applied sciences

MDPI

Article

TMVDPatch: A Trusted Multi-View Decision System for Security Patch Identification

Xin Zhou [1], Jianmin Pang [1,*], Zheng Shan [1,2], Feng Yue [1,*], Fudong Liu [1], Jinlong Xu [1], Junchao Wang [1], Wenfu Liu [1,3] and Guangming Liu [1]

[1] State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450000, China; qf_zhouxin@126.com (X.Z.)
[2] Songshan Laboratory, Zhengzhou 450000, China
[3] State Key Laboratory of Complex Electromagnetic Environment Effects on Electronics and Information System, Luoyang 471000, China
* Correspondence: jianmin_pang@hotmail.com (J.P.); firstchoiceyf@sina.com (F.Y.)

Abstract: Nowadays, the time lag between vulnerability discovery and the timely remediation of the vulnerability is extremely important to the current state of cybersecurity. Unfortunately, the silent security patch presents a significant challenge. Despite related work having been conducted in this area, the patch identification lacks interpretability. To solve this problem, this paper first proposes a trusted multi-view security patch identification system called TMVDPatch. The system obtains evidence from message commit and code diff views respectively, and models the uncertainty of each view based on the D-S evidence theory, thereby providing credible and interpretable security patch identification results. On this basis, this paper performs weighted training on the original evidence based on the grey relational analysis method to improve the ability to make credible decisions based on multi-views. Experimental results show that the multi-view learning method exhibits excellent capabilities in terms of the complementary information provided by control dependency and data dependency, and the model shows strong robustness across different hyperparameter settings. TMVDPatch outperforms other models in all evaluation metrics, achieving an accuracy of 85.29% and a F1 score of 0.9001, clearly verifying the superiority of TMVDPatch in terms of accuracy, scientificity, and reliability.

Keywords: security patch; multi-view learning; evidential deep learning; grey relational analysis

Citation: Zhou, X.; Pang, J.; Shan, Z.; Yue, F.; Liu, F.; Xu, J.; Wang, J.; Liu, W.; Liu, G. TMVDPatch: A Trusted Multi-View Decision System for Security Patch Identification. *Appl. Sci.* **2023**, *13*, 3938. https://doi.org/10.3390/app13063938

Academic Editor: Jose Machado

Received: 20 January 2023
Revised: 2 March 2023
Accepted: 3 March 2023
Published: 20 March 2023

1. Introduction

In recent years, cyber attacks have emerged in a constant stream within turbulent network environment, and software vulnerabilities are still the main threat to network security. The most effective way to address this threat is still to patch the vulnerability. According to the Snyk report [1], approximately 35% of the vulnerabilities are fixed within 20 days, while 36% of the vulnerabilities require 70 days or more, resulting in an average resolution time of 68 days. At the same time, its 2022 Open Source Security Trends Analysis [2] finds that supply chain attacks are increasingly prevalent. The entire supply chain is an attractive attack vector because cyber attackers can attack vulnerabilities in development pipelines without changing software repositories. This means that after vulnerability discovered, it often requires a lot of time for vulnerability patch, and the security threat brought by the response time to address vulnerability is greatly amplified in software supply chain attacks.

For instance, a high impact remote code execution vulnerability (CVE-2021-44228 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228 (accessed on December 2021)) was disclosed in Apache Log4j2 in 2021. Apache Log4j is the foundational logging component, with widespread use across numerous applications. According to the armis report [3], attackers around the world have been trying to exploit the vulnerability since its discovery. Numerous vendors have observed attempts to distribute coin miners,

ransomware, remote access Trojans, web shells, and botnet malware. In just one week, approximately 35% of users were active attacks through the vulnerability, and 31% were exposed to Log4j-related threats on unmanaged devices. The security vendor said it had observed as many as 30,000 attack attempts against its users. Other vendors have reported similar activity. At the same time, according to the armis report [4], the Log4j versions released from 2013 to 2021 have vulnerabilities, the library is ubiquitous. To go a step further, it may take years to get rid of the danger, and a large number of programs may never be patched.

Therefore, the time difference between vulnerability discovery and actual vulnerability patch is extremely important for the current state of network security. After the manufacturer publishes the announcement of the vulnerability patch, the developers often package and upload the code changes with descriptions through submission. Ideally, after the vendor releases the patch, other projects involving this vulnerability will be updated with the patch in a timely manner. But the reality is that there is frequently a significant delay between the actual fix and the release of the patch. In order to fix the vulnerability, the maintainers of each project need to monitor the patch announcement released by the manufacturer, so as to complete the emergency patching in time.

Unfortunately, the silent fix of the vulnerability poses a significant challenge. Out of fear of reputation and the security threat brought by the vulnerability disclosure, vendors often secretly implement vulnerability fixes without assigning the CVE numbers, or even avoid descriptions of vulnerabilities in the commits. According to GitHub's 2019 report [5], about 7.6 million security alerts were fixed, primarily through code changes. Meanwhile, only 12,174 CVEs were disclosed in the same year. As a result, a large number of security patches will not be associated with CVE(Common Vulnerabilities and Exposures) numbers, but will be silently fixed without official reports. Despite this situation, attackers can still directly analyze the code differences to find related vulnerabilities and develop tools for secret exploitation. However, in addition to the security patches, there are a large number of non-security patches in the code changes submitted by the vendors. Additionally, non-security patches may include bug fixes and functional patches. Table 1 summarizes the related work in the security patch identification.

Table 1. The related work in the security patch identification.

Ref	Year	Program Language	Labeled Data	Open Source Dataset	Open Source Code	Method	Object of Study
[6]	2019	C/C++	3272	✗	✗	Heuristics	Code Diff
[7]	2021	C/C++	40,523	Part	✗	Deep learning	Commit Message & Code Diff
[8]	2020	C/C++	9247	✓	✓	Heuristics	Commit Message & Code Diff
[9]	2018	JAVA	2715	✓	✗	Classic Machine Learning	Commit Message & Code Diff
[10]	2021	JAVA	1950	Part	✗	Deep learning	The prior and posterior versions of source code in the commit
[11]	2020	C/C++	82,403	Invalidation	✓	Deep learning	code diff
[12]	2019	C/C++	82,403	Invalidation	✓	Deep learning	Commit Message & Code Diff
[13]	2021	C/C++	38,041	✓	✓	Deep learning	Commit message & The prior and posterior versions of source code in the commit

Table 1. *Cont.*

Ref	Year	Program Language	Labeled Data	Open Source Dataset	Open Source Code	Method	Object of Study
[14]	2020	C/C++	341,767	✗	✗	Symbolic Interpretation	The prior and posterior versions of source code in the commit
[15]	2021	C/C++	3,329,286 Sample Pairs	✗	✗	Heuristics	Commit Message & Code Diff

Therefore, how to efficiently identify security patches for vulnerability fixes in a large number of code change submissions is crucial. To solve this problem, this paper proposes an efficient and reliable security patch identification system called TMVDPatch. In this paper, we propose a multi-view decision system for security patch identification that accounts for classification uncertainty using D-S(Dempster-Shafer) evidence theory. The method parses the control dependency graph (CDG) and data dependency graph (DDG) of the source code, and combines the graph diff with commit message to capture the semantic information and syntax information of the code change. And then, the system uses the results as evidence in the multi-view learning framework, and determines the relevance of different views as the weight of evidence input according to the Grey Relational Analysis method with self-learning correction. Using this approach to self-learning and correction, we can improve the ability of trusted multi-view decision-making.

In order to effectively evaluate the identification ability of the system, we conducted related experiments on the large-scale real patch dataset called PatchDB [16]. The experimental results show that we can achieve a comprehensive accuracy of 85.29%, and the F1 score is 0.9001. At the same time, the precision is 85.42% and the recall is 95.11%.

In summary, the contributions of this work are summarized below:

1. We first propose a trusted multi-view security patch identification system called TMVDPatch. The system adopts a multi-view evidence fusion strategy. Based on the subjective uncertainty learned from each view, the evidence is weighted by the Grey Relational Analysis method to provide credible and interpretable decisions, thereby achieving efficient and scientific security patch identification.
2. We propose a representation method for the semantic and syntactic information of security patches. The method regards the multi-view evidence, including control dependency information, data dependency information and description information, so as to comprehensively represent the semantic and syntax information of security patches. Compared to learning from commit message alone, the multi-view representation method shows an excellent ability in control-dependent and data-dependent complementary information.
3. We conduct a large number of experiments on the model hyperparameter settings and importance evaluation. The results demonstrate the capability boundary of TMVDPatch and the robustness of the model to the hyperparameter settings.
4. We conduct related experiments on PatchDB, a large-scale real patch dataset. The results show that the TMVDPatch model outperforms other models in all metrics, achieving an accuracy of 85.29% and a F1 score of 0.9001. The superiority of TMVDPatch in terms of accuracy, scientificity and reliability is clearly verified.

2. Background and Related Work

2.1. Security Patch Identification

A software patch is a record of changes made to source code between two versions, and the patch file is generated by the diff command. A patch file consists of "blocks" of code modifications (parts prefixed with @@) that indicate the location of modifications, which represent code patches by recording additions and deletions of the code lines. The two

main types of software patches are security patches and non-security patches. Non-security patches include primarily bug fixes and functional patches. Bug fix patches are mainly used for program optimization, and functional patches are mainly used to add new functions.This paper focuses on identifying security patches that fix program vulnerabilities.

In order to clearly illustrate the security patch and the non-security patch, we provide the corresponding cases in Listings 1 and 2. Listing 1 is an example of security patch for an integer underflow vulnerability (CVE-2019-13602), which prevents a vulnerability related to integer underflow in function MP4_EIA608_Convert by modifying the constraints of the i_bytes local variable (lines 16 and 17). Listing 2 shows a non-security patch in the FFmpeg software (9e588125193115189b5a609eef6af678a55f6ecf). This patch enables FFmpeg to automatically call the check_bitstream interface when running writing_frame (the call to check_bitstream is encapsulated in the do_packet_auto_bsf), thereby improving the running efficiency of FFmpeg.

Listing 1. Example of security patch for an integer underflow vulnerability (CVE-2019-13602).

```
1  diff --git a/modules/demux/mp4/mp4.c b/modules/demux/mp4/mp4.c
2  index 77b46de1c3..83f36db1a7 100644
3  --- a/modules/demux/mp4/mp4.c
4  +++ b/modules/demux/mp4/mp4.c
5  @@ -536,10 +536,10 @@ static block_t * MP4_EIA608_Convert( block_t * p_block )
6       } while( i_bytes >= 2 );
7
8
9       /* cdt2 is optional */
10 -     if ( i_remaining >= 10 &&
11 -          (i_bytes = GetDWBE(p_read)) &&
12 -          (i_bytes <= i_remaining) &&
13 -          !memcmp("cdt2", &p_read[4], 4) )
14 +     i_bytes = GetDWBE(p_read);
15 +
16 +     if (10 <= i_bytes && i_bytes <= i_remaining &&
17 +         !memcmp("cdt2", &p_read[4], 4))
18       {
19           p_read += 8;
20           i_bytes -= 8;
```

At present, related work has been done on the identification of security patches. This paper summarizes the related work in this field in recent years' study in Table 1.

Traditional security patch identification mainly adopts heuristic method. Wang et al. [6] propose the effective feature for distinguishing security patches and non-security patches. There are 61 features, including basic features, syntax features and semantic features. On this basis, they use a voting algorithm to integrate five classic classification algorithms, Random Forest, Bayes Net, SGD (Stochastic Gradient Descent), SMO (Sequential Minimal Optimization), and Bagging. The experimental results show that the model has a TPR (True positive rate) of 79.6% and an FPR (False positive rate) of 41.3%. Sawadogo et al. [8] achieve security patch identification using commit message and code diff information for identification. Specifically, they propose a semi-supervised method with co-training to train two classifiers with commit messages and code changes, respectively, addressing the unlabeled problem of patch data. Tan et al. [15] propose a new approach to transform the security patch localization search problem into a ranking problem. Therefore,they propose a new technique for ranking commits related to vulnerabilities and implement a ranking-based security patch localization method called PatchScout.

With the rise of artificial intelligence, recent research on security patch identification primarily uses machine learning methods. Sabetta and Bezzi [9] treat code changes as documents written in natural language and classifies them using standard text classification methods. Hoang et al. [12] propose a patch classification tool based on hierarchical deep learning called PatchNet.They learn relevant features from commit message and code diff, with experimentally verified in Linux kernel patch identification. In order to obtain a deep understanding of the vector representation of code changes, Hoang et al. [11] propose a neural network model based on deep learning called CC2Vec. They use an attention

mechanism to model the hierarchy of code changes, producing a distributed representation of code diff. Zhou et al. [7] propose a security patch identification system based on deep learning called SPI. The system consists of two neural networks: one is a neural network for commit message, which uses commit message for word vector pre-training; the other is a neural network for code diff, which uses subtractive code change and additive code change to learn.

Listing 2. An example of non-security patch in FFmpeg.

```
diff --git a/libavformat/mux.c b/libavformat/mux.c
index 5cb0ca7482..d674bd4a76 100644
--- a/libavformat/mux.c
+++ b/libavformat/mux.c
@@ -893,6 +893,10 @@ int av_write_frame(AVFormatContext *s, AVPacket *pkt)
         return 1;
     }

+    ret = do_packet_auto_bsf(s, pkt);
+    if (ret <= 0)
+        return ret;
+
 #if FF_API_COMPUTE_PKT_FIELDS2 && FF_API_LAVF_AVCTX
     ret = compute_muxer_pkt_fields(s, s->streams[pkt->stream_index], pkt);
```

However, the above work focuses solely on code diff and does not consider the overall information of the source code before and after modification. To solve this problem, Cabrera Lozoya et al. [10] propose a new method called Commit2Vec based on code2vec [17] to capture code change representation from the source code. By analyzing the prior and posterior versions of source code in the commit, the method compares the differences observed in the abstract syntax tree (AST) before and after the code change, and implements the vector representation of AST changes based on neural networks. At the same time, Wang et al. [13] also propose a security patch identification system called PatchRNN. They use the TextRNN model to obtain the vector representation of commit message. And then, they use the prior and posterior versions of source code in the commit to learn code change representation. Finally, they aggregate them and realize the identification of security patches based on a fully connected network.

In addition to heuristic methods and machine learning, Machiry et al. [14] propose a formal method for security patch identification based on symbolic interpretation. They define security patches from a formal perspective for the first time. Moreover, they propose a tool for security patch identification called SPIDER, which performs security patch identification through the source code of the original version and the patched version.

Therefore, based on the analysis of the above work, the key of source code patch analysis is to learn the syntax and semantics of code patches, rather than simply treating them as ordinary text. Learning the control and data dependencies in the source code is crucial for obtaining the essential information of security patches. This paper utilizes commit message and source code of the original version and the patched version to capture a more comprehensive patch representation. In our work, we parse the control dependency graph and data dependency graph of the source code, and combine the graph diff with commit message to capture the semantic information and syntax information of the code change.

At the same time, the key to machine learning-based security patch identification work is a modest dataset labelled with ground truth. As PatchRNN [13] uses the same input information and conducts experiments with the public dataset. In our work, we select the same dataset to carry out related experiments.

2.2. Multi-View Learning

In recent years, multi-view learning has achieved information complementarity by fusing features from multiple modes, and has been widely used in practical work. Typical algorithms include co-training mechanism [18], subspace learning methods [19] and multiple kernel learning (MKL) [20].

The traditional representative method for multi-view learning is Canonical correlation analysis (CCA). CCA seeks to maximize the correlation between views by linearly transforming the data of different views. Traditional CCA can only cope with linear dependencies between views. To solve this problem, related works have proposed nonlinear extensions of CCA, such as kernel CCA [20–23] and deep CCA [24]. Kernel CCA, as a typical nonlinear CCA algorithm, mainly maps low-dimensional data to a high-dimensional kernel function space, and performs correlation analysis in the high-dimensional space through the kernel function. However, although the kernel CCA solves the nonlinear problem, the model is complex and the training cost is large, so Andrew et al. [24] propose the deep CCA. Specifically, deep CCA learns nonlinear correlations between different views by combining DNN(Deep Neural Networks) and CCA. Based on it, Wang et al. [25] propose DCCAE(deep canonically correlated autoencoders) in combination with autoencoders, which consists of two autoencoders and optimizes the combination of canonical correlations between the learned representation and the reconstruction errors of the autoencoders.

Although the above methods have achieved multi-view classification, they do not consider the reliability of the classification results. Han et al. [26,27] propose a trustworthy multi-view classification method that provides interpretability for multi-view classification results. However, the proposed method does not analyze the correlation between different views, which may ignore some important information or introduce some redundant information. The correlation analysis between different views refers to evaluating the interrelationship between the information provided by different views, such as whether there are complementarity, conflict or duplication. This can help select the most useful views or adjust their weights, thereby improving the classification performance. In this paper, we introduce Grey Relational Analysis to dynamically adjust the weights between views, which can speed up model convergence and increase accuracy. Meanwhile, after combining Grey Relational Analysis, we can quantify the weights between views and further evaluate the credibility of multi-view learning.

Therefore, on this basis, we implement a trusted multi-view security patch identification system called TMVDPatch. In our work, we use the fused evidence to assign weights to the original evidence of different views, thereby guiding reliable and credible identification results.

2.3. Uncertainty and the Theory of Evidence

At present, in the work of realizing security patch identification based on neural network, the softmax output is often used as the classification result. However, this method leads to high over-confidence even for false identification [28,29]. Dempster Shafer theory provides a solution to this problem, as a theory on belief functions, which allows beliefs from different sources to be combined with various fusion operators to obtain a new belief that considers all available evidence. In addition, Tang [30] proposes a fuzzy soft set approach in decision making based on Grey Relational Analysis and Dempster–Shafer theory of evidence. They use the Grey Relational Analysis to determine the uncertain degrees of various parameters. Moreover, subjective logic proposes a solution to the quantification and consolidation of multi-view.

Grey Relational Analysis [31–33] is an approach for multi-criteria decision making that measures the similarity or dissimilarity between variables of a process. It can help assess the impact of different factors on the outcome. Evidence theory is a mathematical framework for handling uncertain information. It can be used to fuse data from multiple views. Based on the above analysis, this paper believes that the advantages of combining Grey Relational Analysis and evidence theory for multi-view learning can be summarized as follows:

- Grey Relational Analysis can preprocess multi-view data, reducing data dimensionality and noise.
- Evidence theory can dynamically integrate multi-view data, considering the uncertainty and weight of each view.

- The performance and robustness of multi-view classification can be enhanced, as well as the confidence and interpretability of the decision.

Therefore, based on subjective logic, this paper obtains identification evidence from the multi-view, and combines the Dirichlet distribution to obtain the belief quality and uncertainty quality of security patch identification, and then solves the problem of softmax over-confidence. At the same time, this paper uses the Dirichlet distribution to give the neural network classification results a probability distribution density, and then realizes the second-order probability and uncertainty of the simulated output. Unlike [30], we determine the relationship of different views as the weight of evidence input according to the Grey Relational Analysis method, and make self-learning correction. The specific details are discussed in detail in the Section 4.5.

Subjective Logic [34–38] provides a theoretical framework for the relationship between Dirichlet distribution parameters and belief and uncertainty. The framework formalizes the concept of belief distribution of DST (Dempster–Shafer Theory of Evidence) [39] as Dirichlet distribution [40,41] in the identification framework. Specifically, subjective logic provides a framework that provides each category with a belief quality b_k and an overall uncertainty quality u. Intuitively, the belief quality b_k is the probability of different categories based on the collected evidence, and the uncertainty quality u is the overall uncertainty of the probability of a specific category, where $k = 1, 2, \cdots, K$ represents the category in the K classification. Moreover, the belief quality and uncertainty quality are both positive numbers, and the sum is 1. The formula is satisfied [42]:

$$u + \sum_{k=1}^{K} b_k = 1 \tag{1}$$

3. Motivation

Based on the previous analysis, in supply chain attacks, it is crucial to quickly identify security patches for vulnerability fixes in a large number of code change submissions. At present, related work has been carried out in this field, but it mainly has the following problems:

1. Related work mainly regards commit messages and code changes as ordinary text, and conducts research based on text classification technology in natural language processing. Although these methods have been proved their effectiveness to a certain extent through experiments, they have not really learned the essential information of security patches. As described in Section 2.1, we may reasonably come to the conclusion that the key of source code patch analysis is to learn the syntax and semantics of code patches, rather than simply treating them as ordinary text. Intuitively, how to learn the control dependency and data dependency in the source code is crucial for learning the essential information of security patches.
2. In recent years, the methods of code2vec and seq2vec have provided a new direction for this field. They build an AST abstract syntax tree by parsing the source code, and convert it into a text sequence by expanding the paths between the leaf nodes. The related work [10] is carried out on this basis, and conducts experiments in the JAVA datasets. However, in the experiment of C source code, we find that the generated AST syntax tree is too large, and this method has certain limitations. This means that the AST abstract syntax tree parsing of C source code will obtain a large number of text sequences after expansion. Moreover, it is difficult to learn the patch information completely and effectively from the results of sampling randomly(extracting a small number of text sequences from a large number of sequences). Meanwhile, it is difficult to directly learn the changes of the data dependencies and control dependencies via the path of the child nodes of the AST abstract syntax tree.
3. Currently in the work of security patch identification, the main ensemble methods are vector concatenation and voting algorithms. However, these ensemble methods are too subjective and do not take into account consistency and complementarity

between different features. Therefore, they do not have the interpretability of the identification results. In the field of patch analysis, how to make use of consistency and complementarity between different views, to find the inherent pattern of patch commit to improve the effectiveness of security patch identification is deemed scientific in this paper. Therefore, we may reasonably come to the conclusion that, in addition to the identification results, what should be known is the confidence and interpretability of the identification results.

4. Design and Implementation

4.1. Overall Design

This paper proposes an efficient and reliable security patch identification system called TMVDPatch. The system adopts a multi-view evidence fusion strategy, including control dependency information, data dependency information and commit message, so as to comprehensively represent the semantic and syntax information of security patches. The method provides credible and interpretable decisions based on subjective uncertainty learned from each view, enabling efficient and scientific security patch identification. Figure 1 shows the specific architecture of the model, which is mainly composed of four modules: data collection and preprocessing, code parsing, evidence obtained and trusted multi-view decision rule.

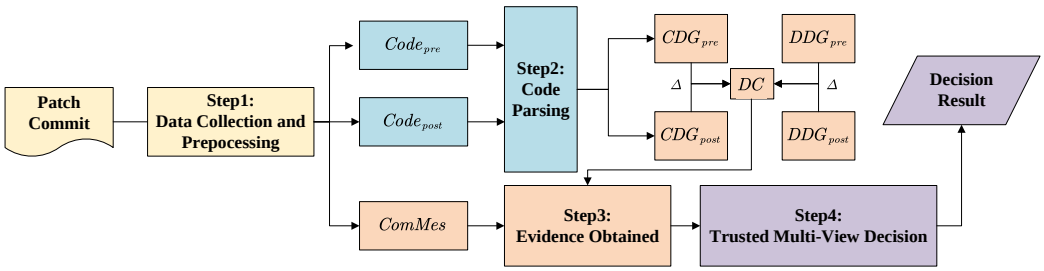

Figure 1. The overall architecture of TMVDPatch.

4.2. Data Collection and Prepocessing

It is well known that the key to security patch identification work is dataset collection and ground truth labeling. To ensure the authenticity of the dataset, this paper adopts the public dataset called PatchDB [16], the labels in this dataset are finally manually verified by three security experts. The dataset provides patch submission files for security patches and non-security patches, which contain commit hashes, commit message text, and code difference information.

On this basis, we preprocess the patch submission files. First, we download the modified source files according to the hash value and code difference information. During the process, some code repository are lost and the source files cannot be downloaded. On this basis, we parse the hunk information and locate the function name modified by the patch. It should be noted that, because the source code needs to be parsed later to generate the control dependency graph and data dependency graph, and feature extraction is performed according to the differences in the dependency graphs. Therefore, in order to avoid the problem of function calls between multiple files, this paper only considers patches involving single-file patches, and filters out patches that modify multiple source files. The PatchDB dataset we obtained contains 23 k non-security patch samples and 12 k security patch samples. In our work, we filtered 5 k non-security patch samples, and 2 K security patch samples. Finally, the relative percentage of security and non-security decreased from 1.9 to 1.8. Since the proportion of samples before and after filtering is slightly different, and the experiments in this paper are carried out on the same dataset, we have reason to believe that this limitation will not affect the results.

Next, we preprocess the submission information text in the patch file, only retains the information text related to the patch description, and filters out irrelevant information such as the subject, date, and reporter email.

Through preprocessing, the source codes before the patch $Code_{pre}$, the source codes after the patch $Code_{post}$, and the descriptions of the submission information called $ComMes$ can be obtained.

4.3. Code Parsing

As described in Section 3, currently in the field of security patch identification, there are two main methods for learning code differences: parsing the source code to build an AST syntax tree and parsing the difference as ordinary text. However, we find the generated AST abstract syntax tree is too large in the experiments of C source code. At the same time, security patches are closely related to data dependencies and control dependencies. It is difficult to directly learn the changes of the data dependencies and control dependencies via the path of the child nodes of the AST abstract syntax tree. Currently, using control dependency graph and data dependency graph to represent semantics are widely used in program analysis. However, how to apply it to the field of source code patch analysis remains to be further studied. Inspired by it, we build the data dependency graph and control dependency graph by parsing the source code, so as to capture more information.

To build the data dependency graph and control dependency graph of a function, the following concepts need to be introduced.

Definition 1. ***Control Flow Graph (CFG)*** *: Consider a function, the CFG of the function is a graph $G = (V, E)$, where $V = \{n_1, n_2, ..., n_i\}$ is a set of nodes with each node representing a statement or control predicate, and $E = \{e_1, e_2, ..., e_i\}$ is a set of direct edges with each edge representing the possible flow of control between a pair of nodes.*

Definition 2. ***Control Dependency****: Consider a function, the CFG of the function is a graph $G = (V, E)$, and two nodes n_i, n_j in V where $i \neq æ$ and n_j is the latter node. If all paths from n_i to the end node in the CFG pass through n_j , then it is said that n_j post-dominate n_i. If there exist a path from n_i to n_j in the CFG, and satisfy n_j post-dominate all nodes on this path except n_i and n_j, and n_j does not post-dominate n_i. Then it is said that n_j is control-dependent on n_i.*

Definition 3. ***Data Dependency****: Consider a function, the CFG of the function is a graph $G = (V, E)$, and two nodes n_i, n_j in V where $i \neq æ$ and n_j is the latter node. If there exist a path from n_i to n_j in the CFG, and a value computed at n_i is used at n_j, then it is said that n_j is data-dependent on n_i.*

In order to clearly illustrate, this paper takes the quicksort program as an example to generate CDG, the details are attached in the Appendix A. In contrast to other tools, the CDG and DDG generated by Joern [43] are not in the unit of statement nodes, but in more fine-grained units of nodes similar to AST nodes. As shown in Appendix A, the 18th line of the source code ($"if(destra > jj)quick\_sort(a, jj, destra);"$) is parsed into two nodes: $(< operator > .greaterThan, destra > jj)$ and $(quick\_sort, quick\_sort(a, jj, destra))$. This fine-grained node representation is more effective for subsequent semantic extraction, as it captures more details. Additionally, the CDG generated by Joern is composed of multiple subgraphs. In this paper, a virtual node (ENTRY, ENTRY) is constructed as the starting node, and all subgraphs are merged to obtain a new CDG.

It should be noted that when we use Joern to parse the code, we found that some functions had recognition problems, and some samples could not show patch differences in the generated CDG and DDG (https://github.com/joernio/joern/issues/1420 (accessed on 1 May 2022)). After verification, the problem is due to the existence of $ifdef$ macro definitions in the source code. Although tools can print a comma-separated list of all preprocessor $ifdef$ and if statements. However, the real-world C/C++ programs tend to have contradicting defines, setting them all at once may lead to code that does not make any sense at all. This issue can be manually reviewed for macro definitions for different patches.

Our method uses the CDG and DDG to represent semantic and syntactic information of the code diff. After obtaining the $Code_{pre}$ and $Code_{post}$, we extract all the methods that

are changed and extract a set of paths over the CDG and DDG. As shown in Figure 1, differently from the AST, CDG and DDG mainly represent dependencies according to the path context. Therefore, different with Code2Vec [17], we do not extract the start-token and the end-token, only keep the path context from the root-node to leaf-node. Next, we discard the contexts that are identical in the code before and after the patch, and use the remaining paths as the basis for the patch code representation. For example, $(METHOD) \downarrow (i\_warning) \downarrow (chmod) \downarrow (METHOD\_RETURN)$. This is an expanded path. We discard any path that is identical in both $Code_{pre}$ and $Code_{post}$, we will discard it.

Moreover, a single-file modification patch may involve multiple modification functions, which have been positioned in Section 4.2. Therefore, given the patch code, we can extract the CDG_{pre} and DDG_{pre} from the $Code_{pre}$, witch represents the path context. Similarly, we can extract the CDG_{post} and DDG_{post} from the $Code_{post}$. Efficiency saving is that we only do the path extraction for the modification functions positioned. We then define the set of contexts called DC describing the semantic and syntactic difference between $Code_{pre}$ and $Code_{post}$. In our work, we use DC to represent the difference in semantic and syntactic information between $Code_{pre}$ and $Code_{post}$. Relevant definitions are as follows:

$$DC = \left(CDG_{pre} \Delta CDG_{post}\right) \cup \left(DDG_{pre} \Delta DDG_{post}\right) \tag{2}$$

$$CDG_{pre} \Delta CDG_{post} = \{c | (c \in CDG_{pre} \cup CDG_{post}) \wedge (c \notin CDG_{pre} \cap CDG_{post})\} \tag{3}$$

$$DDG_{pre} \Delta DDG_{post} = \{d | (d \in DDG_{pre} \cup DDG_{post}) \wedge (d \notin DDG_{pre} \cap DDG_{post})\} \tag{4}$$

Intuitively, the set of contexts DC contains the semantic and syntactic difference between before and after code revision.

4.4. Evidence Obtained

Neural network is used as evidence extractor to capture evidence from input. Therefore, we obtained evidence from code diff view and commit message view respectively based on neural network. Figure 2 shows the neural network structure for evidence obtained from the code diff view.

In the code diff view, we extract evidence based on the CDG and DDG difference sequences obtained in parsing of the code diff. In our work, we use a neural network embedding layer to vectorize each node, and then use the final state of a bidirectional LSTM(Long Short Term Memory) with self-attention to encode the entire sequence, thereby representing each path as a fixed-length vector.

Next, we take all the path-vectors as the sequence input, using the self-attention mechanism and the fully connected layer as the evidence extractor to obatin evidence. In the network, we replace the softmax layer in the classical classifier with the softplus layer to ensure that the neural network output is non-negative. Finally, we get evidence from the code diff view.

For the message commit view, we use a bidirectional LSTM with self-attention mechanism and a softplus layer as the evidence extractor. The message commit view is treated as the natural language input text.

4.5. Trusted Multi-View Decision Rule

In this subsection, we mainly elaborate on trusted multi-view decision rules based on the evidence obtained. As described in Section 2.3, using a softmax output as confidence for predictions often leads to high confidence values, even for erroneous predictions since the largest softmax output is used for the final prediction. In addition, subjective logic associates the parameters of Dirichlet distribution with belief distribution. Related work [26,27] first apply this work to the field of multi-view learning in the framework of deep learning.

They introduce a new paradigm theory in multi-view classification to provide credible and interpretable decisions based on the uncertainty of each view.

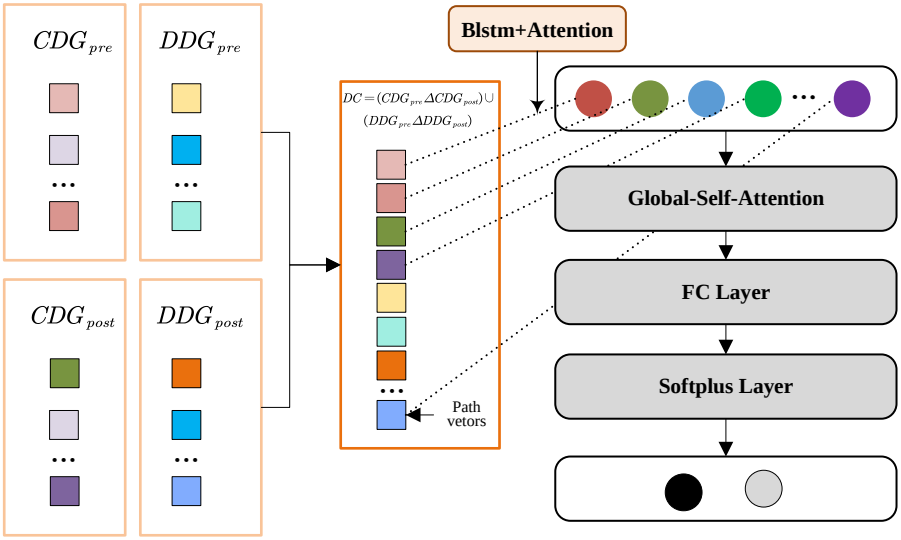

Figure 2. The neural network structure for evidence obtained from the code diff view.

Taking into account all factors, how to fully utilize the patch submission information in an interpretable way is crucial to the identification of security patches. In cutting edge research [26,27], they propose a new multi-view classification algorithm, which aims to integrate multi-view information for trusted decision-making. This coincides with our motivation. Therefore, we obtain multi-view evidence, including control dependency information, data dependency information and description information, to comprehensively represent the semantic and syntax information of security patches.

In our work ,we first learn the patch information based on the neural network model proposed in Section 4.4, and takes the results as evidence in the multi-view learning framework. And then, this paper uses the theoretical framework to observe the possibilities (believe masses) of different views and overall uncertainty (uncertainty mass) based on the evidence collected from patch commit. Note that the evidence refers to the metrics collected from the input to support the classification and is closely related to the concentration parameters of Dirichlet distribution.

Based on the original framework, this paper determines the relevance of different views as the weight of evidence input according to the Grey Relational Analysis method, and makes self-learning correction. In this way, we can improve the ability of trusted multi-view decision-making. Figure 3 shows the methods of trusted multi-view decision rule.

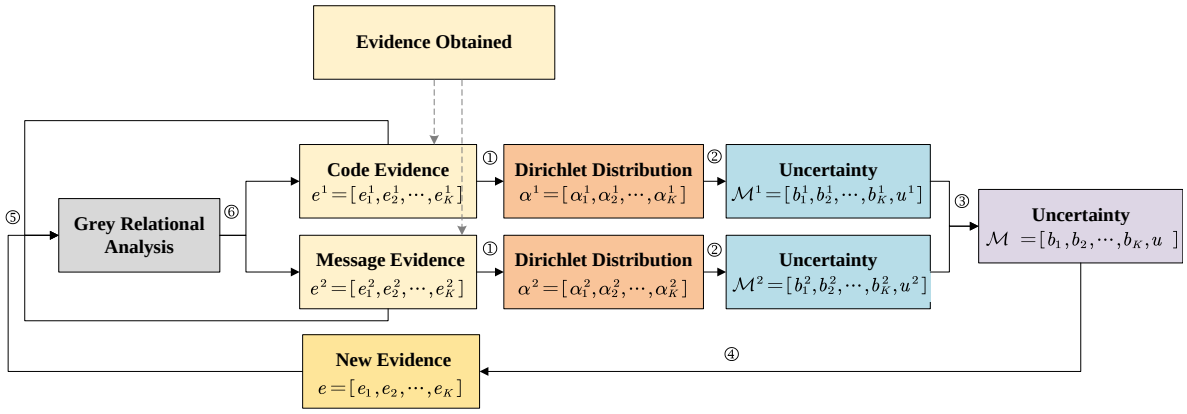

Figure 3. The process of trusted multi-view decision.

In our work, for the view v, according to Step① in the Figure 3, the subjective logic associates
$e^v = [e_1^v, e_2^v, \cdots, e_K^v]$ with the parameters of the Dirichlet distribution $\alpha^v = [\alpha_1^v, \alpha_2^v, \cdots, \alpha_K^v]$. Especially, e^1 represents the evidence from the code diff view and e^2 represents the commit message evidence. Its subjective opinion corresponds to a Dirichlet distribution with parameter $\alpha_k^v = e_k^v + 1$.

Then, this paper calculates the belief quality b_k^v and uncertainty quality u^v according to Step② in the Figure 3. For the category k under the view v, its belief quality b_k^v is calculated from its corresponding evidence e_k^v, and the calculation formula is as follows:

$$b_k^v = \frac{e_k^v}{S^v} \; and \; u^v = \frac{K}{S^v} \tag{5}$$

$S^v = \sum_{i=1}^{K} \alpha_i^v = \sum_{i=1}^{K} \left(e_i^v + 1\right)$ is Dirichlet strength. Therefore, uncertainty u^v is inversely proportional to the totality of evidence S^v.

Next, this paper calculates the joint probability mass assignment set $\mathcal{M} = \{\{b_k\}_{k=1}^K, u\}$ according to Step③ in the Figure 3. It is obtained by combining the sets of probability mass assignments $\mathcal{M}^v = \{\{b_k^v\}_{k=1}^K, u^v\}$ from different views v. In this paper, the specific calculation formula is as follows.

$$\begin{aligned} &\mathcal{M} = \mathcal{M}^1 \oplus \mathcal{M}^2 \; and \; M = \{b_k\}_{k=1}^K, u \\ &b_k = \frac{1}{1-C}(b_k^1 b_k^2 + b_k^1 u^2 + b_k^2 u^1) \\ &u = \frac{1}{1-C} u^1 u^2 \end{aligned} \tag{6}$$

$C = \sum_{i \neq j} b_i^1 b_j^2$ is a measure of the conflicting values of the two views, and $1/(1-C)$ is the normalization factor.

On the basis of the joint probability mass assignment set, according to Step ④ in the Figure 3, calculate the new evidence $e = [e_1, e_2, \cdots, e_K]$. It uses b_k and u, and calculates the new Dirichlet distribution parameters according to the formulas in Step ① and ② to obtain new evidence e.

Next, according to the Grey Relational Analysis method in Step ⑤ in the Figure 3, this paper calculates the correlation between the new evidence e and the original evidence e^1, e^2. Grey Relational Analysis a multi-factor statistical analysis method, which can be used to evaluate the correlation between the parent sequence and other subsequences. Intuitively, it provides a way to measure the distance between vectors. In our work, we implement the measurement of the contribution of evidence from different views to the final evidence based on the Grey Relational Analysis method. The specific method is as follows:

First, the fused evidence $e = [e_1, e_2, \cdots, e_K]$ is used as the parent sequence, and the original evidence from different views $e^v = [e_1^v, e_2^v, \cdots, e_K^v]$ is used as the subsequence, and the data is normalized.

Next, calculate the grey correlation coefficient using the formula [31]:

$$\zeta_v(k) = \frac{\min_v \min_k |e(k) - e^v(k)| + \rho \cdot \max_v \max_k |e(k) - e^v(k)|}{|e(k) - e^v(k)| + \rho \cdot \max_v \max_k |e(k) - e^v(k)|} \tag{7}$$

ρ is the resolution coefficient. $\zeta_v(k)$ represents the correlation coefficient between the evidence in the kth dimension of the view v and the evidence after fusion. In our work, we sets ρ as 0.5. And then, the correlation coefficient corresponding to the view v is obtained by calculating the mean value of the correlation coefficient γ_v under each view v. The specific calculation formula is as follows:

$$\gamma_v = \frac{1}{K} \sum_{k=1}^{K} \zeta_v(k) \tag{8}$$

Finally, this paper obtains the correction weight w_v under the corresponding view v according to the correlation γ_v, and then corrects the evidence under different views $e^v = [e_1^v, e_2^v, \cdots, e_K^v]$, and obtains the corrected evidence e^{vnew}. Furthermore, we set the initial value of correction weight w_v to 1, so that the model can get evidence correction at the beginning of the second epoch. The specific calculation formula is as follows:

$$\begin{aligned} w_v &= \frac{\gamma_v}{\max_v(\gamma_v)} \\ e^{vnew} &= \frac{e^v + w^v e^v}{2} \end{aligned} \tag{9}$$

4.6. Model Training

In the work of neural network-based security patch identification, we use cross-entropy [44] as the loss function:

$$\mathcal{L}_{ce} = -\sum_{j=1}^{K} y_{ij} \log(p_i j) \tag{10}$$

y_{ij} is the labeling situation where the true label of the sample i is j, and p_{ij} is the probability that the sample i is predicted by the model as j. In this paper, the Dirichlet distribution is used to give the neural network classification results a probability distribution density, so the adjusted cross-entropy loss function [42] is:

$$\begin{aligned} \mathcal{L}_{ace}(\alpha_i) &= \int \left[\sum_{j=1}^{K} -y_{ij} \log\left(p_{ij}\right) \right] \frac{1}{B(\alpha_i)} \prod_{j=1}^{K} p_{ij}^{\alpha_{ij}-1} dp_i \\ &= \sum_{j=1}^{K} y_{ij} \left(\psi(S_i) - \psi\left(\alpha_{ij}\right)\right) \end{aligned} \tag{11}$$

$\psi(\cdot)$ is the digamma function. Although this loss function enables the model to obtain more correct label evidence, it cannot guarantee that the wrong label evidence is as small as possible. Therefore, this paper introduces the KL (Kullback-Leibler) divergence term to regularize the evidence to expect the misplaced label evidence to become 0. The specific formula is as follows [26]:

$$\begin{aligned} KL[D(\boldsymbol{p}_i|\tilde{\boldsymbol{\alpha}}_i) \| D(\boldsymbol{p}_i|1)] &= \log\left(\frac{\Gamma\left(\sum_{k=1}^{K} \tilde{\alpha}_{ik}\right)}{\Gamma(K) \prod_{k=1}^{K} \Gamma(\tilde{\alpha}_{ik})} \right) \\ + \sum_{k=1}^{K} (\tilde{\alpha}_{ik} - 1) \left[\psi(\tilde{\alpha}_{ik}) - \psi\left(\sum_{j=1}^{K} \tilde{\alpha}_{ij}\right) \right] \end{aligned} \tag{12}$$

The loss value under single view is:

$$\mathcal{L}(\alpha_i) = \mathcal{L}_{ace}(\alpha_i) + \lambda_t KL[D(\boldsymbol{p}_i|\tilde{\boldsymbol{\alpha}}_i) \| D(\boldsymbol{p}_i|1)] \tag{13}$$

Under the multi-view framework, the total loss is as follows:

$$\mathcal{L}_{overall} = \sum_{i=1}^{N} \left[\mathcal{L}(\alpha_i) + \sum_{v=1}^{V} \mathcal{L}(\alpha_i^v) \right] \tag{14}$$

5. Experiments and Results

5.1. Dataset and Experimental Environment

As mentioned in Section 2.1, the key to machine learning-based security patch identification work is a modest dataset labelled with ground truth. Therefore, to increase code reliability and reproducibility, we employ PatchDB datasets to conduct experiments. More-

over, we will use PatchRNN on the same dataset to conduct experiments as a comparison model. The dataset contains a large number of security patches and non-security patches written in C/C++ language, not only the security patches with vulnerability numbers indexed by the National Vulnerability Database (NVD), but also a large number of security patches obtained from the wild. Among them, the security patch dataset obtained from the wild is collected from commits on GitHub. The dataset is manually validated by three security experts on the labels of security patches and non-security patches, and the authenticity of labels is ensured by cross-checking. Therefore, this paper has reason to believe that the labels of this dataset are convincing, and conduct relevant experiments on this dataset.

Our experiment is conducted using Python 3.9.6, while the TMVDPatch is designed based on Pytorch 1.9.1. Our model is carried out in the Ubuntu 20.04.1 LTS environment running in AMD Ryzen Threadripper 3960X 24-Core Processor CPU with 125-GB RAM. We realize the neural network training by employing the CUDA 11.4 toolkit with 2 NVIDIA GeForce RTX 3090 GPUs.

We use the classical parameter optimizer as the SGD optimizer. For other hyperparameter settings, the batch size is set to 64, and the number of epochs is set to 20.

5.2. Evaluation Metrics

In this paper, the accuracy, precision, recall and $F1$ score are used as evaluation metrics to evaluate the security patch identification effect of TMVDPatch. In the evaluation, we consider four statistical types, namely true positive (TP), true negative (TN), false positive (FP) and false negative (FN). If the sample is actually a security patch and is identified as a security patch, we count this test case as TP. If the sample is actually a non-security patch and is identified as a non-security patch, we count this test case as TN. If the sample is actually a security patch and is identified as a non-security patch, we count this test case as FN. If the sample is actually a non-security patch and is identified as a security patch, we count this test case as FP.

Accuracy measures the number of patches identified accurately as a proportion of all samples. This metric can measure the overall identification effect of the model on security patches and non-security patches. The formula is as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{15}$$

Precision measures the proportion of actually security patches among samples identified by the model as security patches. This metric can measure the false positive rate of the model for the identification of security patches. The specific formula is as follows:

$$Precision = \frac{TP}{TP + FP} \tag{16}$$

Recall measures the proportion of security patch samples identified by the model among all actual security patch samples. The specific formula is as follows:

$$Recall = \frac{TP}{TP + FN} \tag{17}$$

F1 Score comprehensively measures the precision and recall of the model. This metric is calculated by weighting the two. The calculation formula is as follows:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{18}$$

5.3. Hyperparameters Tuning

This subsection mainly discusses the problem of hyperparameter setting in the model. In order to achieve better tuning parameters, we use Optuna for hyperparameter search.

Optuna [45] is a hyperparameter optimization tool that optimizes tree-based hyperparameter search and relies on Bayesian probability to iteratively adjust the search for hyperparameters.

The specific details are shown in Table 2. This subsection mainly adjusts the 8 hyperparameters listed in the Table 2. Except the learning rate is sampled by logarithmic uniform distribution, other hyperparameters are sampled by step. At the same time, we maps the 200 trials to the state points in the three-dimensional space in the form of "X-Recall, Y-Precision, Z-Accuracy", and the red series of points are the trials that are judged be the best trials. The color of the points only represents the trials ID. The specific results are shown in Figure 4, where values represent the accuracy, precision and recall values respectively, and params is the parameter setting of the current best trial.

Table 2. The hyperparameters tuning details.

View	Hyper-Parameter	Search Space	Best Value
Code Diff	codediff_embedding	[40,120]	100
	codediff_global_dropout	[0.1,0.5]	0.2
	codediff_lstm	[32,128]	128
	codediff_lstm_dropout	[0.1,0.5]	0.4
Commit Message	commit_dropout	[0.1,0.5]	0.3
	commit_embedding	[40,120]	120
	commit_lstm	[32,128]	64
Overall	lr	[0.00001, 0.1]	0.000557

In addition, Figure 5 draws a parallel coordinate graph based on the $F1$ evaluation metric to highlight the relationship between hyperparameters in the search space.

Among them, the vector dimension may affect the security patch identification performance of the model. If the dimensions are too large, the model may be so complex that become overfit. However, too small dimension will also lead to poor model learning ability. Therefore, how to select the optimal parameter combination in the hyperparameter search space is critical to the model performance.

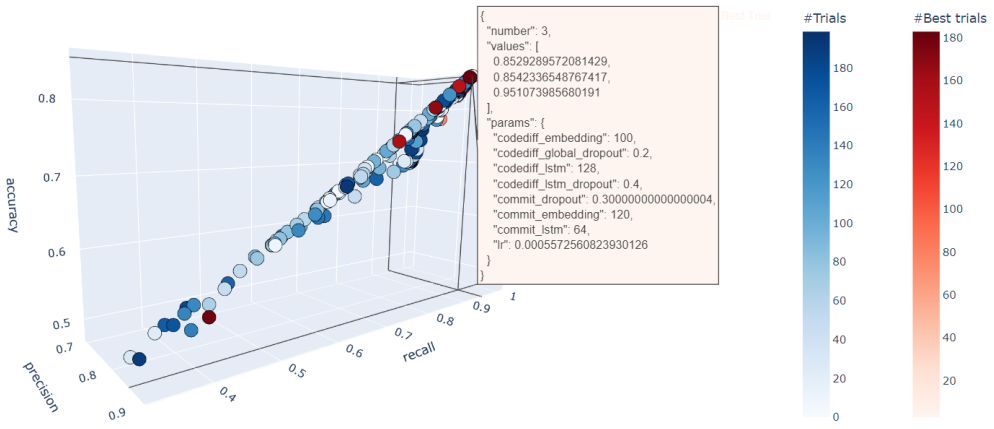

Figure 4. The hyperparameters tuning trials in the three-dimensional space.

As shown in the Figure 5, for the code diff view, both the LSTM hidden layer and the embedding layer require a higher vector dimension to achieve better model performance. But for the message commit view, it needs to achieve better patch identification results by keeping the embedding layer dimension high and the LSTM hidden layer dimension low. The main reason is that the model is more complicated in the message commit view. Because of this, the dropout layer parameter in the LSTM self-attention mechanism is set to 0.3, thereby improving the generalization ability of the model and preventing the model from overfitting. In the same way, in the code diff view, the global dropout layer

parameters are set lower than the dropout layer parameters in the LSTM self-attention mechanism. Moreover, the F1 scores of the 200 trials in Figure 5 are mostly concentrated between 0.8 and 0.9, which further indicates that the model has strong robustness to hyperparameter settings.

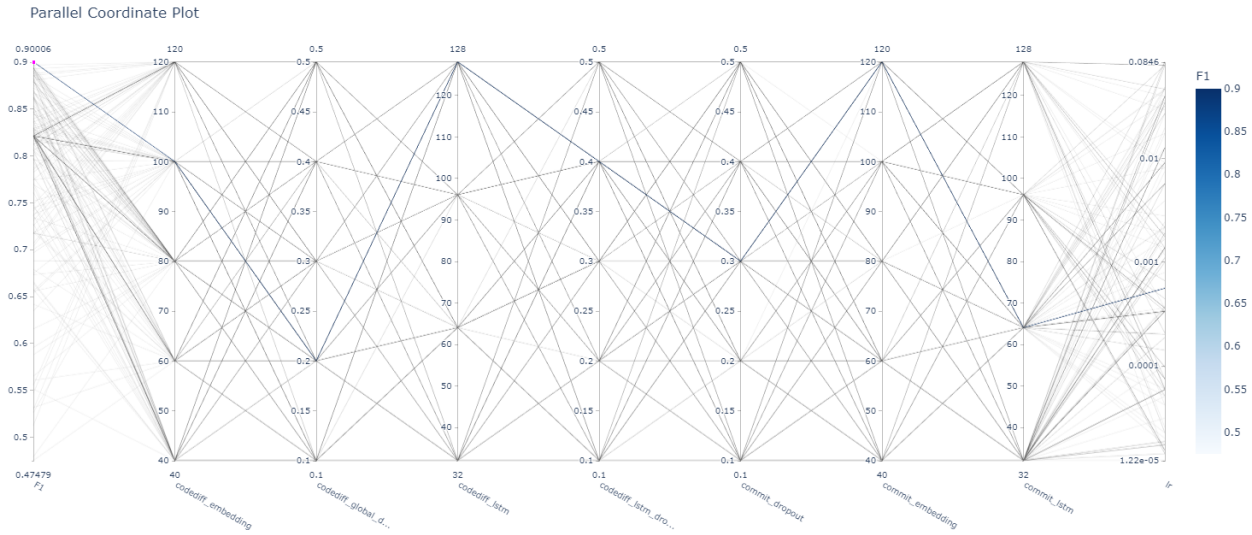

Figure 5. The parallel coordinate graph based on the F1 evaluation metric to highlight the relationship between hyperparameters in the search space.

In order to more intuitively explain the influence of parameters on each evaluation metric, we evaluate the importance of hyperparameters based on Optuna. The specific results are shown in Figure 6, where the sum of the importance of each parameter is 1 under each metric. It is not difficult to see that for all evaluation metrics, the learning rate setting is extremely critical. At the same time, the hyperparameters in the code diff view are slightly more important to accuracy and F1 than the message commit view. Moreover, the F1 score is used as a comprehensive measure of the precision and recall of the model, and the importance of each parameter is distributed evenly.

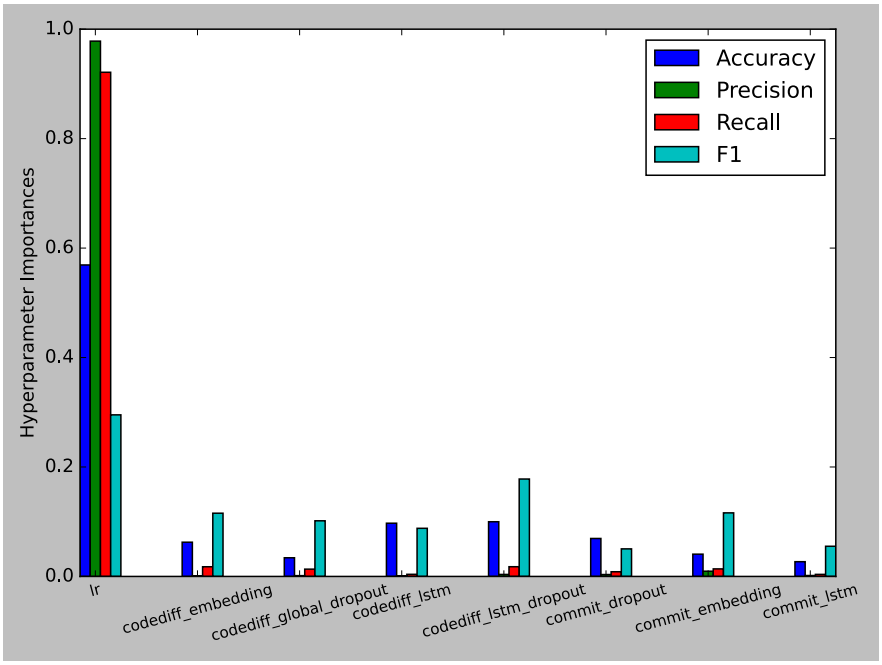

Figure 6. The importance of hyperparameters to evaluation metrics.

5.4. Model Comparison

For the problem of security patch identification, TMVDPatch fully considers the relevant data submitted by the patch (including control dependency information, data dependency information and submission message). In order to evaluate the effectiveness of the TMVDPatch proposed in this paper, different classification models are selected for comparative experiments in this subsection.

In our work, we use SV-CMS and SV-CDP to represent single-view classification models using message commits and code changes alone, respectively. Since PatchRNN uses the same dataset as this paper for security patch identification, and also considers all source code information before and after patch submission, we choose PatchRNN as the comparison model. In addition, we treat code changes as normal text, and build a security patch identification model based on BLSTM(Bi-directional Long Short-Term Memory) called SQ-CDP. On this basis, we combine the SQ-CDP with the SV-CMS(single-view classification models using message commits) to construct a new model called SQ-A.

The comparative experimental results are shown in Table 3. Compared with single-view SV-CMS and SV-CDP, multi-view learning can effectively improve the identification ability of the model, which shows that the evidence between different views can be complementary. At the same time, SV-CDP can achieve a higher recall, because only using single-view learning will lead to better identification of non-security patches by the model, resulting in the phenomenon of model bias. Under the multi-view, the problem can be effectively solved based on comprehensive evidence. Compared with SQ-CDP and SQ-A, TMVDPatch can learn control dependency and data dependency information from code patches. It further illustrates the effectiveness of the representation method of security patches proposed in this paper. Moreover, compared with PatchRNN, TMVDPatch can achieve better identification effect in the same dataset.

5.5. Ensemble Methods

In our work, the TMVDPatch combines the evidence of the commit message view and the code diff view. It should be noted that the credible and interpretable decisions discussed in this paper are essentially an ensemble strategy. In order to fully evaluate the effectiveness of the ensemble method proposed in this paper, this subsection conducts comparative experiments on different ensemble methods. Based on the single-view classification models SV-CMS and SV-CDP, we use three different feature fusion methods (vector summation, vector concatenation and vector averaging) to obtain MV_{sum}, MV_{cat} and MV_{avg}, respectively. Since the parameter settings of the LSTM hidden layer between different views are inconsistent, we set them to 64 and 128, respectively, to obtain the corresponding MV_{sum_64}, MV_{sum_128}, MV_{avg_64} and MV_{avg_128}. In order to further evaluate the TMVDPatch, we use TMVD-NG to represent a trusted multi-view decision model without Grey Relational Analysis feedback to conduct related experiments.

Table 3. The results of model comparison.

Model	Accuracy	Precision	Recall	F1
SV-CMS	0.7391	0.7617	0.9099	0.8293
SV-CDP	0.7013	0.7020	**0.9922**	0.8222
SQ-CDP	0.6954	0.7012	0.9803	0.8176
SQ-A	0.7798	0.7909	0.9296	0.8546
PatchRNN	0.8357	0.7572	0.7366	0.7468
TMVDPatch	**0.8529**	**0.8542**	**0.9511**	**0.9001**

The experimental results are shown in Table 4. The trusted multi-view decision system proposed in this paper outperforms other ensemble models in all four metrics, achieving an accuracy of 0.8529 and a F1 score of 0.9001. In addition, by comparing the experimental results of TMVD-NG, it is confirmed that adding Grey Relational Analysis feedback can effectively improve the identification ability of the model.

Table 4. The results of different ensemble methods.

Model	Accuracy	Precision	Recall	F1
MV_{cat}	0.7798	0.7877	0.9362	0.8555
MV_{sum_64}	0.7329	0.8409	0.7601	0.7985
MV_{sum_128}	0.7574	0.8100	0.8520	0.8302
MV_{avg_64}	0.7736	0.8503	0.8109	0.8301
MV_{avg_128}	0.7615	0.7783	0.9195	0.8430
TMVD-NG	0.8400	0.8502	0.9350	0.8906
TMVDPatch	**0.8529**	**0.8542**	**0.9511**	**0.9001**

6. Discussion and Future Work

At present, the identification of security patches based on neural networks is often not interpretable and cannot fully capture patch information. To solve this problem, this paper proposes an efficient and reliable security patch identification system called TMVD-Patch. The system obtains evidence from message commit and code diff views respectively, and models the uncertainty of each view based on the D-S evidence theory, thereby providing credible and interpretable security patch identification results. On this basis, this paper performs weighted training on the original evidence based on the Grey Relational Analysis method to improve the credible multi-view decision-making ability. Experimental results show that the multi-view learning method exhibits excellent capabilities in terms of control-dependent and data-dependent complementary information, and the model exhibits strong robustness against hyperparameter settings. TMVDPatch outperforms other models in all evaluation metrics, achieving an accuracy of 85.29% and a F1 score of 0.9001, clearly verifying the superiority of TMVDPatch in terms of accuracy, scientificity, and reliability.

One of the limitation of our proposed method is that it depends on the analytical results obtained by the Joern tool, which may not be accurate or complete for some code structures. A possible direction for future work is to incorporate evidence from other views, such as syntactic, semantic or structural features of the code, into our framework to enhance the robustness and generalization ability of our model. Another limitation of our method is that it only handles patches that involve single file changes. This may miss some important information or introduce some errors when dealing with patches that span multiple files and have function calls between them. To overcome this challenge, future work can improve the analysis tool to capture the dependencies and interactions between multiple files and handle multi-file patches more effectively.

The main problem we address is how to provide credible and interpretable decisions for security patch identification. Therefore, we do not discuss neural network models in detail in this paper. In our work, we only use an attention-based BLSTM model to obtain evidence and we do not compare it with other neural network models. Future work can explore neural network model design for this task. For example, different neural network models can be used as multiple views and then investigate their complementarity or conflict situations.

In addition to security patch identification, the framework proposed in this paper also offers potential implications for code analysis in other domains. Our framework can be extended to new solutions for other applications, such as malware detection.

Author Contributions: Conceptualization, X.Z. and W.L.; Methodology, X.Z., J.P. and Z.S.; Software, X.Z., J.P., F.L. and G.L.; Validation, J.W.; Formal analysis, Z.S.; Investigation, W.L. and G.L.; Resources, Z.S. and F.L.; Data curation, F.Y. and J.X.; Writing—original draft, X.Z.; Writing—review & editing, X.Z.; Visualization, J.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China 61802433 and 61802435.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All datasets used in this study are available from the PatchDB website (https://sunlab-gmu.github.io/PatchDB/ accessed on 19 January 2023). The data can also be obtained by contacting the authors.

Acknowledgments: The authors thank the editor and the anonymous reviewers for their constructive feedback and valuable advice. Meanwhile, the authors thank SunLab for providing the PatchDB dataset used in this study.

Conflicts of Interest: The authors declare that they have no conflict of interests and no competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Control Dependence Graph

Listing A1. An example of quick sort.

```
#include <stdio.h>
#include <stdlib.h>
void quick_sort(int *a, int sinistra, int destra)
{
int ii,jj;
int pivot;
printf("-");
pivot = a[sinistra];
jj = sinistra + 1;
ii = destra;
while( ii > jj ){
while((a[jj]<=pivot)&&(jj<=destra)) jj++;
while((a[ii]>pivot)||(ii==jj)) ii--;
if(ii>jj) scambia(&a[ii],&a[jj]);
}
if((ii!=sinistra)&&(a[ii]<a[sinistra])) scambia(&a[ii],&a[sinistra]);
if(sinistra<(ii-1)) quick_sort(a,sinistra,ii-1);
if(destra>jj) quick_sort(a,jj,destra);
}
```

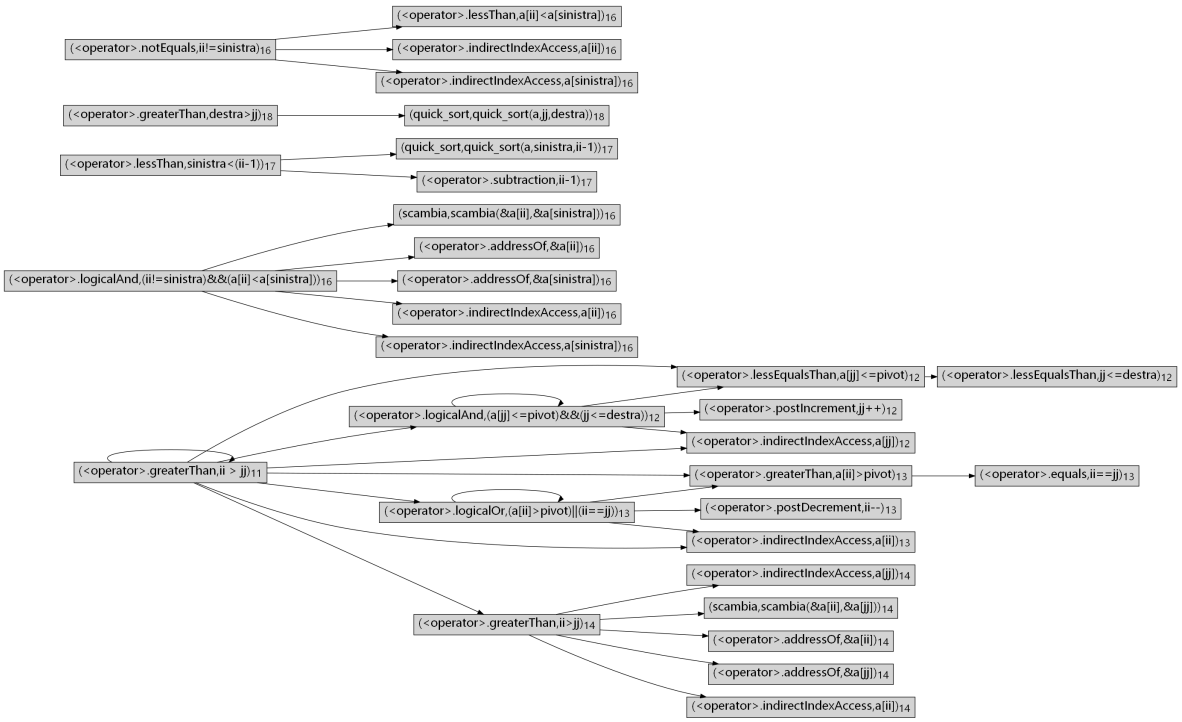

Figure A1. The initial control dependency graph of quick sort.

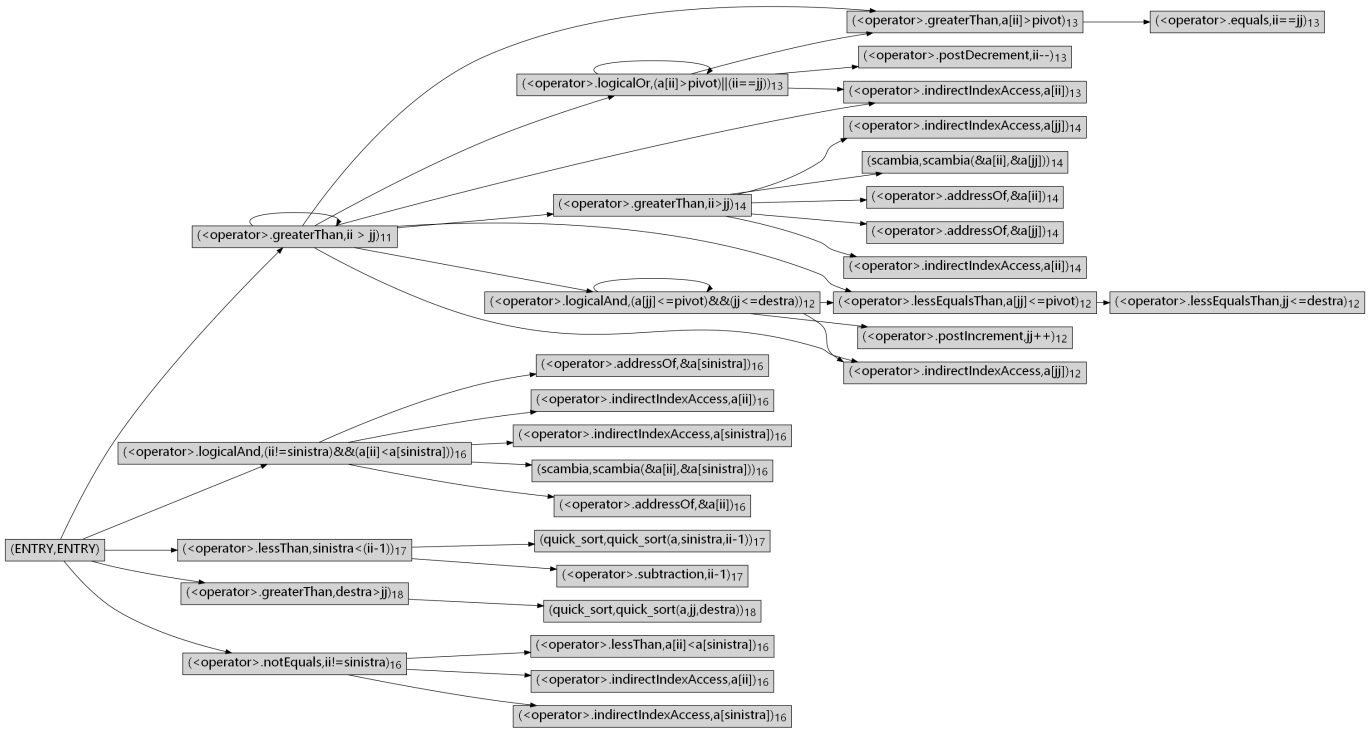

Figure A2. The processed control dependency graph of quick sort.

References

1. Snyk. The State of Open Source Security Report 2020, 2020. Available online: https://go.snyk.io/SoOSS-Report-2020.html (accessed on 19 January 2023).
2. Snyk. Addressing Cybersecurity Challenges in Open Source Software, 2022. Available online: https://snyk.io/reports/open-source-security/ (accessed on 19 January 2023).
3. ARMIS. Log4j Vulnerability. 2021. Available online: https://www.armis.com/log4j (accessed on 19 January 2023).
4. ARMIS. The Long Tail Matters with Apache Log4j, 2021. Available online: https://www.armis.com/blog/he-long-tail-matters-with-apache-log4j/ (accessed on 19 January 2023).
5. Github. The State of the Octovers, 2019. Available online: https://octoverse.github.com/2019/ (accessed on 19 January 2023).
6. Wang, X.; Sun, K.; Batcheller, A.; Jajodia, S. Detecting "0-day" vulnerability: An empirical study of secret security patch in OSS. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, OR, USA, 24–27 June 2019; pp. 485–492.
7. Zhou, Y.; Siow, J.K.; Wang, C.; Liu, S.; Liu, Y. SPI: Automated Identification of Security Patches via Commits. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2021**, *31*, 1–27. [CrossRef]
8. Sawadogo, A.D.; Bissyandé, T.F.; Moha, N.; Allix, K.; Klein, J.; Li, L.; Traon, Y.L. Learning to catch security patches. *arXiv* **2020**, arXiv:2001.09148.
9. Sabetta, A.; Bezzi, M. A practical approach to the automatic classification of security-relevant commits. In Proceedings of the 2018 IEEE International conference on software maintenance and evolution (ICSME), Madrid, Spain, 23–29 September 2018; pp. 579–582. Available online: https://research.com/conference/icsme-2018 (accessed on 19 January 2023).
10. Cabrera Lozoya, R.; Baumann, A.; Sabetta, A.; Bezzi, M. Commit2vec: Learning distributed representations of code changes. *SN Comput. Sci.* **2021**, *2*, 1–16. [CrossRef]
11. Hoang, T.; Kang, H.J.; Lo, D.; Lawall, J. Cc2vec: Distributed representations of code changes. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 518–529.
12. Hoang, T.; Lawall, J.; Oentaryo, R.J.; Tian, Y.; Lo, D. PatchNet: A tool for deep patch classification. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montréal, QC, Canada, 25–31 May 2019 pp. 83–86. Available online: https://conf.researchr.org/home/icse-2019 (accessed on 19 January 2023).
13. Wang, X.; Wang, S.; Feng, P.; Sun, K.; Jajodia, S.; Benchaaboun, S.; Geck, F. PatchRNN: A Deep Learning-Based System for Security Patch Identification. In Proceedings of the MILCOM 2021–2021 IEEE Military Communications Conference (MILCOM), San Diego, CA, USA, 29 November–2 December 2021; pp. 595–600.
14. Machiry, A.; Redini, N.; Camellini, E.; Kruegel, C.; Vigna, G. Spider: Enabling fast patch propagation in related software repositories. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–20 May 2020; pp. 1562–1579.

15. Tan, X.; Zhang, Y.; Mi, C.; Cao, J.; Sun, K.; Lin, Y.; Yang, M. Locating the Security Patches for Disclosed OSS Vulnerabilities with Vulnerability-Commit Correlation Ranking. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Copenhagen, Denmark, 26–30 November 2021; pp. 3282–3299.
16. Wang, X.; Wang, S.; Feng, P.; Sun, K.; Jajodia, S. Patchdb: A large-scale security patch dataset. In Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Taipei, Taiwan, 21–24 June 2021; pp. 149–160.
17. Alon, U.; Zilberstein, M.; Levy, O.; Yahav, E. code2vec: Learning distributed representations of code. *Proc. ACM Program. Lang.* **2019**, *3*, 1–29. [CrossRef]
18. Kumar, A.; Daumé, H. A co-training approach for multi-view spectral clustering. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, DC, USA, 28 June–2 July 2011; pp. 393–400.
19. Xue, Z.; Du, J.; Du, D.; Lyu, S. Deep low-rank subspace ensemble for multi-view clustering. *Inf. Sci.* **2019**, *482*, 210–227. [CrossRef]
20. Bach, F.R.; Jordan, M.I. Kernel independent component analysis. *J. Mach. Learn. Res.* **2002**, *3*, 1–48.
21. Lai, P.L.; Fyfe, C. Kernel and nonlinear canonical correlation analysis. *Int. J. Neural Syst.* **2000**, *10*, 365–377. [CrossRef] [PubMed]
22. Akaho, S. A kernel method for canonical correlation analysis. *arXiv* **2006**, arXiv:cs/0609071.
23. Socher, R.; Fei-Fei, L. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 13–18 June 2010; pp. 966–973.
24. Andrew, G.; Arora, R.; Bilmes, J.; Livescu, K. Deep canonical correlation analysis. In Proceedings of the International conference on machine learning. PMLR, Atlanta, GA, USA, 16–21 June 2013; pp. 1247–1255.
25. Wang, W.; Arora, R.; Livescu, K.; Bilmes, J. On deep multi-view representation learning. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1083–1092.
26. Han, Z.; Zhang, C.; Fu, H.; Zhou, J.T. Trusted multi-view classification. *arXiv* **2021**, arXiv:2102.02051.
27. Han, Z.; Zhang, C.; Fu, H.; Zhou, J.T. Trusted Multi-View Classification with Dynamic Evidential Fusion. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, 45, 2551–2566. [CrossRef] [PubMed]
28. Kull, M.; Perello Nieto, M.; Kängsepp, M.; Silva Filho, T.; Song, H.; Flach, P. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. *Adv. Neural Inf. Process. Syst.* **2019**, *32*. [CrossRef]
29. Bai, Y.; Mei, S.; Wang, H.; Xiong, C. Don't just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 566–576.
30. Tang, H. A novel fuzzy soft set approach in decision making based on grey relational analysis and Dempster–Shafer theory of evidence. *Appl. Soft Comput.* **2015**, *31*, 317–325. [CrossRef]
31. Kuo, Y.; Yang, T.; Huang, G.W. The use of grey relational analysis in solving multiple attribute decision-making problems. *Comput. Ind. Eng.* **2008**, *55*, 80–93. [CrossRef]
32. Ng, D.K. Grey system and grey relational model. *ACM Sigice Bull.* **1994**, *20*, 2–9. [CrossRef]
33. Yang, C.C.; Chen, B.S. Supplier selection using combined analytical hierarchy process and grey relational analysis. *J. Manuf. Technol. Manag.* **2006**, *17*, 926–941. [CrossRef]
34. Jøsang, A. *Subjective Logic*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 3.
35. Oren, N.; Norman, T.J.; Preece, A. Subjective logic and arguing with evidence. *Artif. Intell.* **2007**, *171*, 838–854. [CrossRef]
36. Gao, W.; Zhang, G.; Chen, W.; Li, Y. A trust model based on subjective logic. In Proceedings of the 2009 Fourth International Conference on Internet Computing for Science and Engineering, Washington, DC, USA, 21–22 December 2009; pp. 272–276.
37. Jøsang, A. Generalising Bayes' theorem in subjective logic. In Proceedings of the MFI 2016, Baden-Baden, Germany, 19–21 September 2016. pp. 462–469.
38. Van Der Heijden, R.W.; Kopp, H.; Kargl, F. Multi-source fusion operations in subjective logic. In Proceedings of the 2018 21st International Conference on Information Fusion (FUSION), Cambridge, UK, 10–13 July 2018; pp. 1990–1997.
39. Sentz, K.; Ferson, S. Combination of Evidence in Dempster-Shafer Theory. 2002. Available online: https://www.osti.gov/biblio/800792 (accessed on 19 January 2023).
40. Minka, T. Estimating a Dirichlet Distribution. 2000. Available online: url:https://www.semanticscholar.org/paper/Estimating-a-Dirichlet-distribution-Minka/51fec0515b11cabc2be4832eab43ca5f6ff387d1 (accessed on 19 January 2023).
41. Lin, J. *On the Dirichlet Distribution*; Department of Mathematics and Statistics, Queens University: Kingston, ON, Canada, 2016.
42. Sensoy, M.; Kaplan, L.; Kandemir, M. Evidential deep learning to quantify classification uncertainty. *Adv. Neural Inf. Process. Syst.* **2018**, *31* .
43. Joern. Joern Documentation, 2022. Available online: https://docs.joern.io/home/ (accessed on 19 January 2023).
44. Kapur, J.N.; Kesavan, H.K. *Entropy Optimization Principles and their Applications*; Springer: Berlin/Heidelberg, Germany, 1992.
45. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage , AK, USA, 4–8 August 2019; pp. 2623–2631.