informatics

MDPI

Article

Balancing Accuracy and Efficiency in Vehicular Network Firmware Vulnerability Detection: A Fuzzy Matching Framework with Standardized Data Serialization

Xiyu Fang [1,2,3,*], Kexun He [2,3], Yue Wu [1], Rui Chen [4] and Jing Zhao [4]

[1] School of Computer Science and Engineering, University of Electronic Science and Technology of China, No. 2006, Xiyuan Avenue, Chengdu Hi-tech Zone (West District), Chengdu 611731, China; ywu@uestc.edu.cn
[2] CATARC Automotive Test Center (Tianjin) Co., Ltd., Xianfeng East Road, Dongli District, Tianjin 300162, China; hekexun@catarc.ac.cn
[3] China Automotive Technology & Research Center Co., Ltd., Xianfeng East Road, Dongli District, Tianjin 300162, China
[4] School of Software Technology, Dalian University of Technology, Tuqiang Road, Jinzhou District, Dalian 116024, China; 72117004@mail.dlut.edu.cn (R.C.); zhaoj9988@dlut.edu.ch (J.Z.)
* Correspondence: fangxiyu@catarc.ac.cn

Abstract

Firmware vulnerabilities in embedded devices have caused serious security incidents, necessitating similarity analysis of binary program instruction embeddings to identify vulnerabilities. However, existing instruction embedding methods neglect program execution semantics, resulting in accuracy limitations. Furthermore, current embedding approaches utilize independent computation across models, where the lack of standardized interaction information between models makes it difficult for embedding models to efficiently detect firmware vulnerabilities. To address these challenges, this paper proposes a firmware vulnerability detection scheme based on statistical inference and code similarity fuzzy matching analysis for resource-constrained vehicular network environments, helping to balance both accuracy and efficiency. First, through dynamic programming and neighborhood search techniques, binary code is systematically partitioned into normalized segment collections according to specific rules. The binary code is then analyzed in segments to construct semantic equivalence mappings, thereby extracting similarity metrics for function execution semantics. Subsequently, Google Protocol Buffers (ProtoBuf) is introduced as a serialization format for inter-model data transmission, serving as a "translation layer" and "bridging technology" within the firmware vulnerability detection framework. Additionally, a ProtoBuf-based certificate authentication scheme is proposed to enhance vehicular network communication reliability, improve data serialization efficiency, and increase the efficiency and accuracy of the detection model. Finally, a vehicular network simulation environment is established through secondary development on the NS-3 network simulator, and the functionality and performance of this architecture were thoroughly tested. Results demonstrate that the algorithm possesses resistance capabilities against common security threats while minimizing performance impact. Experimental results show that FirmPB delivers superior accuracy with 0.044 s inference time and 0.932 AUC, outperforming current SOTA in detection performance.

Keywords: firmware vulnerability detection; protocol buffers (ProtoBuf); program analysis; resource-constrained; instruction embedding

Academic Editor: Antony Bryant

Received: 5 June 2025
Revised: 5 July 2025
Accepted: 7 July 2025
Published: 9 July 2025

Citation: Fang, X.; He, K.; Wu, Y.; Chen, R.; Zhao, J. Balancing Accuracy and Efficiency in Vehicular Network Firmware Vulnerability Detection: A Fuzzy Matching Framework with Standardized Data Serialization. *Informatics* **2025**, *12*, 67. https://doi.org/10.3390/informatics12030067

1. Introduction

The Internet of Vehicles (IoV) is a heterogeneous network system composed of vehicular ad hoc networks and mobile communication networks. Through real-time interconnection and perception among vehicles, roads, and cloud infrastructure, the IoV significantly enhances vehicle safety and intelligence in road environments while providing users with diversified information services and optimizing the driving experience [1]. However, IoV security protection faces numerous severe challenges. Security issues in vehicular networks can not only potentially cause property losses to drivers but may in severe cases also endanger the lives of other road users [2].

According to publicly released data from the China Automotive Research Center, vulnerabilities in IoV devices have triggered multiple serious security incidents, including door lock security vulnerabilities, unauthorized access to vehicle cameras, and privacy issues such as vehicle location information leakage. Relevant statistics indicate that 80% of IoV devices employ weak cryptographic algorithms with insufficient security strength; 70% of IoV communications are implemented without encryption protection, while 90% of IoV firmware contains potential security risks [3]. Due to constraints imposed by hardware architectural differences and complex hardware environments, effective detection is difficult to achieve. Simultaneously, existing embedding methods compute independently across models and lack standardized interaction information mechanisms between models, resulting in low efficiency of embedding models during firmware vulnerability detection processes.

The challenges in IoV security detection underscore a broader issue in the field of cybersecurity, namely, the insufficiency of current analytical methods to accurately extract and analyze program semantics, which is crucial for identifying and mitigating such vulnerabilities. Existing similarity analysis methods exhibit insufficient precision in extracting program semantic information. Current analytical approaches primarily fall into two paradigms, code-based and graph-based. Code-based methods mainly focus on instruction contextual relationships while neglecting critical dynamic information such as data flow. Conversely, graph-based methods emphasize structural information analysis such as control flow, but overlook semantic associations between instructions. Consequently, binary programs may dynamically adjust their behavioral characteristics during execution based on input parameters and runtime environments. Thus, relying solely on either static or dynamic analysis methods makes it difficult to comprehensively capture all potential behavioral patterns of a program [1,4,5].

Furthermore, authentication mechanisms and firmware analysis face significant computational efficiency bottlenecks under the current cascaded trust center architecture in vehicular networks, particularly in V2V (Vehicle-to-Vehicle) communication encryption/decryption application scenarios. The current research domain has yet to establish unified instruction transmission format specification standards, preventing effective support for feature input processing by instruction embedding models. This lack of standardization makes cross-platform data processing consistency difficult to guarantee, thereby severely affecting detection models' operational efficiency and accuracy [6–8].

On the other hand, instruction embedding based on high-precision vulnerability detection technology provides effective technical support for vehicular network firmware security analysis. However, these studies generally lack in-depth discussion and empirical analysis regarding how unified structured data representation resolves cross-architecture instruction differences. Notably, existing experimental validations also lack systematic empirical verification through simulation experimental environments. These methodological limitations lead to questioning of the persuasiveness and practicality of research results,

making it difficult to provide reliable theoretical and practical foundations for optimizing actual vehicular network system security architectures [2,9,10].

In light of the aforementioned challenges, this paper proposes a firmware vulnerability detection scheme that balances accuracy and efficiency to achieve efficient operation of instruction embedding models in resource-constrained vehicular network environments. This scheme is specifically designed for resource-constrained vehicular network environments, utilizing a code similarity fuzzy matching analysis algorithm based on statistical inference that is capable of efficient security vulnerability identification in vehicular networks under limited computational resources.

First, beginning at the instruction level of binary programs, the analysis and computation of semantic similarity are gradually extended to the procedural level, with assembly code undergoing normalization processing. During the matching operation phase, precise instruction comparison is performed by applying dynamic programming algorithms for longest common subsequence to candidate functions, combined with path search and neighborhood search techniques to generate high-quality candidate sets for functions awaiting matching. This scientifically derives similarity metrics for file execution semantics based on binary file similarity.

Second, a data transmission serialization format specification based on inter-detection models is constructed by uniformly defining instruction representation format standards across multiple processor architectures including ARM/x86/MIPS. Simultaneously, a specialized instruction conversion preprocessor is developed to transform various binary instructions into standardized ProtoBuf message object structures, significantly enhancing processing efficiency of detection model data serialization and effectively resolving the detection efficiency issues caused by heterogeneous architecture data transmission differences.

Finally, the effectiveness of the proposed solution is validated through systematic model detection experiments and vehicular network simulation experiments based on the NS-3 network simulator [11]. These experiments integrate the IEEE 802.11p wireless communication protocol stack with a cascaded trust center architecture, implementing a complete security verification process from certificate chain management to V2V communication encryption. Additionally, efficient transmission and processing of multidimensional data including location information and vehicle status are supported through the ProtoBuf standardized message encapsulation mechanism.

Above all, the major contributions of this work are as follows:

- We design a code similarity fuzzy matching analysis algorithm based on statistical inference for resource-constrained vehicular network environments, achieving efficient security vulnerability identification under limited computational resources to resolve the application difficulties of traditional methods in resource-constrained environments.
- We develop a semantic similarity analysis framework extending from instruction level to procedural level, implementing scientific mapping from binary file similarity to execution semantic similarity through dynamic programming algorithms for the longest common subsequence in combination with the path search and neighborhood search techniques. This addresses the insufficient precision of existing methods in semantic information extraction.
- We construct a data transmission serialization format specification between detection models. The proposed specification uniformly defines instruction representation formats for multiple processor architectures, including ARM/x86/MIPS. In addition, we develop a dedicated instruction conversion preprocessor to resolve

cross-platform data processing consistency issues and improve the detection model's operational efficiency.

- By combining model detection experiments with vehicular network simulation experiments based on NS-3 and integrating the IEEE 802.11p protocol stack with a cascaded trust center architecture, we establish a complete security verification system from certificate chain management to V2V communication encryption, thereby filling the gap in existing research regarding systematic empirical verification.

2. Background and Literature Review

Recent studies have shown that representational learning outperforms traditional heuristic-based methods on many instruction embedding tasks, including binary similarity detection [8,12], function name prediction, and function boundary identification. The key to its success is learning vector representations of assembly instructions (also known as embeddings) at the granularity of instructions, basic blocks, or functions, which can then be applied to various downstream tasks such as instruction search and firmware vulnerability detection [7]. While compressing instruction embedding models significantly reduces storage requirements, it results in poor performance on embedding assembly instructions [13–15]. Thus, the compressed models cannot run in IoV environments.

Previous work has focused on building pretrained models that can generate such embeddings, either by fine-tuning these pretrained models or directly applying the obtained embeddings to downstream tasks [7,8,16]. Based on prior research on learning embeddings, these can be categorized into unsupervised and supervised learning, both of which have been explored for instruction embedding and trusted communication in IoV networking. In the early days of instruction embedding research, many studies employed unsupervised learning to produce binary code embeddings. The advantage of unsupervised methods is the lack of requirement for any labeled training data, meaning that it is only necessary to disassemble binary files (i.e., to identify instructions and function boundaries) and construct intra- and/or inter-procedural control flow graphs. DeepBindiff [17] and Asm2Vec [18] are typical examples of such methods; among these, DeepBindiff embeds instructions and basic blocks using Word2Vec [19].

As research deepened and downstream tasks demanded instruction embedding more strongly, supervised instruction embedding methods (first proposed by [12]) have shown advantages. Most previous works trained Siamese architectures on labeled binary code pairs. Gemini by Xu et al. [12] represents each function as an Augmented Control Flow Graph (ACFG) and uses them as inputs to a Structure2Vec model. Two identical Structure2Vec models sharing parameters are constructed as a Siamese network and trained on ACFG pairs with similarity labels [12].

Today, software supports cross-operating system and instruction set architectures (ISAs). Software that runs across operating systems may implement the same high-level functionality but follow different OS interfaces. Depending on the deployed hardware, binaries can be compiled for different ISAs, which is common among firmware for various ISAs such as ARMv7, ARMv8, MIPS, etc. Dealing with such heterogeneity poses challenges for instruction embedding [8,10]. Therefore, platform-agnostic instruction embedding approaches have garnered great interest. Previous research has proposed various platform-independent instruction embedding techniques for applications such as vulnerability search, reuse vulnerability detection, binary similarity detection, etc. The key promise is that analytical work can seamlessly transfer across platforms. For example, error signatures only need to be defined once and can be used to search errors in binaries from different ISAs [20]. Previous works have achieved this by abstracting binaries into OS/ISA-agnostic

assembly instructions. In contrast, Gemini and InnerEye [21] use Siamese models to directly learn similarity between binaries compiled for different platforms.

Upon reviewing related instruction embedding work, we find that current instruction embedding research trends towards large models. However, specific domains such as IoV networking have urgent needs that remain under-explored due to resource constraints. Motivated by communication scenarios in IoV networking, we study state-of-the-art trusted communication techniques in VANETs [1,5] to facilitate effective instruction embedding under such environments.

The architecture of VANETs can be divided into three domains based on communication range: intra-vehicle, vehicle-to-infrastructure (V2I), and inter-vehicle communications. Intra-vehicle communication refers to interactions between the on-board unit (OBU) chipset and in-vehicle control components such as air conditioning, ABS, ECUs, and infotainment systems. The circuits and programs are predefined by manufacturers in this closed setting. There are few external interfaces and the programs are relatively simple with higher security. User terminals can be dedicated devices or integrated remote controls to operate specific in-vehicle systems, mostly via the CAN bus or relatively secure wireless means.

V2I communication involves interactions between OBUs and roadside unit (RSU) infrastructure to enable vehicle-road coordination, driving assistance, and self-driving. Inter-vehicle communication usually refers to radio frequency direct communications (V2V) between OBUs of vehicles. By broadcasting basic safety messages via DSRC or C-V2X, vehicles share their location, speed, and acceleration for safety applications to generate warnings about hazards and risks. This allows drivers and passengers to prepare in advance in order to prevent accidents or mitigate adverse consequences [22].

VANETs can be divided into three primary domains: intra-vehicle communication, vehicle-to-infrastructure (V2I) communication. Intra-vehicle communication mainly involves the interactions between the on-board unit (OBU) chipset [1] and various in-vehicle control components such as air conditioning, ABS, ECUs, and infotainment systems [23]. In such communication scenarios, related circuits or programs are usually predefined by manufacturers. Due to fewer external interfaces and relatively simple program design, their security is higher. User terminals can be either a dedicated device or an integrated remote control for operating specific in-vehicle systems. Such connections are mostly through the Controller Area Network (CAN) bus, with some secure wireless options available. V2I communication refers to the interactions between OBUs and roadside units (RSUs), enabling functionalities like vehicle-road coordination, driving assistance, and self-driving.

Certificate authentication plays an indispensable role in today's information systems, mainly in confirming the legitimacy of user identities and granting corresponding access permissions when users access target system resources over the internet. For VANET systems, authentication is the first and most important line of defense. In researching authentication techniques, ref. [24] proposed a new decentralized VANET authentication protocol using a novel group signature scheme which provides threshold authentication, efficient revocation, unforgeability, anonymity, and traceability. In [25], the authors introduced a novel conditional privacy-preserving authentication protocol for VANETs which binds public keys into pseudonyms to ensure non-repudiation by obtaining real IDs of misbehaving vehicles. In [26,27], a multi-domain vehicle authentication architecture was proposed by introducing blockchain technology to establish distributed trust and share cross-domain information among multiple administrative domains. In [28], the authors proposed a novel privacy-preserving authentication protocol based on a certificateless aggregated signature scheme that allows users to generate fuzzy identities to hide their real identities; in this scheme, even the private key can remain unchanged if the corresponding pseudonym is updated.

Today, infrastructure-based VANET is the primary scheme for deploying intelligent connected vehicle applications, attracting extensive attention and support from stakeholders such as automakers, chipmakers, transportation authorities, and ISPs. However, data security and privacy protection must be fully considered when designing VANET system architectures and communication workflows, which imposes new requirements on VANET architectures. This paper conducts in-depth analyses of process design and potential security risks in VANETs, then proposes a trusted communication scheme based on VANET infrastructure combined with industry realities and standards to enable more effective instruction embedding for IoV networking.

3. Proposed Method

3.1. Fuzzy Matching Analysis for Code Similarity

3.1.1. Function Identification

In the training phase, we initially generate binary code for a series of candidate processes and determine the starting address of each function. Based on these addresses, we construct a prefix tree to store and match different processor optimization processes. The prefix tree efficiently stores strings by sharing common prefixes, with each node representing a potential function start instruction. Node weights are determined by the recognition rate $|TP|/(|TP| + |FP|)$. This structure facilitates rapid identification and classification of function entry points. Table 1 provides the notation and abbreviations.

Table 1. Notations and abbreviations.

Notations or Abbreviations	Description
XML	Extensible Markup Language
ProtoBuf/PB	Google Protocol Buffers
IoV	Internet of Vehicles
CAN	controller area network
V2V	Vehicle-to-Vehicle
ECU	electronic control unit
OBU	On-board Unit
LCS	Longest Common Subsequence
BFS	Breadth-First Search
CFG	Control Flow Graph
FMENS	Fuzzy Matching Enhanced Neighborhood Search
CRL	Certificate Revocation Lists
CA	Certificate Authorities
MTA	Main Trust Authority
STA	Trust Authorization Center
ERI	Electronic Registration Identification
M	the predetermined number of pseudonym certificates
OOV	Out-of-Vocabulary

During the classification phase, we determine which instructions may constitute function entry points based on node weights. After identifying function starts, we employ a static CFG recovery algorithm to generate the instruction set for the entire function body from the specified address. We construct the CFG graph by recursively connecting directly related instructions. When processing direct and indirect jump instructions, we appropriately extend boundaries to ensure the correctness and completeness of control flow. This includes special handling of function calls and jump instructions to ensure that all possible execution paths can be traced.

(1) Instruction Extraction Phase: For the set of function start positions and corresponding end positions obtained from debug information, we extract instruction sets of length k

from the start position. If the instruction length in the current (b, f) is less than k, we use all instructions in (b, f) for training. Specifically, for each function's (b, f), we extract its first k instructions $I[b : b + k]$; if the instruction length of (b, f) is less than k, we retain all information from (b, f).

(2) Prefix Tree Generation Phase: Based on the characteristics of prefix trees described earlier, we construct a prefix tree using the instruction sets produced in Phase 1. A path extending from the root node to a leaf node represents an instruction sequence. Here, k represents the maximum length of instruction sequences. In actual analysis, the instruction length may not equal k; if the length is less than k, we directly use the actual length for training, while if it is greater than k we use k for training. During experimentation, instruction sequences undergo normalization operations. This normalization enables matching operations to handle similar but non-equivalent instructions, effectively improving analysis precision and recall rates. For instruction sequences of length k, we generate a prefix tree as shown in Figure 1, where each non-root node v represents an instruction in the sequence. The execution process of a function's instructions can then be represented as the instruction sequence from the root node to v.

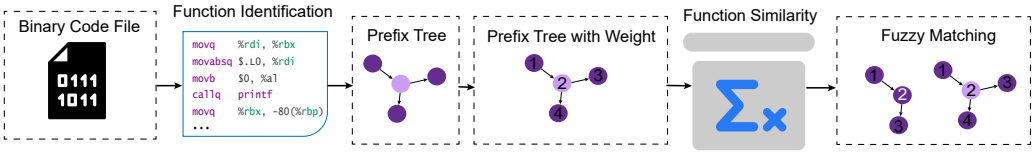

Figure 1. Fuzzy matching analysis for code similarity framework.

(3) Weight Calculation Phase: In specific experiments, we divide the function set into training and testing sets. By assigning a weight to each prefix tree node in the training set, we can determine whether the statement sequences (maximum k instructions) generated in Phase 2 represent function entry points. The weight calculation process is as follows: for the entire training set D, if a node in the prefix tree is not a function entry point in D then we store it in set D_-; otherwise, we store it in set D_+. We then record the number of actual function starts in D_+. Note that errors may occur in this process; for example, an instruction sequence corresponding to a path in the prefix tree may not be an actual function start, necessitating a reduction in weight. We set the node weight as $\omega = D_+/(D_+ + D_-)$.

3.1.2. Function Similarity with Fuzzy Matching

Dynamic programming (DP) is a mathematical method for solving multi-stage decision optimization problems, with its theoretical foundation derived from Bellman's optimality principle. DP decomposes complex problems into interconnected subproblems and employs either bottom-up recursive methods or top-down memo-ization strategies for resolution. Unlike traditional divide-and-conquer approaches, DP-processed subproblems typically exhibit overlapping characteristics. This requires maintaining state transition tables in order to avoid redundant computation, resulting in significantly enhanced algorithmic efficiency. In DP, "programming" actually refers to the table-filling process, in which solutions to subproblems are systematically stored in a multi-dimensional state space to facilitate subsequent computational queries.

The longest common subsequence (LCS) algorithm is crucial for sequence similarity measurement, identifying shared non-contiguous element sequences across multiple sequences. Distinct from the longest common substring (which requires continuity), the LCS permits elements to exist non-contiguously in the original sequence, making it particularly applicable to instruction sequence similarity analysis. From Equation (1), the LCS problem can be efficiently solved through DP, as shown below.

$$\begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1, & \text{if } X_i = Y_j \\ \max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)), & \text{if } X_i \neq Y_j \end{cases} \quad (1)$$

Breadth-first search (BFS) is a graph traversal algorithm extended to control flow graph (CFG) traversal in function similarity analysis. This study proposes an enhanced BFS algorithm that expands the search space layer by layer from the CFG entry vertex, generating mapping relationships and similarity scoring matrices between basic blocks by matching control flow structures with candidate functions. During the matching process, we employ instruction sequence normalization strategies to standardize instruction sequences within basic blocks through semantic equivalence transformations, which eliminates surface differences caused by compiler optimizations, instruction reordering, and other factors. Subsequently, we utilize the aforementioned LCS algorithm to calculate the similarity of normalized instruction sequences. In this way, we obtain the maximum common subsequence in execution paths between target and candidate functions, thereby establishing initial mapping relationships between basic blocks.

To further enhance the precision and efficiency of function matching, this research introduces the Fuzzy Matching Enhanced Neighborhood Search (FMENS) algorithm. Based on initial basic block mappings, this algorithm expands the search space outward using control flow dependencies while tolerating a certain degree of structural variation, thereby enabling identification of function variants that have undergone code refactoring, local modifications, or optimizations. FMENS iteratively updates the similarity matrix, ultimately generating global similarity scores between functions that comprehensively consider instruction semantics, control flow structure, and data dependency relationships, which provides a reliable similarity measurement foundation for binary code analysis.

3.1.3. Semantic Similarity Calculation

Let us assume that the state $s_i \in S_P$ of program P at a specific time point (where S_P represents the set of all possible states) can be represented as a triplet (pos_i, x_i, w_i), where pos_i denotes the position of variable x_i at that moment and w_i represents the current value of variable x_i. A program's execution trace $\tau \in S_P^*$ is a sequence of states $s_0, s_1, \ldots, s_n$, while the set of all possible execution traces constitutes E_P. For trace τ, we use $init(\tau)$ and $term(\tau)$ to represent its initial and terminal states, respectively.

We denote the mapping relationship between two states s_a and s_b as ρ, which maps variable x_a in s_a to variable x_b in s_b, written as $x_a \to x_b$. If the condition $s_a(u) = s_b(v)$ is satisfied for all mapping pairs $(u, v) \in \rho$, then s_a and s_b are considered equivalent under mapping ρ, denoted as $s_a \cong_\rho s_b$. For two execution traces τ_a and τ_b, if they are equivalent under mapping ρ, we denote this as $\tau_a \cong_\rho \tau_b$. For a program pair (P_a, P_b), the set of all possible variable mapping relationships is denoted as $M(P_a, P_b)$.

Definition 1. *Code Fragment Equivalence: For two code fragments C_a and C_b, they are equivalent under mapping ρ, i.e., $C_a \cong_\rho C_b$, if and only if: (1) each input in C_a has a corresponding input in C_b, and (2) for any execution pair $(\tau_a, \tau_b) \in (C_a, C_b)$, if they are equivalent at the input points ($\tau_a \cong_\rho \tau_b$), then the following condition is satisfied:*

$$\forall (in_a, in_b) \in (\rho \cap (inputs(C_a) \times inputs(C_b))) : init(\tau_a)(in_a) = init(\tau_b)(in_b). \quad (2)$$

Definition 2. *Code Fragment Similarity: For two code fragments, their similarity is defined as the proportion of the maximum number of matching variables under a mapping relationship, namely, Equation (3):*

$$Similarity(C_a, C_b) = \frac{\max\{|\rho| | \forall(\tau_a, \tau_b) \in (C_a, C_b) : \tau_a \cong_\rho \tau_b\}}{|vars(C_a)|}. \quad (3)$$

The significance of the above similarity formula lies in its direct computation of similarity based on the core semantics of code fragments rather than merely relying on output value comparisons. This method reduces potential errors associated with judging similarity solely through output values, thereby enabling the generation of meaningful similarity scores for code that appears unrelated on the surface but is semantically similar. For states s_a and s_b, the proportion of matching values in s_a is defined as $Similarity(s_a, s_b) = \frac{|\rho_{opt}|}{|s_a|}$, where $|\rho_{opt}|$ represents the size of the optimal mapping that makes s_a and s_b equivalent, i.e., the maximum mapping satisfying $s_a \cong_{\rho_{opt}} s_b$.

3.2. Training Information Formatted Using ProtoBuf

To enhance network transmission efficiency during model encoding and decoding processes, we implement Google Protocol Buffers (ProtoBuf) as our serialization format. Compared to XML and JSON, ProtoBuf offers significant advantages: (1) ProtoBuf-serialized data typically require 20–100% less space than JSON, substantially reducing network bandwidth consumption; (2) the binary format of ProtoBuf enables parsing speeds 20–100 times faster than JSON, minimizing model latency; (3) data structures defined through .proto files provide rigorous type checking, reducing runtime errors.

Our implementation utilizes the ProtoBuf message structure shown below.

```
syntax = "proto3";

message ModelInput {
        repeated float instruction
        _features = 1 [packed=true];
        int32 sequence_length = 2;
        int32 feature_dimension = 3;
}

message ModelOutput {
        repeated float embedded_
        representation = 1 [packed=true];
        repeated float attention
        _scores = 2 [packed=true];
        float loss = 3;
}
```

This structure enables efficient serialization and transmission of model input features and output representations, leading to significantly improved system throughput and response times, particularly in distributed training and inference scenarios.

Following the design improvements in the transformer model described earlier, this section discusses the use of ProtoBuf data format to transfer data across different compilation architectures and various models. Utilizing ProtoBuf formatted training information, ProtoBuf acts as a "translation layer" and "bridging technology" within the firmware vulnerability detection framework. It addresses the issue of inconsistent code representation caused by architectural differences, solving the core problem of cross-architecture instruction representation. This enables code compiled from different architectures but from the same source code to be compared and analyzed through a unified data structure.

In this section, we introduce a method for cross-architecture basic block embedding based on the transformer architecture, which includes an encoder and a decoder. To better handle and standardize instruction data, we propose a ProtoBuf-based instruction representation standardization framework. The details of this framework and its application in cross-architecture embedding are discussed below.

3.2.1. Introduction to ProtoBuf

ProtoBuf [29], led by Google, is a structure suitable for network transmission that facilitates the serialization and deserialization of structured data. This structure boasts high portability and is applicable across various programming languages, operating systems, and development platforms; additionally, its serialization mechanism features scalability. ProtoBuf can be compared to Extensible Markup Language (XML) and JSON, but offers significant advantages in terms of the size of information as well as the efficiency of serialization and deserialization operations. It is smaller, more efficient, and simpler to use for the same amount of information. For users, it only requires building the structure of the data to be transmitted one time at both the sending and receiving ends, including the types, order, hierarchy, and nesting. Then, using compilation tools, the constructed data structure is compiled into source code and integrated into the project, allowing for easy serialization, deserialization, or reading and writing of various data streams. This operation is not restricted by any programming language.

The advantages of ProtoBuf are primarily reflected in the efficiency of data serialization and high flexibility during development and maintenance. For example, a data structure defined in ProtoBuf is several times smaller than one in the more commonly-used XML format, and can be serialized tens of times faster. Its flexibility is demonstrated by the fact that updates to the data structure can be made without breaking existing older programs.

3.2.2. ProtoBuf Instruction Representation Framework

Initially, we defined a ProtoBuf message format to standardize the representation of instructions. This not only aids in model training and inference but also enhances the efficiency of data storage and exchange. The specific ProtoBuf message definition is shown below.

```
syntax = "proto3";
package instruction;
// Standardized representation
// of an instruction
message Instruction {
        // Architecture of the
        // instruction,
        // e.g., "x86" or "ARM"
        string architecture = 1;
        // List of operands
        repeated string operands = 2;
        // Opcode
        string opcode = 3;
        // Positions of the instruction
        // within the basic block
        repeated int32 positions = 4;
}
```

3.2.3. Encoder

As shown in Figure 2, the encoder receives instruction embeddings as input, which are obtained through word embedding and positional encoding. The encoder consists of six enhanced transformer blocks, each including a multi-head self-attention layer and a feed-forward neural network layer.

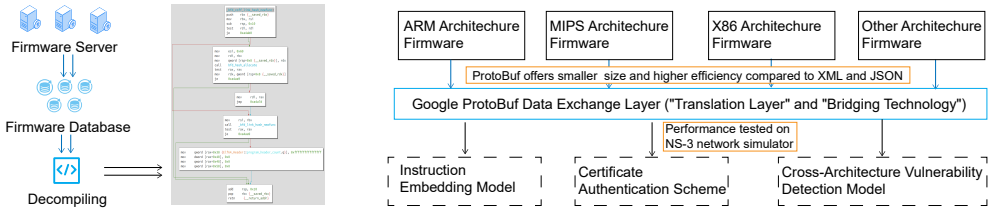

Figure 2. Cross-architecture firmware vulnerability detection model based on ProtoBuf.

The multi-head self-attention layer constructs the basic block embedding representation by capturing complex relationships between instruction embeddings. Instruction embeddings are first linearly transformed through three learnable weight matrices W_Q, W_K, W_V to generate the matrices Q, K, V. These matrices are then divided into h heads, each producing Q_i, K_i, V_i. Using the dot-product attention mechanism, each head computes attention scores A_i while scaling the dot products to avoid gradient issues, ultimately producing the output Z_i of the multi-head self-attention layer.

3.2.4. Decoder

The structure of the decoder is similar to that of the encoder, consisting of six analogous transformer blocks. Through the encoder–decoder attention mechanism, the decoder accesses vector representations of the input sequence from the encoder and predicts the next instruction in the output sequence.

3.2.5. Training and Prediction

In cross-architecture embedding operations, an X86 encoder is initially trained, followed by parameter adjustments to accommodate different architectures. During the training phase, instruction sequences from the X86 architecture serve as input for the encoder, while instruction sequences from the ARM architecture are used as input for the decoder. This approach allows the model to learn how to map instructions from one architecture to another.

By integrating this ProtoBuf-based instruction representation with the transformer architecture, we effectively implement cross-architecture basic block embedding, thereby enhancing the model's generalization capabilities and its ability to handle complex tasks.

3.3. ProtoBuf Structure-Based Pseudonym Certificate Verification Method for Vehicular Networks

Security agencies face a critical issue in current IoV network applications, particularly in scenarios such as multi-level interchanges where large numbers of vehicles converge. From Figure 3, they need to ensure that the efficiency of pseudonym certificate verification meets or exceeds 1000 verifications per second. To address this, our research proposes a pseudonym certificate generation and verification scheme based on the protocol buffer (PB) structure, leveraging its widespread industrial use to test whether the required verification efficiency can be achieved.

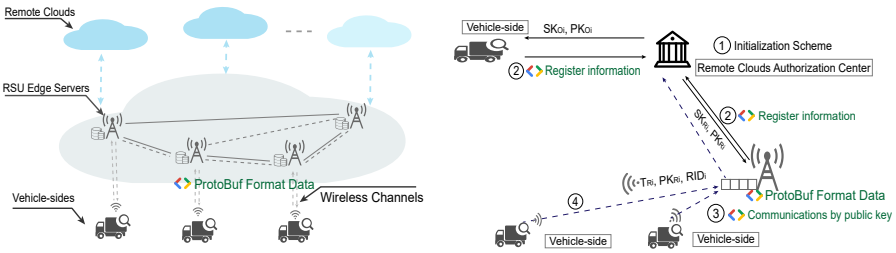

Figure 3. Certificate authentication scheme based on protocol buffers for enhancing data serialization efficiency and development flexibility.

3.3.1. Initialization Phase

The purpose of the initialization phase is to generate the initial data required for subsequent processes, such as public–private key pairs used by various communication entities (e.g., different levels of trust centers), certificate revocation lists (CRLs), and related identity markers. Vehicles register their information and obtain public keys from multi-level certificate authorities (CAs).

During the initialization phase, vehicles organize their registration messages and upload them via the nearest roadside unit (RSU) using a trusted wired or wireless network to the main trust authority (MTA). Subsequently, the MTA verifies the vehicle information, saves the records either in the cloud or locally, and generates the public key certificate for the vehicle.

The CA within the MTA generates a random private key, which becomes the system's master key; the corresponding public key is then generated from this private key. The CA initializes the CRL and distributes the CA's public key to vehicles and RSUs in the network. Vehicles use the CA's public key to compute their own public–private key pairs.

After the initialization phase, vehicles have obtained the CA's public key and generated their own public–private key pairs. Cascading from the trust authorization center (STA) and RSUs, vehicles also receive regional symmetric keys.

3.3.2. Pseudonym Certificate Generation Phase

This phase is designed to ensure the privacy of data when vehicles participate in communications. When a vehicle intends to engage in subsequent vehicle-to-vehicle (V2V) communications in IoV networks, the vehicle terminal enters this phase. The vehicle sends a pseudonym certificate request to the CA, which includes the vehicle's unique identifier (ID) and the vehicle's public key, then encrypts these using the CA's public key and sends it to the CA. Upon receipt, the CA decrypts the request using its private key to obtain the true identity of the terminal vehicle and verify this identity. The vehicle's request information is formatted as follows:

$$Request_{vi \to CA} = E_{PK_{CA}}(ID_{vi}, PK_{vi}) \tag{4}$$

where ID is the vehicle's electronic registration identification (ERI) and PK is the vehicle's public key. After decryption, the CA obtains the vehicle's true identity and conducts verification.

If the verification is successful, the CA generates a pseudonym ID and a pseudonym certificate for the vehicle along with an expiration time and then randomly generates a private key for the pseudonym, from which the public key is computed. If verification fails, this may indicate that the vehicle has previously engaged in violations or malicious activities and its certificate has been revoked, necessitating resolution through the transportation management authority.

The main trust authority (MTA) maintains a list in which the key is the vehicle terminal's real identity and the value is its corresponding set of pseudonym certificate identities.

After a specified number of pseudonyms are generated, they are encrypted using the vehicle's public key and sent to the corresponding vehicle terminal.

The response message from the CA to the vehicle terminal is formatted as follows:

$$Message_{CA \to Vi} = \{PID_{v_{i,k}}, Cert_{PV_{i,k}}, PK_{PV_{i,k}}\}, K \in [1, M] \quad (5)$$

where M is the predetermined number of pseudonym certificates. Upon receiving the message, the vehicle terminal decrypts it using its own private key and stores the pseudonym information locally.

3.3.3. Vehicle-to-Vehicle Secure Communication Phase

This phase is designed to ensure the security of data during vehicle-to-vehicle (V2V) communications. After obtaining pseudonym certificates, vehicle terminals within the same certification area validate the pseudonym certificates through the subordinate trust authority (STA) in order to ascertain the vehicle's true identity. They also check the revocation status of the vehicle's certificate to determine whether the vehicle is eligible to receive communication keys. Subsequently, the STA issues keys for V2V communication.

Vehicles certified within the same certification area possess identical communication keys. During communication, symmetric encryption is employed to maintain the security and integrity of the data.

3.4. Certificate Authentication Communication Data Structure Based on ProtoBuf

In the final stage of the architecture process for IoV communication, vehicles transmit Basic Security Message (BSM) information. In 2013, the IEEE completed the WAVE protocol stack based on wireless networks for inter-vehicle communication, which consists of IEEE 802.11p, IEEE 1609 series protocols, and the SAE J2735 standard [30]. SAE J2735 specifies the contents of vehicle BSMs, including vehicle ID, operational status, speed, direction of travel, etc. The messages transmitted between onboard units (OBUs) and roadside units (RSUs) are the basic units of application-layer data packet transmission, and are composed of different categories of message bodies.

3.4.1. Key Negotiation Process Design

Key negotiation is used for parties A and B to jointly establish a session key.

In this study, the negotiated key is primarily the regional symmetric key for the final phase of the entire security architecture process. The key negotiation process must satisfy several characteristics:

First, the key negotiation process should coexist with the encryption negotiation process, both of which are essential components of key negotiation.

Second, the negotiated key should have a validation mechanism based on an expiration time, ensuring that the negotiated communication key is regularly and forcibly replaced. In the same communication area, after a vehicle obtains the regional symmetric key, the negotiated key will expire after a specified time and cannot continue to be used. This prevents vehicles with revoked certificates from participating in V2V communications and posing a security threat. Additionally, the entire key negotiation process itself should have related threshold times as the basis for timeout judgment to ensure that the negotiation process is not too lengthy. If the negotiation process itself is prolonged, this likely indicates an error or malicious attack, in which case the process should be considered failed and needs to be renegotiated.

Lastly, to prevent leakage of information in communications, the data exchanged between the parties during the negotiation process should not appear in plaintext. Negotiation should proceed using a scheme similar to asymmetric key systems, and the encrypted data should include an identity marker to verify the initiator as the genuine communicating party.

3.4.2. Implementation of Certificates Based on PB Structure

The generation process for certificates using the PB structure can be divided into three parts.

First, according to the content structure of certificates in Chapter 3, each part is written in sequence and hierarchy into the corresponding .proto file, with each field and its type in the certificate edited into the .proto file following the PB syntax.

Second, the ProtoBuf toolkit is used to compile the .proto file written in the first part. This file defines the hierarchical structure and content framework of the certificate. These data structures serve as the format agreed upon by the communicating parties and are intended for software developers and business logic, which dictate the format of these data structures; however, this format is not particularly friendly for network transmission and storage, as it requires serialization, deserialization, and read–write operations of structured data related to the business logic. ProtoBuf provides corresponding interface code for these operations. Through the 'protoc' compiler, our .proto files can be transformed into corresponding data structure operation interfaces.

Interface code can be generated with the following command: $protoc - I = \$SRC\_DIR - -cpp\_out = \$DST\_DIR\$SRC\_DIR/xxx.proto$.

The third part involves calling the above-generated interfaces in the project to implement serialization, deserialization, and read–write operations. The .proto files from the first part are compiled by 'protoc' in the second part into corresponding interface code files, which appear as header files in C++. These generated interface files are incorporated into the project, after which the data structures defined in the .proto files can be used, such as for writing, reading, serializing, and deserializing.

4. Experimental Results and Analysis

4.1. Experiment Setup

Our experimental framework is structured into three distinct phases. The first phase focuses on developing cross-architecture basic block embeddings. We implemented Python scripts using Binary Ninja to extract instruction sequences from attributed control flow graphs (ACFGs), followed by neural network model construction using PyTorch (Python3) to achieve effective embedding of cross-architecture basic blocks. We selected three established baseline models for comparative analysis: Gemini [12], DeepVSA [31], and PalmTree [7]. The experiments were executed on a high-performance cloud server equipped with a 42-core CPU, 240 GB RAM, three NVIDIA RTX 3090 GPUs (each with 24 GB VRAM), and 100 GB SSD storage. The software environment consisted of PyTorch 1.10.0 and CUDA 11.3.

The experimental requirements included C-V2X mode4 communication capabilities and an OpenSSL-based cryptographic environment. We initially established an IoV network communication environment based on NS-3 [11], employing Ubuntu 18.04 as the Linux development platform due to its stability. The simulation experiments were conducted on the NS-3 simulator, with the simulation scenario consisting of a 200-m unidirectional roadway populated with ten communication terminal nodes (nine IoV terminals and one roadside unit) uniformly distributed along the roadway. These nodes maintained constant velocities while traveling forward, with the roadside unit (RSU) positioned in the central

region of the roadway. The communication methodology implemented LTE-based C-V2X mode4 communication protocols.

For our comparative analysis, we developed both baseline embedding models and our proposed FirmPB framework in PyTorch. While traditional BERT architectures typically employ twelve layers, twelve attention heads, and 768-dimensional hidden states, our FirmPB implementation utilizes a more streamlined configuration with twelve layers, eight attention heads, and 64-dimensional hidden states to optimize computational efficiency and minimize training resource requirements.

To thoroughly evaluate our methodological contributions, we created two variants of our framework: the complete FirmPB implementation incorporating our novel assembly instruction call graph algorithm, and a reduced version (FirmPB-W/o-G) that omits this graph generation component, allowing us to isolate and quantify its impact.

Our evaluation leveraged the publicly available MIRROR dataset [8], comprising source code from five prominent C/C++-based open-source projects: Binutils 2.30, Coreutils 8.29, FFmpeg n3.2.13, OpenSSL 1.1.1b, and Redis 5.0.5. Using the LLVM [32] compiler infrastructure, we generated corresponding x86 and ARM assembly code datasets from these source repositories.

Experimental evaluation was conducted on a high-performance computing environment running Ubuntu 18.04 and featuring an Intel Xeon E5-2683 V4 processor, dual GeForce RTX 3090 graphics accelerators, and 124 GB system memory. The FirmPB pretraining phase utilized 128,000 randomly selected sample pairs from our compiled ARM and x86 assembly datasets. We initialized our model using the "bert-base-uncased" pretrained weights from Hugging Face [16] and conducted training over 40 epochs, focusing on sentence semantic equivalence objectives. As depicted in Figure 4A, the model successfully converged on the cross-architectural (x86/ARM) basic block equivalence task within this training regime. Subsequent evaluations of downstream task performance were conducted using this optimized cross-architecture instruction embedding model.

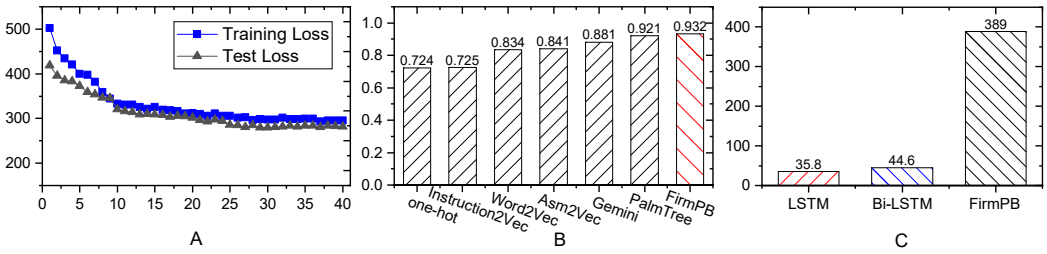

Figure 4. (**A**) Training and test loss, (**B**) AUC values of Gemini, and (**C**) storage space of trained model.

4.2. Performance Evaluation of Model on Downstream Tasks

4.2.1. Cross-Architecture Instruction Embedding for Firmware Vulnerability Detection Using ProtoBuf Bridging Technology

In implementing cross-architecture instruction embedding, we first constructed standardized vocabulary repositories for both X86 and ARM architectures. Although machine instructions consist of mnemonics and operands, the diverse representations of hexadecimal address variables pose challenges to comprehensive vocabulary coverage during training. To mitigate potential out-of-vocabulary (OOV) issues, we devised an instruction normalization preprocessing pipeline: (1) immediate values were uniformly replaced with the IMMV identifier; (2) hexadecimal addresses such as 0x3402c8 were converted to ADDRESS; (3) variable references were standardized to VAR; (4) function invocations were represented as CALL; (5) register references were systematically normalized according to a classification framework, with general-purpose 64-bit registers designated as reg_gen_64 and register pointers as reg_pointer.

After generating instruction embeddings, we applied an aggregation algorithm to integrate instruction sequence vectors into basic block representations. Concurrently, we leveraged the BERT pretrained models within the PalmTree framework to separately construct instruction embeddings for ARM and X86, obtaining basic block representations through identical aggregation functions. This dual-pathway design enabled us to evaluate the efficacy of ProtoBuf-based cross-architecture embedding by computing vector distances between basic block embeddings produced through both methods. The training loss and validation accuracy curves presented in Figure 4A demonstrate that performance metrics reached convergence after 20 iterations.

Figure 4B depicts the results of architecture-independent training, clearly showing that without ProtoBuf bridging technology, basic block embeddings from different architectures occupy distinctly separate vector subspaces. To further validate our ProtoBuf-based unified representation approach for cross-architecture analysis, we randomly selected 50 pairs of basic blocks with identical source code but different compilation architectures as positive-anchor sample pairs, quantified their similarity in embedding space using Euclidean distance, and calculated the distance variations between 50 negative samples and the anchor points. The results provide compelling evidence that our methodology effectively bridges semantic representations across different architectures, establishing a reliable foundation for cross-architecture firmware vulnerability detection.

4.2.2. Instruction Search Evaluation

We randomly selected a set of untrained assembly instruction basic blocks, with x86 and ARM samples each occupying 50%. We computed the embedding for each basic block (instruction sequences with only one entry and one exit) by averaging the instruction embeddings within. Given a basic block, we can find semantically equivalent basic blocks based on the cosine distance between two basic block embeddings. As shown in Figure 5A, we present the basic block search results using x86 basic blocks to search ARM instruction basic blocks and vice versa within the same ARM or x86 architecture. We plotted the ROC (receiver operating characteristic) curves for basic block search using embeddings from Instruction2Vec [33], Word2vec [19], Asm2Vec [18], PalmTree [7], and FirmPB.

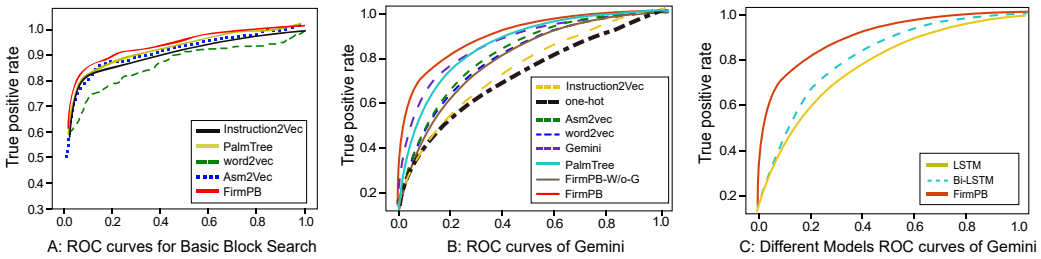

Figure 5. Experiments on downstream task.

The ROC curves in Figure 5A–C demonstrate significant differences in embedding performance across various models. Figure 5A illustrates that the FirmPB model substantially outperforms traditional approaches such as Instruction2Vec, word2vec, Asm2Vec, and PalmTree in basic block search tasks. The ROC curve's position closer to the upper left-hand corner indicates superior balance between true positive and false positive rates. This superior performance can be attributed to FirmPB's enhanced capability to capture semantic features of basic blocks, whereas traditional models exhibit limitations when processing complex structures in firmware binary code.

Figure 5B,C further validates FirmPB's advantages. In Figure 5B, FirmPB achieves higher detection accuracy within the Gemini framework compared to other embedding methods, particularly when contrasted with its FirmPB-W/o-G variant that lacks graph

structural information. These results demonstrate the critical importance of integrating graph structures for model performance. Figure 5C confirms FirmPB's significant advantage over conventional sequence models such as LSTM and Bi-LSTM, likely because FirmPB is better suited for handling nonlinear control flows and data dependencies in firmware code. This effectively reduces false positive rates while maintaining high true positive rates

From the ROC curves, it is possible to intuitively observe the ranking of area under curve (AUC) values for different embedding schemes. It can be seen that (1) word2vec again performs the worst; (2) the manually designed Instruction2Vec embedding outperforms even the automatically learned word2vec; (3) Asm2Vec and PalmTree perform quite well, but still lag behind FirmPB; (4) our proposed FirmPB instruction embedding model achieves better AUC than other baselines, showing consistent performance improvement.

4.2.3. Similarity Analysis of Instruction Embedding Vectors

In this study, we conducted a comparative evaluation of various instruction embedding methods and their collaborative effects using the Gemini model. The evaluated objects include Instruction2Vec [33], word2vec [19], Asm2Vec [18], FirmPB-W/o-G, and the complete FirmPB. We also introduced one-hot encoding combined with the embedding layer as a baseline method (labeled as "one-hot") while using the original Gemini basic block features as a control group (labeled as "Gemini").

As shown in Figure 4B, although Gemini demonstrated high AUC values in its original research, this might be attributed to overfitting phenomena resulting from data homogeneity, as both the training and testing sets consisted of OpenSSL code compiled by LLVM [32] under identical architectures. To comprehensively examine the model's adaptability to heterogeneous data, we constructed a more challenging testing environment by utilizing the LLVM compiler to process source code from Binutils 2.30, Coreutils 8.29, FFmpeg n3.2.13, OpenSSL 1.1.1b, and Redis 5.0.5, creating a diverse test set spanning both the x86 and ARM architectures.

Figure 4B presents the AUC performance of Gemini when using inputs generated by various models. Based on the experimental results, we derive the following key findings:

(1) Despite Gemini's impressive performance in the original literature, its generalization effectiveness is significantly insufficient in a novel data environment.
(2) Artificially designed embedding methods (including Instruction2Vec and one-hot vectors) perform poorly, indicating that manual features may excessively depend on specific application scenarios.
(3) FirmPB maintains excellent performance even when faced with test sets drastically different from training data, surpassing other solutions. This demonstrates that FirmPB can significantly enhance the general adaptability of downstream tasks.
(4) All three pretraining tasks contribute to FirmPB's final effectiveness. Notably, the complete FirmPB exhibits distinct advantages compared to its simplified version FirmPB-W/o-G, which experiences a decrease in performance to a level similar to Gemini (as shown in Figure 5B). This indicates that FirmPB achieves superior instruction embedding effects compared to traditional methods through its unique assembly instruction relationship generation algorithm.

4.3. Communication Cost of Vehicle-Side and RSU Trusted Communication Scheme

In our simulation, we analyzed the delay overhead introduced by the scheme across four distinct phases. First, the initialization phase encompasses the time required for the VA to generate identity marks and public–private key pairs for vehicle terminals and RSUs as well as the blockchain storage of these credentials. Second, in the pre-communication phase, vehicle terminals retrieve regional RSU public keys and both parties submit detection keys

to the blockchain for consistency verification. Third, the symmetric key negotiation phase between vehicle terminals and RSUs utilizes asymmetrically encrypted and signed random numbers that vehicle terminals decrypt with their private keys. Fourth, the firmware verification phase involves symmetric key encryption and decryption operations.

The detailed time consumption statistics across these phases are presented in Table 2. Our results show an average total overhead of approximately 40–50 milliseconds. Given that vehicle communication applications typically require delays under 50 milliseconds, we evaluated our scheme against the five business scenario categories defined by international IoV networking organizations and 3GPP [34] specifications (Table 3). The findings demonstrate that our trusted communication scheme's approximately 50-millisecond delay satisfies the performance requirements for most VANET application scenarios.

Table 2. Delay testing of the four parts in the trusted communication scheme.

Times	Initialization Scheme (s)	Generate Pk/Sk (s)	Communications By Pk (s)	Instruction Embedding (s)	Total Delay (s)
1	0.000301	0.006618	0.000425	0.041301	0.048645
2	0.000271	0.003636	0.000586	0.052277	0.05677
3	0.000266	0.004524	0.000625	0.0460287	0.0514437
4	0.000628	0.004308	0.00062	0.0410363	0.0465923
5	0.000291	0.005184	0.000441	0.0210282	0.0269442
6	0.000301	0.005001	0.000402	0.0390221	0.0447261
7	0.000297	0.004869	0.000221	0.0380332	0.0434202
8	0.000311	0.004422	0.000441	0.040233	0.045407
9	0.000456	0.004367	0.000543	0.042299	0.047665
10	0.000324	0.003635	0.00045	0.033361	0.03777
avg.	0.0003446	0.0046564	0.0004754	0.03946195	0.04493835

Table 3. Reliability testing in different application scenarios.

Scenario	Effective Range/(m)	Absolute Speed/(km/h)	Relative Velocity/(km/h)	Maximum Latency/(ms)	Receiving Reliability
suburb	200	50	100	100	90%
highway1	320	160	280	100	90%
highway2	320	280	280	100	80%
NLOS/City	100	50	100	100	90%
Urban intersection	50	50	100	100	95%
Campus/business district	50	30	30	100	90%
Emergency collision	20	80	160	20	95%

The implementation leverages protocol buffers (PB) for certificate authentication, delivering significant advantages over traditional serialization methods. This approach substantially improves data serialization efficiency, thereby reducing processing overhead and network transmission times. Additionally, the PB structure provides enhanced development flexibility and maintainability through its strongly-typed schema definition, version compatibility, and cross-platform support. These benefits are particularly valuable in the dynamic VANET environment, where efficient message processing and adaptable system architecture are critical. Consequently, our PB-based secure communication privacy scheme not only meets the stringent performance demands of vehicular networks but also offers superior development efficiency and system scalability.

4.4. Security Evaluation of Trusted Communication Scheme

For implementation on memory-limited IoV hardware, we investigated LSTM architectures as potential replacements for BERT in instruction embedding generation, motivated by their reduced memory requirements and enhanced convergence efficiency [13,14]. Our eval-

uation encompassed instruction retrieval experiments and vector similarity analysis across downstream applications. The storage footprint comparisons are visualized in Figure 4C. Despite Bi-LSTM and standard LSTM configurations demonstrating storage advantages relative to FirmPB [13–15], their performance metrics reveal significant shortcomings. The results presented in Figure 5C indicate substantial deficiencies in ROC measurements, with performance falling below acceptable thresholds compared to FirmPB-based approaches. This performance gap precludes direct deployment of these lightweight architectures for on-vehicle firmware vulnerability assessment. Consequently, we developed a vehicle-to-infrastructure communication framework as an alternative solution addressing current technical limitations while fulfilling automotive network security requirements. The subsequent sections provide comprehensive security and performance analyses of our proposed communication protocol.

To ensure the rigor of the security research in this study, the following security assumptions are made about the test environment. Regarding the environment, we assume that the IoV network infrastructure is fully equipped, i.e., that the RSUs and security modules are complete and have no hardware deficiencies. This security scheme prioritizes security guarantees; any scheme process failures due to hardware reasons will disable inter-vehicle communications or trigger other security workflows.

First, we assume that the wired or wireless connections between trusted authorities in this paper are managed by dedicated personnel and are highly trusted, serving as trust anchors. Second, we assume all vehicles undergo relevant departmental inspections when entering the communication environment, with no illegal hardware modifications. Finally, we assume that all participating vehicle-side and RSUs communicate following the designed communication security and privacy scheme in this paper. To test the security of the simulation scheme in this study, we investigated security threats in IoV network communications. Research shows there are five main types of vulnerabilities: (1) eavesdropping, (2) impersonation, (3) tampering, and (4) repudiation (the fifth type, man-in-the-middle, is excluded from this analysis).

(1) In the simulation, eavesdropping was tested by intercepting communication between two vehicle nodes to determine whether the security scheme avoids such threats. Simulations in NS3 showed that because communications between vehicles are encrypted regionally and messages are encrypted with the recipient's public key when sent, only the private key can decrypt them; thus even if intercepted, the encrypted information cannot be directly read. Moreover, key negotiation in this scheme uses blockchain to verify public keys instead of certificate validation, avoiding centralized data tampering issues while skipping certificate verification steps.

(2) Impersonation is tested by falsifying the identity of a vehicle node during communications to determine whether the scheme prevents such threats. In this scheme, vehicles first obtain certified public keys corresponding to their real identities from the TA in CS. The blockchain-verified public keys are then used for communications. Prior to sending, the sender encrypts a random number (agreed symmetric key) with its private key. When an attacker impersonates a vehicle or RSU during ongoing communications, the receiver performs decryption (signature verification) using the stored public key and detect fakes via comparison against the blockchain. In addition, falsified messages cannot be decrypted using the receiver's public key.

(3) Tampering is tested by intercepting and altering messages. In this scheme, regional encryption, signed messages, and authenticated entities prevent such threats. In simulations, malicious third parties cannot obtain plaintexts to tamper with due to lacking the regional symmetric key after intercepting encrypted messages.

(4) Repudiation is tested by denying sent messages in order to verify the scheme's capabilities. In this scheme, vehicle messages are digitally signed with the vehicle's private key and pseudonym certificates are logged with the administration entity, allowing a lookup table to be queried in order to prove the sender and prevent repudiation.

4.5. Scalability Comparison of Firmware Analysis Methods

We conducted a systematic comparison of three state-of-the-art firmware analysis methods: FIRMRCA [35], POMP [36], and POMP++ [37], in Table 4. These methods were selected based on their shared characteristics with FirmPB; all consider instruction semantics, providing analysis results with high instructional value for analysts. POMP performs backward taint analysis on core dump files and execution traces collected by Intel PT [38] to precisely identify firmware analysis causes. POMP++ further enhances POMP's memory alias resolution capabilities by introducing value set analysis techniques. Although these methods were originally designed for x86 Linux platforms, we have ported them to the 32-bit ARM Cortex-M architecture to ensure fairness and scientific validity in our evaluation.

Table 4. Performance Comparison of Firmware Analysis Methods Across Various Input Sizes (in instructions).

Model	800	1600	3200	6400	12,800
FirmPB	35.44 s	76.26 s	150.79 s	300.34 s	630.26 s
FirmRCA [35]	43.78 s	122.70 s	348.12 s	2.1 h	4.7 h
POMP [36]	53.78 s	138.45 s	331.62 s	1.8 h	5.7 h
POMP++ [37]	63.65 s	141.32 s	1.1 h	3.1 h	5.7 h

To quantify performance differences, we randomly selected five test cases containing over 180,000 instructions and calculated the average total time overhead. The experimental results demonstrate that FIRMRCA and POMP exhibit clear exponential growth trends as analysis depth increases linearly, while POMP++ shows polynomial-level growth characteristics. In contrast, FirmPB displays only moderate time complexity growth as the analysis depth increases. This significant performance difference conclusively proves FirmPB's stability and efficiency during the instruction embedding phase. By minimizing the impact of unresolved memory aliases, FirmPB can efficiently perform fault localization even with extremely long execution traces. This capability is particularly crucial in complex firmware crash scenarios, where firmware analyses are often deeply embedded within lengthy execution traces and can be difficult to quickly locate.

In the field of firmware disassembly analysis, fuzzy matching frameworks are widely employed to identify similar code fragments across different binary files even when these code segments exhibit variations due to compilation or optimization processes. These traditional approaches primarily rely on techniques such as instruction sequence similarity, control flow graph features, graph isomorphism algorithms, and edit distance metrics. However, these conventional methods typically demonstrate exponential complexity growth as analysis depth increases and face significant challenges when handling unresolved memory aliases.

In contrast, the FirmPB approach proposed in this paper utilizes deep learning-based semantic embedding methods, which demonstrate substantial advantages. The proposed approach not only captures implicit semantic relationships between instructions but also exhibits only moderate time complexity growth as analysis depth increases. Furthermore, FirmPB demonstrates superior performance in addressing memory alias issues and presents

enhanced cross-architecture generalization capabilities, enabling comprehensive understanding of deep code semantics rather than merely identifying surface patterns. These characteristics provide FirmPB with significant advantages in complex firmware analysis tasks, particularly in vulnerability detection and firmware analysis scenarios.

4.6. Discussion on Balancing Accuracy and Efficiency

In firmware vulnerability detection, achieving an optimal balance between accuracy and efficiency represents a critical challenge that warrants further exploration beyond the current analysis. The ideal balance involves maximizing detection accuracy while minimizing computational overhead and time consumption, which is particularly crucial in resource-constrained IoV environments.

An ideal balance would enable real-time vulnerability detection with high precision, allowing FirmPB to maintain its superior AUC performance (0.932) while operating within the 50 ms latency threshold required for most VANET applications. This would allow for comprehensive firmware security verification without compromising vehicle communication requirements. The integration of protocol buffers demonstrates a step toward this balance by improving serialization efficiency; however, the relationship between model complexity and detection performance requires deeper examination.

When this balance is suboptimal, several consequences emerge. Prioritizing accuracy over efficiency (as seen with the full FirmPB implementation versus lightweight alternatives) may result in systems that, while highly accurate, cannot operate within vehicular timing constraints. This necessitates vehicle-to-infrastructure offloading solutions such as those implemented in this study. Conversely, emphasizing efficiency at the expense of accuracy (as demonstrated by the LSTM and Bi-LSTM models) produces systems with performance falling below acceptable thresholds, creating dangerous security gaps where vulnerabilities remain undetected despite timely processing. Our experimental results clearly illustrate this tradeoff, showing that lightweight models achieve better efficiency but exhibit substantial deficiencies in ROC measurements that preclude their direct deployment for on-vehicle firmware vulnerability assessment.

5. Conclusions

This research presents an innovative solution to address critical challenges in firmware vulnerability detection for embedded devices. The paper establishes a firmware vulnerability detection framework based on statistical inference and code similarity fuzzy matching analysis within resource-constrained vehicular network environments, successfully achieving a balance between detection accuracy and computational efficiency. The main contributions of this research can be summarized in three aspects. First, we develop a normalized segmentation and semantic equivalence mapping mechanism for binary code through dynamic programming and neighborhood search techniques. This effectively extracts similarity metrics for function execution semantics, resolving the accuracy limitations of traditional instruction embedding methods that neglect program execution semantics. Second, we innovatively introduce Google Protocol Buffers (ProtoBuf) as a serialization format for inter-model data transmission, serving as a "translation layer" and "bridging technology" within the detection framework. In addition, we propose a ProtoBuf-based certificate authentication scheme that significantly enhances vehicular network communication reliability and data serialization efficiency. Finally, through secondary development on the NS-3 network simulator, we construct a vehicular network simulation environment to comprehensively validate the functionality and performance characteristics of the proposed architecture. Experimental results demonstrate that the method proposed in this research not only possesses resistance capabilities against common security threats but

also minimizes performance impact. This achievement holds significant implications for enhancing firmware security in resource-constrained environments and provides a new technical approach for vulnerability detection in embedded systems. Future research could further explore the adaptability of detection methods across more heterogeneous platforms, along with advanced semantic analysis techniques combined with deep learning to further improve the accuracy and generalization capabilities of the detection framework.

Author Contributions: Conceptualization, X.F.; methodology, X.F. and K.H.; software, K.H.; validation, X.F., Y.W. and R.C.; formal analysis, X.F. and K.H.; investigation, Y.W. and J.Z.; resources, X.F.; data curation, K.H. and R.C.; writing—original draft preparation, X.F. and Y.W.; writing—review and editing, X.F., K.H. and J.Z.; visualization, K.H. and R.C.; supervision, X.F.; project administration, X.F.; funding acquisition, X.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available at https://dx.doi.org/10.14722/bar.2020.23002, accessed on 5 July 2025.

Conflicts of Interest: Xiyu Fang is employed by CATARC Automotive Test Center (Tianjin) Co., Ltd. and China Automotive Technology & Research Center Co., Ltd. and has not received funding from each of these companies. CATARC Automotive Test Center (Tianjin) Co., Ltd., as a company, is not involved in this research, and has no potential conflicts of interest related to the study itself. China Automotive Technology & Research Center Co., Ltd., as a company, is not involved in this research, and has no potential conflicts of interest related to the study itself. Kexun He is employed by CATARC Automotive Test Center (Tianjin) Co., Ltd. and has not received funding from CATARC Automotive Test Center (Tianjin) Co., Ltd. CATARC Automotive Test Center (Tianjin) Co., Ltd., as a company, is not involved in this research, and has no potential conflicts of interest related to the study itself. Yue Wu is employed by the University of Electronic Science and Technology of China and has not received funding from the university. The University of Electronic Science and Technology of China, as an institution, is not involved in this research, and has no potential conflicts of interest related to the study itself. Rui Chen and Jing Zhao are employed by Dalian University of Technology and have not received funding from the university. Dalian University of Technology, as an institution, is involved in this research, and has no potential conflicts of interest related to the study itself.

References

1. Zhao, J.; Wang, R. Fedmix: A sybil attack detection system considering cross-layer information fusion and privacy protection. In Proceedings of the 2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), Virtual Conference, 20–23 September 2022; pp. 199–207.
2. Chen, R.; Younas, W.; Zhao, J. Firm-vehicle: Trusted communication enabled instruction embedding model for resource-constrained vanet environments. In Proceedings of the 20th International Conference, ICIC 2024, Tianjin, China, 5–8 August 2024; pp. 391–402.
3. Ferguson, J. *Reverse Engineering Code with IDA Pro*; Syngress: Rockland, MA, USA, 2008.
4. Feng, X.; Zhu, X.; Han, Q.-L.; Zhou, W.; Wen, S.; Xiang, Y. Detecting vulnerability on iot device firmware: A survey. *IEEE/CAA J. Autom. Sin.* **2022**, *10*, 25–41.
5. Chen, H.; Liu, J.; Wang, J.; Xun, Y. Towards secure intra-vehicle communications in 5g advanced and beyond: Vulnerabilities, attacks and countermeasures. *Veh. Commun.* **2023**, *39*, 100548.
6. Kim, G.; Hong, S.; Franz, M.; Song, D. 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, Virtual Conference, 18–22 July 2022; pp. 151–163.
7. 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, Virtual Conference, 15–19 November 2021; pp. 3236–3251.

8. Zhang, X.; Sun, W.; Pang, J.; Liu, F.; Ma, Z. Similarity metric method for binary basic blocks of cross-instruction set architecture. In Proceedings of the 2020 Workshop on Binary Analysis Research, San Diego, CA, USA, 23 February 2020; Volume 10.
9. Park, J.; Lee, S.; Hong, J.; Ryu, S. Static analysis of jni programs via binary decompilation. *IEEE Trans. Softw. Eng.* **2023**, *49*, 3089–3105.
10. Song, Q.; Zhang, Y.; Wang, B.; Chen, Y. Inter-bin: Interaction-based cross-architecture iot binary similarity comparison. *IEEE Internet Things J.* **2022**, *9*, 20018–20033.
11. Riley, G.F.; Henderson, T.R. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 15–34.
12. Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; Song, D. 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, Dallas, TX, USA, 30 October–3 November 2017; pp. 363–376.
13. Tang, R.; Lu, Y.; Liu, L.; Mou, L.; Vechtomova, O.; Lin, J. Distilling task-specific knowledge from bert into simple neural networks. *arXiv* **2019**, arXiv:1903.12136.
14. Lin, J.; Liu, Z.; Wang, H.; Han, S. Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 784–800.
15. Park, S.; Choi, W. Regulated subspace projection based local model update compression for communication-efficient federated learning. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 964–976.
16. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
17. Duan, Y.; Li, X.; Wang, J.; Yin, H. Deepbindiff: Learning program-wide code representations for binary diffing. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2020.
18. Ding, S.H.; Fung, B.C.; Charland, P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–22 May 2019; pp. 472–489.
19. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 27th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; Volume 26.
20. Chua, Z.L.; Shen, S.; Saxena, P.; Liang, Z. Neural nets can learn function type signatures from binaries. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 99–116.
21. Zuo, F.; Li, X.; Young, P.; Luo, L.; Zeng, Q.; Zhang, Z. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv* **2018**, arXiv:1808.04706.
22. Chen, S.; Hu, J.; Shi, Y.; Peng, Y.; Fang, J.; Zhao, R.; Zhao, L. Vehicle-to-everything (v2x) services supported by lte-based systems and 5g. *IEEE Commun. Stand. Mag.* **2017**, *1*, 70–76.
23. Sun, H.; Sun, M.; Weng, J.; Liu, Z. Analysis of id sequences similarity using dtw in intrusion detection for can bus. *IEEE Trans. Vehicular Technol.* **2022**, *71*, 10426–10441.
24. Molina-Masegosa, R.; Gozalvez, J.; Sepulcre, M. Configuration of the c-v2x mode 4 sidelink pc5 interface for vehicular communication. In Proceedings of the 2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), Shenyang, China, 6–8 December 2018; pp. 43–48.
25. Tan, H.; Zheng, W.; Vijayakumar, P.; Sakurai, K.; Kumar, N. An efficient vehicle-assisted aggregate authentication scheme for infrastructure-less vehicular networks. *IEEE Trans. Intell. Transp.* **2022**, *24*, 15590–15600.
26. Yang, Y.; Wei, L.; Wu, J.; Long, C.; Li, B. A blockchain-based multidomain authentication scheme for conditional privacy preserving in vehicular ad-hoc network. *IEEE Internet Things J.* **2021**, *9*, 8078–8090.
27. Adil, M.; Ali, J.; Attique, M.; Jadoon, M.M.; Abbas, S.; Alotaibi, S.R.; Menon, V.G.; Farouk, A. Three byte-based mutual authentication scheme for autonomous internet of vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 9358–9369.
28. Zhou, Y.; Cao, L.; Qiao, Z.; Xia, Z.; Yang, B.; Zhang, M.; Zhang, W. An efficient identity authentication scheme with dynamic anonymity for vanets. *IEEE Internet Things J.* **2023**, *10*, 10052–10065.
29. Blyth, D.; Alcaraz, J.; Binet, S.; Chekanov, S.V. ProIO: An event-based I/O stream format for protobuf messages. *Comput. Phys. Commun.* **2019**, *241*, 98–112.
30. *IEEE Std 1609.0-2019 (Revision of IEEE Std 1609.0-2013)*; IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture. IEEE: Piscataway, NJ, USA, 2019; pp. 1–106.
31. Guo, W.; Mu, D.; Xing, X.; Du, M.; Song, D. {DEEPVSA}: Facilitating value-set analysis with deep learning for postmortem program analysis. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1787–1804.

32. Lattner, C.; Adve, V. LLVM: A compilation framework for lifelong program analysis and transformation. In Proceedings of the International Symposium on Code Generation and Optimization, 2004. CGO 2004, San Jose, CA, USA, 20–24 March 2004; pp. 75–88.
33. Lee, Y.J.; Choi, S.-H.; Kim, C.; Lim, S.-H.; Park, K.-W. Learning binary code with deep learning to detect software weakness. In Proceedings of the KSII the 9th International Conference on Internet (ICONI) 2017 Symposium, Vientiane, Laos, 17–20 December 2017.
34. 3GPP TR22. 886 V16. 2.0. Study on Enhancement of 3GPP Support for 5G V2X Services. 2018. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3108 (accessed on 5 July 2025).
35. Chang, B.; Zhao, B.; Zhang, Q.; Liu, P.; Tian, Y.; Beyah, R.; Ji, S. FirmRCA: Towards Post-Fuzzing Analysis on ARM Embedded Firmware with Efficient Event-based Fault Localization. In Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 12–14 May 2025; pp. 3783–3800.
36. Xu, J.; Mu, D.; Xing, X.; Liu, P.; Chen, P.; Mao, B. Postmortem program analysis with hardware-enhanced post-crash artifacts. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 17–32.
37. Mu, D.; Du, Y.; Xu, J.; Xu, J.; Xing, X.; Mao, B.; Liu, P. POMP++: Facilitating postmortem program diagnosis with value-set analysis. *IEEE Trans. Softw. Eng.* **2019**, *47*, 1929–1942.
38. Intel. Collecting Intel® Processor Trace (Intel Pt) in Intel® System Debugger. Available online: https://www.intel.com/content/www/us/en/developer/videos/collecting-processor-trace-in-intel-system-debugger.html?wapkw=intel%20pt (accessed on 5 July 2025).