sustainability

MDPI

Article

Toward Sustainable Model Services for Deep Learning: A Sub-Network-Based Solution Integrating Blockchain with IPFS and a Use Case in Intelligent Transportation

Rui Jiang *,†, Jiatao Li †, Weifeng Bu † and Chongqing Chen

College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China; 202130310119@stu.shmtu.edu.cn (J.L.); 202010311045@stu.shmtu.edu.cn (W.B.); 202140310003@stu.shmtu.edu.cn (C.C.)

* Correspondence: rjiang@shmtu.edu.cn

† These authors contributed equally to this work.

Abstract: In the era of deep learning as a service, ensuring that model services are sustainable is a key challenge. To achieve sustainability, the model services, including but not limited to storage and inference, must maintain model security while preserving system efficiency, and be applicable to all deep models. To address these issues, we propose a sub-network-based model storage and inference solution that integrates blockchain and IPFS, which includes a highly distributed storage method, a tamper-proof checking method, a double-attribute-based permission management method, and an automatic inference method. We also design a smart contract to deploy these methods in the blockchain. The storage method divides a deep model into intra-sub-network and inter-sub-network information. Sub-network files are stored in the IPFS, while their records in the blockchain are designed as a chained structure based on their encrypted address. Connections between sub-networks are represented as attributes of their records. This method enhances model security and improves storage and computational efficiency of the blockchain. The tamper-proof checking method is designed based on the chained structure of sub-network records and includes on-chain checking and IPFS-based checking stages. It efficiently and dynamically monitors model correctness. The permission management method restricts user permission based on the user role and the expiration time, further reducing the risk of model attacks and controlling system efficiency. The automatic inference method is designed based on the idea of preceding sub-network encrypted address lookup. It can distribute trusted off-chain computing resources to perform sub-network inference and use the IPFS to store model inputs and sub-network outputs, further alleviating the on-chain storage burden and computational load. This solution is not restricted to model architectures and division methods, or sub-network recording orders, making it highly applicable. In experiments and analyses, we present a use case in intelligent transportation and analyze the security, applicability, and system efficiency of the proposed solution, particularly focusing on the on-chain efficiency. The experimental results indicate that the proposed solution can balance security and system efficiency by controlling the number of sub-networks, thus it is a step towards sustainable model services for deep learning.

Keywords: deep learning; model services; sustainability; blockchain; InterPlanetary File System; intelligent transportation

check for updates

Citation: Jiang, R.; Li, J.; Bu, W.; Chen, C. Toward Sustainable Model Services for Deep Learning: A Sub-Network-Based Solution Integrating Blockchain with IPFS and a Use Case in Intelligent Transportation. *Sustainability* **2023**, *15*, 15435. https://doi.org/10.3390/su152115435

Academic Editors: Wei Liang, Asis Kumar Tripathy, Tien-Hsiung Weng and Xiong Li

Received: 18 September 2023
Revised: 16 October 2023
Accepted: 24 October 2023
Published: 30 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Deep learning has entered the era of large models. Due to limitations in computing resources, expertise, and time, it is challenging for ordinary users to deploy, maintain, and utilize deep models. This has given rise to the concept of Deep Learning as a Service (DLaaS) [1,2]. In DLaaS, model services are at the core. Two fundamental model services are to store models for model owners, referred to as model storage, and to generate results for model users using the stored models, referred to as model inference. Sustainability

is an essential requirement for model services. Model services are provided through a service system. Sustainable model services require the system to be able to provide services stably over the long term, which necessitates the system to ensure the security of the model. A model involves intellectual property and service quality, and if it is attacked, related model services may become disrupted or ineffective, resulting in losses for both model owners and users. Particularly in fields closely related to public safety, such as intelligent transportation, attacks on models can lead to significant incidents. Various attacks and protection methods for deep models have been widely discussed. Model security vulnerabilities include model tampering [3], model inputs tampering [4,5], model extraction [6] and model leakage [7]. Currently, methods to protect model security can be divided into two main categories. One category includes algorithms to defend against attacks [8]. The other focuses on secure model storage techniques to ensure model security by minimizing the possibility of attacks [9]. Sustainable model services also require the system to be applicable to all deep models. Only a widely applicable model service system can continuously meet various demands; otherwise, it may be replaced by more versatile systems. Sustainable model services also involve the optimal utilization of resources, necessitating the system to balance storage efficiency and computational efficiency while ensuring security. If the efficiency of a model service system is low, it may face obsolescence because it cannot meet the requirements for resource sustainability.

Existing works have already explored to maintain sustainability for deep model services. Blockchain [10], as a highly secure digital ledger technology, is commonly used as a secure storage platform for building a model service system. To further enhance the security and efficiency of the system, integrating blockchain with other technologies has become a recent research focus. Common technologies include data encryption [11,12], auditing and monitoring [13,14], access control [15], and the InterPlanetary File System (IPFS) [16]. How to design and integrate specific techniques to create a secure, efficient, and highly applicable system that supports sustainable model services for deep learning is an issue worthy of in-depth research. To address this issue, ref. [17] introduced an analytical framework for machine learning that combines blockchain with data encryption to protect model security, while also proposing the use of a binary-based analysis storage format (ASTORE) to store models, improving the system efficiency. Ref. [18] combined blockchain with data encryption, auditing and monitoring, proposing a method called DeepRing to protect deep neural networks. This method can prevent model tampering. Ref. [19] introduced a deep model evaluation framework. This work suggested using the IPFS as an off-chain auxiliary part to store the entire model, only recording the model in the blockchain, managing model permissions using single-attribute-based on-chain access control, and utilizing trusted off-chain computational resources for centralized model inference. This approach is versatile and practical to some extent.

The above-mentioned works introduce new ideas for maintaining the sustainability of deep model services but also have some limitations. Ref. [17] only employs data encryption to protect model security. Although it proposes using a new format for model storage, it has not discussed how to manage model usage permissions or introduce auditing and monitoring mechanisms to prevent model tampering, resulting in significant security gaps. Additionally, ref. [17] does not provide a corresponding inference method for models in the ASTORE format. While DeepRing [18] significantly improves security, it assumes that each layer has weights, biases, and activation functions, and that each layer has only one subsequent layer. This means that the method cannot handle complicated network architectures and lacks universality. Moreover, storing the entire model in the blockchain and performing model inference on-chain significantly reduces system efficiency. The solution proposed in [19] has issues as well. On one hand, it does not restrict usage time for models. On the other hand, model information is concentrated in the IPFS, which could lead to the leakage of all model information if the model address is compromised. Additionally, the centralized model inference also poses security risks.

To achieve sustainable model services for deep learning that have good applicability and balance security and system efficiency, we propose a sub-network-based solution for model storage and inference that integrates blockchain with IPFS. The specific contributions are as follows:

(1) A highly distributed storage method based on blockchain and IPFS is proposed. Specifically, a deep model is decomposed into a series of sub-networks, dividing the whole model into intra-sub-network and inter-sub-network information. Sub-network files are uploaded to the IPFS, while the encrypted address of the sub-networks and their connection information are organized into a chained structure recorded in the blockchain. This method makes it difficult to leak the complete model information and helps alleviate the storage burden in the blockchain, thus enhancing system efficiency, and can achieve a balance between security and system efficiency by adjusting the number of sub-networks.
(2) A two-stage tamper-proof checking method based on the chained structure of sub-network records is designed. This method checks the information of stored models, including an on-chain checking stage and an IPFS-based checking stage. It can dynamically and continuously monitor model correctness, further protecting the model from tampering caused by security vulnerabilities in the blockchain and the IPFS.
(3) A double-attribute-based model usage permission management method is presented. This method employs attribute-based on-chain access control, granting permission to a user based on the role and the expiration time. Only authorized users with permission are allowed to perform inference using the model but are restricted from accessing and downloading the model. This enhances supervision on users, further mitigates security risks, and controls system efficiency.
(4) An automatic inference method based on preceding sub-network encrypted address lookup is proposed. This method divides the entire model inference process into distributed off-chain sub-networks inference processes. Model inputs and sub-network outputs are uploaded to the IPFS, and their encrypted address are recorded in the blockchain. This records and pretects the inference results while further alleviating the on-chain storage and computational burden.
(5) A smart contract called Model Contract is designed to deploy all four methods mentioned above into the blockchain, enabling them to be automatically executed in a decentralized manner.
(6) A use case in the field of intelligent transportation is provided to demonstrate the enhancement of vehicle detection reliability using the proposed solution. The sustainability of this solution is analyzed based on this use case, including its security, applicability, and system efficiency. Additionally, an analysis of the on-chain efficiency sustainability of this solution is conducted based on a simulation experiment.

The rest of this paper is organized as follows. Section 2 presents key issues, related technologies, and existing works of model services for deep learning. Section 3 describes the proposed solution. In Section 4, we provide a use case in intelligent transportation and conduct analyses on the proposed solution. Section 5 serves as the conclusion to this paper.

2. Related Works

In this section, we first discuss the key issues of model services for deep leaning, then introduce the fundamental technologies for building a system for model services, and finally present the current research status of model services for deep learning.

2.1. Key Issues of Model Services for Deep Learning

The development of deep models has shown a trend towards larger scales and more complex architectures. Early deep convolution neural networks were constructed by stacking convolutional and fully connected layers together. The scales have evolved from small to large over time. For instance, LeNet-5 [20], proposed in 1998, includes two convo-

lutional layers and three fully connected layers, with around 60 thousand parameters. AlexNet [21], introduced in 2012, comprises eight convolutional layers and three fully connected layers, with approximately 60 million parameters. Visual Geometry Group (VGG) networks [22] were introduced in 2014, and the VGG-16 has thirteen convolutional layers and three fully connected layers, in total around 138 million parameters. As models grew in scale, there were architectural innovations. For example, ResNet [23], proposed in 2015, introduced residual connections. Inception [24], also proposed in 2015, featured parallel branches with various-sized convolution kernels. More recently, Google's Transformer [25] architecture gained attention. It is designed for sequence data and incorporates encoder-decoder architecture, multi-head attention and self attention mechanisms, skip connections, and layer normalization layers, and can handle long-range dependencies effectively. This success led to models built on this architecture, including Google's Bidirectional Encoder Representations from Transformers (BERT) [26] and OpenAI's Generative Pre-trained Transformers (GPT) [27]. The BERT-large has around 300 million parameters, while GPT-3 has about 175 billion parameters. Deep models like BERT and GPT fall into the category of large models.

The security of deep models is also becoming increasingly important, especially in fields closely related to public safety, such as intelligent transportation. The security vulnerabilities of models involve the following aspects. On one hand, if a model can be easily accessed by attackers, they can manipulate the model. Manipulation can take the form of directly deleting or damaging the existing architecture and parameters of the model, as well as implanting specific architectures such as backdoors causing erroneous outputs when the input triggers the backdoor condition. On the other hand, if model information is easily obtained by attackers [7], they can study the model and add designed perturbations to the model input, resulting in incorrect outputs. These attack forms are referred to as adversarial attacks [5] in the literature. Adversarial attacks initiated based on complete model information are known as white-box attacks, while those initiated based solely on the model's input and output are known as black-box attacks. Additionally, if attackers have unlimited access to the model, they can also use machine learning to extract models for illicit purposes [6]. Currently, methods to protect model security can be categorized into two types. One enhances the model robustness against attacks by designing algorithms, for example, adversarial training techniques [8] to withstand adversarial attacks. The other type explores secure storage techniques [9] to ensure model security by minimizing the likelihood of attacks. The proposed model storage method in this paper falls into the latter category.

Training, deploying, maintaining and using deep models are difficult tasks for ordinary users due to limitations related to computing resources, expertise, and time. This challenge has given rise to the concept of Deep Learning as a Service (DLaaS) [1,2]. The core of DLaaS is model services, which includes two fundamental services: model storage, involving the storage of models for model owners, and model inference, involving the generation of results for model users using the stored models. Model services are provided through a service system. The system needs to be capable of consistently delivering services over the long term while considering the efficient utilization of resources to ensure the sustainability of model services. Specifically, the system must ensure the security of models to protect intellectual property and service quality. The system also needs to be applicable to all deep models to ensure good service versatility. Additionally, the system must balance storage and computational efficiency to conserve resources and energies while maintaining model services.

2.2. Fundamental Technologies for Building Model Service Systems for Deep Learning

There are many fundamental technologies involved in building model service systems for deep learning. Technologies related to this paper include data encryption, access control, auditing and monitoring, blockchain, and the IPFS.

Data encryption is one of the most prevalent secure data technologies [11,12,28]. It involves converting data into ciphertext, ensuring that authorized users can decrypt the data. Encryption techniques can be divided into symmetric and asymmetric encryption, each differing in terms of encryption and decryption processes, speed, and transmission security. Firstly, symmetric encryption [29,30] uses one key for both encryption and decryption, whereas asymmetric encryption [31,32] employs two keys: a public one and a private one. Secondly, symmetric encryption and decryption are faster, suitable for processing large volumes of data. Asymmetric encryption and decryption take more time and are relatively slower, making them more suitable for small-scale data. Taking security and efficiency into consideration, the solution proposed in this paper adopts symmetric encryption.

Access control is a frequently used information security technology [15], and has been applied in various fields such as online courses [33], medical device supply chains [34], port supply chains [35], and the Internet of Things [36]. It refers to managing or restricting users' access permissions to data, ensuring the information inaccessible to unauthorized users. The solution proposed in this paper adopts the Attribute-Based Access Control (ABAC) [37]. The ABAC determines access based on users' attributes, providing a better solution for complex security requirements. The general process of verifying access control permissions is as follows: access control policies are predefined by data owners to establish rules for resource access control. Access control requests are submitted by users, and then compared against the predefined access control policies. If they match, authorization is granted; otherwise, access is denied. Centralized access control can pose risks. Having access control policies stored centrally can lead to a single-point-of-failure. And, centralized access control requires third-party supervision, also introducing the tampering risk.

Auditing and monitoring are technologies used to keep the security and compliance of data in various systems [9,14], originally applied in financial systems. These techniques aim to ensure that data is adequately protected to prevent data leaks, tampering, or other security vulnerabilities. In the field of data storage, event logs are used to record important events that occur in the storage system, such as accesses, modifications, deletions, etc., for the purpose of auditing. Integrity monitoring is employed to prevent and detect data tampering, verifying the integrity of data through techniques like digital signatures [13]. Real-time monitoring involves ongoing checks on the storage system to promptly identify anomalies. Compliance checks refer to examining whether the stored data adheres to specific requirements.

Blockchain is a highly secure digital ledger technology, characterized by decentralization, distributed consensus, and immutability [10,38]. It is often used as a secure data storage platform. The decentralization implies that data is stored across multiple nodes, enhancing system robustness. The distributed consensus mechanism uses consensus algorithms to verify and confirm transactions; only verified transactions can be added, preventing malicious transactions and tampering. The immutability refers to the fact that each block includes the hash value of the preceding block, forming a chained data structure. Modifying information within a block would change the hash values of all subsequent blocks, making tampering difficult. Furthermore, blockchain supports smart contracts, enabling automated on-chain operations and reducing the risk of third-party intervention. Despite its high security, the security of the blockchain is not absolute. For instance, the 51% attack is a common attack on the blockchain [39]. Attackers gain control over more than 51% of the computational power or the number of validating nodes in the blockchain network, enabling them to maliciously manipulate and attack the network. In such cases, attackers can tamper with information in the blockchain, reverse transactions, initiate attacks, and disrupt the confirmation of legitimate transactions, thereby compromising the data security.

The IPFS is a decentralized peer-to-peer file storage system [16]. Unlike traditional web protocols that rely on location-based addressing, the IPFS uses content addressing. This

means that a file's hash value, known as the Content Identifier (CID), serves as its unique identifier. This approach allows files to be located and retrieved based on their content, independent of specific servers or locations. This content-addressing method helps achieve secure data storage and can also be used to detect file tampering. Compared to traditional centralized cloud storage services, the IPFS offers advantages such as decentralized storage and high-performance data transmission. Unlike the blockchain, the IPFS does not support smart contracts. The IPFS has already been employed for secure storage of industrial images [40], medical data [41], and vehicle network data [42]. While the IPFS provides a certain level of security, its security is not absolute. Factors such as the presence of malicious nodes, CID leakage, or external attacks can impact the security and availability of data.

2.3. Current Research Status of Model Services for Deep Learning

Integrating multiple technologies effectively based on blockchain is a recent research focus in the field of building model service system for deep learning. Ref. [17] proposed the integration of blockchain and data encryption, utilizing a specialized binary data format known as Analyzed Store Format (ASTORE) to store machine learning models, thus enhancing data efficiency within the system. However, ref. [17] did not discuss model usage permissions or an auditing and monitoring mechanism for model tamper resistance, nor did they elucidate how model inference occurs within the ASTORE format. Ref. [18] combined data encryption, auditing and monitoring, and blockchain to introduce a protective approach for deep neural networks named DeepRing. This method designs each layer of the deep network as a block, with each block containing functionalities to store layer parameters, compute layer outputs, update a ledger after output computation, and validate subsequent layer outputs to prevent model tampering. Nonetheless, DeepRing assumes that each layer possesses weights, biases, and activation functions, and that each layer only connects to one another layer. This limitation implies that the approach cannot handle complicated network structures. Furthermore, storing the entire model in the blockchain could potentially have an adverse impact on system efficiency, as it requires a significant amount of on-chain storage space and on-chain computational resources. To alleviate the storage and computational burden in the blockchain, ref. [19] proposes the use of the IPFS as an off-chain auxiliary storage for model files [43], reserving the blockchain solely for recording model-related information. Moreover, ref. [19] deploys access control through a smart contract to manage model usage permissions, while model inference is performed using trusted off-chain computational resources. One drawback of the solution proposed by [19] is that, if the CID of a model is leaked, all model information is compromised. Additionally, adopting a centralized processing approach for model inference can introduce security vulnerabilities. Overall, current research in model services for deep learning has not yet proposed mechanisms that are capable of effectively balancing security and system efficiency, and achieved good applicability, the study of sustainability in deep model services is still in its early stage.

3. Proposed Sub-Network-Based Solution Integrating Blockchain with IPFS for Deep Model Storage and Inference

To address the issues faced by current deep model services, a sub-network-based solution integrating blockchain with IPFS for fundamental model services, including storage and inference, is proposed. This solution incorporates mechanisms for balancing security and system efficiency, and based on this solution, systems supporting sustainable model services for deep learning can be developed. The solution involves four aspects: storage, tamper-proof checking, permission management, and inference, as illustrated in Figure 1. Subsequently, a detailed description will be provided for each aspect. The functions employed in the algorithms are listed in Table 1.

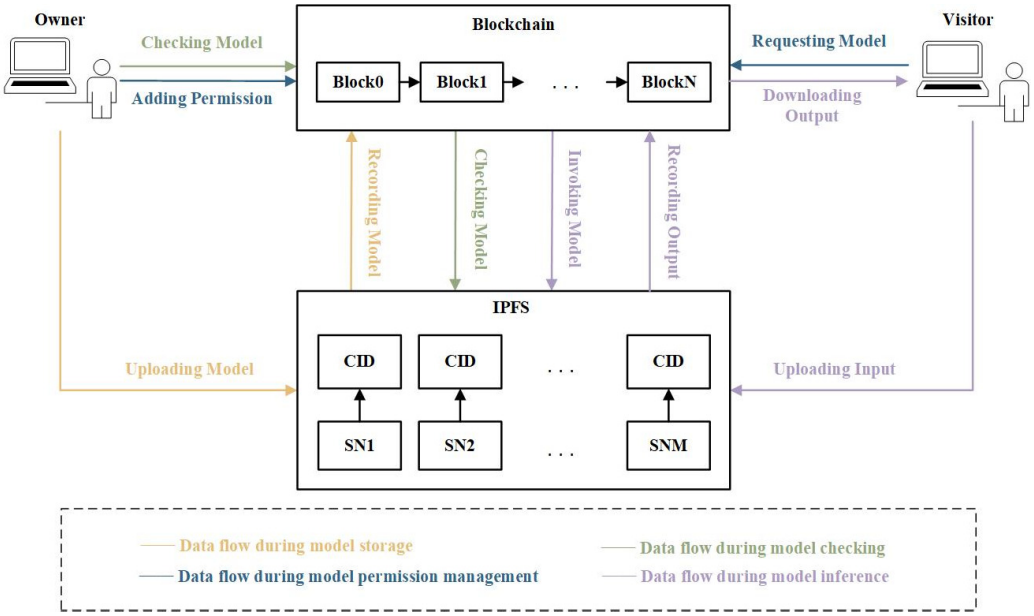


Figure 1. Proposed sub-network-based solution integrating blockchain with IPFS for deep model storage and inference.

Table 1. Descriptions of functions.

Function	Description
Size()	Returning the length of a list
UploadToIPFS(v)	Uploading the file v to the IPFS and retrieve its CID
Add(r)	Adding an element r to a list or dictionary
Encrypt(x, k)	Encrypting x with key k
Hash($x_1, x_2, ..., x_n$)	Returning the identifier of a record through $x_1, x_2, ..., x_n$
APIstub.PutState(x, v)	Uploading v to the blockchain and assigning x as its index
Marshal($x_1, x_2, ..., x_n$)	Packaging $x_1, x_2, ..., x_n$
Linux.time.Now()	Retrieving the current time in Linux system
APIstub.GetState(x)	Searching with index x from the blockchain state database
Reverse()	Reversing the elements of a list
Decrypt(x, k)	Decrypting x with key k
DownloadFromIPFS(x)	Downloading the file with CID x from the IPFS
Inference(L, M)	Using a input list L to infer the output of a network M

3.1. Model Storage

Considering the model security and the system efficiency, the proposed solution extends the storage space of the blockchain using the IPFS. Model storage employs a highly decentralized approach. Model owners first decompose the deep model to be stored into a series of sub-networks according to their needs, thereby dividing the information of the entire model into the information within the sub-networks, i.e., the files corresponding to the sub-networks, and the information between the sub-networks, i.e., the interconnections between sub-networks. When storing the model, the first step involves uploading files of the sub-networks to the IPFS, obtaining the corresponding CID for each file. Subsequently, the smart contract encrypts the CIDs using the key provided by the owner, i.e., *ModelKey*. Then, based on the interconnections between sub-networks provided by the owner, all encrypted CIDs are uploaded to the blockchain as records of the sub-networks in the blockchain. The structure of this record is designed as follows:

$$SNRecord = \{SNID, PreSNID, PreSNEncryCID, SNEncryCID\},$$

where $SNEncryCID = \text{Encrypt}(CID, ModelKey)$ represents the encrypted CIDs of the sub-network file in the IPFS using the $ModelKey$. $PreSNEncryCID$ is a list of encrypted CIDs of the previous sub-networks adjacent to the current sub-network.

$$SNID = \text{Hash}(PreSNID, PreSNEncryCID, SNEncryCID) \tag{1}$$

serves as the identifier for the current sub-network record. $PreSNID$ refers to the identifier of the preceding record, making the $PreSNID$ of the first record empty. Based on the sub-network records in the blockchain, the model's record in the blockchain is designed as follows:

$$ModelRecord = \{ModelID, SNID_List, ModelName, OwnerID, Time\},$$

where $SNID_List$ is a list composed of $SNID$s from the sub-network records, and its order represents the sequence of recording the sub-networks. We do not impose any restrictions on the order of recording the sub-networks, and the order of $SNID_List$ does not need to match the sequence in which the sub-networks appear in the model.

$$ModelID = \text{Hash}(SNID_List, Time). \tag{2}$$

$ModelName$, $OwnerID$, and $Time$ are the model name, the identifier of the model owner, and the time of recording the model, respectively. The design of the sub-network records and the model record is related to tamper-proof checking, permission management, and automated inference, which will be explained in the following sections.

To accomplish decentralized and efficient model management, a smart contract named Model Contract (MC) has been designed. AddModel() is a method within MC, used for uploading the sub-network files to the IPFS and recording the sub-networks and the model in the blockchain, which is displayed in Algorithm 1. FindByID() is another method within MC, serving the purpose of searching for a specific record within the blockchain.

Algorithm 1 AddModel(): Uploading sub-network files of a model to the IPFS and recording the sub-networks and the model in the blockchain

Require: $SNF_List, PreSNF_List, ModelKey, ModelName, OwnerID$
Ensure: OK
for $k = 1$ **to** SNF_List.Size() **do**
  $SNCID \leftarrow \text{UploadToIPFS}(SNF_List[k])$
  $SNCID_List$.Add($SNCID$)
end for
$PreSNID \leftarrow$ Null
$SNID_List \leftarrow$ Null
for $k = 1$ **to** $SNCID_List$.Size() **do**
  $PreSNCID \leftarrow [SNCID | SNF \in PreSNF_List[k]$ and $SNCID \in SNCID_List]$
  for $j = 1$ **to** $PreSNCID$.Size() **do**
    $PreSNEncryCID$.Add(Encrypt($PreSNCID[j], ModelKey$))
  end for
  $SNEncryCID \leftarrow \text{Encrypt}(SNCID_List[k], ModelKey)$
  $SNID \leftarrow \text{Hash}(PreSNID, PreSNEncryCID, SNEncryCID)$
  APIstub.PutState($SNID$, Marshal($PreSNID, PreSNEncryCID, SNEncryCID$))
  $PreSNID \leftarrow SNID$
  $SNID_List$.Add($SNID$)
end for
$Time \leftarrow$Linux.time.Now()
$ModelID \leftarrow \text{Hash}(SNID_List, Time)$
APIstub.PutState($ModelID$, Marshal($SNID_List, ModelName, OwnerID, Time$)))
return OK

3.2. Model Tamper-Proof Checking

The proposed solution chooses to store the model in the IPFS and the blockchain due to their immutability. However, as mentioned in Section 2, this immutability is not absolute. In order to further prevent model tampering caused by security vulnerabilities in the IPFS and the blockchain, we designed a two-stage tamper-proof checking method for the stored models. Inspired by the anti-tampering mechanism of the blockchain, the identifier of the current sub-network record, i.e., $SNID$, is designed as a function of the identifier of the preceding sub-network record, i.e., $PreSNID$, while also being related to the encrypted CIDs in the record, i.e., $PreSNEncryCID$ and $SNEncryCID$, as shown in Equation (1). This implies that sub-networks are recorded using a chained structure. If $PreSNID$, $PreSNEncryCID$, or $SNEncryCID$ changes, $SNID$ will also change. Due to this design, in the proposed method, the model owner needs to back up the $SNID_List$, referred to as $OrigSNID_List$, and all $SNEncryCID$s, referred to as $OrigSNEncryCID$s, upon completion of the initial model storage. This backup is used for tamper-proof checking and restoring the model information in the blockchain.

Tamper-proof checking should be completed before the model is delivered for use. It includes two stages: the first stage checks whether the model information in the blockchain has been tampered with, and the second stage checks whether the model information in the IPFS has been tampered with. Specifically, in the first stage, a comparison is made between the current $SNID_List$ and $OrigSNID_List$ to determine their consistency. This comparison is performed in reverse order starting from the last element of both lists, until the first inconsistent $SNID$ is found. The reason for the change in this $SNID$ is that the corresponding $PreSNEncryCID$ or $SNEncryCID$ has been altered. The model owner examines the specific reason for the change in this record and re-records the corresponding sub-network and model. The process of checking tampered sub-network records is designed as a method in the smart contract MC, as illustrated in Algorithm 2. For the new model record, repeat the above checking and restoration process until the $SNID_List$ of the latest model record is identical to $OrigSNID_List$. This concludes the first checking stage. In the second stage, the smart contract decrypts the $SNEncryCID$ of each sub-network record and retrieves the corresponding CID from the IPFS. If the CID is not found, it indicates that the sub-network file has been tampered with or deleted. In this case, the model owner needs to re-upload the file of that sub-network to the IPFS. The process of checking tampered sub-network files is designed as one of the methods in the smart contract MC, as illustrated in Algorithm 3. The second checking stage concludes when the CIDs of all sub-networks recorded in the blockchain can be retrieved from the IPFS.

Algorithm 2 TPCI(): Checking whether the model information in the blockchain has been tampered with in the first stage of the tamper-proof checking

Require: $OrigSNID_List$, $ModelID$
Ensure: OK or Error
1: $ModelRecord \leftarrow$ APIstub.GetState($ModelID$)
2: **if** $ModelRecord ==$ Null **then**
3: **return** Error("The model record cannot be found.")
4: **end if**
5: $f \leftarrow$ Null
6: **for** $k = 1$ **to** $OrigSNID_List$.Size() **do**
7: **if** $OrigSNID_List$.Reverse()[k] ! = $ModelRecord.SNID_List$.Reverse()[k] **then**
8: $f \leftarrow OrigSNID_List$.Size() $- k + 1$
9: **end if**
10: **end for**
11: **if** f ! = Null **then**
12: **return** Error("The fth subnetwork record has been tampered with!")
13: **end if**
14: **return** OK

Algorithm 3 TPCII(): Checking whether the model information in the IPFS has been tampered with in the second stage of the tamper-proof checking

Require: $ModelID$, $ModelKey$
Ensure: OK or Error
1: $ModelRecord \leftarrow$ APIstub.GetState($ModelID$)
2: **for** $k = 1$ **to** $ModelRecord.SNID_List$.Size() **do**
3: $SNRecord \leftarrow$ APIstub.GetState($ModelRecord.SNID_List[k]$)
4: $SNCID \leftarrow$ Decrypt($SNRecord.SNEncryCID, ModelKey$)
5: $response \leftarrow$ DownloadFromIPFS($SNCID$)
6: **if** $response == $ Null **then**
7: break;
8: **end if**
9: **end for**
10: **if** $response == $ Null **then**
11: **return** Error("The subnetwork with the $SNCID$ has been tampered with in the IPFS!")
12: **end if**
13: **return** OK

3.3. Model Permission Management

In addition to strengthening the supervision of model correctness, the proposed solution also enhances the restrictions on usage permission to the model. This helps maintain model security and system efficiency. By deploying the ABAC model in the blockchain, it manages model usage permissions based on user roles and expiration time. Specifically, a permission is defined as

$$Permission = \{UserID, Role, ModelID, Time\}.$$

Here, $UserID$ represents the user's identifier, $Role$ indicates the user's identity, which can be an owner or a visitor. Owners have the authority to add the visitor role for other users. The visitors are only granted permission to utilize the model but are not allowed to access or download the model. Users without any roles have no permissions regarding the model. $ModelID$ is the identifier of the model. $Time$ represents the expiration time for this permission. The record structure for each permission in the blockchain is set as $\{PermissionID, Permission\}$. $PermissionID$ is the identifier, calculated as

$$PermissionID = \text{Hash}(Permission.UserID, Permission.ModelID, Permission.Role).$$

User's usage permission to the model must undergo an permission verification, following these specific steps: first, the user submits an permission request as

$$Request = \{UserID, ModelID, Role\}.$$

Then, the identifier for this request is generated as

$$RequestID = \text{Hash}(Request.UserID, Request.ModelID, Request.Role).$$

Permission verification attempts to retrieve the permission with $PermissionID$ equal to $RequestID$ from the on-chain state database. If no permission is found, it means the user does not have the right to use the model. If a permission record is found, further permission verification is conducted based on the expiration time. After the permission verification is successful, the model owner checks the model and provides the latest $ModelID$ and $ModelKey$ to the smart contract, which are not disclosed to the user.

To achieve decentralized model usage permission management, the smart contract MC has been designed with the AddPermission(), QueryPermission() and CheckPermission()

methods. AddPermission() is used to add permission records to the blockchain, where owner permissions are added by the smart contract during model uploading, and visitor permissions are added by the owner. QueryPermission() utilizes the FindByID() method defined in Section 3.1 to search for permission records. CheckPermission() handles permission verification by first generating a model usage permission request, then using QueryPermission() to search for a permission record. Subsequently, it further verifies the permission based on the expiration time and returns the verification result, as outlined in Algorithm 4.

Algorithm 4 CheckPermission(): Checking a user's permission

Require: *Request*
Ensure: OK or Error
1: **/** Generating a request for using the model. **/**
2: **if** *Request.UserID* == Null or *Request.ModelID* == Null or *Request.Role* == Null **then**
3: **return** Error("The request is incomplete.")
4: **end if**
5: *RequestID* ← Hash(*Request.UserID*, *Request.ModelID*, *Request.Role*)
6: **/** QueryPermission(). **/**
7: *Permission* ← APIstub.GetState(*RequestID*)
8: **if** *Permission* == Null **then**
9: **return** Error("The permission has not been submitted.")
10: **end if**
11: **/** Verifying permission with expiration time. **/**
12: **if** *Request.Role* == *Permission.Role* and *Request.Role* == Owner **then**
13: **return** OK
14: **else if** *Request.Role* == *Permission.Role* and *Request.Role* == Visiter **then**
15: **if** Linux.time.Now() > *Permission.Time* **then**
16: **return** Error("The permission has expired.")
17: **else**
18: **return** OK
19: **end if**
20: **end if**
21: **return** Error("The request is invalid.")

3.4. Model Inference

In response to the proposed model storage method, an automatic inference method based on encrypted CID lookup of preceding sub-networks has been designed. This method is independent of the order of sub-network recording. Model users upload the model input, i.e., *ModelInput*, to the IPFS and provide the encrypted key for the model output, i.e., *OutputKey*, which is also used for encrypting and decrypting the CID of the model input. Model owners provide the *ModelID* and *ModelKey*. The smart contract invokes the sub-networks by looking up the encrypted CID of the preceding sub-networks, following these steps: firstly, using the *ModelID*, the smart constract calls the *SNID_List* of the *ModelRecord*, then iterates through the *SNID_List* to find *SNRecord*s with an empty *PreSNEncryCID*. These *SNRecord*s correspond to the sub-networks that appear earliest in the model. And thus, these sub-networks can be inferred using the provided model inputs, the results of the sub-network inference are stored in the IPFS, with the encrypted CIDs of these outputs being stored in the blockchain. Subsequently, the *SNEncryCID* of sub-networks that have undergone inference are recorded in a list, referred to as *InferedSNEncryCID_List*. Among the remaining *SNRecord*s, the smart contract searches for *SNRecord*s with *PreSNEncryCID* being a subset of the current *InferedSNEncryCID_List*. This signifies that the inputs for these sub-networks have been generated, enabling inference to be performed on them. The outputs of these

sub-networks are stored in the IPFS, and the encrypted CIDs of the outputs are stored in the blockchain. The smart constract updates the $InferedSNEncryCID_List$ thereafter, then continue following the above process to perform inference on the remaining sub-networks and record their outputs in the blockchain until all sub-networks have been inferred, that is, until the lengths of $InferedSNEncryCID_List$ and $SNID_List$ are the same. The record of a model output in the blockchain is designed as follows:

$$ModelOutputRecord = \{ModelOutputID, ModelID, ModelInputEncryCID, SNOutput_Dict, Time\},$$

where $SNOutput_Dict = \{SNEncryCID{:}SNOutputEncryCID\}$, and

$$ModelOutputID = \text{Hash}(ModelID, ModelInputEncryCID, Time). \quad (3)$$

Here, $ModelInputEncryCID$ is the encrypted CID of the $ModelInput$, $SNOutputEncryCID$ is the encrypted CID of the outputs of the sub-networks. We have designed the above process as a method named ModelInference() within the smart contract MC, as shown in Algorithm 5. There are three implementation methods for sub-network inference: off-chain inference, on-chain inference, and hybrid inference. Off-chain inference refers to the scenario where, during automated inference, the sub-networks in the IPFS are not uploaded to the blockchain. Instead, the smart contract allocates off-chain trusted computing resources randomly, the corresponding computing platform, using the decrypted CIDs, retrieves the sub-network and input files from the IPFS and performs the off-chain inference, then uploads the encrypted CIDs of the output file to the blockchain. On-chain inference refers to the scenario where, during automated inference, the sub-network and input files in the IPFS are gradually uploaded to the blockchain as the smart contract invoked. Then the smart contract uses on-chain computing resources to perform the inference. Hybrid inference refers to a situation where, during automated inference, the smart contract can utilize both off-chain and on-chain computing resources. Algorithm 5 utilizes the off-chain inference method; see Algorithm 6.

Algorithm 5 ModelInference(): Performing model inference after tamper-proof checking and permission granting.

Require: $ModelID, ModelKey, OutputKey, ModelInput$
Ensure: OK or Error

$ModelInputCID \leftarrow$ UploadToIPFS($ModelInput$)
$ModelInputEncryCID \leftarrow$ Encrypt($ModelInputCID, OutputKey$)
$SNOutput_Dict \leftarrow$ Null
$InferedSNEncryCID_List \leftarrow$ Null
$InferedSNID_List \leftarrow$ Null
while $InferedSNID_List$.Size() ! = $SNID_List$.Size() **do**
    $CurrSNID_List \leftarrow ModelRecord.SNID_List - InferedSNID_List$
    $CurrSNInputEncryCID_List \leftarrow$ Null
    $CurrSNEncryCID_List \leftarrow$ Null
    for $k = 1$ **to** $CurrSNID_List$.Size() **do**
        $SNRecord \leftarrow$ APIstub.GetState($CurrSNID_List[k]$)
        if $SNRecord.PreSNEncryCID \subseteq InferedSNEncryCID_List$ **then**
            $CurrSNEncryCID_List$.Add($SNRecord.SNEncryCID$)
            if $SNRecord.PreSNEncryCID ==$ Null **then**
                $CurrSNInputEncryCID_List$.Add($[ModelInputEncryCID]$)
            end if
            $CurrSNInputEncryCID_List$.Add($[SNOutput_Dict(SNRecord.PreSNEncryCID)]$)
            $InferedSNID_List$.Add($SNRecord.SNID$)
        end if
    end for
    for $k = 1$ **to** $CurrSNEncryCID_List$.Size() **do**
        $CurrSNOutputEncryCID \leftarrow$ OffChainSNInference($CurrSNInputEncryCID_List[k]$,
        $CurrSNEncryCID_List[k], ModelKey, OutputKey$)
        if $CurrSNOutputEncryCID ==$ Error **then**
            return Error
        end if
        $SNOutput_Dict$.Add($CurrSNEncryCID_List[k] : CurrSNOutputEncryCID$)
    end for
    $InferedSNEncryCID_List$.Add($CurrSNEncryCID_List$)
end while
$Time \leftarrow$ Linux.time.Now()
$ModelOutputID \leftarrow$ Hash($ModelID, ModelInputEncryCID, Time$)
APIstub.PutState($ModelOutputID$, Marshal($ModelID, ModelInputEncryCID$,
$SNOutput_Dict, Time$)
return OK

Algorithm 6 OffChainSNInference(): Performing sub-network inference through off-chain computation.

Require: $SNInputEncryCID_List, SNEncryCID, ModelKey, OutputKey$
Ensure: $SNOutputEncryCID$ or Error
1: $SNInput_List \leftarrow$ Null
2: **for** $k = 1$ **to** $SNInputEncryCID_List$.Size() **do**
3: $SNInputCID \leftarrow$ Decrypt($SNInputEncryCID_List[k], OutputKey$)
4: $SNInput \leftarrow$ DownloadFromIPFS($SNInputCID$)
5: $SNInput_List$.Add($SNInput$)
6: **end for**
7: $SNCID \leftarrow$ Decrypt($SNEncryCID, ModelKey$)
8: $SN \leftarrow$ DownloadFromIPFS($SNCID$)
9: $SNOutput \leftarrow$ Inference($SNInput_List, SN$)
10: **if** $SNInput_List == $ Null or $SN == $ Null or $SNOutput == $ Null **then**
11: **return** Error("The inference fails.")
12: **end if**
13: $SNOutputCID \leftarrow$ UploadToIPFS($SNOutput$)
14: $SNOutputEncryCID \leftarrow$ Decrypt($SNOutputCID, OutputKey$)
15: **return** $SNOutputEncryCID$

4. Experiments and Analyses

In this section, we first apply the proposed solution to vehicle detection in intelligent transportation, demonstrating how this solution enhances the reliability of vehicle detection by improving model security through a functional demonstration. Then, a security and applicability analysis of the proposed solution is conducted. Finally, we present experimental results and analysis regarding system efficiency of the proposed solution, consisting of two parts: an analysis of system efficiency based on the above use case and an analysis of the on-chain efficiency through results of a simulation experiment. All experiments were conducted using an platform based on the Hyperledger Fabric consortium blockchain [44]. A distributed container network comprising four nodes was built on the Docker Swarm platform. Data encryption was performed using the Advanced Encryption Standard [30]. The hashing algorithm used is SHA-256 [45], and when using SHA-256, all inputs were concatenated into a single string before being input into the algorithm.

4.1. A Use Case in Intelligent Transportation

4.1.1. Background and Settings

Vehicle detection is an important issue in intelligent transportation, and methods based on deep learning have become a research focus in this field [46]; one category of methods is based on deep learning-based foreground segmentation [47,48]. This experiment is conducted based on a foreground segmentation network [49]. The network consists of an encoder, a Feature Pooling Module (FPM), and a decoder. Two skip-connection modules are used between the encoder and decoder to merge low-level and high-level features, thereby improving detection performance. We trained this model using a portion of frames from the "highway" video in the CDnet2014 dataset [50] along with their annotations, making it a vehicle detection network, as shown in Figure 2. The training process strictly followed the experimental settings in Section 2 of [49]. And the used training code is from https://github.com/lim-anggun/FgSegNet_v2 (accessed on 10 August 2023). Figure 3a displays three video frames from "highway" that were not used for training, while Figure 3b shows the inference results of the trained model on these three video frames. It can be observed that the model performs well in detecting vehicles in different scenarios of the same scene. All vehicles are segmented, and their contours are fairly accurate. However, if the model encounters a tampering attack, the detection results can change significantly. Figure 3c shows the inference results of injecting Gaussian noise into the skip connection

module SN5's convolutional kernels. It can be seen that all detection results become unrecognizable. In real-life scenarios, computer vision companies develop deep models for vehicle detection and provide detection services to users, such as traffic management departments, through a model service system. Vehicle detection must be reliable, and if the model is subjected to the tampering attacks mentioned above during the service, it could lead to severe incidents. Sustainable model services are crucial to ensuring reliable vehicle detection.

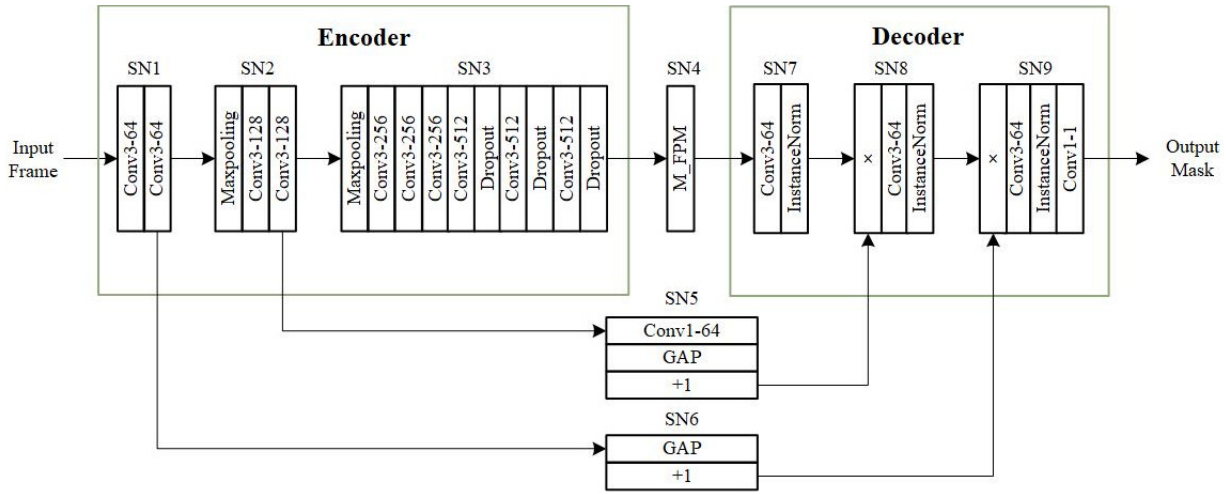


Figure 2. Architecture of the used vehicle detection network [49]. (SN: sub-network.)

4.1.2. Functional Demonstration

In this section, we demonstrate how the proposed solution, based on the vehicle detection model in Figure 2, ensures the reliability of detection from the perspectives of storage, tamper-proof checking, permission management, and inference. The following experiment involves two users, User1 and User2, with User1 acting as the model owner, and User2 as the model requester. Two servers, each equipped with a RTX 3090 GPU, were used for model inference.

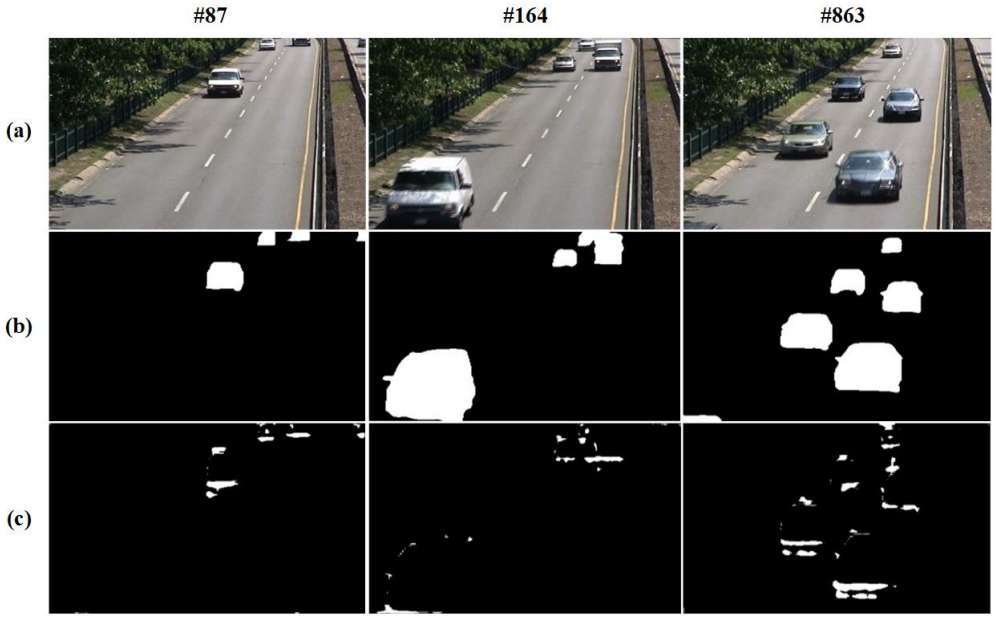


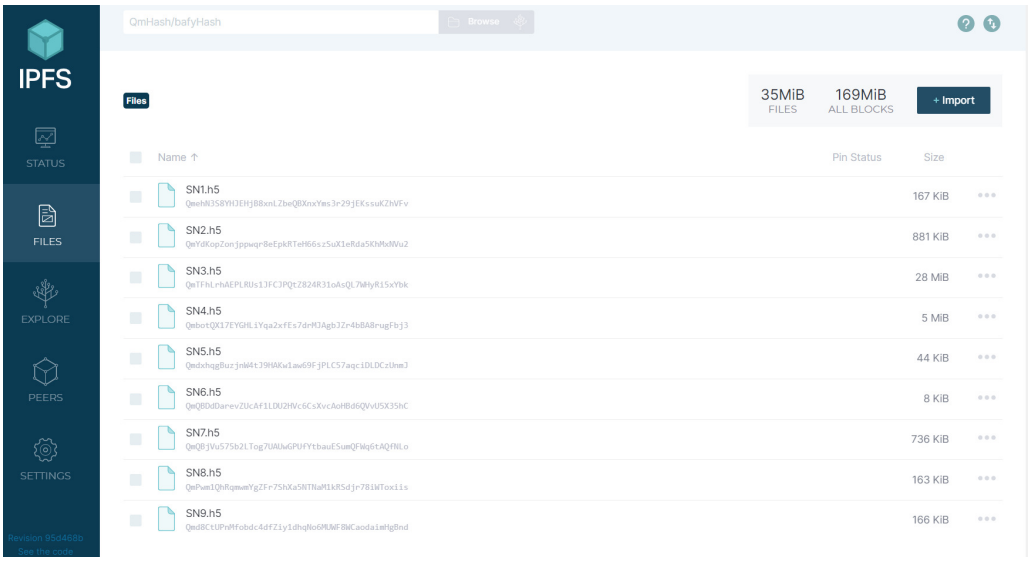
Figure 3. Three frames (**a**) from the "highway" along with the inference results of the normal vehicle detection model (**b**) and the tampered model (**c**). The tampered model is obtained by adding Gaussian noise following $N(0, 256)$ to the convolution parameters of SN5 in the normal model.

First, the vehicle detection model was well-distributed and stored based on the blockchain and the IPFS. The dispersion of model information increases the difficulty of obtaining the complete model information and prepares for tamper-proof checking, permission management, and inference in terms of data structure. The model owner initially divided the model into sub-networks. There are various ways to achieve this model division, and one of these ways is depicted in Figure 2. The model was divided into nine sub-networks. Subsequently, based on this division, the model owner uploaded these sub-network files to the IPFS, as illustrated in Figure 4a. In Figure 4a, SN1.h5, SN2.h5,..., SN9.h5 represent the files corresponding to the sub-networks, and beneath each file name is its CID. The interconnections among the nine sub-networks are defined using two lists:

$$SNF_List = [\mathrm{SN1}, \mathrm{SN2}, \mathrm{SN4}, \mathrm{SN5}, \mathrm{SN6}, \mathrm{SN7}, \mathrm{SN9}, \mathrm{SN8}, \mathrm{SN3}],$$

$$PreSNF_List = [[], [\mathrm{SN1}], [\mathrm{SN3}], [\mathrm{SN2}], [\mathrm{SN1}], [\mathrm{SN4}], [\mathrm{SN6}, \mathrm{SN8}], [\mathrm{SN5}, \mathrm{SN7}], [\mathrm{SN2}]].$$

In this setup, *SNF_List* keeps track of the order in which the sub-networks are uploaded to the IPFS, while *PreSNF_List* records the preceding sub-networks adjacent to each sub-network in *SNF_List*. By using the AddModel() method in the smart contract MC, the records of the sub-networks and the model were stored in the blockchain, completing the model storage process. Subsequently, in Figure 4b, we used the FindByID() method in MC, indicated by blue underlines, to display the records of SN8, SN3, and the model in the blockchain. As can be seen, the record of SN3 is preceded by the record of SN8, therefore the *PreSNID* of the record of SN3 is the *SNID* of the record of SN8. The *PreSNEncryCID* of the record of SN8 is the *SNEncryCID* of SN5 and SN7, and the *PreSNEncryCID* of the record of SN3 is the *SNEncryCID* of SN2. The order of *SNID_List* in the record of the model is consistent with that of *SNF_List*. The model owner backed up the *SNEncryCID* from each sub-network record, indicated by red underlines, and the *OriginalSNID_List* from the model record, also indicated by red underlines, is the basis for model tamper-proof checking.



(**a**)

Figure 4. *Cont.*



(**b**)

Figure 4. (**a**) Off-chain storage of the sub-network files of the vehicle detection model using the IPFS. (**b**) Invoking FindByID() to retrieve the records of the vehicle detection model and its sub-networks SN8, SN3 in the blockchain.

Although the model has been stored in both the blockchain and the IPFS, its security is not absolute, and there is a possibility of model tampering due to security vulnerabilities. To ensure the reliability of detection, it is essential to ensure the correctness of the vehicle detection model. This requires a tamper-proof checking method to dynamically and continuously monitor the correctness of the model. For the model tamper-proof checking, we demonstrate the results under two scenarios. The first scenario involves the tampering of model information in the blockchain. Instead of directly tampering with the model information in the blockchain, we uploaded a tampered sub-network record to the blockchain. Based on the design of the sub-network record and model record structures, the $SNID$ of subsequent sub-network records changed as a result, leading to changes in the model record. Therefore, we also uploaded the subsequent sub-network records and the corresponding model record to the blockchain. Specifically, for the sub-network SN8, we removed the encrypted CID of SN5 from its $PreSNEncryCID$. As a result, in Figure 5a, the $PreSNEncryCID$ of SN8's record has changed compared to Figure 4b, causing a change in its $SNID$. The subsequent record after SN8's record, which is the record of SN3, also experienced $SNID$ changes. Ultimately, the model record has changed as well. We utilized the AddModel() method from the smart contract MC to upload the tampered sub-network and model records to the blockchain, and then utilized the FindByID() method from MC to display these records, as indicated by blue underlines. By inputting the $OriginalSNID_List$ and the $ModelID$ of the tampered model record, we conducted tamper-proof checking on the model using the TPCI() method in MC, as indicated by blue underlines. The results are shown in Figure 5b. It can be observed that the $SNID$ of the first tampered sub-network record, which corresponds to the record of SN8 marked with a red underline, has been detected during the checking and indicated as an error, as highlighted by a red underline. The second scenario involves tampering with the information stored in the IPFS. We deleted a file belonging to one of the sub-networks in the IPFS. Specifically, the file of SN5 was deleted. By inputting the original model record's $ModelID$ and the CID encryption key $ModelKey$, as indicated by a blue underline, we conducted tamper-proof checking on the model using the TPCII() method in the smart contract MC. The results are shown in Figure 6. It is evident that the CID associated with SN5 cannot be retrieved, resulting in errors highlighted by red underlines.



(**a**)



(**b**)

Figure 5. (**a**) Invoking FindByID() to retrieve the tampered records of the vehicle detection model and its sub-networks SN8, SN3 in the blockchain. (**b**) Invoking TPCI() to check the record of the tampered vehicle detection model in the blockchain.

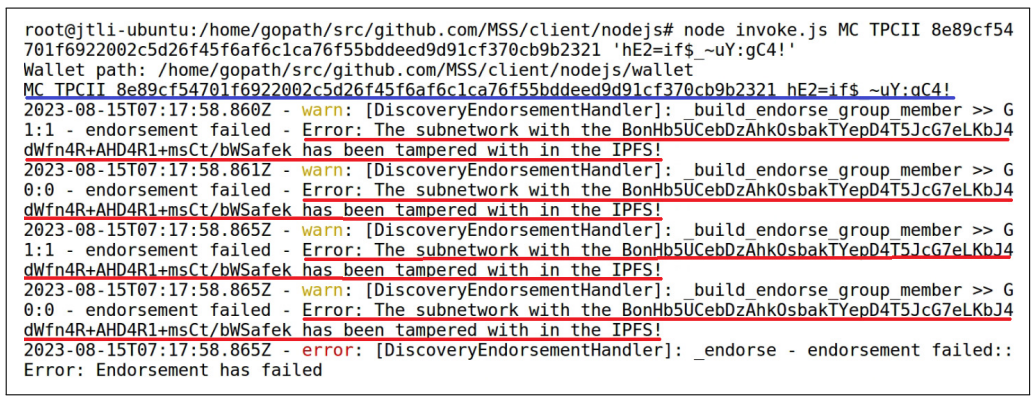


Figure 6. Invoking TPCII() to check the tampered vehicle detection model in the IPFS.

The vehicle detection model service is not open to the public. Considering model security, service providers need to control user access to the model. In the proposed solution, this is achieved through role-and-time-based permission management. After storing the

model in the IPFS and the blockchain, User1 called the AddPermission() method of MC to grant User2 permission to use the model. Subsequently, User2 generated and verified a request to use the model by invoking the CheckPermission() method of the smart contract MC. Figure 7a illustrates the successful addition of a permission record through the QueryPermission() method of MC. During the query, the *RequestID* was used as input, as indicated by the blue underline. The result marked with a red underline confirms the existence of this permission record, indicating that User1 allows User2 to use the model. The time displayed here represents the expiration time of the permission. After successfully granting permission, User1 input the model information into the smart contract for model inference without providing this information to User2. When the permission expires, User2 called the CheckPermission() method again to verify the request to use the model, as indicated by the blue underline in Figure 7b, and the system displays that the permission has expired, as shown by the red underline.

```
root@jtli-ubuntu:/home/gopath/src/github.com/MSS/client/nodejs# node invoke.js MC QueryPermissio
n 8a70dcd7d7ee063204a3c697b297c5795f04324b29c58103c420ce98f76a9a0c
Wallet path: /home/gopath/src/github.com/MSS/client/nodejs/wallet
MC QueryPermission 8a70dcd7d7ee063204a3c697b297c5795f04324b29c58103c420ce98f76a9a0c
Transaction has been submit, result is: {"ModelID":"8e89cf54701f6922002c5d26f45f6af6c1ca76f55bdd
eed9d91cf370cb9b2321","Role":"Visitor","Time":"2023-08-20 17:35:41","UserID":"User2"}
```

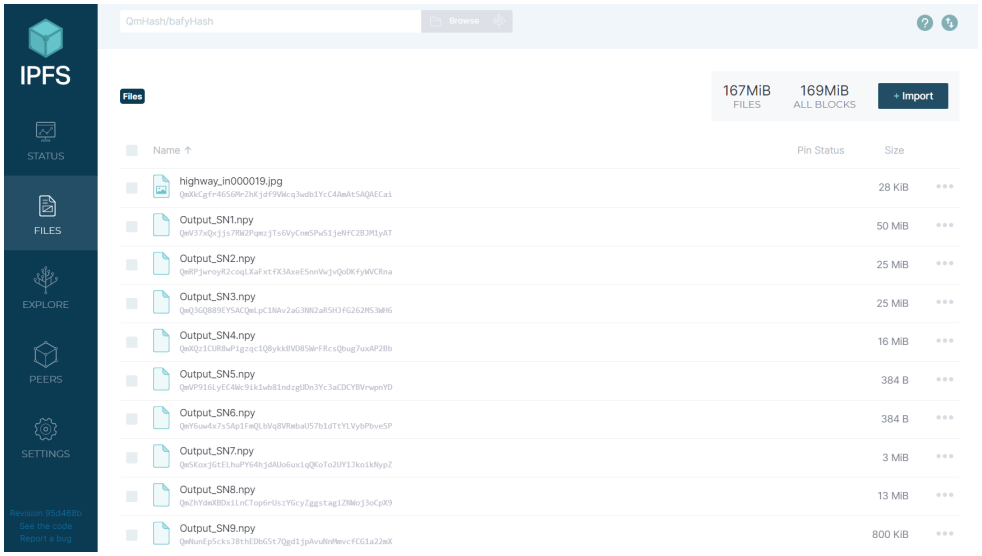
(**a**)

```
root@jtli-ubuntu:/home/gopath/src/github.com/MSS/client/nodejs# node invoke.js MC CheckPermissio
n User2 8e89cf54701f6922002c5d26f45f6af6c1ca76f55bddeed9d91cf370cb9b2321 Visitor
Wallet path: /home/gopath/src/github.com/MSS/client/nodejs/wallet
MC CheckPermission User2 8e89cf54701f6922002c5d26f45f6af6c1ca76f55bddeed9d91cf370cb9b2321 Visito
r
2023-09-05T12:38:35.093Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G
1:1 - endorsement failed - Error: The permission has expired
2023-09-05T12:38:35.094Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G
0:0 - endorsement failed - Error: The permission has expired
2023-09-05T12:38:35.097Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G
1:1 - endorsement failed - Error: The permission has expired
2023-09-05T12:38:35.097Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G
0:0 - endorsement failed - Error: The permission has expired
2023-09-05T12:38:35.098Z - error: [DiscoveryEndorsementHandler]: _endorse - endorsement failed::
Error: Endorsement has failed
```

(**b**)

Figure 7. (**a**) Invoking QueryPermission() to query the added permission record used for granting model usage permission. (**b**) Invoking CheckPermission() to complete permission verification.

The model inference produces the vehicle detection results, and this step also considers the security of both the model and the detection results. Once the permission granting was successful, User1 input the *ModelID* and *ModelKey* into the ModelInference() method of the smart contract MC and User2 input the *ModelInput* and *OutputKey* into ModelInference(). The automatic model inference process began. In our experiment, considering the model security, the inference of sub-networks took place in two off-chain servers. Sub-network SN1 started the inference, followed by SN2 and SN6. Next were SN3 and SN5, then SN4, SN7, SN8, and finally SN9. *ModelInput* and the outputs from each sub-network were sequentially uploaded to the IPFS, see Figure 8a. The *ModelInput* is a frame from the "highway" video named "highway_in000019.jpg". The CIDs of the model input and sub-network inference outputs were encrypted and then uploaded to the blockchain as parts of the record of the model inference result. Figure 8b displays the records of the model outputs in the blockchain by invoking the FindByID() method, as indicated by a blue underline. The *SNOutput_Dict*, marked with a red underline, is a dictionary containing the mapping of encrypted CIDs between each sub-network and its inference result. We have printed them all for reference.



(**a**)

```
root@jtli-ubuntu:/home/gopath/src/github.com/MSS/client/nodejs# node invoke.js MC FindByID 67d06
945cb5e86aa7ea26e9e15b82cf042256a68499e5fea490cee583cecada0
Wallet path: /home/gopath/src/github.com/MSS/client/nodejs/wallet
MC FindByID 67d06945cb5e86aa7ea26e9e15b82cf042256a68499e5fea490cee583cecada0
Transaction has been submit, result is: {"ModelID":"8e89cf54701f6922002c5d26f45f6af6c1ca76f55bdd
eed9d91cf370cb9b2321","ModelInputEncryCID":"/1KQqLmXjtvw1vetCi74SJYbI3Z420KJZxCjlM29cNJVHhPAI7lV
A2aZsuTrJ9oV","ModelOutputID":"67d06945cb5e86aa7ea26e9e15b82cf042256a68499e5fea490cee583cecada0"
,"SNOutput_Dict":"[[4nLBZsUQOGJR0LK4wDxBkGI6+kIRwIMUmluoz7thJEKSqqV/GIQngl1Mlfn+zBy0:xHyERonXUAm
pXOyBZ7uHpnEHMORclZ8eo9s92qs4pb+G/Wo7nvYGNkIWrQ8962Ow],[TGG/uWyLyAtsFjrXXN8//kbECTMsfADagRpOA/o4
bTk8KtbH/xbV6nOE5QPCH5Z9:s9GaBNrvBTLftuSEW+8IES4gC+RrWDpTIlPCIrxHmQkyMM3aXX0qB93iCXsuspCb],[McJp
81Tjdy8XUPaxU6lmF9lw5AijfcUXHuIqMI9z2Ro4gW2gHN2uZ7bnagP1SIyR:XWMcTdLFh+6VZXoYuJTMGe43Ua9HNPaM2xN
OHV5XoE7ss7AjZx3IEVgnNIXX2Uki],[jRdT3c8y0yE076H8UTG50nj180KSZi2vUV1b5VmZI1NZaiake06bqvLpHzWJppt4
:u0aUTuaNtFPVnRLeKg2J4oKImZ83+5fFK6nB59RF6bDAlkF+Ko9pO9Tg+tHDOkdT],[BonHb5UCebDzAhkOsbakTYepD4T5
JcG7eLKbJ4dWfn4R+AHD4R1+msCt/bWSafek:+ZUjqkdRiT8UoIs8AmIPlcU7bQA7by2SdaZ6nOf4jVliwZpjYOSO+GwXsRp
XRXd7],[cgrNm2VfaAWAXtSzIY73YS2JjqlyM2kxPeFamoSeYXeGewW5BDL13ZATPWTsHTJR:dzMjrzDbHAy4cAayqGbfNdh
qjkIHun+jORLFqxXytPhQjSAHh3ahX3IVTQ0pXjIq],[pbSFCvD1MRR5BN4QaRzP8rfMLv7PTWaMGGuk76EqzXtaPqkTyIv5
BQK6cp3RcxUa:0LbBwjEBfAXbgrCiIMWKsiCgbOwiQSPKlHHDERhPpkunIg1TEzL8PLNQn+/qtfgV],[/g048Hi05Ymab7RM
AEHu9G5DpGJCxAAWkoyg8tllWFitjcYDZjzXund4uSyHNu8E:YSYdJ0r/T/RIrExNw/xdKyZ+Ro+hHsWSZ6TnCxQz0j9xrFY
MWKaDNmOl+4VhpKbu],[K0XeYh4DbvNfQcydU1uia6Y8HGS/81POAshZA8WCvRMAOhPmJmwgct4QOauTulPx:x5MRyNjdAYm
zZpF+2bdeZrOyCAkOYyITIP7YRbgQrvn5Xhwg/aUfldeFSrEwbDAF]]","Time":"2023-08-16 13:23:18"}
```

(**b**)

Figure 8. (**a**) Off-chain storage of the input and sub-network outputs of the vehicle detection model using the IPFS. (**b**) Invoking FindByID() to retrieve the record of the inference result of the vehicle detection model when the input is "highway_in000019.jpg".

4.2. Security and Applicability Analysis

In this section, we begin by comparing the proposed solution with three existing solutions [17–19] based on the used technologies and their applicability to deep models, as shown in Table 2. We then proceed to further analyze the security and applicability of the proposed solution in the context of the use case presented in Section 4.1.

Table 2. Comparisons of the proposed solution and some existing solutions on security and applicability.

Perspective	Solution of [17]	DeepRing [18]	Solution of [19]	Proposed Solution
Storage	Blockchain (the model is in the blockchain.)	Blockchain (the model is in the blockchain.)	IPFS + Blockchain (the model is in the IPFS.)	IPFS + Blockchain (the model is in the IPFS and the blockchain.)
Data Encryption	Considered (the type is not specified.)	Asymmetric encryption and symmetric encryption	Symmetric encryption	Symmetric encryption
Auditing and Monitoring	Not considered	Layer-wise asymmetric encryption and decryption	Not considered	Tamper-Proof checking

Table 2. *Cont.*

Perspective	Solution of [17]	DeepRing [18]	Solution of [19]	Proposed Solution
Access Control	Not considered	Not considered	ABAC based on user roles	ABAC based onuser roles and expiration time
Inference	Not considered	Not considered	Centralized inference	Distributed inference
Applicability to Deep Models	All	Part	All	All

In the proposed model storage method, the entire model information is not fully stored in either the IPFS or the blockchain. Instead, the information of the entire model is divided into intra-subnetwork information and inter-subnetwork connection information. The information within a sub-network is stored in the IPFS, while the inter-subnetwork connection information is stored in the blockchain. Compared to the solutions proposed in [17–19] that store the model fully in the blockchain or the IPFS, the proposed model storage method is more decentralized, making it more difficult to steal the complete model information and thereby enhancing security. Moreover the proposed model storage method is not constrained by model architectures, model division methods, or sub-network recording sequences, making it applicable to all deep models. In terms of data encryption, we have chosen a symmetric encryption method to strike a balance between security and system efficiency.

A method for tamper-proof checking on stored models is presented in the proposed solution. This auditing and monitoring method has not been considered in existing solutions [17–19]. This method effectively prevents subsequent errors caused by tampered models, thereby enhancing model security. For example, in the first scenario presented in Section 4.1.2, due to the proposed model inference method invoking sub-network files in the IPFS according to the sub-network records in the blockchain, when the $SNEncryCID$ of sub-network SN5 is deleted from the $PreSNEncryCID$ of SN8's record, issues arise with SN8's input, causing the automated inference fails. In the second scenario presented in Section 4.1.2, although the sub-network records in the blockchain are correct, there is no SN5 file available in the IPFS for invocation, causing SN5 to fail to be inferred, leading to a failure of the model inference. However, it does not mean that automated inference cannot function properly whenever model information is tampered with. Consider the scenario in Figure 3c where Gaussian noise is added to the convolution kernel of SN5—if its file in the IPFS is tampered with, there is a change in its CID. And the changed CID is encrypted and uploaded to the blockchain, thus tampering with the model information in the blockchain as well. In this case, automated inference can proceed normally, but the inference results will be compromised. To rectify this scenario, the TPCI() method can be used to check and restore the tampered parts in the blockchain, followed by using the TPCII() method to check and restore the tampered parts in the IPFS.

In the proposed solution, we enhance model security by utilizing permission management based on user roles and expiration times, a feature not considered in existing solutions [17–19]. While [19] also employs the ABAC for granting permissions to users, this method is solely based on user roles. In contrast, our proposed permission management method combines user roles and expiration times, aligning more closely with the practical requirements of model services for deep learning. Furthermore, unlike [19], in our solution, once usage permissions are granted for a model, the model owner does not provide users with the checked and updated $ModelID$ or $ModelKey$; instead, this information is directly input into the smart contract. As a result, users can only use the model without being able to access or download it, further enhancing model security.

We also introduce a distributively executable model inference method in our proposed solution, which has not been considered in existing solutions [17–19]. The implementation of automated inference involves three possible strategies: off-chain inference, on-chain inference, and hybrid inference. Off-chain inference has lower security, as there is a possibility

of model leakage or tampering when off-chain computational resources are compromised. Nevertheless, unlike [19], due to the highly decentralized nature of our storage method, an individual computing resource are used only for several sub-networks during inference, and the connection information of sub-networks is stored in the blockchain, minimizing the likelihood of complete model information leakage. On-chain inference offers higher security during the inference process, with a lower likelihood of model leakage or tampering. However, this will impose a significant storage and computational burden on the blockchain. For hybrid inference, the security and system efficiency fall between that of off-chain inference and on-chain inference.

4.3. Experimental Results and Analyses on System Efficiency

4.3.1. System Efficiency Analysis Based on the Use Case in Intelligent Transportation

In this part, we analyze the system efficiency of the proposed solution based on the use case presented in Section 4.1. In the experiment conducted in Section 4.1, we collected experimental results on nine aspects reflecting system efficiency, involving required storage space, time consumed for data transmissions, and time consumed for computations. The results are presented in the third column of Table 3. Under the same experimental conditions, we also conduct the same experiment using the solution presented in [19], which does not involve model division and stores the entire model file in the IPFS while recording the model in the blockchain. The results are shown in the last column of Table 3. As analyzed in Section 4.2, it is already known that the proposed solution offers better security compared to the solution proposed in [19].

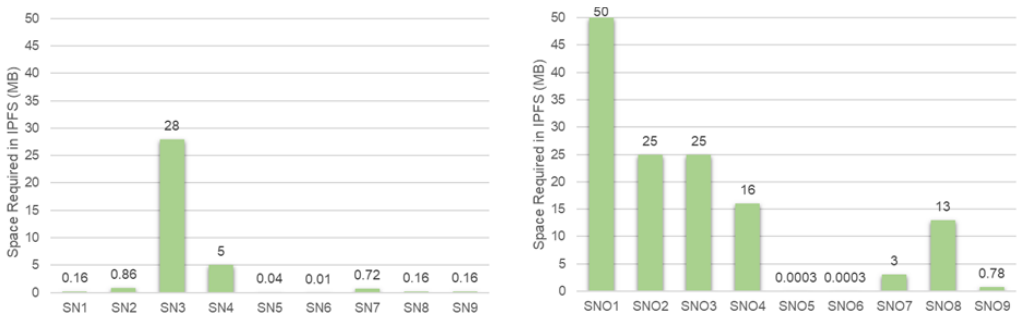
Table 3. Comparisons of the proposed solution and the solution proposed in [19] on some aspects of the system efficiency.

Perspective		**Proposed Solution**	**Solution Proposed in [19]**
Storage	Storage space required in IPFS during model storage (MB)	35	64
	Storage space required in blockchain during model storage (10^{-3} MB)	0.75	0.06
	Storage space required in blockchain during model inference (MB)	132	0.81
	Storage space required in blockchain during model inference (10^{-3} MB)	0.34	0.09
Data Transmission	Time consumed for file uploading to IPFS during model storage (s)	0.77	0.34
	Time consumed for file uploading to IPFS during model inference (s)	1.25	0.13
	Time consumed for file downloading from IPFS during model inference (s)	1.93	0.53
Computation	Time consumed for hashing during model storage (10^{-6} s)	33.86	1.75
	Time consumed for off-chain model inference (s)	6.02	6.76

In terms of storage space, we measured the space required in both the blockchain and the IPFS during the model storage and inference processes. In the proposed solution, during the model storage, nine sub-network files were stored in the IPFS, consuming less space compared to storing the entire model file in the IPFS. However, recording the models in the blockchain requires nine sub-network records and one model record for the proposed solution. In contrast, the solution without model division only requires one model record. It is evident that the proposed solution consumes more on-chain space. The proposed solution actually requires less total storage space in the model storage. During the model inference, the inference results are stored in the IPFS, occupying a significant amount of storage space, its on-chain record includes the mappings between encrypted CIDs of each sub-network and its inference result, which increases the required on-chain storage space. Regarding data transmission, the main time consumption is concentrated on uploading data to the IPFS and downloading data from the IPFS. In the proposed solution, during the model storage, the sub-network files were uploaded to the IPFS. We measured the time it took to upload each sub-network file, as shown in Figure 9c. Comparing Figure 9a,c, it

is evident that the uploading time is related to the file size, but not solely dependent on it; it is also influenced by factors like network speed. The same applies to downloading, as shown in Figure 9e. Although the size of the entire model file is close to two times of the size of the sub-network files, the time it takes to upload the entire model file to the IPFS is only less than half of the time it takes to upload all the sub-network files. The model inference involves the uploading and downloading of the model input and sub-network inference results in the IPFS, as well as the downloading of sub-network files. We also measured the time for uploading and downloading each file, as shown in Figure 9d–f. Because it involves the uploading and downloading of intermediate model output results, the proposed solution consumes significantly more time than the solution that does not divide the model into sub-networks. In terms of computation, we respectively examine the time consumed for on-chain hashing during model storage and the time consumed for off-chain sub-network inference. On-chain hashing during model storage is used to compute identifiers for the sub-network records and the model record. The specific time consumption can be seen in Figure 10a. Because ten hashing calculations are required, the time consumed is significantly higher compared to when the model is not divided into sub-networks, which requires only one hashing. The proposed solution employs off-chain inference for sub-networks. Although off-chain inference has lower security compared to on-chain inference, it is currently a more feasible inference method considering system efficiency. Off-chain inference is conducted on two servers, and we measured the inference time for each sub-network, as shown in Figure 10b. In this process, inference for SN2 and SN6 can be performed simultaneously, and so can SN3 and SN5, resulting in a shorter total time compared to the scenario where the model is not divided and inferred on a single server. It should be noted that because each server is equipped with only one RTX 3090 GPU, the inference time here does not meet the requirements for real-time detection.

From the above experimental results, it can be seen that while the proposed solution has certain advantages in aspects such as the space required in the IPFS during model storage and the time consumed for off-chain inference, it still consumes more in terms of overall space and time compared to the solution without model division. This is the cost of increased security. We found that the number of sub-networks is an important parameter in the proposed solution, and all aspects examined in Table 3 are related to it. In the use case in Section 4.1, dividing the vehicle detection model into nine sub-networks may be not the best choice. As the number of sub-networks increases, the security of the model improves, but the storage and computations required by the system also increase. In the practical use of the proposed solution, a balance between security and system efficiency can be achieved by selecting an appropriate number of sub-networks to maintain the sustainability of model services. This mechanism, which can balance security and system efficiency, is a feature of the proposed solution that existing solutions [17–19] have not considered.



(**a**) (**b**)

Figure 9. *Cont.*

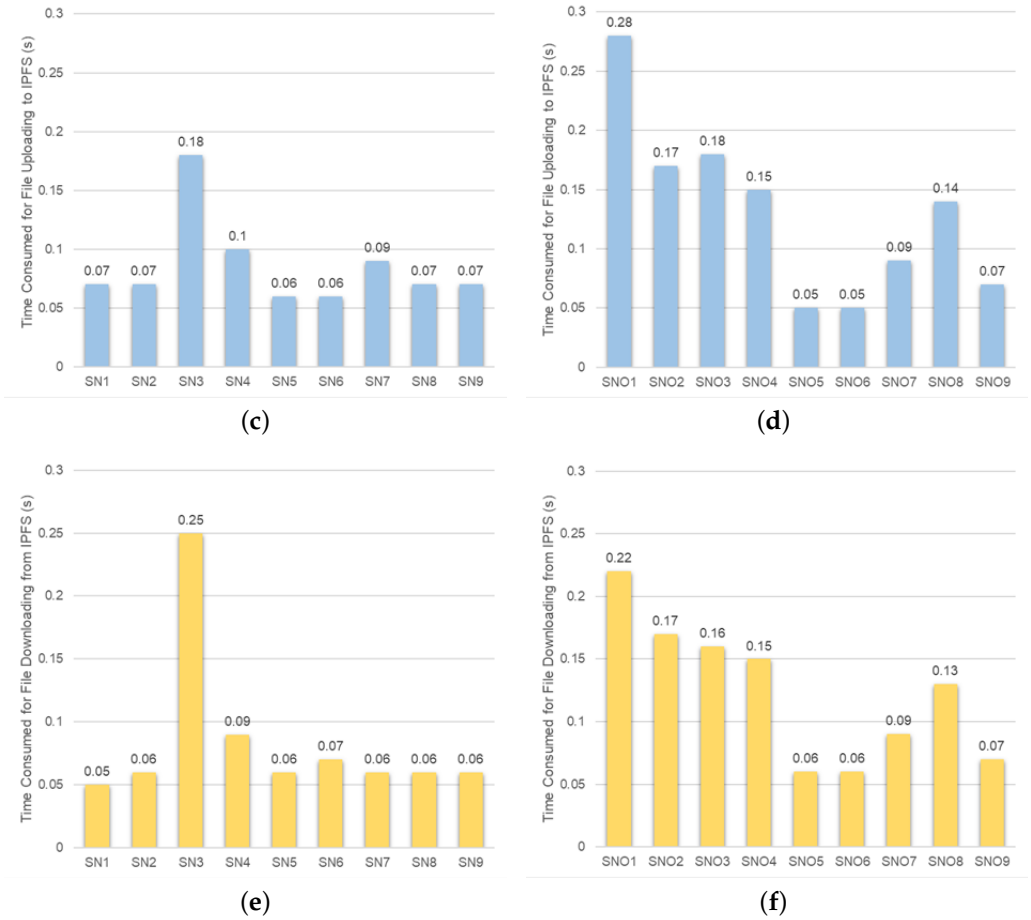


Figure 9. Sizes of sub-network files (**a**) and their inference result files (**b**) in the use case of Section 4.1. (**c**,**d**): Time consumed to upload these files to the IPFS. (**e**,**f**): Time consumed to download these files from the IPFS. (SN: sub-network; SNO: sub-network inference output).

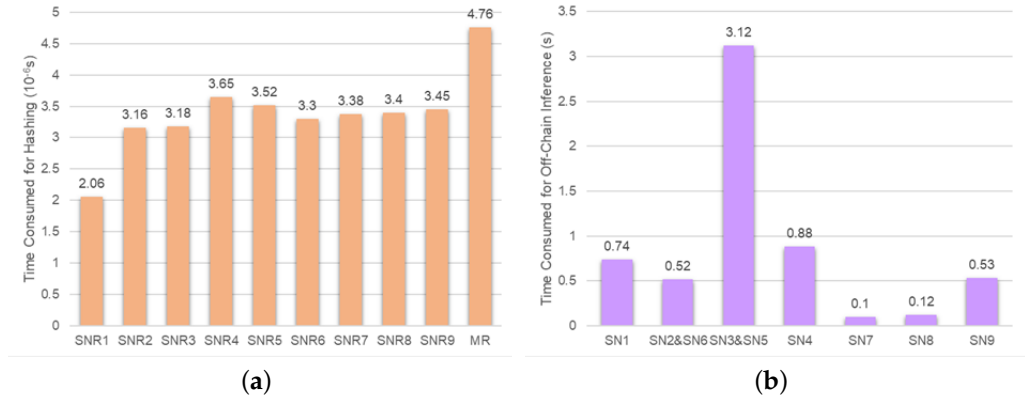
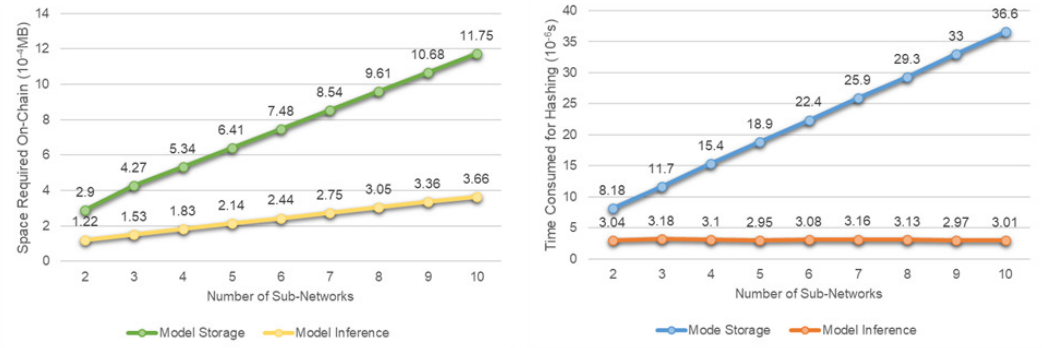


Figure 10. Time consumed for hashing during the model storage (**a**) and time consumed for sub-network off-chain inference (**b**) in the use case of Section 4.1. (SN: sub-network; SNR: sub-network record; MR: model record).

4.3.2. On-Chain Efficiency Analysis Based on A Simulation Experiment

In this part, we analyze the efficiency and sustainability of blockchain from the perspective of on-chain storage and computation. We first conduct a simulation experiment and then proceed with the analysis based on the results of this experiment. To simplify the experiment, we make the assumption that, except for the first sub-network of the model, each subsequent sub-network has only one preceding sub-network. Since the proposed

solution does not impose any restrictions on the order of recording sub-networks, we also assume that the first recorded sub-network is not the first sub-network of the model. Based on these assumptions, the simulation experiment examines the impact of increasing the number of sub-networks on on-chain storage and computation. Specifically, we consider the number of sub-networks to be 2, 3, 4, 5, 6, 7, 8, 9, and 10, and analyze the on-chain space required by the sub-network and model records, as well as the time consumed by hashing for these records. The experimental results are shown in Figure 11.



(**a**) (**b**)

Figure 11. Trend of on-chain storage space required (**a**) and trend of time consumed for on-chain hashing (**b**) for model storage and inference as the number of sub-networks increases in the simulated experiment of Section 4.3.2.

From the results, it can be observed that the on-chain storage space required by the sub-network and model records, as well as the model inference result record, exhibits a trend of basic linear growth with the increase in the number of sub-networks. When the number of sub-networks is 10, these records occupy approximately 1.5 KB of storage space. Therefore, in the proposed solution, the space occupied by model and its inference result records in the blockchain is relatively small, providing room for other on-chain information, such as permission records, ensuring sustainability in terms of storage. During the model storage, hashing is required to compute identifiers for all sub-network records and the model record, as shown in Equations (1) and (2). The time consumed for this operation also exhibits basic linear growth with the increase in the number of sub-networks. During the model inference, hashing is only needed once for recording the inference result. Since the length of the input string remains unchanged, see Equation (3), the time consumed for this operation remains relatively constant. When the number of sub-networks is 10, the total time consumed for on-chain hashing is approximately 40 µs. Thus, in the proposed solution, the computational time involved in on-chain recording of the model and its inference results is not substantial. The complexity of model tamper-proof checking and on-chain computations for model inference is also related to the number of sub-networks. With reasonable control over the number of sub-networks, the computational load within the blockchain remains manageable. Furthermore, the proposed permission management method can restrict model usage based on user roles and expiration times, which also helps control the computational load in the blockchain. Therefore, on-chain computations in the proposed solution exhibit sustainability.

5. Conclusions

This paper proposes a sub-network-based solution integrating blockchain and IPFS for model storage and inference. This solution aims to balance model security and system efficiency while being widely applicable, serving as a foundation for the development of systems that support sustainable model services. The proposed solution utilizes a highly decentralized storage method, dividing a deep model into sub-networks and inter-sub-network components. Sub-networks are stored in the IPFS, while connectivity between

sub-networks, as well as records of sub-networks and models, are stored following a chained structure in the blockchain. To prevent model security issues arising from system vulnerabilities, a two-stage tamper-proof checking method is designed to monitor model security. In addition to enhancing the supervision of model correctness, the proposed solution strengthens control over model usage by employing permission management based on user roles and expiration times. The solution introduces an automated inference method and utilizes the IPFS for storing model inputs and sub-network outputs, alleviating the on-chain storage burden. Trusted off-chain computing resources are used for distributed inference, mitigating the on-chain computational burden. The proposed solution is applicable to all deep models. We applied this solution to vehicle detection in intelligent transportation and analyzed its security, applicability, and system efficiency. Experimental results demonstrate that the proposed solution offers good security and applicability. By choosing an appropriate number of sub-networks and employing the dual-attribute-based model permission management method, model security and system efficiency can be balanced, ensuring the sustainability of on-chain storage and computational resources. Issues for further research on this proposed solution include determining how to select the number and division method of sub-networks for a given deep model, as well as whether there are quantitative criteria for the selection. We will address these issues in future work.

Author Contributions: Conceptualization, R.J. and J.L.; Data curation, R.J. and J.L.; Formal analysis, R.J., J.L. and W.B.; Funding acquisition, R.J.; Investigation, R.J. and J.L.; Methodology, R.J., J.L. and W.B.; Project administration, R.J.; Resources, J.L. and W.B.; Software, J.L., W.B. and C.C.; Supervision, R.J.; Validation, J.L. and W.B.; Visualization, R.J., J.L. and W.B.; Writing—original draft, R.J., J.L. and W.B.; Writing—review & editing, R.J., J.L. and W.B. 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 (grant number 61702323), the National Training Program of Innovation and Entrepreneurship for Undergraduates (grant number G20220602), and the Shanghai Maritime University's Top Innovative Talent Training Program for Graduate Students in 2022 (grant number 2022YBR005).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: http://changedetection.net/ (accessed on 10 August 2023).

Acknowledgments: The authors are grateful to the reviewers and editors for their valuable suggestions improved the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Wu, Y.; Liu, L.; Pu, C.; Cao, W.; Sahin, S.; Wei, W.; Zhang, Q. A comparative measurement study of deep learning as a service framework. *IEEE Trans. Serv. Comput.* **2019**, *15*, 551–566.
2. Cui, L.; Chen, Z.; Yang, S.; Chen, R.; Ming, Z. A secure and decentralized DLaaS platform for edge resource scheduling against adversarial attacks. *IEEE Trans. Comput.* **2021**. [CrossRef]
3. Xu, G.; Li, H.; Ren, H.; Yang, K.; Deng, R.H. Data security issues in deep learning: Attacks, countermeasures, and opportunities. *IEEE Commun. Mag.* **2019**, *57*, 116–122.
4. Liu, Y.; Ma, X.; Bailey, J.; Lu, F. Reflection backdoor: A natural backdoor attack on deep neural networks. In Proceedings of the 16th European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; pp. 182–199.
5. Qiu, S.; Liu, Q.; Zhou, S.; Wu, C. Review of artificial intelligence adversarial attack and defense technologies. *Appl. Sci.* **2019**, *9*, 909.
6. Kesarwani, M.; Mukhoty, B.; Arya, V.; Mehta, S. Model extraction warning in mlaas paradigm. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 371–380.
7. Hu, X.; Liang, L.; Chen, X.; Deng, L.; Ji, Y.; Ding, Y.; Du, Z.; Guo, Q.; Sherwood, T.; Xie, Y. A systematic view of model leakage risks in deep neural network systems. *IEEE Trans. Comput.* **2022**, *71*, 3254–3267.

8. Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2805–2824.
9. Aldoseri, A.; Al-Khalifa, K.N.; Hamouda, A.M. Re-thinking data strategy and integration for artificial intelligence: Concepts, opportunities, and challenges. *Appl. Sci.* **2023**, *13*, 7082.
10. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decent. Bus. Rev.* **2008**, 21260.
11. Simmons, G.J. Symmetric and asymmetric encryption. *ACM Comput. Surv. (CSUR)* **1979**, *11*, 305–330.
12. Han, D.; Pan, N.; Li, K.C. A traceable and revocable ciphertext-policy attribute-based encryption scheme based on privacy protection. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 316–327.
13. Liu, H.; Han, D.; Cui, M.; Li, K.C.; Souri, A.; Shojafar, M. IdenMultiSig: Identity-based decentralized multi-signature in internet of things. *IEEE Trans. Comput. Soc. Syst.* **2023**, *10*, 1711–1721. [CrossRef]
14. Han, D.; Zhu, Y.; Li, D.; Liang, W.; Souri, A.; Li, K.C. A blockchain-based auditable access control system for private data in service-centric IoT environments. *IEEE Trans. Ind. Inform.* **2021**, *18*, 3530–3540. [CrossRef]
15. Sandhu, R.S.; Samarati, P. Access control: Principle and practice. *IEEE Commun. Mag.* **1994**, *32*, 40–48. [CrossRef]
16. Benet, J. Ipfs-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
17. Wang, T.; Du, M.; Wu, X.; He, T. An analytical framework for trusted machine learning and computer vision running with blockchain. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Seattle, WA, USA, 14–19 June 2020; pp. 6–7.
18. Goel, A.; Agarwal, A.; Vatsa, M.; Singh, R.; Ratha, N. DeepRing: Protecting deep neural network with blockchain. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA, USA, 16–17 June 2019; pp. 2821–2828.
19. Jiang, R.; Li, J.; Bu, W.; Shen, X. A blockchain-based trustworthy model evaluation framework for deep learning and its application in moving object segmentation. *Sensors* **2023**, *23*, 6492. [CrossRef]
20. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324.
21. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
22. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
24. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
25. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
26. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
27. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
28. Li, H.; Han, D.; Tang, M. A privacy-preserving storage scheme for logistics data with assistance of blockchain. *IEEE Internet Things J.* **2021**, *9*, 4704–4720. [CrossRef]
29. Coppersmith, D. The data encryption standard (des) and its strength against attacks. *IBM J. Res. Dev.* **1994**, *38*, 243–250. [CrossRef]
30. Rijmen, V.; Daemen, J. Advanced encryption standard. *Proc. Fed. Inf. Process. Stand. Publ. Natl. Inst. Stand. Technol.* **2001**, *19*, 22.
31. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
32. Koblitz, N.; Menezes, A.; Vanstone, S. The state of elliptic curve cryptography. *Des. Codes Cryptogr.* **2000**, *19*, 173–193. [CrossRef]
33. Li, D.; Han, D.; Zheng, Z.; Weng, T.H.; Li, H.; Liu, H.; Castiglione, A.; Li, K.C. MOOCsChain: A blockchain-based secure storage and sharing scheme for MOOCs learning. *Comput. Stand. Interfaces* **2022**, *81*, 103597. [CrossRef]
34. Li, J.; Han, D.; Wu, Z.; Wang, J.; Li, K.C.; Castiglione, A. A novel system for medical equipment supply chain traceability based on alliance chain and attribute and role access control. *Future Gener. Comput. Syst.* **2023**, *142*, 195–211. [CrossRef]
35. Gao, N.; Han, D.; Weng, T.H.; Xia, B.; Li, D.; Castiglione, A.; Li, K.C. Modeling and analysis of port supply chain system based on Fabric blockchain. *Comput. Ind. Eng.* **2022**, *172*, 108527. [CrossRef]
36. Liu, H.; Han, D.; Li, D. Fabric-IoT: A blockchain-based access control system in IoT. *IEEE Access* **2020**, *8*, 18207–18218. [CrossRef]
37. Hu, V.C.; Kuhn, D.R.; Ferraiolo, D.F.; Voas, J. Attribute-based access control. *Computer* **2015**, *48*, 85–88. [CrossRef]
38. Li, D.; Han, D.; Weng, T.H.; Zheng, Z.; Li, H.; Liu, H.; Castiglione, A.; Li, K.C. Blockchain for federated learning toward secure distributed machine learning systems: A systemic survey. *Soft Comput.* **2022**, *26*, 4423–4440. [CrossRef] [PubMed]
39. Ye, C.; Li, G.; Cai, H.; Gu, Y.; Fukuda, A. Analysis of security in blockchain: Case study in 51%-attack detecting. In Proceedings of the 2018 5th International Conference on Dependable Systems and Their Applications (DSA), Dalian, China, 22–23 September 2018; pp. 15–24.

40. Kumar, R.; Tripathi, R.; Marchang, N.; Srivastava, G.; Gadekallu, T.R.; Xiong, N.N. A secured distributed detection system based on IPFS and blockchain for industrial image and video data security. *J. Parallel Distrib. Comput.* **2021**, *152*, 128–143. [CrossRef]
41. Sun, J.; Yao, X.; Wang, S.; Wu, Y. Blockchain-based secure storage and access scheme for electronic medical records in IPFS. *IEEE Access* **2020**, *8*, 59389–59401. [CrossRef]
42. Sangeeta, N.; Nam, S.Y. Blockchain and interplanetary file system (IPFS)-based data storage system for vehicular networks with keyword search capability. *Electronics* **2023**, *12*, 1545. [CrossRef]
43. Nath, P.; Mushahary, J.R.; Roy, U.; Brahma, M.; Singh, P.K. AI and blockchain-based source code vulnerability detection and prevention system for multiparty software development. *Comput. Electr. Eng.* **2023**, *106*, 108607. [CrossRef]
44. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
45. Penard, W.; van Werkhoven, T. On the secure hash algorithm family. In *Cryptography in Context*; Wiley: Hoboken, NJ, USA, 2008; pp. 1–18.
46. Wang, Z.; Zhan, J.; Duan, C.; Guan, X.; Lu, P.; Yang, K. A review of vehicle detection techniques for intelligent vehicles. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 3811–3831. [CrossRef]
47. Jiang, R.; Zhu, R.; Su, H.; Li, Y.; Xie, Y.; Zou, W. Deep learning-based moving object segmentation: Recent progress and research prospects. *Mach. Intell. Res.* **2023**, *20*, 335–369. [CrossRef]
48. Jiang, R.; Zhu, R.; Cai, X.; Su, H. Foreground segmentation network with enhanced attention. *J. Shanghai Jiaotong Univ. (Sci.)* **2023**, *28*, 360–369. [CrossRef]
49. Lim, L.A.; Keles, H.Y. Learning multi-scale features for foreground segmentation. *Pattern Anal. Appl.* **2020**, *23*, 1369–1380. [CrossRef]
50. Wang, Y.; Jodoin, P.M.; Porikli, F.; Konrad, J.; Benezeth, Y.; Ishwar, P. CDnet 2014: An expanded change detection benchmark dataset. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 387–394.

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