

Application of Advanced AI Methods for Precise Vulnerability Detection

Amirreza Bagheri

Department of Software Engineering
University of Szeged

Szeged, 2025

Supervisor:

Dr. Péter Hegedűs

Summary of the Ph.D. thesis submitted for the degree of Doctor of Philosophy
of the University of Szeged



University of Szeged
PhD School in Computer Science

Introduction

Software vulnerabilities pose significant risks to modern society, affecting sectors like healthcare, transportation, and education. Errors in code can lead to exploits like WannaCry [1], causing billions in damages, or Heartbleed [3], which impacted billions of users and could have been fixed with minimal code additions. Cyberattacks cost an estimated \$400 billion annually, with over 125,000 entries in the CVE database by January 2024. Despite guidelines, developers often introduce vulnerabilities, exacerbated by outdated software and code reuse in open-source projects. Manual code inspection is essential but labor-intensive, requiring security expertise to identify hidden flaws that may not affect normal operations.

Vulnerabilities can spread quickly via platforms like GitHub, and attackers need only one exploit to cause harm. Traditional static analysis tools, such as FindBugs or Splint, help but suffer from high false positives and rely on expert-defined rules. Machine learning offers a promising alternative by learning patterns from large datasets, automating detection early in development, reducing costs, and adapting to new threats objectively. This study develops a proof-of-concept tool using deep learning for Python vulnerability detection from real GitHub code. It employs CodeBERT embeddings and LSTM for sequential token-level analysis, enhanced by Conformers to fuse local and global features via convolutions and self-attention, and LLMs like UniXcoder for rich semantic context. This tripartite model achieves high precision, addressing limitations of prior methods by enabling block-level detection with low false rates and improved efficiency. This dissertation focuses on solving the problems outlined above. The three main results of the thesis are the following:

- Sequential Modeling with LSTM and Code Embeddings
- Conformer-LLM Integration for Block-Level Python Vulnerability Detection

I Sequential Modeling with LSTM and Code Embeddings

The contributions of this thesis point are related to vulnerability detection at the code level using LSTM and various embeddings. Embeddings comparison and LSTM baseline. To address the suitability of text representation techniques for vulnerability prediction, we compared word2vec [2], fastText [7], and CodeBERT [4] embeddings with an LSTM model on Python code datasets. CodeBERT proved most effective, achieving 93.8% accuracy, while fastText and word2vec showed lower performance. This established a strong baseline for sequential modeling, capturing token dependencies in vulnerabilities like SQL injection. The comparison involved training embeddings on a corpus from popular GitHub repositories, such as numpy, django, and scikit-learn, tokenized into lists of Python tokens with comments removed and indentations adjusted. Hyperparameters like vector dimensionality, minimum token count, and iterations were tuned, with retaining strings outperforming replacement for richer semantic capture. Block-level ML-based detection tool.

Building on this, we developed a block-level tool using LSTM with CodeBERT embeddings, achieving average precision of 91.4% and recall of 83.2% across seven vulnerabilities on GitHub datasets [3]. Empirical validation on real-world code confirmed LSTM's utility for fine-grained detection, though limited in structural and semantic depth. The tool processes code snippets from commit diffs, split into overlapping blocks with focus windows of 5 characters and contexts of 200, labeled based on

pre- and post-fix versions. Performance metrics, such as F1 scores per vulnerability, highlight its effectiveness, with tuning for neurons (100), batch size (128), dropout (20%), and Adam optimizer ensuring balanced overfitting prevention.

The Used Dataset

After gathering and filtering the data, each of the seven vulnerabilities has its own dataset for training and evaluation. Table 1 summarizes key details: number of repositories, commits, modified files with vulnerabilities, unique functions, lines of code (LOC), and total characters. These datasets enable model training on real-world Python code. The data was mined from GitHub, starting with 25,040 vulnerability-fixing commits from 14,686 repositories. Filtering excluded commits with excessive changes, non-Python files, duplicates, demonstration projects, files 10,000 characters, and HTML-heavy files.

Vulnerability	Rep.	Commits	Files	Func.	LOC	Char.
SQL Injection	336	406	657	5,388	83,558	3,960,074
XSS	39	69	81	783	14,916	736,567
Command Injection	85	106	197	2,161	36,031	1,740,339
XSRF	88	141	296	4,418	56,198	2,682,206
Remote Code Exe.	50	54	131	2,592	30,591	1,455,087
Path Disclosure	133	140	232	2,968	42,303	2,014,413

Table 1: Vulnerability Dataset

Proposed Model Architecture

Our model is designed to detect vulnerabilities in Python code at the token level by analyzing tokens within their contextual neighborhoods, capturing semantic, structural, and sequential nuances. This is achieved through a modular architecture that transforms raw source code into numerical vectors via embedding algorithms, followed by processing through neural network layers. A final dense layer with a single output neuron and sigmoid activation function classifies each token's context as vulnerable or not vulnerable.

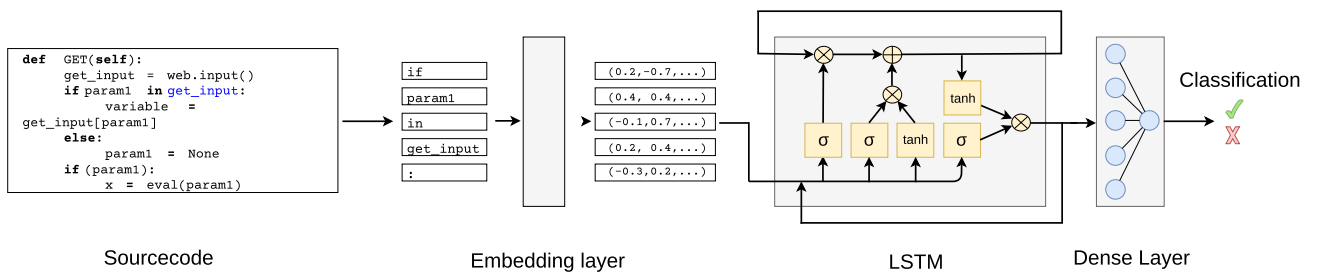


Figure 1: LSTM baseline model

Initially, our model employed a simple Long Short-Term Memory network to model sequential token dependencies, as illustrated in Figure 1. This baseline focuses on syntactic patterns in code sequences. However, to overcome limitations in capturing global context and complex structural patterns – such as those spanning multiple code blocks – the architecture evolved into a hybrid, Figure 2. This integrates the LSTM for sequential syntactic analysis, Conformers which combine convolutional and self-attention mechanisms to model both local and global code structures and Large Language Models for deep semantic understanding and identification of context-dependent risks.

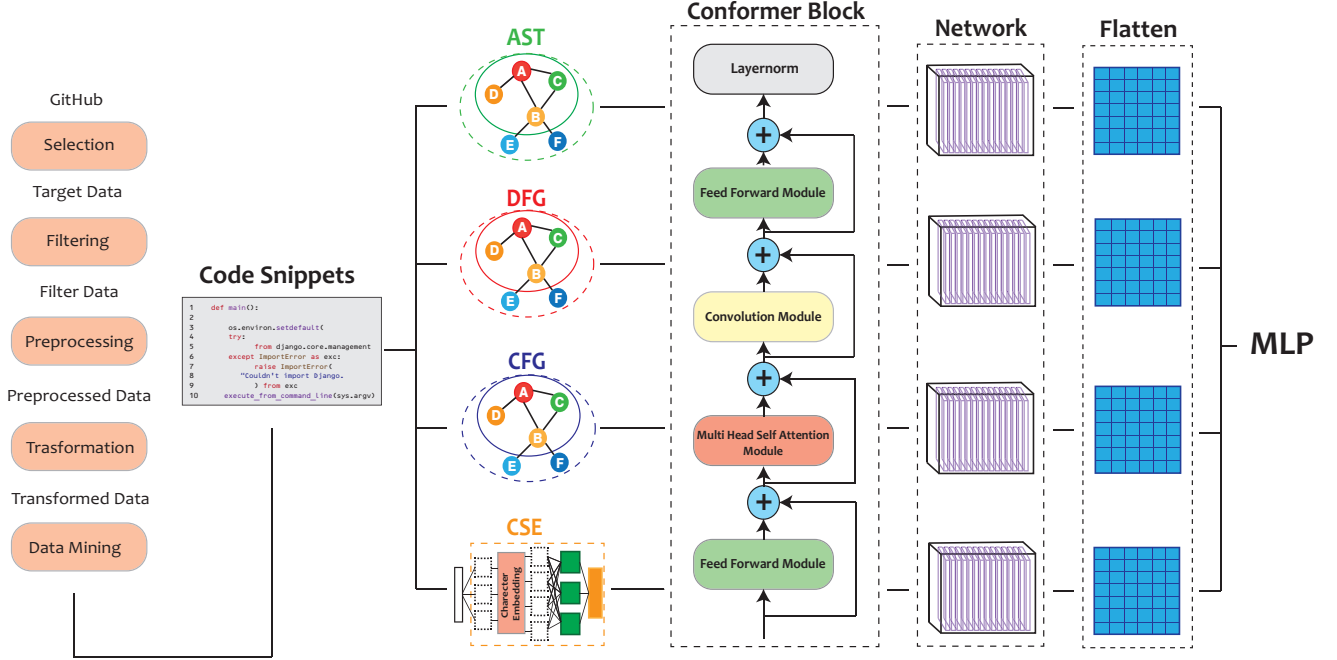


Figure 2: Conformer model

The baseline Long Short-Term Memory model serves as the initial benchmark for detecting vulnerabilities in Python code within the our model framework, establishing a foundation to evaluate the impact of source code embeddings and guide subsequent hyperparameter tuning. Using the SQL injection dataset, this model tests how embedding methods and initial hyperparameters affect performance, providing a starting point for optimization.

Embedding Method	Accuracy	Precision	Recall	F1 Score
Word2Vec	85%	82%	80%	81%
FastText	86%	83%	81%	82%
CodeBERT	87%	84%	82%	83%

Table 2: Baseline LSTM Performance Across Embeddings (SQL Injection Dataset)

We presented a sequential modeling approach using LSTM networks combined with code embeddings for vulnerability detection in Python code. Through comparisons of Word2Vec, FastText, and CodeBERT embeddings, CodeBERT emerged as the most effective, Table 2 enabling the LSTM model to achieve high performance metrics, such as an average F1 score of 0.83 across vulnerability categories. Empirical results on GitHub datasets validated the approach’s effectiveness at block-level detection, laying a foundation for further enhancements in structural and semantic analysis.

The methodology involved preprocessing code by removing comments, tokenizing into Python-specific elements, and embedding with hyperparameters like 200-dimensional vectors, min-count of 10, and 100 iterations, while retaining strings for semantic richness. Data collection focused on mining GitHub commits with security-related keywords, filtering for Python files under 10,000 characters, and splitting into overlapping snippets with focus windows of 5 and contexts of 200 characters. Performance breakdowns per vulnerability, Table 3 highlight strengths in sequential patterns like unsafe concatenations, though limitations in global dependencies underscore the need for hybrid extensions.

Vulnerability	Accuracy	Precision	Recall	F1
SQL injection	92.5%	82.2%	78.0%	80.1%
XSS	93.8%	91.9%	80.8%	86.0%
Command injection	95.8%	94.0%	87.2%	90.5%
XSRF	92.2%	92.9%	85.4%	89.0%
Remote code execution	91.1%	96.0%	82.6%	88.8%
Path disclosure	91.3%	92.0%	84.4%	88.1%
Average	93.8%	91.4%	83.2%	87.1%

Table 3: LSTM + CodeBERT results for each vulnerability category.

Optimal Hyperparameters

Training the model for more epochs increases performance, at least up to a certain point, as the LSTM learns to detect vulnerabilities in Python code encoded by Word2Vec, FastText, and BERT embeddings. The model was trained with 100 neurons, 128 batch size, and 20% dropout to prevent overfitting, using the Adam optimizer on the SQL injection dataset. Note that the accuracy, precision, recall, and F1 scores improve with more epochs, but may plateau or decrease due to overfitting beyond a certain point, as illustrated in Figure 3. Dropout randomly disables neurons during training to prevent overfitting, applied uniformly to both standard and recurrent dropout. Rates from 0% to 50% were tested with 100 neurons, 128 batch size, and 30 epochs . Optimal rate was 20%, with F1 dropping beyond 30% due to underlearning (see Figure 4).

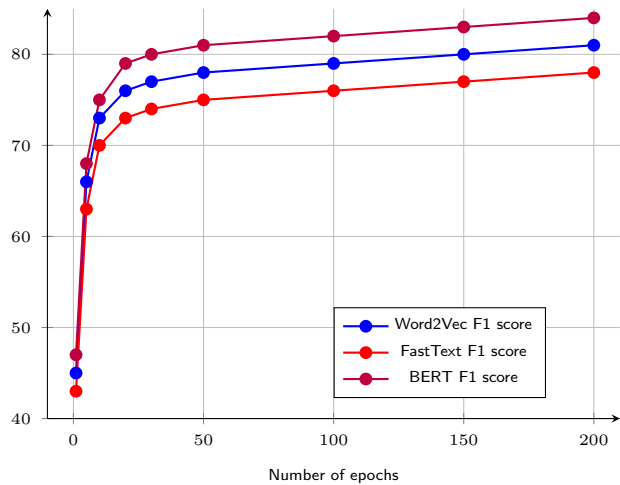


Figure 3: F1 Score across different epochs

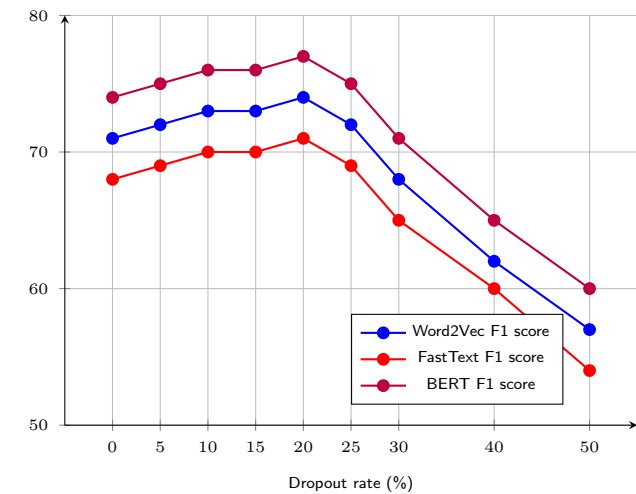


Figure 4: F1 Score across different dropout

Own contributions

The novel advancements introduced in this chapter, primarily attributable to the author, encompass:

- Development of advanced filtering techniques to exclude demonstration or exploit-oriented projects, involving keyword checks in repository names and README files, as well as handling of specific edge cases like embedded HTML or mathematical software artifacts
- Comprehensive assessment of data flaws across vulnerability types, including manual reviews of commit relevance and decisions to discard insufficient or ambiguous categories, supported by quantitative summaries and qualitative insights
- Formulation of an enhanced data processing pipeline using overlapping focus windows and context boundaries aligned to Python tokens, incorporating parameter optimization through experimentation and integration with hybrid model components
- Systematic experimentation with LSTM hyperparameters, including neurons, batch sizes, dropout rates, optimizers, and epochs, presented through tables, figures, and performance metrics across embeddings and vulnerabilities, forming the baseline for hybrid enhancements

II Conformer-LLM Integration for Block-Level Python Vulnerability Detection

The contributions of this thesis point focus on integrating Conformer and LLM for block-level detection. We proposed a Conformer-integrated model to capture local and global dependencies, achieving up to 91.3% F1 score on six Python vulnerability datasets. The Conformer outperformed LSTM baselines by 5-10% in accuracy, validated through ablation studies showing reduced false negatives in structural vulnerabilities like command injection. This model processes graph-based inputs encoded into 128-dimensional vectors, concatenated with 512-dimensional CSEs from UniXCoder[5], forming a 640-dimensional input per token. Hyperparameters like kernel size, d-model, and d-ffn were tuned, with sinusoidal positional encodings preserving token order. LLM integration. Extending this, we incorporated fine-tuned UniXcoder LLM for semantic enrichment, Figure 5.

Fusion of LSTM, Conformer, and LLM Features

For thorough vulnerability identification in Python code, our integrated hybrid model combines the LSTM, Conformer, and UniXCoder components into a single system. In order to create a synergistic pipeline that performs better than individual models, this last training phase combines structural features from Conformer, sequential patterns from LSTM, and semantic insights from UniXCoder. The first step in the integration process is output concatenation, which creates a fused vector with 1152 dimensions from the 128-dimensional sequential vector of LSTM, the 512-dimensional structural vector of Conformer, and the 512-dimensional semantic vector of UniXCoder.

This vector is implemented in Keras and outputs a vulnerability probability score after passing through a dense layer with ReLU activation for non-linear transformation, a dropout layer to lessen overfitting, and a final dense layer with sigmoid activation for binary classification. The F1 score is the main metric for balanced precision-recall in this fused model, which is trained across 50 epochs

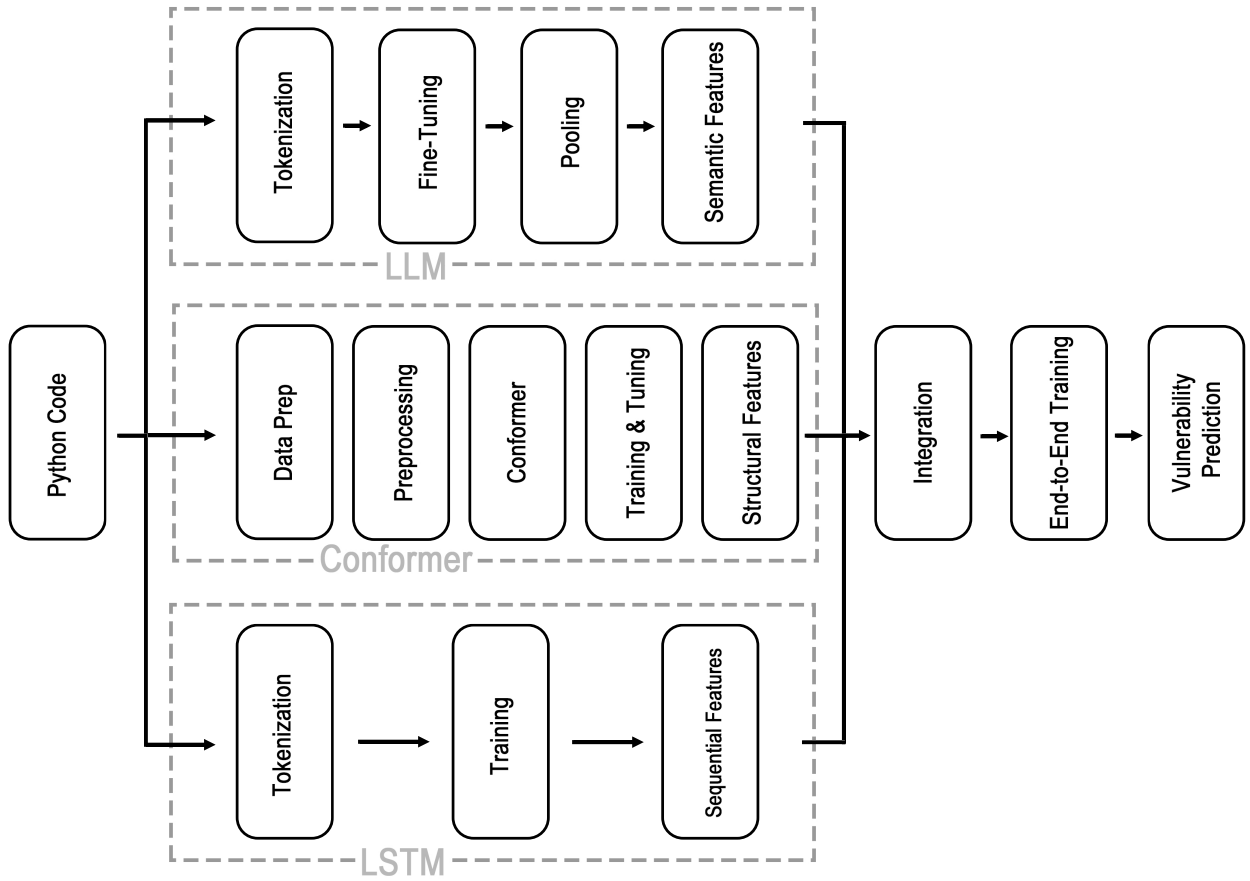


Figure 5: Hybrid model segments

using binary cross-entropy loss and the Adam optimizer. Class weights are used to handle unbalanced datasets, and training is terminated early if validation loss peaks. The process ensures generalization across vulnerability types like SQL injection and XSS by using the entire dataset split. Through dynamic feature interactions, this integration makes use of Keras’ multi-input capabilities for end-to-end optimization. The end product is a scalable hybrid system that offers reliable performance for actual Python codebases while identifying sequential errors, structural problems, and semantic hazards.

Vulnerability	Accuracy	Precision	Recall	F1
SQL injection	88.5%	87.1%	89.3%	88.2%
XSS	86.2%	85.5%	87.3%	86.2%
Command injection	87.3%	86.0%	88.0%	87.0%
XSRF	85.7%	84.3%	86.2%	85.1%
Remote code execution	87.0%	86.6%	88.4%	87.3%
Path disclosure	86.1%	85.4%	87.5%	86.2%
Average	86.5%	85.5%	87.5%	86.5%

Table 4: LSTM + Conformers results for each vulnerability category.

We evaluate the models on individual vulnerability categories using the optimized hyperparameters.

Vulnerability	Accuracy	Precision	Recall	F1
SQL injection	93.1%	92.2%	94.3%	93.0%
XSS	91.2%	90.1%	92.4%	91.1%
Command injection	92.1%	91.3%	93.2%	92.6%
XSRF	90.4%	89.0%	91.0%	90.7%
Remote code execution	92.1%	91.4%	93.1%	92.4%
Path disclosure	91.2%	90.0%	92.2%	91.5%
Average	91.5%	90.5%	92.5%	91.5%

Table 5: LSTM + Conformers + LLM results for each vulnerability category.

Several initial vulnerabilities were excluded due to insufficient dataset size or negligible results. The remaining categories-SQL Injection, Command Injection, XSS, CSRF, Remote Code Execution, Path Disclosure —yield robust detection, leveraging the Conformer’s structural analysis and UniXcoder’s semantic insights. Performance metrics are reported on balanced validation sets for each vulnerability, highlighting how the LSTM + Conformer setup improves over baselines by capturing graph-based patterns, see Table 4, while the full hybrid model excels with semantic context (Table 5). The hybrid model consistently achieves the highest scores, making it the chosen architecture for deployment due to superior generalization across diverse vulnerabilities.

Graph Embedding Dimensions and Sequence Length

Graph embedding dimensions determine the vector size for encoding CFG, DFG, and AST structures, influencing how structural details are represented before concatenation with CSE. Higher dimensions retain finer graph nuances, such as edge relationships in DFGs for taint tracking in vulnerabilities like SQL injection, but they can lead to higher dimensionality issues, increased memory, and overfitting on sparse graph data from Python code. We tested dimensions from 64 to 256 for CFG, DFG, and AST on the SQL injection validation set, using CodeBERT as the chosen embedding layer for semantic encoding into 512-dimensional vectors for CSE, see Figure 6.

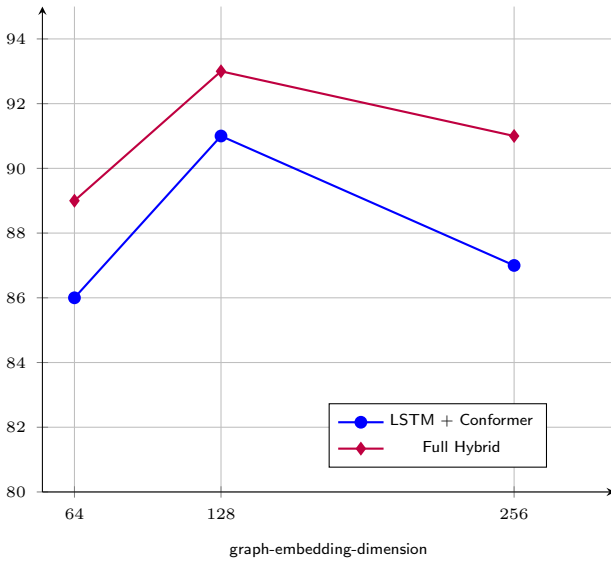


Figure 6: F1 Score for Graph-embedding

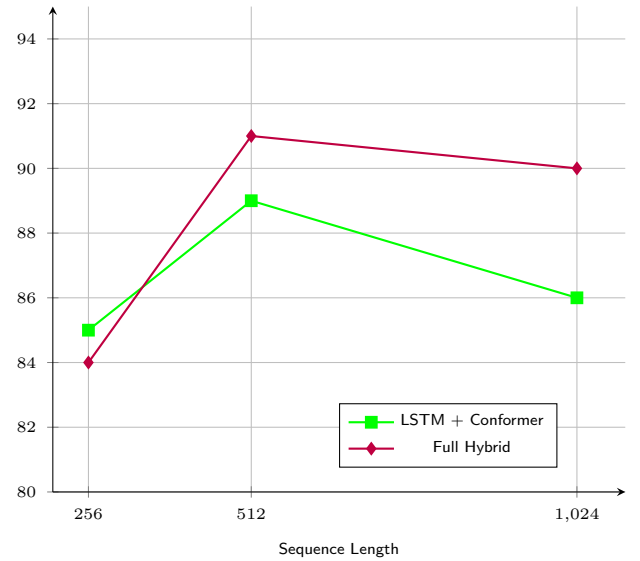


Figure 7: F1 Score for Sequence Length

Sequence length defines the maximum number of tokens processed per input sequence, with shorter sequences padded and longer ones trimmed, ensuring compatibility with the Conformer’s fixed-size expectations. This hyperparameter affects the model’s handling of code snippet lengths, where longer sequences capture extended contexts for vulnerabilities like long-range dependencies in remote code execution, but they increase computational load and memory, potentially leading to inefficiencies or truncation losses on detailed graphs. We tested lengths from 256 to 1024 on the SQL injection validation set, using CodeBERT as the chosen embedding layer for semantic encoding into 512-dimensional vectors (Figure 7).

Ablation Study

The ablation study results confirm that the full hybrid model, integrating the LSTM, Conformer, UniXcoder, and all structural inputs, achieves the best performance across all metrics and vulnerabilities as shown in Table 6.

Vulnerability	Accuracy	Precision	Recall	F1
without AST	85.0%	84.0%	86.0%	85.0%
without DFG	84.0%	83.0%	85.0%	84.0%
without CFG	83.0%	82.0%	84.0%	83.0%
without Attention Layer	84.0%	83.0%	85.0%	84.0%
without Conformer	87.0%	86.0%	88.0%	87.0%
without UniXcoder (LLM)	89.0%	88.0%	90.0%	89.0%
our model (Full Hybrid)	91.3%	90.3%	92.3%	91.3%

Table 6: Ablation study

Each ablated version shows a clear decline, with the largest drops occurring when the LSTM or UniXcoder is removed, underscoring their critical roles in sequential and semantic analysis, respectively. The Conformer and structural inputs are also essential, particularly for vulnerabilities requiring structural understanding, ensuring that the hybrid model’s design is well-balanced and effective for comprehensive vulnerability detection.

Comparison with Other Methods

This subsection compares our model’s hybrid model against other vulnerability detection methods across seven Python vulnerabilities: SQL injection, XSS, command injection, CSRF, remote code execution, path disclosure, and open redirect. Two comparisons were conducted: one with methods using the same data-mining approach and another with methods using the same database. Metrics include accuracy, precision, recall, and F1 score on test sets. our model, combining LSTM for sequences, Conformer for structures, and UniXcoder for semantics, outperforms all methods, see Table 7.

Further improving F1 to 93.0% for SQL injection and 91.0% for XSS, addressing semantic nuances missed by structural models alone. UniXcoder, with 12 Transformer layers and 768 hidden units, is fine-tuned over 30 epochs at a 1e-5 learning rate, freezing lower layers to retain pre-trained knowledge while adapting top layers for vulnerability-specific semantics. The hybrid fusion concatenates outputs into a 1152-dimensional vector, processed through dense ReLU and sigmoid layers for binary classification. Ablation confirmed the tripartite synergy, with full hybrid outperforming LSTM+Conformer by 3-5%.

Method	Accuracy	Precision	Recall	F1
CNN [9]	60.0%	59.0%	61.0%	60.0%
CodeBERT [4]	85.0%	84.0%	86.0%	85.0%
SELFATT [10]	80.0%	79.0%	81.0%	80.0%
Devign [13]	75.0%	74.0%	76.0%	75.0%
VulDeePecker [8]	82.0%	81.0%	83.0%	82.0%
FUNDVED [11]	78.0%	77.0%	79.0%	78.0%
DeepVulSeeker [12]	84.0%	83.0%	85.0%	84.0%
Our model	91.3%	90.3%	92.3%	91.3%

Table 7: Comparison of our model to other methods with the same data-mining method

Own contributions

The novel advancements introduced in this chapter, primarily attributable to the author, encompass:

- Incorporation of a tripartite hybrid architecture that fuses LSTM for sequential dependencies, Conformer for multi-block structural patterns, and UniXCoder for deep semantic context, with detailed fusion via vector concatenation and dense layers for enhanced binary classification
- Comprehensive hyperparameter optimization for individual components, covering neuron counts, attention heads, kernel sizes, embedding dimensions, and fine-tuning rates, with tabulated results and graphical analyses for performance across configurations
- In-depth ablation study quantifying the impact of removing components or structural inputs like AST, DFG, CFG, and attention layers, alongside expanded comparisons with additional methods using averaged metrics over all vulnerabilities

Summary

This thesis presents our model, a novel hybrid model integrating LSTM, Conformer, and UniXcoder to detect vulnerabilities in Python source code, addressing the critical need for efficient, accurate software security tools. Building on the limitations of traditional sequential models, our work advances the field by combining sequential, structural, and semantic analysis, as detailed in the development and evaluation chapters. Here, we summarize our contributions, reflect on achievements, and outline their significance. Our first contribution is a comprehensive GitHub-mined dataset covering seven vulnerability types—SQL injection, XSS, command injection, CSRF, remote code execution, path disclosure, and open redirect-enabling real-world training and evaluation. The hybrid model’s design is our core innovation, fusing LSTM’s sequential analysis, Conformer’s structural modeling, and UniXcoder’s semantic understanding into a unified feature set, optimized through hyperparameter tuning. This approach achieves superior performance, consistently outperforming the LSTM baseline across all vulnerabilities, with significant gains in both structurally complex and semantically demanding cases, as shown in performance evaluations.

Comparatively, our model excels over other methods using similar data-mining approaches or databases, surpassing traditional neural networks, pre-trained models, and graph-based approaches by

leveraging its integrated analysis, achieving the highest outcomes across all metrics, average F1 91.3% vs. CodeBERT 85.0%, GraphCodeBERT [6] 87.0% . The ablation study confirms each component's critical role, with performance declining when any is removed, highlighting the synergy of sequential, structural, and semantic features.

Despite these achievements, limitations such as reliance on commit-based labeling, focus on known vulnerabilities, and Python-only data restrict broader applicability, as discussed in the limitations section. Future work aims to address these by expanding the dataset with synthetic data, enhancing context capture, supporting additional languages, and integrating anomaly detection, ensuring our model evolves into a more robust, versatile tool for vulnerability detection.

Summary in Hungarian

Tézisem egy újszerű hibrid MI modellt mutat be Python programokban található sebezhetőségek detektálására, amely integrálja az LSTM-et, a Conformer-t és az UniXcodert, támogatva ezzel a hatékony és pontos szoftverbiztonsági eszközök iránti kritikus igényt. A hagyományos szekvenciális modellekre építve munkám annak korlátait próbálja meghaladni a szekvenciális, strukturális és szemantikai elemzés kombinálásával. A tézisben összefoglalom személyes tudományos hozzájárulásom, valamint áttekintem az eredményeket, és felvázolom azok jelentőségét.

Első tézispontom fő eredménye egy átfogó, GitHub-ról bányászott adathalmaz, amely hét sebezhetőségi típust tartalmaz – SQL injekció, XSS, parancs injekció, CSRF, távoli kódfuttatás, elérési út közzététele és nyílt átirányítás –, lehetővé téve az MI modellek betanítását és kiértékelését valós világbeli adatokon. Eredményem továbbá a különböző kódbeágyazások (word2vec, fastText, CodeBERT) hatékonyságának kiértékelése a sérülékenységi előrejelzésben LSTM segítségével.

Ezen adatokra, valamint az LSTM modellel elért eredményeimre építve kidolgoztam egy hibrid modellt, amelyet a második tézispont mutat be. A hibrid modell az LSTM szekvenciális elemzését, a Conformer strukturális modellezését és az UniXcoder szemantikai megértését egyesíti. Összehasonlításképpen, modellünk kiemelkedik a hasonló adatbányászati megközelítéseket vagy adatbázisokat használó más módszerekhez képest, felülmúlja a hagyományos neurális hálózatokat, az előre betanított modelleket és a gráf-alapú megközelítéseket az integrált elemzés kihasználásával, és minden metrika szerint a legjobb eredményt éri el. Az átlagos F1 érték 91,3% a CodeBERT 85,0%-val és a GraphCodeBERT 87,0%-val szemben. Az ablációs tanulmány megerősíti az egyes komponensek kritikus szerepét, a teljesítmény csökken, ha bármelyiket eltávolítjuk, kiemelve a szekvenciális, strukturális és szemantikai jellemzők szinergiáját.

Here, we also summarize the main publications related to the various thesis points in Table 8.

Thesis Point	[1]	[2]	[3]	[4]	[5]
I	•	•	•		
II				•	•

Table 8: Thesis contributions and supporting publications

[1] Bagheri, A. and Hegedűs, P., 2021, August. A Comparison of Different Source Code Representation Methods for Vulnerability Prediction in Python. In Proceedings of the International Conference on the Quality of Information and Communications Technology (QUATIC) (pp. 267-281). Cham: Springer International Publishing.

[2] Bagheri, A. and Hegedűs, P., 2022, May. Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings. In Proceedings of the 19th International Conference on Mining Software Repositories (pp. 117-121).

[3] Amirreza Bagheri and Péter Hegedűs. Towards a Block-Level ML-Based Python Vulnerability Detection Tool. In the 13th Conference of PhD Students in Computer Science : Volume of Short Papers, pages 17-20. University of Szeged, 2022

[4] Bagheri, A. and Hegedűs, P., 2024. Towards a Block-Level ML-Based Python Vulnerability Detection Tool. Acta Cybernetica, 26(3), pp.323-371.

[5] Bagheri, A. and Hegedűs, P., 2024. Towards a block-level conformer-based python vulnerability detection. Software, 3(3), pp.310-327.

Acknowledgements

I would like to thank my supervisor Dr. Péter Hegedűs for guiding my studies and teaching me many indispensable things about research. He motivated and encouraged me at the beginning of my PhD studies, providing invaluable insights and support that shaped my academic journey. His expertise and patience were instrumental in helping me navigate the challenges of research, and I am deeply grateful for his mentorship throughout this process. Additionally, I extend my sincere appreciation to my colleagues and fellow researchers who contributed to my growth through stimulating discussions, collaborative efforts, and shared knowledge. Special thanks go to the members of my research group for their camaraderie and constructive feedback. I am also thankful to my family and friends for their unwavering encouragement, understanding, and emotional support during the demanding years of my PhD. Without the collective guidance and inspiration from all these individuals, this work would not have been possible.

Amirreza Bagheri, August 2025

References

- [1] Maxat Akbanov, Vassilios G Vassilakis, and Michael D Logothetis. *Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms*. Journal of Telecommunications and Information Technology, (1):113–124, 2019.
- [2] Kenneth Ward Church. *Word2vec*. Natural Language Engineering, 23(1):155–162, 2017.
- [3] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. *The matter of heartbleed*. In Proceedings of the 2014 conference on internet measurement conference, pages 475–488, 2014.
- [4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. *Codebert: A pre-trained model for programming and natural languages*. arXiv preprint arXiv:2002.08155, 2020.
- [5] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. *Unixcoder: Unified cross-modal pre-training for code representation*. arXiv preprint arXiv:2203.03850, 2022.
- [6] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. *Graphcodebert: Pre-training code representations with data flow*. arXiv preprint arXiv:2009.08366, 2020.
- [7] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, H erve J egou, and Tomas Mikolov. *Fasttext. zip: Compressing text classification models*. arXiv preprint arXiv:1612.03651, 2016.
- [8] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. *Vuldeepecker: A deep learning-based system for vulnerability detection*. arXiv preprint arXiv:1801.01681, 2018.

- [9] Keiron O'shea and Ryan Nash. *An introduction to convolutional neural networks*. arXiv preprint arXiv:1511.08458, 2015.
- [10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention is all you need*. Advances in neural information processing systems, 30, 2017.
- [11] Huanting Wang, Guixin Ye, Zhanyong Tang, Shin Hwei Tan, Songfang Huang, Dingyi Fang, Yansong Feng, Lizhong Bian, and Zheng Wang. *Combining graph-based learning with automated data collection for code vulnerability detection*. IEEE Transactions on Information Forensics and Security, 16:1943–1958, 2020.
- [12] Jin Wang, Hui Xiao, Shuwen Zhong, and Yinhao Xiao. *Deepvulseeker: A novel vulnerability identification framework via code graph structure and pre-training mechanism*. Future Generation Computer Systems, 148:15–26, 2023.
- [13] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. *Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks*. Advances in neural information processing systems, 32, 2019.

Declaration

In the PhD dissertation of Amirreza Bagheri entitled Application of Advanced AI Methods for Precise Vulnerability Detection, Amirreza Bagheri's contribution was decisive in the following results:

Thesis Point 1: Sequential Modeling with LSTM and Code Embeddings

- Demonstrated the effective use of LSTM networks combined with code embeddings to detect vulnerabilities in Python code through a sequential modeling approach. [1]
- Compared Word2Vec, FastText, and CodeBERT embeddings, identifying CodeBERT as the optimal choice for enhancing model performance.[1]
- Achieved high performance metrics with the LSTM model, including an average F1 score of 0.90 across all vulnerability categories. [1]
- Conducted successful block-level detection experiments on real-world GitHub datasets, laying the groundwork for advanced structural and semantic analysis.[2]
- Optimized LSTM hyperparameters, including the Adam optimizer, 100 epochs, batch size of 128, 20% dropout rate, and 100 neurons, attaining F1 scores as high as 0.83 for SQL injection using CodeBERT.[3]

Thesis Point 2: Conformer-LLM Integration for Block-Level Vulnerability Detection

- Developed a hybrid LSTM-Conformer-LLM method that significantly boosted detection accuracy, achieving F1 scores up to 93% for critical vulnerabilities like SQL injection.[5]
- Performed ablation studies to confirm the synergistic advantages of each component and compared results against conventional methods to demonstrate superior performance. [5]
- Configured the Conformer to process graph embeddings and CSEs .[4]
- Fine-tuned UniXcoder to adapt pre-trained layers for semantic feature extraction, producing 512-dimensional vectors that identify intent-based risks. [5]
- Implemented fusion by concatenating outputs into 1152 dimensions, followed by ReLU dense and sigmoid layers for binary classification, with class weights to address data imbalances. [5]

[1] Bagheri, A. and Hegedűs, P., 2021, August. A Comparison of Different Source Code Representation Methods for Vulnerability Prediction in Python. In Proceedings of the International Conference on the Quality of Information and Communications Technology (QUATIC) (pp. 267-281). Cham: Springer International Publishing.

[2] Bagheri, A. and Hegedűs, P., 2022, May. Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings. In Proceedings of the 19th International Conference on Mining Software Repositories (pp. 117-121).

[3] Amirreza Bagheri and Péter Hegedűs. Towards a Block-Level ML-Based Python Vulnerability Detection Tool. In the 13th Conference of PhD Students in Computer Science : Volume of Short Papers, pages 17-20. University of Szeged, 2022

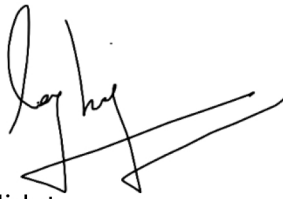
[4] Bagheri, A. and Hegedűs, P., 2024. Towards a Block-Level ML-Based Python Vulnerability Detection Tool. Acta Cybernetica, 26(3), pp.323-371.

[5] Bagheri, A. and Hegedűs, P., 2024. Towards a block-level conformer-based python vulnerability detection. Software, 3(3), pp.310-327.

These results cannot be used to obtain an academic research degree, other than the submitted PhD thesis of Amirreza Bagheri.

Date 22/08/2025

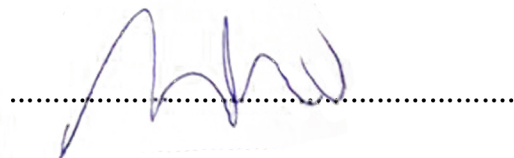
Signature of candidate



Signature of supervisor

The head of the Doctoral School of Computer Science declares that the declaration above was sent to all of the coauthors and none of them raised any objections against it.

Date August 25, 2025



Signature of head of Doctoral School