Firm-Vehicle: Trusted Communication Enabled Instruction Embedding Model for Resource-Constrained VANET Environments

Rui Chen(✉), Waleed Younas, and Jing Zhao

School of Software Technology, Dalian University of Technology, Dalian 116024, Liaoning, China
chenrui_dut@163.com, 12017026@mail.dlut.edu.cn, zhaoj9988@dlut.edu.cn

Abstract. Instruction embedding is an essential technique in firmware security research and serves as a fundamental component in many vulnerability detection and security analysis methods for VANET. Instruction embedding maps the semantic information of instructions into fixed-dimensional vectors. However, the current instruction embedding approaches often neglect the operational requirements of resource-constrained environments typical of VANET. Moreover, the dependency on third-party disassembly tools for the extraction of instruction call graphs presents challenges related to runtime environment, thereby complicating the feasible application of instruction embedding techniques. To address these limitations, we propose a novel cross-architecture firmware instruction embedding model called Firm-Vehicle, specifically tailored for resource-constrained environments in VANET. Primarily, we devise a lightweight algorithm for extracting instruction call graphs, eliminating the reliance on third-party disassembly tools and improving the efficiency of call graphs extraction, thereby enhancing the instruction embedding model. Through evaluations and comparison with other approaches, Firm-Vehicle not only reduces the required time for instruction call graphs extraction but also enhances the stability of the instruction embedding model, enabling secure and efficient operation within the VANET environment.

Keywords: Instruction Embedding · VANET · Resource Constrained · Model Compression · Model Distillation

1 Introduction

The Vehicular Ad-hoc Networks (VANET) is a heterogeneous network vehicle-side and roadside unit (RSU) networks, and human mobile communication networks. It leverages real-time interconnection and perception among vehicles, roads, and the cloud to enhance vehicle safety and intelligence on the road, while providing various information services

W. Younas and J. Zhao—Contributing authors.

D.-S. Huang et al. (Eds.): ICIC 2024, LNCS 14869, pp. 391–402, 2024.
https://doi.org/10.1007/978-981-97-5603-2_32

to users, thereby improving the driving and driving experience. However, the security of VANET faces severe challenges. Security issues in the VANET not only result in financial losses for drivers but can also expose threats to the safety of others, even endangering lives [26]. Numerous security issues were identified in VANET devices, such as car door lock vulnerabilities, unrestricted access to vehicle cameras, and exposure of vehicle location privacy. Statistical data reveals that 80% of VANET devices adopt weak password algorithms, 70% of VANET communications are unencrypted, and 90% of VANET firmware have security vulnerabilities [1, 7].

Research on firmware vulnerability detection has become a top priority for VANET security [11, 14, 17, 21, 25]. Among them, instruction embedding is a key technique that maps instructions to fixed dimensional vectors. By designing and implementing fast and effective firmware instruction embedding models for VANET devices, it can help security researchers quickly discover security issues in VANET devices, and also help VANET manufacturers quickly locate code design defects. This will better promote the development of the VANET industry, and research on instruction embedding for VANET device firmware is of great significance.

However, current instruction embedding research primarily employs large models and deploys instruction embedding services on central servers, neglecting research into resource-constrained scenarios [14, 21, 25]. As a result, existing instruction embedding solutions fail to provide effective services for resource-constrained VANET environments, where firmware detection is urgently needed. At the same time, in current instruction embedding research, third-party disassembly tools are commonly used to generate instruction call graphs from disassembled code, in order to provide more semantic information about the instructions to the model [14, 25]. Moreover, using third-party instruction call graphs extraction tools raises issues of authorization and environment deployment. For example, IDA pro [8] is a powerful but expensive tool with high requirements for deployment environments, such as inability to deploy on Linux, which greatly limits the progress of instruction embedding related tasks.

Therefore, to enable efficient execution of instruction embedding models in the resource-constrained VANET environments, we proposes a novel cross-architecture instruction embedding model Firm-Vehicle tailored for VANET communication scenarios [20, 26]. Specially, we first devise a lightweight instruction call graph extraction algorithm based on jump instructions' context, avoiding authorization and deployment environment issues during model inference. Meanwhile, efficient deployment and execution of instruction embedding models in VANET faces some challenges. Then, we leverage model distillation techniques to compress the complex instruction embedding model that has been trained on a central server. The objective of this process is to efficiently transfer the rich information from the teacher model to the student model, thereby constructing a lightweight model that is suitable for deployment in resource-constrained VANET environments. This approach ensures minimal performance degradation while achieving effective compression of the model structure, and it meets the operational requirements within resource-limited VANETs. Finally, through the above designs and evaluations, our approach not only accelerates the instruction call graph generation process, but also enhances the stability of instruction embedding models for efficient execution in resource-constrained environments.

Above all, we are the major contribution of this work is as follows:

- We propose Firm-Vehicle, a novel cross-architecture firmware instruction embedding model specifically designed for resource-constrained VANET communication scenarios. Additionally, in order to eliminate reliance on third-party disassembly tools, we devise a lightweight instruction call graph extraction algorithm, that significantly speeds up the generation of instruction call graphs, thereby facilitating the creation of high-quality instruction embeddings.
- We propose a model compression approach based on knowledge distillation to streamline complex instruction embedding models through model distillation, reducing them to lightweight models suitable for deployment in resource-constrained VANET environments, thereby ensuring the effective operation of the embedding models within the VANETs.
- We evaluate the performance of our model by conducting downstream tasks related to command embedding and compare it against existing approaches in terms of security and performance. Experimental results demonstrate that Firm-Vehicle not only accelerates the generation of command invocation relationships in resource-constrained VANET environments but also enhances the stability of the command embedding model, enabling safe and efficient operation.

2 Related Work

Recent studies indicate that representational learning surpasses traditional heuristic-based methods in instruction embedding tasks like binary similarity detection [24, 25], function name prediction, and function boundary identification. This success hinges on learning vector representations, or embeddings, of assembly instructions at various granularities, individual instructions, basic blocks, or entire functions, which are crucial for applications such as instruction search and firmware vulnerability detection [14]. However, compressing these models to reduce storage needs has resulted in poor performance in embedding assembly instructions and limited usability in VANET environments [15, 18, 23].

Earlier research focused on developing and fine-tuning pre-trained models to generate such embeddings for both direct application and downstream tasks [4, 14, 25]. These methods are divided into unsupervised and supervised learning categories. Initially, many studies used unsupervised learning to generate binary code embeddings without needing labeled data, merely requiring the disassembly of binary files and the construction of control flow graphs. Examples include DeepBindiff [6] and Asm2Vec [5], which utilize Word2Vec techniques [16, 28].

Many modern software applications support multiple operating systems and instruction set architectures (ISAs), implementing the same functionalities but interacting with different OS interfaces and compiling binaries for various ISAs like ARMv7, ARMv8, and MIPS. This heterogeneity presents challenges for instruction embedding. Consequently, there is significant interest in platform-agnostic instruction embedding techniques [21, 25], aimed at applications such as vulnerability search and binary similarity detection. These techniques enable seamless transfer of analysis work across different platforms; for instance, a single error signature can be applied to binaries from various ISAs. Approaches vary from abstracting binaries into OS/ISA-agnostic assembly

instructions to using Siamese models like Gemini [24] and InnerEye [27], which learn direct similarities between binaries compiled for different platforms.

The Hinton model distillation technique mainly focuses on distilling the output of the teacher model's prediction layer [10]. However, engineers often only consider the input-output correlation of the teacher model, potentially leading to issues like overfitting due to neglect of the model's internal structure. This is particularly problematic with large Transformer-based models, where distilling only output predictions might not capture the extensive semantic and syntactic knowledge in the intermediate layers. To overcome this, researchers have introduced a hierarchical distillation method that aligns the hidden representations of the student and teacher models across all layers, facilitating a more comprehensive transfer of knowledge from intermediate layers [3].

Upon comparison with existing works, we have identified the need for an efficient instruction embedding model for resource-constrained VANETs. To address this, we devised an algorithm to generate required call graphs without third-party tools. We also employ model distillation to refine complex embedding models into a lightweight model tailored for VANETs, ensuring effectiveness under constraints. The model details will be elaborated upon.

3 Problem Formulation

Firmware security has become a crucial challenge in the VANET domain, with instruction embedding being a key research focus. Current instruction embedding models face two main limitations: heavy reliance on third-party decompilation tools for generating assembly instructions and their relationships, and designs that do not suit resource-constrained environments due to their size.

Addressing these issues, our study introduces a lightweight algorithm for extracting instruction call relationships and a novel instruction embedding model. Initially deployed on resource-rich roadside units, the model is adapted for VANETs through knowledge distillation. This technique transfers knowledge from complex "teacher" models to more manageable "student" models, enhancing their predictive capabilities and generalization. We utilize model distillation on central servers to effectively compress these models, optimizing them for the limited resources and high processing demands of VANETs, while striving to maintain performance.

4 Proposed Method

This section introduces the Firm-Vehicle instruction embedding model, based on the Hugging Face pre-trained BERT model (bert-base-uncased) with an integrated graph structure extraction algorithm, detailed in Fig. 1A. Knowledge distillation techniques transfer expertise from a large-scale to a lightweight model, ensuring efficient operation and accurate detection in resource-constrained VANET environments. Thus, the model meets VANETs' stringent real-time performance and resource efficiency requirements, effectively performing code similarity analysis tasks even with limited computational resources.

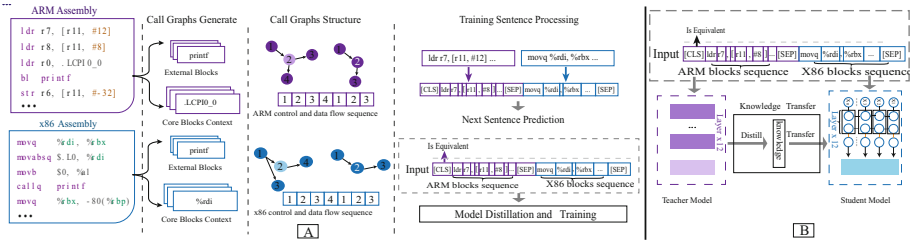

Fig. 1. A: The Firm-Vehicle system extracts call graphs between assembly instructions to train a language model using next sentence prediction, enabling cross-architecture instruction embeddings. B: For model compression, a lightweight Bi-LSTM student network is trained using outputs from a BERT-based teacher model through distillation technique.

4.1 Assembly Instruction Call Graphs Extraction Algorithm

In existing binary code similarity research, instruction call graphs extraction often relies on third-party disassembly tools. However, these third-party tools capture excessive information. In instruction call graph analysis, too many dependent call graphs are added, including the assembly code of auxiliary source programs. The increase of such useless information results in higher time costs for model training. On the other hand, with the presence of many irrelevant instructions, the model cannot accurately identify the semantics of actual executed assembly. Also, the use of third-party tools may involve fees and licenses, cannot be combined with model data processing due to runtime environment issues.

Therefore, in this section, we designed an assembly instruction call graphs extraction algorithm, mainly for analyzing assembly instruction call graphs information and reducing the complexity of instruction call graphs dependent on information. As shown in Algorithm 1, this algorithm can also remove pseudo instructions from compilers. Compared with third-party tools, the instruction call graphs extraction of this algorithm is not limited by environment and fees, and can be flexibly deployed on RSU devices to more effectively assist downstream tasks such as detecting firmware vulnerabilities in vehicles.

From Fig. 1A, in the processing after obtaining each line of assembly instruction, first skip empty lines and instructions with specific labels. In assembly, .Ltmp and .LPC labels do not indicate block semantics and should be ignored directly. For non-empty lines containing new labels, extract the label name and add it to a dictionary as the key, creating a corresponding empty list as the value. For instruction lines starting with a tab or space, remove leading and trailing whitespace based on architecture type after processing. For valid instructions, remove comments and add the instruction to the list for the corresponding label after processing. This allows classifying and storing assembly instructions by labels for convenient subsequent processing or analysis.

Algorithm 1 Vehicle Firm Assembly Instruction Call Graphs Extraction Algorithm

```
for line in assembly code do
    Ignore empty line and instruction containing .Ltmp and LPC
    if line does not start with a tab or space then
        label =get the part before ":" from line
        while blocks contains label do
            append "1" to label
        end while
        create an empty list in blocks with the key as label
    else if label is not empty then
        instruction = remove leading and trailing whitespaces from line
        if instruction does not start with "#" or "@" then
            if arch is 'x86' then
                remove the annotation from instruction using regular expression and
add it to blocks[label]
        else if arch is 'arm' then
                remove the annotation from instruction using regular expression and
add it to blocks[label]
            end if
        end if
    end if
end for
```

Due to the large number of vocabularies (e.g. constants, strings) in binary code that cannot be covered in the vocabulary, the OOV (out-of-vocabulary) problem often occurs, requiring processing of unlabeled words. For constant data, replace with [data] tag; for string data, replace with [str] tag; for jump addresses, replace with [addr] tag; for called function names, replace with [function] tag; replace the rest unknowns with [unk] tag. By substituting low-frequency vocabularies, the OOV problem encountered by NLP models in processing assembly language can be addressed.

To determine the contextual relationships in the call graph structure between different assembly blocks in ARM assembly, the ldr instruction is used to store and load memory, and LCPI0_0 contains the offset. We can find the assembly content starting with ldr LCPI through the regex $ldr\backslash sr0$, $\backslash s(\backslash .LCPI\backslash d*\_\backslash d*)bl\backslash sprintf$; to obtain the offset represented by LCPI. For example, if the offset of LCPI0_0 is found, we can extract the content contained in the asciz instruction pointed to by the offset, which represents the execution context order of the block. Similarly, for x86 assembly, we adopt the same method, only replacing the regex $movabsq\backslash s\$(\backslash .L\backslash d*)$, $\backslash \%rdi$; $movb\backslash \%al$; $callq\backslash sprintf$; for finding calls to other blocks with.

4.2 Model Distillation

From Fig. 1B, this paper denotes the output vector given by the teacher model $Model_t$ just before the Softmax function in the output layer as z_t, where $\widehat{y_t} = Softmax(z_t)$ represents the inferred probability distribution of input sample classes based on the model. Similarly, the output vector of the student model is denoted as z_s, with the predicted probability distribution being $\widehat{y_s}$. Therefore, the definition of the loss function

used during the training process of the student model deployed at node k, $Model_s^k$, is as follows.

$$L_s = \lambda L_{sl} + (1 - \lambda) L_{kd}$$
$$L_{sl} = H(max(y_s), y_r)$$
$$L_{kd} = \tau^2 KL(y_s, S_k(y_t))$$

The term L_{sl} represents the loss generated by the discrepancy between the predicted result $\widehat{y_s}$ of the student model during regular training and the actual result y_r, where $H()$ denotes the cross-entropy function. The term L_{kd} is the loss function resulting from the difference between the outputs of the teacher model and the student model, where τ is the distillation temperature hyperparameter, which should be adjusted and optimized according to training effectiveness, with $y_s = Softmax(z_s/\tau)$ and $y_t = Softmax(z_t/\tau)$; here, $KL()$ signifies the Kullback-Leibler divergence function, and $S_k()$ is the mapping function based on the prior traffic distribution at node k.

Firstly, the teacher model in this study adopts a BERT-based architecture with specific parameter settings: the number of layers (Layers) is 12, the number of heads in the multi-head attention mechanism (Head) is 12, and the dimension of the hidden layer (Hidden_dimension) is 128. During the training process of the Firm-Vehicle teacher model, we adopted fundamental training methods including the Masked Language Model (MLM). By applying the next sentence prediction task, the model is capable of recognizing connections between code blocks of different architectures; concurrently, the model's next instruction prediction works in conjunction with the Masked Language Model to aid in a deeper understanding of the contextual relationships between assembly instructions. Utilizing the Masked Language Model, the model can perform predictive inference on masked tokens during training, thereby more accurately grasping the semantic connotations of assembly instructions.

When building the student model, we use the last layer output of the teacher model to guide its training, effectively distilling the teacher's knowledge. This not only transfers the teacher's information to the student model but also improves its ability to generalize on new data. This distillation method maximizes the potential of the teacher model, allowing the student model to reach or come close to the teacher's performance in a more compact form. The student model, a 12-layer Bi-LSTM with a hidden layer size of 64, is fine-tuned for next sentence prediction by training on paired input code block sets, similar to the teacher model. These pairs, half from adjacent blocks within the same document and half from different documents, use the teacher's outputs as training labels to predict the sequence of the code blocks, enhancing the model's ability to generalize across different contexts.

5 Experimental Results and Analysis

In this section, we focus on evaluating different instruction embedding studies and communication schemes for model deployment environments. For this purpose, we designed and implemented a comprehensive evaluation framework to evaluate Firm-Vehicle and baseline methods. The evaluation can be divided into three categories: model metric evaluation, communication cost evaluation and communication security evaluation for

the model deployment environment. In the rest of this section, we first introduce our evaluation framework and experimental setup, then discuss the results.

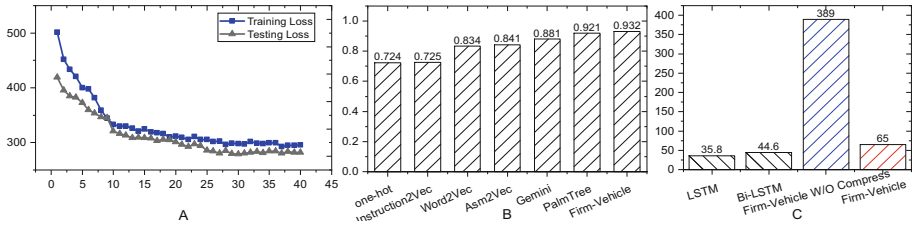

Fig. 2. A: Training and Test Loss. B: AUC values of Gemini. C: Storage space of trained model.

5.1 Experiment Setup

In this study, we evaluated three binary analysis tasks: Gemini [24] for binary code similarity detection, DeepVSA [9] for value set analysis, EKLAVYA [2] for function type signature inference, and PalmTree [14] for cross-architecture analysis through instruction embedding. We used the original implementations for experiments conducted in consistent datasets and environments. Additionally, we proposed a trusted communication scheme between vehicle-side and RSU units in VANET environments, to be verified using the NS3 [19] simulator.

We also assessed the effectiveness of our assembly instruction call graph extraction algorithm by comparing two versions of our Firm-Vehicle model: one with the call graph algorithm and one without (Firm-Vehicle-W/o-G).

For our dataset, we used the MIRROR paper's public dataset, which includes source code from five notable C/C++ open-source projects: Binutils 2.30, Coreutils 8.29, FFmpeg n3.2.13, OpenSSL 1.1.1b, and Redis 5.0.5. We compiled these into x86 and ARM assembly using the LLVM compiler.

Our experiments were performed on an Ubuntu 18.04 workstation with an Intel Xeon E5-2683 V4 CPU, two GeForce RTX 3090 GPUs, and 124GB memory. During the pretraining phase of Firm-Vehicle, we used 128,000 sample pairs from the ARM and x86 datasets and trained for 40 epochs on a sentence semantic equivalence task using the bert-base-uncased model from Hugging Face [4]. From Fig. 2A, after 40 epochs, training converged on the task of establishing equivalence between x86 and ARM basic blocks. We will use this trained model to assess performance in downstream tasks.

5.2 Metric Functions for Instruction Embedding

In the Firm-Vehicle network, the MEAN strategy calculates the embedding vector of binary functions using the last hidden layer of non-first instructions. To compare embedding vector similarities, four methods are utilized: cosine similarity, Manhattan distance, Euclidean distance, and dot product similarity. Cosine similarity, notably, measures the cosine of the angle between two vectors, focusing on directional differences rather than

magnitude or distance. The formula for cosine similarity is $u \cdot v/|u| \cdot |v|$, where u and v are vectors, $u \cdot v$ is their dot product, and $|u|$ and $|v|$ are their norms. Cosine similarity values range from -1 to 1, with values closer to 1 indicating higher similarity due to a smaller angle between vectors.

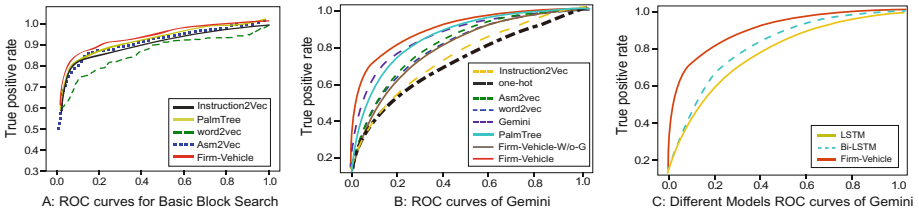

Fig. 3. Experiments on Downstream Task.

5.3 Performance Evaluation of Model on Downstream Tasks

Instruction Search and Similarity Evaluation. We conducted an experiment using a dataset comprising untrained x86 and ARM assembly instruction basic blocks, with each architecture representing 50% of the samples. For each basic block, embedding was computed by averaging the embeddings of the contained instructions. We then identified semantically equivalent basic blocks using cosine distance between their embeddings, as shown in Fig. 3A. The results, visualized through ROC curves, compared the effectiveness of various embedding techniques including Instruction2Vec, Word2vec, Asm2Vec, PalmTree, and our Firm-Vehicle model. The ROC curves revealed the ranking of AUC values among the techniques: Word2vec performed the poorest, Instruction2Vec outperformed Word2vec, Asm2Vec and PalmTree showed good performance but were surpassed by the Firm-Vehicle model, which demonstrated superior AUC, indicating a consistent performance improvement.

Additionally, we evaluated the Gemini model using these embeddings and included a one-hot vector approach as an additional baseline. Despite high AUC values reported in the original Gemini study, concerns about overfitting arose due to using training and testing sets from the same source. To assess model generalization more effectively, we compiled source code from multiple projects (Binutils, Coreutils, FFmpeg, OpenSSL, Redis) using LLVM to generate diverse x86 and ARM assembly datasets.

Figure 2B presents the AUC values of Gemini using different models to generate inputs. Based on the results, we can draw the following conclusions:

(1) Despite the superior performance reported in the original Gemini paper, we observe that the original Gemini [24] model does not generalize well on completely new testing data.
(2) Manually designed embeddings, Instruction2Vec [13], and one-hot vectors perform poorly, implying hand-picked features may only be suitable for specific tasks.
(3) Although tested on significantly different datasets from training, Firm-Vehicle still performs remarkably well and outperforms other schemes. This demonstrates Firm-Vehicle's ability to greatly enhance generalization for downstream tasks.

(4) All three pretraining tasks contribute to the final Firm-Vehicle model. However, Firm-Vehicle does not show obvious advantages over other baselines, meaning only Firm-Vehicle, through its algorithm generating assembly instruction relations, can perform better instruction embedding than previous methods on this downstream task. As shown in Fig. 3B, our Firm-Vehicle-W/o-G without the proposed relation generation algorithm sees a performance drop, on par with Gemini.

Table 1. Reliability Testing in Different Application Scenarios

Scenario	Effective range/m	Absolute Speed (km/m)	Relative Speed (km/h)	Maximum Latency (ms)
suburb	200	50	100	86
highway1	320	160	280	95
highway2	320	280	280	65
NLOS/Urban	100	50	100	96
Intersection	50	50	100	86
Compus and Business distribute	50	30	30	76

Performance Comparison of Instruction Embedding Models. To deploy instruction embedding models on resource-constrained vehicle-side devices, considering the advantages of LSTM over BERT in terms of memory footprint and faster convergence, we attempted using Bi-LSTM and LSTM to replace BERT for instruction embedding training [15, 18]. Similarly, we conducted instruction search tests and similarity detection of instruction embedding vectors on downstream tasks. As illustrated in Fig. 2C, we plotted the model storage requirements. We have observed that after knowledge distillation, the storage footprint of the Firm-Vehicle model has been significantly reduced, becoming nearly equivalent to that of the LSTM and Bi-LSTM models. In the following sections, we will further compare the performance of these three types of models in the context of instruction embedding.

In order to deploy instruction embedding models on resource-constrained VANET, and considering the advantages of LSTM over BERT in terms of memory footprint and faster convergence, we attempt to use Bi-LSTM and LSTM to replace BERT for training instruction embeddings. Likewise, we conducted instruction retrieval tests and similarity detection for instruction embedding vectors in downstream tasks. As depicted in Fig. 2C, we have illustrated the storage requirements of the models. Our findings suggest that although the proposed Firm-Vehicle model utilizes lightweight recurrent neural networks after distillation from the BERT model, it demonstrates significant performance benefits.

From Fig. 3C, our study compares the Firm-Vehicle with LSTM and Bi-LSTM models that have not undergone knowledge distillation, utilizing the ROC metric. The results indicate that the ROC performance of the Firm-Vehicle is superior to that of both LSTM and Bi-LSTM models, which is in line with our expectations.

Upon completion of the knowledge distillation process, the compressed Firm-Vehicle model was deployed to VANET. We conducted response time tests for the model under various scenarios. From Table 1, performance metrics for different application scenarios were obtained by deploying the Firm-Vehicle model in an embedded device environment, specifically on a Raspberry Pi 4B, using the NS3 simulator. The test results indicate that the latency of the trustworthy communication scheme proposed in this paper is approximately 70–100 ms, which meets the requirements of most application scenarios. Therefore, the instruction embedding model compression method based on knowledge distillation proposed in this study is capable of operating effectively within resource-constrained VANET environments.

6 Conclusion

We designed an algorithm to efficiently generate assembly call graphs required by the instruction embedding model without relying on third-party tools. Our approach accelerated call graph generation and improved model stability for efficient instruction embedding under limitations. Moreover, we utilized model compression techniques to distill the complex embedding model with instructions on the central server through model distillation, obtaining a lightweight model suitable for deployment in the resource-constrained VANET environment, effectively operate in the resource-limited VANET to meet the stringent requirements for computational resources and fast response capabilities in VANET.

References

1. Chen, H., Liu, J., Wang, J., et al.: Towards secure intra-vehicle communications in 5G advanced and beyond: vulnerabilities, attacks and countermeasures. Veh. Commun. **39**, 100548 (2023)
2. Chua, Z.L., Shen, S., Saxena, P., et al.: Neural nets can learn function type signatures from binaries. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 99–116 (2017)
3. Dalvi, F., Sajjad, H., Durrani, N., et al.: Analyzing redundancy in pretrained transformer models. http://arXiv.org/abs/2004.04010 (2020)
4. Devlin, J., Chang, M., Lee, K., et al.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805. http://arXiv.org/abs/1810.04805 (2018)
5. Ding, S.H., Fung, B.C., Charland, P.: Asm2vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 472–489. IEEE (2019)
6. Duan, Y., Li, X., Wang, J., et al.: DeepBinDiff: learning program-wide code representations for binary diffing. In: Network and Distributed System Security Symposium (2020)
7. Feng, X., Zhu, X., Han, Q.L., et al.: Detecting vulnerability on IoT device firmware: a survey. IEEE/CAA J. Automatica Sinica **10**(1), 25–41 (2022)
8. Ferguson, J.: Reverse engineering code with IDA Pro. Syngress (2008)
9. Guo, W., Mu, D., Xing, X., et al.: DEEPVSA: facilitating value-set analysis with deep learning for postmortem program analysis. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 1787–1804 (2019)
10. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint: http://arXiv.org/abs/1503.02531 (2015)

11. Kim, G., Hong, S., Franz, M., et al.: Improving cross-platform binary analysis using representation learning via graph alignment. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 151–163 (2022)
12. Lattner, C., Adve, V.: LLVM: a compilation framework for lifelong program analysis and transformation, San Jose, CA, USA, pp. 75–88 (2004)
13. Lee, Y.J., Choi, S.H., Kim, C., et al.: Learning binary code with deep learning to detect software weakness. In: KSII the 9th International Conference on Internet (ICONI) 2017 Symposium (2017)
14. Li, X., Qu, Y., Yin, H.: PalmTree: learning an assembly language model for instruction embedding. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 3236–3251 (2021)
15. Lin, J., Liu, Z., Wang, H., et al.: AMC: AutoML for model compression and acceleration on mobile devices. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11211, pp. 815–832 (2018). https://doi.org/10.1007/978-3-030-01234-2_48
16. Mikolov, T., Sutskever, I., Chen, K., et al.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
17. Park, J., Lee, S., Hong, J., et al.: Static analysis of JNI programs via binary decompilation. IEEE Trans. Softw. Eng. (2023)
18. Park, S., Choi, W.: Regulated subspace projection based local model update compression for communication-efficient federated learning. IEEE J. Sel. Areas Commun. **41**(4), 964–976 (2023)
19. Riley, G.F., Henderson, T.R.: The *ns-3* network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12331-3_2
20. Shen, M., Lu, H., Wang, F., et al.: Secure and efficient blockchain-assisted authentication for edge-integrated internet-of-vehicles. IEEE Trans. Veh. Technol. **71**(11), 12250–12263 (2022)
21. Song, Q., Zhang, Y., Wang, B., et al.: Inter-bin: interaction-based cross-architecture IoT binary similarity comparison. IEEE Internet Things J. **9**(20), 20018–20033 (2022)
22. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, vol. 27 (2014)
23. Tang, R., Lu, Y., Liu, L., et al.: Distilling task-specific knowledge from BERT into simple neural networks. arXiv preprint: http://arXiv.org/abs/1903.12136 (2019)
24. Xu, X., Liu, C., Feng, Q., et al.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376 (2017)
25. Zhang, X., Sun, W., Pang, J., et al.: Similarity metric method for binary basic blocks of cross-instruction set architecture. In: Proceedings 2020 Workshop on Binary Analysis Research (2020)
26. Zhao, J., Wang, R.: FedMix: a Sybil attack detection system considering cross-layer information fusion and privacy protection. In: 2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 199–207. IEEE (2022)
27. Zuo, F., Li, X., Young, P., et al.: Neural machine translation inspired binary code similarity comparison beyond function pairs. arXiv preprint: http://arXiv.org/abs/1808.04706 (2018)
28. Zuo, S., Zhang, Q., Liang, C., et al.: MoeBERT: from BERT to mixture-of-experts via importance-guided adaptation. arXiv preprint: http://arXiv.org/abs/2204.07675 (2022)