block will contain a line of code to be analysed, accompanied by  $n$  lines of context before and after, allowing the model to analyse the relationships and patterns that appear in the code to determine whether it is vulnerable or not. A vulnerable code fragment is processed by generating blocks with two lines of context before and after the target line. In each block, the first two lines correspond to the preceding context, the third line is the main line analysed for vulnerability, and the last two lines represent the subsequent context. The blocks are generated sequentially by shifting the main analysed line by one position each time until the entire code file has been processed.

In addition, as each block is being generated, the information from the lines removed from the commit is used to check whether the line that is the focus of attention is vulnerable or not. If the line matches one of the removed lines, the block is labelled as 1; otherwise, it is labelled as 0.

### 3.3. Model Training

For this work, two models were chosen for testing: first, BERT, due to its proven performance in this type of task in recent research; finally, an LSTM network, which has already been proven to offer good results in these tasks [9] and also requires less computational power and is much lighter than the LLMs that have become so popular recently. In order for neural networks to be able to interpret the language of programming code, the code must be transformed into numerical vectors known as embeddings. These embeddings are generated by training a neural network, which estimates for each word the probability of finding each of the other words in the vocabulary in its context. This ensures that two semantically similar words have similar embeddings.

## 4. Results

To test the model's results and performance, a computer with an Intel Core i7-7920HQ CPU at 3.10 GHz, 16.0 GB of RAM, and a GeForce RTX 3070 OC Edition GPU with 8 GDDR6 was used. To ensure a rigorous and balanced evaluation of the model that allows for the assessment of model variability, the experiments were carried out using 5-fold cross-validation. In these experiments on the LSTM network, the learning rate chosen was  $1 \times 10^{-5}$ . To avoid overfitting, a dropout of 0.2 was established and early stopping was implemented, taking the F1 score as the reference metric. The batch size was set to 64 and a maximum of 30 epochs was established for each fold. To optimize training by adjusting the learning rate, the ADAM algorithm [10] was used. In the case of the BERT model, the BERT-base-uncased version was used, which consists of approximately 110 million parameters. A learning rate of  $2 \times 10^{-5}$  was set, and to avoid overfitting, a weight decay of 0.01 was set. The batch size was set to 16, and as in the LSTM model, early stopping was added with a maximum number of 30 epochs.

The results of the trained models can be seen in Table 2, which shows the final cross-validation test metrics for each model.

The three-line BERT model has obtained positive metrics close to 0.875 across all metrics, without observing a reduction in Precision and F1 as seen in some state-of-the-art proposals, indicating that the model performs solidly in the experiments conducted. By adding more context to the 5-line model, the model increases its metrics to 0.897, showing that increasing the context helps the model find the patterns that make a block of code vulnerable or not. On the other hand, in the LSTM model, being an inferior model, it was predictable that the metrics would decrease, and this has been the case, with an F1 score of

0.80 in the three-line model and 0.81 in the five-line model. Comparing the LSTM and BERT models, we can see that the increase in metrics in the LSTM network when increasing the context is lower than in the BERT model, and the accuracy metrics decrease considerably in the LSTM model compared to the other metrics.

**Table 2.** Experiments Results.

| Model | Context | Accuracy | Precision | Recall | F1     |
|-------|---------|----------|-----------|--------|--------|
| BERT  | 3 lines | 0.8751   | 0.8769    | 0.8751 | 0.8756 |
| BERT  | 5 lines | 0.8969   | 0.8970    | 0.8969 | 0.8970 |
| LSTM  | 3 lines | 0.8308   | 0.7562    | 0.8526 | 0.8012 |
| LSTM  | 5 lines | 0.8430   | 0.7778    | 0.8508 | 0.8125 |

## 5. Conclusions

This work has begun the development of a tool that helps programmers identify vulnerabilities in Python when working on Artificial Intelligence algorithms. The tests carried out so far in the TensorFlow framework are positive and promising, demonstrating that these tools can be implemented to detect security flaws in this field and many others, as seen in the state of the art. The experiments carried out show that BERT is more effective than LSTM for this task, especially when the amount of context is increased. BERT's ability to generate robust contextual embeddings is reflected in its better results, while LSTM, although it improves with more context, is not sufficient in comparison with the BERT model, so although it requires less computing power, its performance needs to be increased to make it viable in a real environment. One of the most significant problems that has arisen during the research is the collection of vulnerable Python code samples specifically related to artificial intelligence algorithms. This scarcity of vulnerable samples may affect the generalization of the model, so to address this issue in the next steps of the research, we propose supplementing the dataset with vulnerabilities from other frameworks already mentioned, such as Torch, MxNet, or Hugging Face.

**Author Contributions:** Conceptualization, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V.; methodology, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V.; validation, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V.; investigation, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V.; writing—original draft preparation, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V.; writing—review and editing, S.M.M., L.A.M.H., A.L.S.O. and L.J.G.V. All authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the European Commission under the Horizon 2020 research and innovation programme, as part of the project HEROES ([heroes-fct.eu](https://heroes-fct.eu), Grant Agreement no. 101021801) and of the project ALUNA ([aluna-isf.eu](https://aluna-isf.eu), Grant Agreement no. 101084929). This work was also carried out with funding from the Recovery, Transformation and Resilience Plan, financed by the European Union (Next Generation EU), through the Chair “Cybersecurity for Innovation and Digital Protection” INCIBE-UCM. In addition this work has been supported by Comunidad Autonoma de Madrid, CIRMA-CM Project (TEC-2024/COM-404). The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in this study are included in the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. OWASP Foundation. OWASP Machine Learning Security Top 10. 2025. Available online: <https://owasp.org/www-project-machine-learning-security-top-10/> (accessed on 14 January 2025).
2. Neuhaus, S.; Zimmermann, T.; Holler, C.; Zeller, A. Predicting Vulnerable Software Components. In Proceedings of the CCS '07: 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 31 October–2 November 2007; pp. 529–540. [CrossRef]
3. Wang, H.; Ye, G.; Tang, Z.; Tan, S.H.; Huang, S.; Fang, D.; Feng, Y.; Bian, L.; Wang, Z. Combining Graph-Based Learning with Automated Data Collection for Code Vulnerability Detection. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 1943–1958. [CrossRef]
4. Zou, D.; Hu, Y.; Li, W.; Wu, Y.; Zhao, H.; Jin, H. mVulPreter: A Multi-Granularity Vulnerability Detection System with Interpretations. *IEEE Trans. Dependable Secur. Comput.* **2022**, 1–12. [CrossRef]
5. Wu, Y.; Zou, D.; Dou, S.; Yang, W.; Xu, D.; Jin, H. VulCNN: An Image-inspired Scalable Vulnerability Detection System. In Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, 21–29 May 2022; pp. 2365–2376. [CrossRef]
6. Harzevili, N.S.; Shin, J.; Wang, J.; Wang, S.; Nagappan, N. Characterizing and Understanding Software Security Vulnerabilities in Machine Learning Libraries. In Proceedings of the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), Melbourne, Australia, 15–16 May 2023; pp. 27–38. [CrossRef]
7. Filus, K.; Domańska, J. Software vulnerabilities in TensorFlow-based deep learning applications. *Comput. Secur.* **2023**, *124*, 102948. [CrossRef]
8. Tensorflow. TensorFlow: An Open Source Machine Learning Framework for Everyone. Available online: <https://github.com/tensorflow/tensorflow> (accessed on 4 January 2025).
9. Wartschinski, L.; Noller, Y.; Vogel, T.; Kehr, T.; Grunske, L. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. *Inf. Softw. Technol.* **2022**, *144*, 106809. [CrossRef]
10. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2017**, arXiv:1412.6980. [CrossRef]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.