Verifying Outsourced Computation in an Edge Computing Marketplace

Christopher Harth-Kitzerow and Gonzalo Munilla Garrido

Department of Informatics, Technical University of Munich, Garching, Germany

Abstract. An edge computing marketplace could enable IoT devices (Outsourcers) to outsource computation to any participating node (Contractors) in their proximity. In return, these nodes receive a reward for providing computation resources. In this work, we propose a scheme that verifies the integrity of arbitrary deterministic functions in the presence of both dishonest Outsourcers and Contractors who try to maximize their expected payoff. We compile a comprehensive set of threats for this adversary model and show that not all of these threats are addressed when combining verification techniques of related work. Our verification scheme fills the gap by detecting or preventing each identified threat. We tested our verification scheme with state-of-the-art pre-trained Convolutional Neural Network models designed for object detection. On all devices, our verification scheme causes less than 1ms computational overhead and a negligible network bandwidth overhead of at most 84 bytes per frame. Our implementation can also perform our verification scheme's tasks parallel to the object detection to eliminate any latency overhead.

Keywords: Edge Computing, Internet of Things, Function Verification, Computing Marketplaces

1 Introduction

Offloading computational tasks from IoT devices to computation resources at the network edge can improve the responsiveness of existing applications and enable novel latency-sensitive use cases [1]. In an edge computing marketplace, we assume that the Outsourcer is a computationally weak IoT device that outsources real-time data to a Contractor to process. The Contractor can be an edge server or any device in proximity to the Outsourcer with enough unutilized computational resources to execute the assigned function reliably and with low latency.

Compared to fixed client-server assignments, an edge computing marketplace may overcome the challenges of limited availability of servers, insufficient quality of service, and idle server resources. Dynamic assignments in an open marketplace could increase availability and competition among edge servers. This allows IoT applications to profit from increased connectivity and responsiveness due to better matching. Edge servers, on the other hand, profit from higher resource utilization due to increased matching rates.

As IoT devices are often computationally weak, it might be difficult for them to verify whether responses returned by a third-party Contractor are valid. In fact, Contractors have an incentive to return a computationally less expensive probabilistic result to save resources while still collecting the reward. Likewise, an Outsourcer has no incentive to pay an honest Contractor right after receiving all computational results, and expensive micro-transactions prohibit real-time payment.

In this work, we propose a verification scheme for computation marketplaces that can verify the integrity of arbitrary deterministic functions. The Outsourcer verifies the integrity of returned responses by sending some inputs to another Contractor in proximity called the Verifier. We refer to this approach as sampling-based re-execution. We evaluate our scheme's performance with outsourced object detection based on a real-time image stream sent by an IoT device to an edge server. We provide the following contributions:

1. We compile a comprehensive list of potential threats that a computing marketplace might be vulnerable to in the presence of dishonest participants.
2. We combine existing verification techniques proposed by related work and introduce two novel ones to address all identified threats.
3. Our resulting verification scheme requires little interaction with a trusted third party (TTP) and is resistant to dishonest Outsourcers, Contractors, and Verifiers. The TTP does not have to be located at the edge and can act with arbitrary latency.
4. Our implementation demonstrates that our verification scheme causes negligible communication and latency overhead.

2 Related Work

In previous work, authors have identified different components that an edge computing marketplace should provide. These components include a matching and price-finding algorithm [2], a payment scheme [3] [4], in some cases privacy preservation [5] [6] [7], and a verification scheme [8] [9]. We focus on designing a verification scheme in this work and assume that the other components are present.

Compared to schemes based on cryptographic techniques such as Secure Multiparty Computation, or Fully Homomorphic Encryption, re-execution adds only a negligible computational and network overhead to the computation. Re-execution can be implemented in several ways. In [10], the authors propose outsourcing computation to multiple Contractors in a multi-round approach. The Outsourcer sends different inputs to each Contractor in every round. A trusted master node compares results if the same input is sent to more than one Contractor. The disadvantage of this scheme is that it requires many available Contractors in proximity to the Outsourcer.

In [11], the authors propose a scheme based on sampling-based re-execution. Within specified intervals, the Contractor commits to signed Merkle root hashes based on the responses it sent to the Outsourcer. The Outsourcer can instruct a third-party Verifier to verify these results by checking if the signed Merkle root's signature is correct and if random samples were computed and returned correctly. The scheme assumes that the Verifier and the Outsourcer are honest.

In [12], the authors propose a scheme based on complete re-execution. The Outsourcer sends inputs to $n = 2$ Contractors and accepts the result if they are identical. If responses do not match, the job gets outsourced again to $n_{new} = n^2$ Contractors until no conflicts arise. This approach is similar yet less efficient than our Contestation protocol introduced in section 3. As their scheme does not distinguish between Contractors and Verifiers, the resulting overhead is higher. Also, their scheme relies on a trusted time-stamping server that monitors communication between edge devices. This TTP can end up being the bottleneck of the ecosystem.

In [13], the authors propose a scheme based on sampling-based re-execution with third-party Verifiers. They utilize smart contracts running on Ethereum to set incentives for Outsourcers and Contractors. The incentives discourage dishonest behavior. Verifiers are assumed to be partially trusted.

In [14], the authors propose a scheme based on sampling-based re-execution. The Contractor commits to a Merkle Tree root hash every few intervals and sends it to the Outsourcer. The Outsourcer then randomly selects a few samples to re-compute them and sends a proof of membership challenge to the Contractor. It aborts the contract if the verified output does not match the Contractor's response or if the proof of membership challenge is unsuccessful. The Outsourcer is assumed to be fully trusted.

3 Design of our Verification Scheme

As our scheme uses re-execution as a verification approach, it can be applied to any deterministic function. We assume the following setting for outsourced computation in an edge computing marketplace.

1. Edge servers are stationary (reappearing actors) and offer outsourced computation for a fee. They can either act as Contractors or Verifiers.
2. Outsourcers are mobile (reappearing and adhoc actors).
3. Outsourcers owe a reward to Contractors and Verifiers for each processed input.
4. Each edge participant may act dishonestly but tries to maximize its expected payoff.
5. A TTP or Blockchain is present that provides a public key infrastructure, a reputation system, and handles payments. We refer to this party as the payment settlement entity (PSE). The PSE does not have to be located at the edge.

Before outsourcing of computation starts, we assume an Outsourcer and a Contractor have agreed to a contract. The contract contains a unique ID, the participants' public keys, a reward per processed input, and the function/model to be used is specified. Additionally, the Contractor and Outsourcer may agree on fines, deposits, and bounties if a participant is caught cheating to increase the protocol's robustness. We assume multiple Verifiers are available and willing to agree on a contract with the Outsourcer to process random sample inputs.

3.1 Preparation Phase

The preparation phase is responsible for assigning a Verifier to a contract while preventing collusion. We refer to planned collusion if two participants know each other beforehand and try to collude. We refer to ad-hoc collusion if two participants do not initially know each other but still try to communicate and collude.

Randomization Randomization ensures that the Outsourcer and the Contractor commit to a random Verifier. Additionally, the Verifier and the Contractor do not learn each other's identities. The protocol consists of the following steps: The Outsourcer signs the hash $h(x)$ of a large random number x and the contract hash ch. It sends $h(x)$ with a signature to the Contractor. The Contractor signs the received hash of the Outsourcer along with a large random number y, the contract hash, and a list of available Verifiers sorted by their public keys. Along with this signed hash, the Contractor sends the value y and the list of available Verifiers to the Outsourcer. By signing the initially sent hash of the Outsourcer, the Contractor commits to x and y without knowing x. Figure 1 illustrates this protocol.

If the list of available Verifiers matches Outsourcer's local list, it contacts the Verifier at $(x+y) \mod n$, where n is the total number of Verifiers. If the Outsourcer contacts a different Verifer, it will not be able to present the necessary Contractor signatures during Contestation. Thus, Randomization prevents planned collusion.

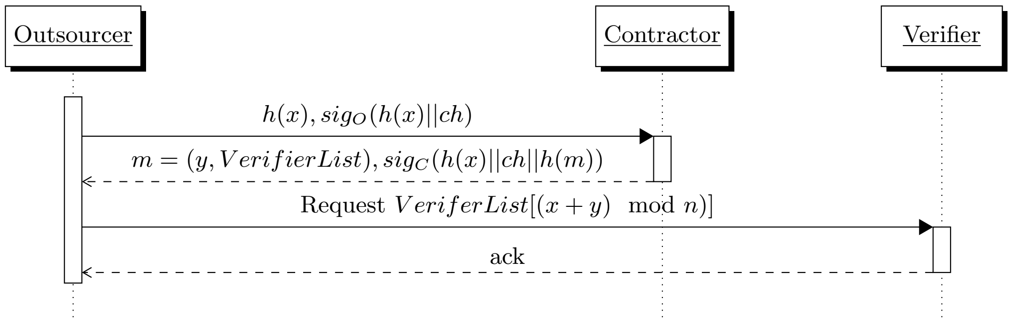
Outsourcer
Contractor
Verifier
$h(x), sig_O(h(x)||ch)$
$m = (y, VerifierList), sig_C(h(x)||ch||h(m))$
Request $VeriferList[(x+y) \mod n)]$
ack

Fig. 1: Randomization

Table 1: Payoff matrix with honesty-promoting incentives

Contractor \ Verifier	Diligent	Dishonest
Diligent	$r - c_h$	$r - c_h + b$
Dishonest	$rq - (f + b)(1 - q) - c_d$	$r - c_d$

Game-theoretic Incentives Even if the Verifier and the Contractor do not know each other, there is a risk of ad-hoc collusion. Suppose there exists a q-algorithm that is computationally inexpensive but provides a correct result with a certain probability q. For object detection, this might be the naive response that no object is detected and, therefore, no bounding boxes at specific coordinates have to be estimated. There is a reward r for returning a valid result and computation costs when computing the desired function honestly c_h or dishonestly c_d. From a game-theoretic perspective [15], there are two nash equilibria [16]. One nash equilibrium exists when both players act honestly, but the other when both players act dishonestly [17] [12].

It is crucial to design incentives that eliminate the Nash equilibrium of both players acting dishonestly. In [12] and [17], the authors have identified a relationship of incentives by adding fees for cheating players and bounties for dishonest players such that being honest is a dominant strategy from a game-theoretic point of view. Table 1 illustrates the payoff matrix of the Contractor with the use of a bounty b and a fee f. The payoff matrix for the Verifier looks identical. Our scheme uses an initial deposit to enforce that a cheating participant pays the fine after detection.

3.2 Execution Phase

After agreeing on a contract with the random Verifier, the Outsourcer starts sending inputs to the Contractor and the Verifier to process.

Sampling Sampling refers to picking one random input out of a collection of inputs. In our verification scheme, the Outsourcer sends samples to the Verifier to check whether its response matches the Contractor's response belonging to the same input. We call this process sampling-based re-execution. Sampling-based re-execution has a significant advantage over complete re-execution. Just with a few samples, a dishonest Contractor with a cheating rate c can be detected with nearly 100% confidence. Thus, we can significantly improve the efficiency of the verification process at a negligible security drawdown.

During sampling, the Outsourcer chooses an interval length l and sends only one random sample per interval to the Verifier. The chance p of detecting a cheating attempt within n intervals is $p = 1 - (1 - c)^n$. Even if the Contractor has a low cheating rate, e.g., $c = 0.1$, the Outsourcer needs less than $n = 50$ intervals to detect cheating with 99% confidence.

Digital Signatures Since participants communicate in an unmonitored, peer-to-peer fashion in our verification scheme, we need a way to securely record payment promises and dishonest behavior. Otherwise, an Outsourcer could claim never to have received any responses from the other participants. Likewise, the Contractor and Verifier could deny that a dishonest result originated from them and could claim to have processed more responses than they did. The PSE can only solve a dispute and hold entities accountable with tamper-proof records.

Figure 2 shows a high-level overview of sampling in combination with digital signatures. "$i = r$" indicates that the current index i matches the random number r generated in the current interval.

When an Outsourcer sends an input, it always attaches a digital signature signed over the current input index, the contract hash, and the input itself. This ensures that each signature can be traced back to one unique input. Also, the Outsourcer includes a number of currently acknowledged outputs n to the message and signature. Thus, the Contractor and the Verifier receive a signed commitment of redeeming n times the specified reward per response. The unique contract hash ensures that each participant can only redeem payment once per contract.

When the Contractor and the Verifier send a result to the Outsourcer, they attach the associated input index, along with a digital signature forged over the contract hash, input index, input signature, and the input itself. This signature serves as proof of being the originator of a fraudulent message when detected cheating. If the signature verification fails, the participant aborts the contract.

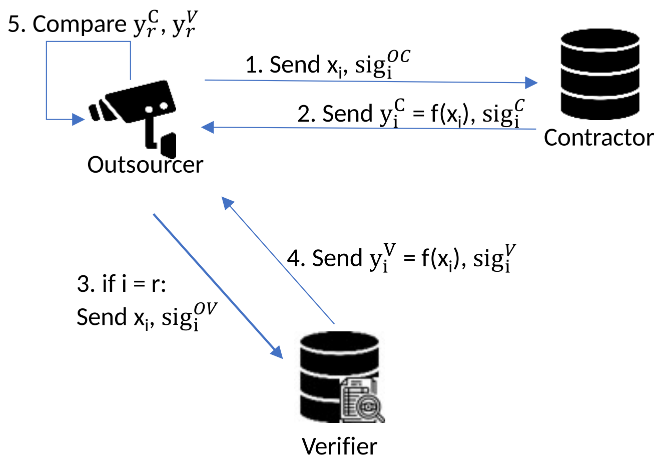
5. Compare y_r^C, y_r^V
1. Send x_i, sig_i^{OC}
2. Send $y_i^C = f(x_i)$, sig_i^C
Contractor
Outsourcer
4. Send $y_i^V = f(x_i)$, sig_i^V
3. if i = r:
Send x_i, sig_i^{OV}
Verifier

Fig. 2: Execution phase

Commitment to Messages Using Merkle Trees Merkle Trees can be used as a data structure to efficiently verify whether data is contained in a large collection. The root hash of a Merkle tree can be used to verify if data is included in the whole tree with $log_2(n)$ steps. Some verification schemes use this attribute of Merkle trees in combination with signatures to commit to large amounts of inputs or outputs by sending the signed Merkle tree root hash [18].

In our scheme, a Contractor utilizing Merkle Trees commits to a collection of responses and receives a proof-of-membership challenge in even intervals. This way, instead of signing all responses, it is sufficient for the Contractor only to sign one Merkle root hash and challenge-response per interval, thus improving efficiency.

3.3 Closing Phase

A participant can terminate a contract at any time. If the contract is terminated according to custom, the Verifier and the Contractor store their last signed input of the Outsourcer, containing the latest number of acknowledged outputs and the signed contract hash. They send the signature and all values to verify it to the PSE. To prevent insufficient quality of service (QoS), a global reputation system and local blocklists per node ensure that participants providing reliable service can be identified. Thus, all participants can submit a review for the other participants at the end of a contract.

The PSE verifies the signatures and deducts the reward per input specified in the contract times the number of acknowledged output contained in the last input on behalf of the Outsourcer after a deadline. Within that deadline, the Outsourcer can report dishonest behavior if it holds two responses that do not match. In this case, the Outsourcer sends the input, both responses, their signatures, and the contracts to the PSE. For scalability reasons, the PSE does not re-execute the computation. It only checks whether all values match their signatures and verifies if responses are indeed unequal. Provisionally, the Contractor is accused of cheating. Within the specified deadline, the Contractor can decide to engage in a protocol we call Contestation to prove that the Verifier's response was incorrect instead.

Contestation Contestation ensures that a falsely accused Contractor or Verifier can prove its innocence. A participant accused of cheating may decide to re-outsource the original input to two additional random Verifiers within a deadline. If both random Verifiers return a response that matches the participant's response, it presents their responses and signatures to the PSE. The participant having the minority of random Verifier support at the end of Contestation is convicted of cheating.

If a Verifier is accused of cheating, it can use the identical protocol to contact two additional random Verifiers and flip the majority of random Verifier support. This protocol might be repeated until no available Verifiers are left. If more than

50% of available Verifiers are non-colluding, Contestation serves as a guarantee that the participant who returned a fraudulent response is found guilty. In combination with a nearly 100% detection rate of cheating using sampling-based re-execution, any cheating parting will be eventually found guilty with a high probability. The participant found guilty at the end of the protocol has to pay the specified fee in its contract, and all additionally consulted Verifiers. In the first round of Contestation, the Outsourcer must prove to the PSE that it contacted the correct Verifier during Randomization by presenting the received Contractor signatures.

Figure 3 illustrates a message sequence chart of Contestation. Notice that the TTP is involved in minimal computation to ensure scalability. It only needs to verify anything if the convicted participant contests conviction. Also, a convicted participant failing to get a majority of Verifier support has no incentive to send the last message and occupy the TTP.

Note that for a dishonest participant, it is irrational to perform Contestation as additional random Verifiers have to be paid for their service. The computational overhead of Contestation is low as only one input has to be recomputed. However, it requires finding multiple available Verifiers in the system. As latency is not critical in this scenario, those random Verifiers do not have to be located at the edge and can be computationally weak devices. We expect that Contestation is usually not performed as its existence alone ensures that a dishonest participant decreases its expected payoff when cheating.

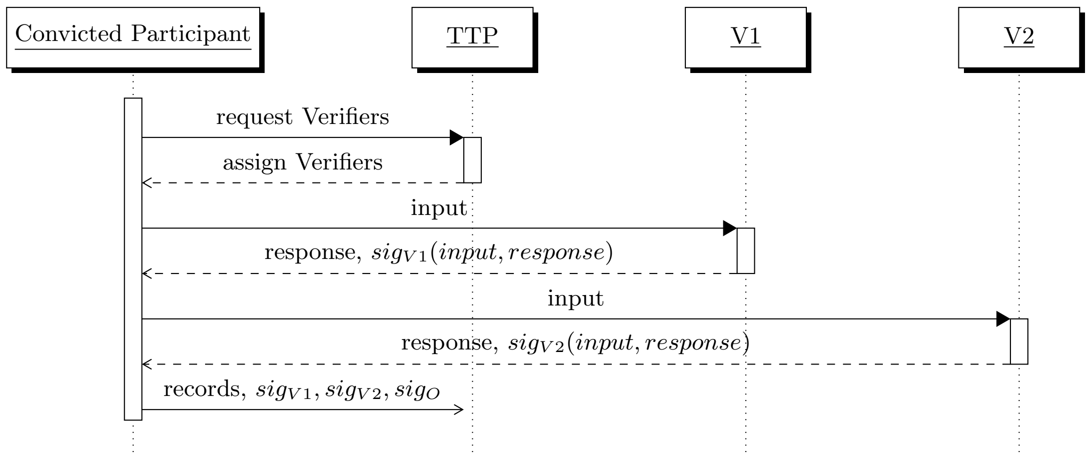
Convicted Participant
TTP
V1
V2
request Verifiers
assign Verifiers
input
response, $sig_{V1}(input, response)$
input
response, $sig_{V2}(input, response)$
records, $sig_{V1}, sig_{V2}, sig_O$

Fig. 3: Contestation

4 Threat Model

In our verification scheme, the Outsourcer, the Contractor, and the Verifier are untrusted and may behave dishonestly. However, we assume that they try to maximize their expected payoff. This type of threat model was first introduced by the authors of [17]. We noticed that existing verification schemes such as [12] assuming payoff-maximizing adversaries address only a subset of possible threats that can result from these assumptions.

Due to the lack of an existing collection of threats in this setting, we compile a comprehensive list of possible threats. We consider internal threats of dishonest behavior by one participant and by collusion. Also, we consider threats of external attackers and quality of service (QoS) violations such as timeouts and low response rates. We assume a distant TTP conducts payments and delegates handling of disputes.

We compiled nine threats in total. These are described in the following paragraphs. Our verification scheme resists all identified threats with high probability. Table 2 summarizes the techniques used by our verification scheme to prevent or detect each possible protocol violation we identified. The confidence column indicates the probability that the protocol violation can be prevented or detected by our techniques. Note that Contestation provides a 100% detection rate of associated protocol violations only if more than 50% of available Verifiers in the ecosystem are non-colluding by not agreeing on an identical incorrect response.

4.1 Contractor sends back false responses to save resources

If a Contractor sends back incorrect responses, the Outsourcer detects this with high probability by sending random samples to the Verifier and comparing if responses belonging to the same input from both participants are equal. In section 3, we show that even with a low number of samples, a cheating Contractor is caught with nearly 100% confidence.

4.2 Verifier sends back false responses to save resources

When the Outsourcer detects two unequal responses from the Verifier and the Contractor, our verification scheme provisionally accuses the Contractor of cheating. However, the Contractor can perform Contestation to prove that the Verifier sent the incorrect response instead.

4.3 Outsourcer sends back different inputs to Contractor and Verifier to refuse payment

The Outsourcer may send two different inputs to the Contractor and the Verifier two receive different responses. It can report these responses to the PSE to refuse payment. This behavior can only be detected if proof is available that the responses resulted from two different inputs. Thus, the Outsourcer must sign each sent raw input with contract-related information. By concatenating the Outsourcer's signature to the Contractor's and Verifier's responses before signing, Contestation can verify if the Outsourcer sent identical raw inputs to both parties.

4.4 Contractor or Verifier tries to avoid global penalties when convicted of cheating or QoS violations

Even when a Verifier or Contractor is detected cheating by an Outsourcer, they may claim never to have sent the reported response. The use of digital signatures prevents this threat.

4.5 Participant refuses to pay even if obliged to by the protocol

Even if the Outsourcer is obliged to reward an honest Contractor or Verifier, there needs to be a way to enforce the payment. Likewise, the Verifier or the Contractor might try to reject paying a penalty fee when detected cheating.

Microtransactions sent for each response are not an option. Usually, the payment scheme can become a latency bottleneck, and each transaction comes with transaction costs. Therefore, payment has to be handled after a contract ends. We assume that the PSE supports deposits and payments on other participants' behalf. It does not need to recompute any values or execute contract-specific functions.

4.6 Outsourcer and Verifier collude to refuse payment and save resources

The Outsourcer and the Verifier may collude to report the Contractor for cheating. Contestation detects this dishonest behavior. If the Verifier is detected cheating by the Contractor through Contestation, it has to pay a fine. If the incentives are set correctly, acting honestly maximizes the expected payoff. Randomization also prevents planned collusion with high probability.

.

4.7 Contractor and Verifier collude to save resources

The Contractor and the Verifier may collude to save computational resources by agreeing on an incorrect response. The Outsourcer checks if both results match and assumes the responses to be correct. Randomization prevents this behavior with a high probability in case of planned collusion.

Additionally, a contract with honesty-promoting incentives maximizes the expected payoff when acting honestly. This measurement makes ad-hoc collusion between Contractor and Verifier unlikely as well. Beyond our verification scheme, an Outsourcer may decide to utilize more than one Verifier or perform additional verification techniques at a lower frequency.

4.8 Timeouts, Low Response Rate, High Response time

In our verification scheme, dishonest Contractors and Verifiers do not receive payments and must pay a fine. In contrast, QoS violations such as timeouts, a low response rate, or a high response time come without monetary consequences. Nevertheless, participating in an ecosystem with insufficient processing or networking capabilities should be discouraged.

Whenever a participant receives a message from another participant that exceeds the QoS thresholds specified in its internal parameters, it may abort the contract. It may also blacklist a participant and submit a negative review. Thus, an unreliable participant misses out on the current contract's ongoing payments and may receive fewer assignments or less payoff from future contracts.

4.9 Message Tampering

An external attacker may attempt to tamper with messages sent between participants to harm a participant. Digital signatures prevent this behavior.

Table 2: Utilized Techniques to Prevent Protocol Violations

Type of Violation	Description	Techniques	Confidence
Dishonest Behavior by Individual	1. Contractor sends back false response to save resources	Sampling-based re-execution, utilization of a third party Verifier	Up to 100%
	2. Verifier sends back false response to save resources	Contestation	100%
	3. Outsourcer sends different input to Contractor and Verifier to refuse payment	Digital Signatures (signature chain), Contestation	100%
	4. Contractor or Verifier tries to avoid global penalties	Digital Signatures	100%
	5. Participant refuses to pay even if obliged to by the protocol	PSE authorized to conduct payment on behalf of another entity	100%
Dishonest Behavior via Collusion	6. Outsourcer and Verifier collude to refuse payment and save resources	Randomization, Game-theoretic incentives, Contestation	100%
	7. Contractor and Verifier collude to save resources	Randomization, Game-theoretic incentives	High confidence
QoS Violation	8. Timeout, Low Response Rate, High Response Time	Blacklisting, Review system, Contract abortion	100%
External Threat	9. Message Tampering	Digital Signatures	100%

4.10 Other Threats

Outsourcers and Contractors can join the ecosystem with multiple identities without hurting the system's security. However, the PSE should issue identity checks of new Verifiers in the ecosystem to prevent Sybil attacks [18]. Otherwise, a participant can increase its probability of matching with colluding participants.

5 Discussion

Out of the different re-execution approaches introduced in section 2, sampling based re-execution is the most practical. We further borrow the following other techniques from related work to improve the outsourcing process.

1. Digitial dignatures to hold verifiable proofs of received messages [9] [19] [11] [18].
2. Merkle Trees to reduce the required number of digital signatures that need to be sent [9] [14] [11] [18].
3. TTPs or Blockchains to resolve payments or record communication [9] [13] [19] [18] [12].
4. A global reputation system to promote honest behavior [10].

Other proposed verification schemes often require a TTP at the network edge or assume that one of the participants is a trusted party (TP). A comparison of the trust assumptions of different schemes is shown in Table 3. In our verification scheme, all participants at the edge may act dishonestly. During outsourcing, they collect publicly verifiable proofs from other participants to later present to a distant trusted PSE. This TTP conducts payments and delegates handling of disputes.

Table 3: Trusted parties required in different schemes

Scheme	TPs involved	TPs involved during execution phase	Assumptions
Ours	1	0	Payment settlement entity is fully trusted
[14]	1	1	Outsourcer is fully trusted
[11]	2	2	Outsourcer and Verifier are fully trusted
[20]	1	1	Outsourcer is fully trusted
[18]	> 3	1	Outsourcer is semi-trusted
[10]	> 2	> 2	Multiple Contractors available for one Contract, Pool of trusted nodes available
[13]	1	1	Verifiers are semi-trusted
[12]	2	1	Trusted timestamp server is available

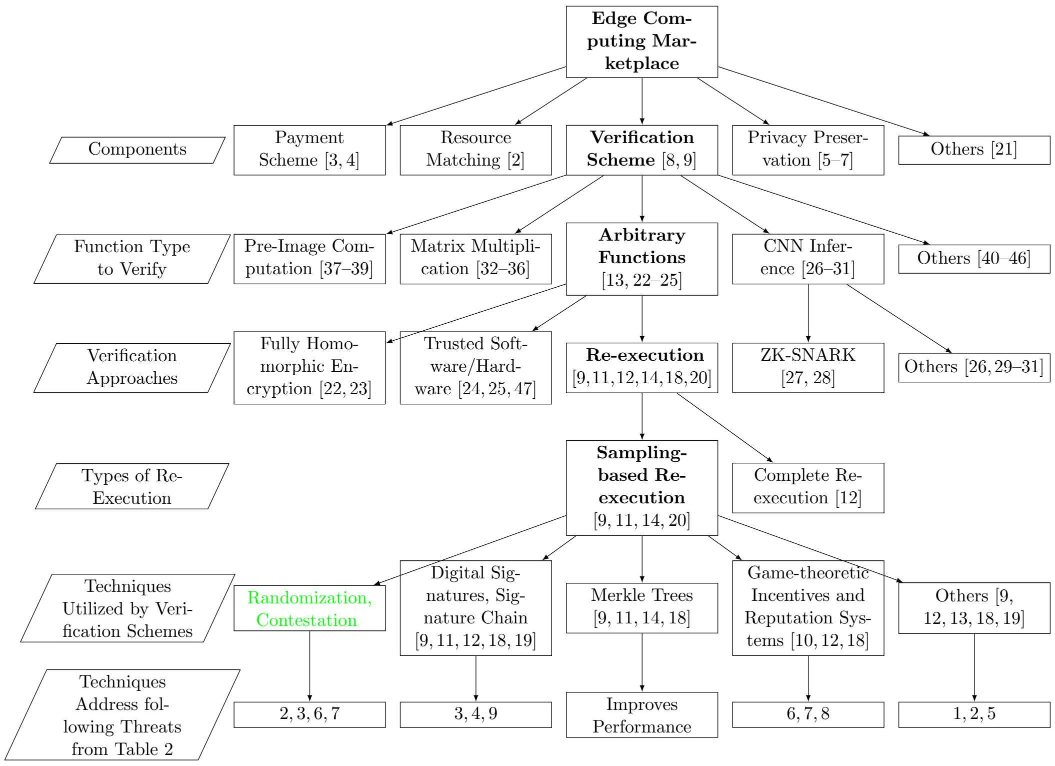
Edge Computing Marketplace
Components
Payment Scheme [3, 4]
Resource Matching [2]
Verification Scheme [8, 9]
Privacy Preservation [5–7]
Others [21]
Function Type to Verify
Pre-Image Computation [37–39]
Matrix Multiplication [32–36]
Arbitrary Functions [13, 22–25]
CNN Inference [26–31]
Others [40–46]
Verification Approaches
Fully Homomorphic Encryption [22, 23]
Trusted Software/Hardware [24, 25, 47]
Re-execution [9,11,12,14,18,20]
ZK-SNARK [27, 28]
Others [26, 29–31]
Types of Re-Execution
Sampling-based Re-execution [9, 11, 14, 20]
Complete Re-execution [12]
Techniques Utilized by Verification Schemes
Randomization, Contestation
Digital Signatures, Signature Chain [9, 11, 12, 18, 19]
Merkle Trees [9, 11, 14, 18]
Game-theoretic Incentives and Reputation Systems [10, 12, 18]
Others [9, 12, 13, 18, 19]
Techniques Address following Threats from Table 2
2, 3, 6, 7
3, 4, 9
Improves Performance
6, 7, 8
1, 2, 5

Fig. 4: Overview: Designed verification scheme

Figure 4 shows an overview of our scheme in the context of our literature analysis. The last two comparisons show different verification techniques utilized by existing verification schemes based on re-execution. Even by combining these existing techniques, we can only solve 7 out of our 9 identified possible threats. We address the remaining two by our Randomization and Contestation protocols (marked green in figure 4):

Threat 6 supposes that the Outsourcer and the Verifier collude to refuse payment to the Contractor. Randomization prevents this behavior with a high probability in case of planned collusion. Contestation prevents this behavior in both cases of adhoc and planned collusion. Threat 3 supposes that the Outsourcer sends different inputs to the Contractor and the Verifier. This behavior is detected by Contestation. We discuss both threats in more detail in section 4.

6 Performance

In our test setup, a Raspberry Pi (Outsourcer) sends a real-time webcam stream to two different machines (Verifier and Contractor) in the local network. The Contractor and the Verifier send back bounding boxes of detected objects in a frame. One test setup uses a regular GPU/CPU for inference, and one uses a Coral USB Accelerator. The Coral USB Accelerator is an entry-level Tensor Processing Unit (TPU) that is specifically designed to perform neural network inference [48]. We use weights that were pre-trained on the Microsoft Coco dataset [49]. We use Yolov4 [50] and MobileNet SSD V2 [51] as models for object detection.

Our implementation uses the NaCl ED25519 signature scheme and can perform the verification scheme's task in parallel to the object detection. This eliminates the latency overhead of our scheme as the verification scheme's task utilizes one CPU thread while the GPU is the bottleneck when performing inference. The source code of our implementation is publicly available here. Our implementation supports multithreading, Merkle Trees, and non-blocking message pattern to improve the efficiency of our scheme. An overview of the test setup is provided in the appendix. Table 4 shows the key results of our test implementation.

Our results show that our verification scheme causes less than 1ms of latency overhead per frame. The Contractor's GPU is the system's bottleneck in our test setup, limiting overall performance to 68.06 fps. As we only used a mid-range GPU and an entry-level edge accelerator in our tests, more potent Contractor hardware could increase overall system performance to match the Outsourcer's performance of more than 200fps. The average 416x416 frame has a size of 120 KB in our tests. The Network bandwidth overhead per participant is negligible at a maximum of 84 bytes per frame. It consists of a 512-bit large signature and, at most, five 32 bit integers such as frame index, acknowledged responses, and other contract-related information. When Merkle Trees are utilized, the Contractor and the Verifier only send signatures when the Outsourcer requests a proof-of-membership challenge.

Table 4: Key Results

Participant	Device	CPU	GPU	Model	$\frac{Frames}{Second}$	Time spent on application (%)	Time spent on verification (%)	Time spent on verification (ms)
Outsourcer	Raspberry Pi Model 4B			MobileNet SSD V2 300×300	236.00	78.70	21.30	0.90
Outsourcer	Raspberry Pi Model 4B			Yolov4 tiny 416×416	146.90	85.10	14.90	1.01
Contractor	Desktop PC	Core i7 3770K	GTX 970	Yolov4 tiny 416×416	68.06	100.00	0.00	0.00
Contractor	Desktop PC	Core i7 3770K	Coral USB Accelerator	MobileNet SSD V2 300×300	63.59	100.00	0.00	0.00
Contractor	Notebook	Core i5 4300U	Coral USB Accelerator	MobileNet SSD V2 300×300	49.30	100.00	0.00	0.00
Verifier	Notebook	Core i5 4300U	Coral USB Accelerator	MobileNet SSD V2 300×300	28.75	-	0.00	0.00

7 Conclusion

In this work, we proposed a scheme for verifying arbitrary outsourced functions in an edge computation marketplace. Our verification scheme is resistant to a comprehensive set of protocol violations that might occur in a computation marketplace with untrusted participants. We benchmarked our verification scheme's performance on consumer hardware and TPUs. Our verification scheme achieves less than 1ms of latency overhead per frame on all tested machines. By utilizing concurrency, the overhead can be reduced to 0 by running the verification scheme's tasks in a parallel thread to the outsourced computation. The network bandwidth overhead of our scheme caused mainly by digital signatures is negligible (at most 84 bytes per frame).

In comparison with verification schemes proposed by the current academic literature, our verification scheme provides additional security by preventing or detecting all threats we identified. At the same time, it only requires third-party involvement outside the network edge. These performance and security characteristics make our scheme ideal for use within an edge computing marketplace that matches computationally weak untrusted IoT devices with untrusted third-party resources to outsource latency-sensitive tasks.

Our verification scheme implements one essential component of a fully functioning edge computing marketplace. As illustrated in figure 4, the remaining components to make an edge computing marketplace viable are: Payment, Matching and price-finding, and Privacy preservation. Future work may optimize and aggregate all components to build an end-to-end system serving as a standalone edge computing marketplace for arbitrary functions.

References

1. W. Tang, X. Zhao, W. Rafique, L. Qi, W. Dou, and Q. Ni, "An offloading method using decentralized p2p-enabled mobile edge servers in edge computing," *Journal of Systems Architecture*, vol. 94, pp. 1–13, 2019.
2. A. Zavodovski, S. Bayhan, N. Mohan, P. Zhou, W. Wong, and J. Kangasharju, "Decloud: Truthful decentralized double auction for edge clouds," 05 2019.
3. R. Rahmani, Y. Li, and T. Kanter, "A scalable distriubuted ledger for internet of things based on edge computing," in *Seventh International Conference on Advances in Computing, Communication and Information Technology-CCIT 2018, Rome, Italy, 27-28 October, 2018*. Institute of Research Engineers and Doctors (IRED), 2018, pp. 41–45.
4. L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, X. Lin, S. Hu, and M. Du, "Veriml: Enabling integrity assurances and fair payments for machine learning as a service," *arXiv preprint arXiv:1909.06961*, 2019.
5. J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18 209–18 237, 2018.
6. Y. Wang, Z. Tian, S. Su, Y. Sun, and C. Zhu, "Preserving location privacy in mobile edge computing," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 05 2019, pp. 1–6.

7. M. Gheisari, Q.-V. Pham, M. Alazab, X. Zhang, C. Fernández-Campusano, and G. Srivastava, "Eca: An edge computing architecture for privacy-preserving in iot-based smart city," *IEEE Access*, vol. PP, pp. 1–1, 08 2019.
8. W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
9. H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen, "Bitcoin-based fair payments for outsourcing computations of fog devices," *Future Generation Computer Systems*, vol. 78, 12 2016.
10. R. Di Pietro, F. Lombardi, F. Martinelli, and D. Sgandurra, "Anticheetah: Trustworthy computing in an outsourced (cheating) environment," *Future Generation Computer Systems*, vol. 48, pp. 28–38, 2015.
11. L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Information sciences*, vol. 258, pp. 371–386, 2014.
12. A. Küpçü, "Incentivized outsourced computation resistant to malicious contractors," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 633–649, 2015.
13. S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, and A. Dubey, "Mechanisms for outsourcing computation via a decentralized market," *arXiv preprint arXiv:2005.11429*, 2020.
14. W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable grid computing," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004, pp. 4–11.
15. M. J. Osborne *et al.*, *An introduction to game theory.* Oxford university press New York, 2004, vol. 3, no. 3.
16. M. Aghassi and D. Bertsimas, "Robust game theory," *Mathematical Programming*, vol. 107, no. 1-2, pp. 231–273, 2006.
17. M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in *Proceedings of the 3rd international workshop on Economics of networked systems*, 2008, pp. 85–90.
18. M. Nabi, S. Avizheh, M. V. Kumaramangalam, and R. Safavi-Naini, "Game-theoretic analysis of an incentivized verifiable computation system," in *International Conference on Financial Cryptography and Data Security.* Springer, 2019, pp. 50–66.
19. X. Chen, J. Li, and W. Susilo, "Efficient fair conditional payments for outsourcing computations," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 6, pp. 1687–1694, 2012.
20. H. Wang, "Integrity verification of cloud-hosted data analytics computations," in *Proceedings of the 1st International Workshop on Cloud Intelligence*, 2012, pp. 1–4.
21. I. Psaras, "Decentralised edge-computing and iot through distributed trust," 06 2018, pp. 505–507.
22. Y. C. Chunming Tang, "Efficient non-interactive verifiable outsourced computation for arbitrary functions," Cryptology ePrint Archive, Report 2014/439, 2014, https://eprint.iacr.org/2014/439.
23. C. Xiang and C. Tang, "New verifiable outsourced computation scheme for an arbitrary function," *International Journal of Grid and Utility Computing*, vol. 7, p. 190, 01 2016.
24. T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.
25. S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "Sgaxe: How sgx fails in practice," 2020.
26. H. Chabanne, J. Keuffer, and R. Molva, "Embedded proofs for verifiable neural networks," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 1038, 2017.
27. S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 584, 2020.
28. J. Groth, "On the size of pairing-based non-interactive arguments," 05 2016, pp. 305–326.
29. X. Chen, J. Ji, L. Yu, C. Luo, and P. Li, "Securenets: Secure inference of deep neural networks on an untrusted cloud," in *ACML*, 2018.

30. Z. Ghodsi, T. Gu, and S. Garg, “Safetynets: Verifiable execution of deep neural networks on an untrusted cloud,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4675–4684.
31. A. A. Badawi, J. Chao, J. Lin, C. F. Mun, J. J. Sim, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, “Towards the alexnet moment for homomorphic encryption: Hcnn, thefirst homomorphic cnn on encrypted data with gpus,” 2018.
32. R. Freivalds, “Probabilistic machines can use less running time.” in *IFIP congress*, vol. 839, 1977, p. 842.
33. X. Lei, X. Liao, T. Huang, and F. H. Rabevohitra, “Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud,” *Inf. Sci.*, vol. 280, pp. 205–217, 2014.
34. Z. Cao and L. Liu, “A note on ”achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud”,” 03 2016.
35. D. Benjamin and M. J. Atallah, “Private and cheating-free outsourcing of algebraic computations,” in *2008 Sixth Annual Conference on Privacy, Security and Trust*, 2008, pp. 240–245.
36. M. J. Atallah and K. B. Frikken, “Securely outsourcing linear algebra computations,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS ’10. New York, NY, USA: Association for Computing Machinery, 2010, p. 48–59. [Online]. Available: https://doi.org/10.1145/1755688.1755695
37. B. Carbunar and M. Tripunitara, “Fair payments for outsourced computations,” in *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2010, pp. 1–9.
38. B. Carbunar and M. V. Tripunitara, “Payments for outsourced computations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 313–320, 2012.
39. P. Golle and I. Mironov, “Uncheatable distributed computations,” vol. 2020, 04 2001, pp. 425–440.
40. G. Xu, G. T. Amariucai, and Y. Guan, “Delegation of computation with verification outsourcing: Curious verifiers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 717–730, 2017.
41. X. Hu and C. Tang, “Secure outsourced computation of the characteristic polynomial and eigenvalues of matrix,” *Journal of Cloud Computing*, vol. 4, 12 2015.
42. C. Wang, K. Ren, J. Wang, and Q. Wang, “Harnessing the cloud for securely outsourcing large-scale systems of linear equations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1172–1181, 2013.
43. X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, “Outsourcing large matrix inversion computation to a public cloud,” *IEEE TRANSACTIONS ON CLOUD COMPUTING*, vol. 1, pp. 78–87, 07 2013.
44. W. Song, B. Wang, Q. Wang, C. Shi, W. Lou, and Z. Peng, “Publicly verifiable computation of polynomials over outsourced data with multiple sources,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2334–2347, 2017.
45. X. Wang, K.-K. R. Choo, J. Weng, and J. Ma, “Comments on “publicly verifiable computation of polynomials over outsourced data with multiple sources”,” *IEEE Transactions on Information Forensics and Security*, vol. PP, pp. 1–1, 08 2019.
46. J. Meena, S. Tiwari, and M. Vardhan, “Privacy preserving, verifiable and efficient outsourcing algorithm for regression analysis to a malicious cloud,” *Journal of Intelligent & Fuzzy Systems*, vol. 32, pp. 3413–3427, 04 2017.
47. V. Costan and S. Devadas, “Intel sgx explained,” *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.
48. A. Ghosh, S. A. Al Mahmud, T. I. R. Uday, and D. M. Farid, “Assistive technology for visually impaired using tensor flow object detection in raspberry pi and coral usb accelerator,” in *2020 IEEE Region 10 Symposium (TENSYMP)*, 2020, pp. 186–189.

49. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
50. A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.
51. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

8 Appendix

Codebase: `https://github.com/chart21/Verification-of-Outsourced-Object-Detection`

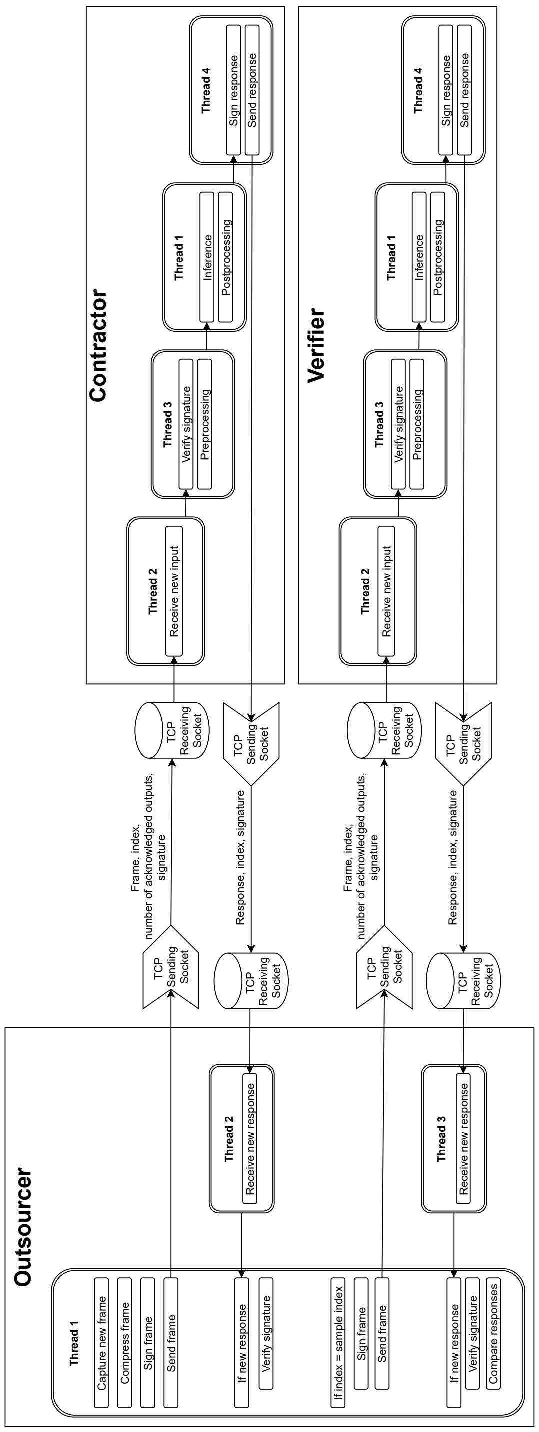

Fig. 5: Test Setup