On Enabling Machine Learning Tasks atop Public Blockchains: a Crowdsourcing Approach

Yuan Lu, Qiang Tang, Guiling Wang
Department of Computer Science, New Jersey Institute of Technology
Email: {yl768, qiang, gwang}@njit.edu

***Abstract*—The recently emerged blockchain (in particular smart contract) technology offers an enticing opportunity for decentralized sharing economy. Machine learning can be one important subroutine in such a decentralized ecosystem. Unfortunately, machine learning programs are usually computational intensive as well as randomized, which fall into the *inherent* limitations of open blockchain where complex and randomized programs cannot be executed by the underlying nodes collectively.**

Given also the limitations of existing verifiable computing techniques, we propose a crowdsourcing idea from the game theoretic perspective to resolve the tension. We design a simple incentive mechanism so that the execution of a wide range of complex programs can be crowdsourced via the blockchain, and any false computing result could be deterred. In particular, our protocol works in the scenarios that there is no trusted third-party involved; Moreover, our protocol not only works in the classical model of non-colluding service providers, but also can tolerate any potential coalition up to $n-1$, where n is the total number of service providers. We also showcase how to use our protocol to crowdsource two typical kinds of machine learning tasks via open blockchain.

We envision that our solution is not only promising to launch decentralized applications involving a wide range of machine learning programs, but also a stepping stone towards a general way to empowering intensive and randomized computations atop the open blockchain.

I. Introduction

The blockchain technology, which was firstly minted by Satoshi Nakamoto in the proposal of a decentralized cryptocurrency called Bitcoin [1], enables a common public ledger to be collectively maintained by permissionless Internet nodes. It was soon realized that as the "transactions" in the ledger could be versatile, the applications of a distributed ledger can go far beyond cryptocurrency. One eminent example is based on the fact that data committed into the distributed ledger can contain the code of some computer programs, the execution of which can be enforced and verified by the network of blockchain nodes, as the same execution environment is shared within the whole blockchain network. Hence, a more exotic application of smart contract [2], which we did not know how to achieve before, is enabled. A smart contract can clearly define and codify the clauses of a "business" contract, so that some predefined actions such as payments/penalties can be automatically enforced when pre-specified conditions are satisfied. The smart contract essentially emulates a reliable party who faithfully facilitates all message deliveries, monetary transfers and computations related to a specified "business" contract (but with no privacy guarantee as it is transparently replicated across the Internet), and therefore becomes very appealing to realize the booming sharing economy without relying on potentially unreliable third-parties [3, 4].

Although the fascinating smart contract technology paves a path to many interesting decentralized applications, the smart contract in (permissionless) blockchain is still at its infant stage, and suffers from several inherent limitations that hinder its wider adoption.

Among several fundamental challenges, one that we are particularly interested in is that currently smart contract can only support very light computation: the complexity of computation tasks of each smart contract is usually strictly bounded. The reason lies in the fact that: during the blockchain mining procedure (the new block generation), when some output of a smart contract is expected to be recorded (that might also affect the validity of future blocks), honest miners are required to execute the program in order to validate the correctness of the outcome. If such a program is computationally intensive, crafty adversarial nodes may simply skip such verification step (or ignore putting the output at all), and go ahead to propose new blocks. Doing this gives the adversarial nodes substantial advantage of winning the chance for proposing new blocks, as honest nodes would not be able to propose any block until the execution of the smart contract finishes. Such an undesirable feature was known as verifier's dilemma [5].

There is another fundamental challenge for smart contracts that they cannot support randomized computations: since randomization lets even honest nodes have inconsistent outputs for the same program with the same input, it obviously prevents the central goal of the blockchain to ensure the consensus among all honest nodes.

Those limitations of smart contracts seriously hinder the applicability to broader interesting scenarios, especially in the settings that involve the execution of machine learning programs. Since those machine learning tasks are often computational costly and randomized (e.g. deep learning using stochastic gradient descent algorithm [6]), it becomes elusive how we can enable the machine learning tasks codified to be auto-executed in decentralized applications. On the other hand, it is a common practice for startup companies to outsource their machine learning tasks [7]. The severe tension between the demands of applications and the restrictions of smart contract is widely acknowledged.

Observing that the security of the consensus protocol restricts only the *on-chain* computations, thus naturally we could

2375-9259/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDMW.2018.00019

turn to the area of verifiable computation for mitigation. In particularly, one may employ advanced cryptographic tools for verifiable computation such as SNARK [3, 8]. The full nodes could simply "outsource" the execution of the smart contracts to some service providers, and ask them further attach a succinct proof to the output. The full nodes now only verify the correctness of the proof during consensus. Unfortunately, although the nice feature of SNARK enables a cheap verification which reduces the on-chain cost (few elliptic curve pairings essentially), the computation and memory cost of generating the SNARK proofs in general for complex programs are still astronomically large, which makes it in fact infeasible in the scenarios of our interests as the machine programs are complex. Another line of researches explored the attestation capability of recently emerged trusted hardwares such as Intel's SGX to carry such outsourcing [9]. The basic idea is to load the program into the protected memory (called trusted enclave), execute the program and sign the output using the secret key hardcoded in the chip. However, the size of the enclave is pretty small, and therefore considerable performance penalties can be brought during attesting memory intensive tasks such as large-scale deep learning and random forest etc. More seriously, a recent attack on SGX could actually extract the attestation key completely [10], making the status of secure hardware is currently unclear.

Our contributions. To enable the feasibility of using smart contracts to crowdsource computation intensive and sometimes randomized programs such as machine learning tasks, with the given limitations on all existing verifiable computation, we take a game theoretic approach. Instead of providing absolute confidence of verifying the correct execution of outsourced "smart contract", we design a simple incentive mechanism (that is enforced by smart contract itself) so that dishonest execution could be deterred. We present a general protocol to allow the chain to outsource heavy and randomized computations to some external computing service providers (called workers). More specifically, our contributions are threefold:

1) Since relying on one worker without verifying the output always leaves chance for misbehavior, we first consider a classical (non-cooperative) game-theoretic setting with two workers that can audit each other. The chain would take results that both workers agree on. Most state-of-the-art solutions along this line assume an opportunistic approach that a trusted party (TTP) comes online to arbitrate, in case an agreement on the result cannot be reached. As opposed, we would get rid of this assumption. Besides properly setting the incentives/payoffs, another subtle point remains due to the *transparency* of the open blockchain: if one worker submits his answer, the other worker can simply send the element without doing any work. This is known as the free-riding problem in crowdsourcing which was also noted in previous works such as [3]. We leverage a cryptographic tool of commitment scheme to tackle this problem. Taking all the issues into account, we design a simple incentive mechanism and prove there is a desirable refinement of Nash equilibria that both workers do the correct computation and return the right answer (c.f. Sec. II for the equilibria refinement). We also consider a class of randomized algorithms that can be considered as programs taking inputs and an extra random coin (which can be supplied by the chain to ensure the consistency for different workers).
   We remark that in our protocol, the expensive and randomized computation is crowdsourced via the application layer to be "asynchronously" executed and separated from the consensus layer. The full nodes/miners can simply put the output into the coming block as the result is submitted, they do not need to wait the execution to finish during mining, and thus the consensus security will not be influenced.
2) We then give two concrete instantiations regarding different type of machine learning tasks, and apply our protocol to enable the "execution" of them over the blockchain. The first example is about checking the quality of a crowdsourced machine learning model, i.e., conditional payment based on the performance on the testing data. The second example is directly for outsourcing machine learning training via the smart contract, which can be an important piece of the social computing puzzle.
   These two interesting instances capture the essence of most potential decentralized crowdsourcing for machine learning tasks. We also note that our solution is actually general to carry a wide range of complex (or randomized) programs as well.
3) Finally, we set force to consider potential coalition(s) in the general setting of more (≥ 3) workers. In particular, we design a protocol that can tolerate any coalition(s), as long as there is no coalition consisting of all workers.

We remark that this work can be seen as a stepping stone towards a general solution to further push boundaries of smart contract in the open blockchain.

Related work. We mainly review related works and briefly discuss their insufficiencies.

Verifiable computation. A verifiable computation [11] allows a prover to produce an output along with a cryptographic proof to convince a verifier that the output is obtained through correctly executing a pre-defined computation. Recent constructive developments of zk-SNARKs [12] enables very efficient verification with the price that proving takes tremendous time and memory. For many heavy tasks, it is infeasible to generate the proof in practice. For example, in [3], we use more than 250 GB memory+swap and 15 hours to prove the majority of 11 RSA-OAEP encrypted answers.

Attestation via trusted hardwares. The recent rise of trusted computation hardware such as Intel SGX [13] brings new techniques to allow people outsource their particular computation tasks [9] to untrusted third-party. However, to "enjoy" such new developments, one has to first trust the manufacturer. Worse still, the design rationale of SGX is a minimalist trusted

machine that has a restricted amount of enclave, so it becomes unclear how to avoid the huge performance penalty during attesting memory intensive programs.

Outsourced computation in game theoretic setting. Outsourced computation have been intensively discussed in game theoretic settings before [14–18]. Most of them rely on a TTP (or an implicit one launched by the requester) to resolve disputes and de-collude coalitions [14, 17, 18]. Some of them consider the absence of TTP under more optimistic scenario where all computing services are non-colluding [15, 16]. We will consider a more stringent setting: n computing services can form any coalition size up to $n-1$, and also there is no TTP to resolve mismatched computing results.

II. Preliminaries

Commitment scheme The cryptographic notion of a commitment scheme realizes a digital "locked box", so the committed content can be hidden, and later be opened uniquely. The commit algorithm and the open algorithm are denoted by $c \leftarrow Commit(m, K)$ and $m \leftarrow Open(c, K)$, respectively. The security requirements for a commitment scheme include: 1) *hiding*: the receiver seeing the commitment c learns nothing about the committed message m; 2) *binding*: the sender cannot cheat the receiver by revealing a different message for the commitment during the open phase. Both above properties should hold with an overwhelming probability.

Game, strategy, equilibrium and practical mechanism A game Γ joined by n players, if described in normal-form, can consist of: 1) a set of players denoted by $\mathbf{W} = \{W_1, \ldots, W_n\}$; 2) a space of players' pure strategies, denoted by $\mathbf{S} = \mathbf{S_1} \times \cdots \times \mathbf{S_n}$, in which $\mathbf{S_i}$ denotes the pure strategies of player W_i, while a strategy σ_i of player W_i is chosen from his strategy space (possibly by a randomized way), and we call $\vec{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$ as a joint strategy of the players; 3) a utility function that defines the utility of each player under different joint strategies.

A Nash equilibrium is a particular joint strategy where no player can realize better utility by unilaterally changing his strategy. Standard Nash equilibrium assumes each player makes decision independently. More generally, we may also consider collusion among part of the players. If a equilibrium can tolerate any coalition size up to k (with $n > k$ players in total), we call it k-resilient Nash equilibrium. We also remark that throughout this paper, we will apply a particular refinement of Nash equilibrium that can be determined by iterated elimination of weakly-dominated strategies, as we always can expect a player would not play a strategy, if there is a better alternative for that strategy. In case that there exists such a refined equilibrium $\vec{\sigma}$ in game Γ, we call $(\Gamma, \vec{\sigma})$ a *practical mechanism*, and $(\Gamma, \vec{\sigma})$ is also called *k-resilient practical mechanism* if the equilibrium $\vec{\sigma}$ can tolerate any deviation by coalition of size up to k.

III. Problem Formulation

In this section, we will present the problem more precisely along with the security requirement, in the game-theory model.

The overview of system. As briefly illustrated in Fig.1, there are three roles in the system. The $blockchain$ network mimics a trusted third-party that is also computationally-restricted. A blockchain user, called $requester$, can request the blockchain to execute some small programs called smart contracts. The chain will internally execute the programs, and "deliver" the computing results to all blockchain users, which is fully trustful. But, the requester cannot expect the blockchain run a computationally-intensive smart contract, as heavy smart contracts are hindered by the intrinsic limit of the blockchain [5]. Alternatively, the blockchain/requester would like to outsource the computations to some external off-chain service providers, called $workers$.

Remark that we will consider the outsourced program has a large output space, and the correct output is unpredictable, namely, for each given input, one cannot guess the output without computing the program with that input. Note that for most machine learning tasks (e.g. the training of DNN classifier for large-scale dataset), such properties can be observed.

Also note that there could be some trivial outputs such as all zeros, all ones, the input or a part of the input etc., we remark that such a set of common knowledge denoted by $\mathbf{E}_{ck}$ (also a subset of the output space) is considered in the paper to capture these publicly known false outputs.[1]

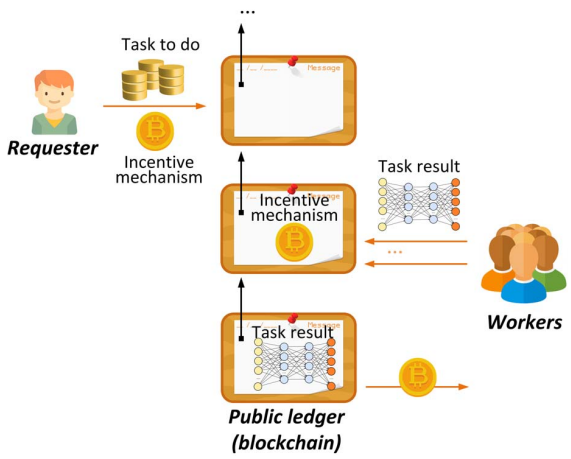

Fig. 1. The model of outsourcing/crowdsourcing complex computations (e.g. analysis for large-scale data) to external workers from requester/blockchain. The computation task is published via the blockchain along with a pre-specified incentive mechanism. The external workers execute the outsourced task and return their results to the blockchain. The incentive mechanism then will reward/punish workers accordingly, such that any misbehavior of submitting incorrect result will be deterred. Remark that if the task to do involves large-scale data, they can be sent through the off-chain way.

In details, the blockchain, the requester, and the workers can be understood as follows:

- $Blockchain$. The blockchain is always "good" for availability and correctness [8, 19]. The blockchain can faithfully execute some pre-specified programs called smart

[1]Remark that the correct output will not fall into the common-knowledge set $\mathbf{E}_{ck}$, as we consider a program whose output is unpredictably placed into a large output space that is much more considerable than $|\mathbf{E}_{ck}|$.

contracts to compute, transfer money, and/or broadcast execution result. But, smart contracts have to be light and deterministic, due to the intrinsic limitations of the underlying blockchain [5]. Worse still, the blockchain is transparent and promise no privacy guarantee, as all its internal states are public to everyone.

It is worth to noticing that we consider the smart contract can detect any particular false output belonging to $\mathbf{E}_{ck}$, as $\mathbf{E}_{ck}$ is actually common knowledge. This is still a scenario much more stringent than the presence of a TTP who has to actively show up to resolve any dispute.

- $Requester$. The $requester$ requests the blockchain to execute a smart contract, which in practice is done through sending the blockchain a transaction pointing to the contract's program code[2]. Also, the requester promises some pre-specified monetary rewards to incentivize the blockchain to enforce the smart contract[3].

  But the $requester$ cannot request the blockchain to run heavy and randomized computations to facilitate his/her business. In such case, s/he will seek to outsource the task to off-chain service providers. Let the outsourced program be randomized, and represent it as $\vec{y} = P(\vec{x}; \vec{r})$, where $\vec{x}$ is the inputs, $\vec{y}$ is the outputs, and $\vec{r}$ is the random coin. We remark our assumption that the outputs $\vec{y}$ will not be predictable as P can extract considerable entropy from the random coin $\vec{r}$ (or from the input $\vec{x}$ as in some cases $\vec{r}$ can be empty), which actually captures many machine learning algorithms.

  Also, the requester promises an incentive mechanism pre-specified via the blockchain to incentivize the external computation services to compute his/her task, and we can view the requester as "good", since all his/her behaviors are codified into self-enforced contract.
- $Workers$. As the blockchain cannot enforce the execution of complex and randomized computations for the requester, the requester/blockchain will further outsource these computation tasks to some external computing services, called $workers$, with announcing a pre-specified incentive mechanism via the blockchain. When workers claim to compute a outsourced task, they have to transfer enough deposits to the blockchain, such that "penalties" can be applied, when the blockchain realizes undesirable behaviors of the workers.

  More importantly, any worker is rational, that means a work only prefers to maximize its utility. If some workers form a coalition, these workers will prefers to maximize the coalition's utility as well.

Monetary parameters. As we consider the problem in the game-theoretic setting, the monetary parameters should be clearly defined. In the following, we will explain the related parameters, and clarify the rationale behind:

- r represents the reward for a task, which is promised by the requester and facilitated by the blockchain;
- d corresponds the deposit should be funded by a worker, if the worker claims to do the outsourced task;
- c means the cost of a worker to do the task, which corresponds to the cost of running a program in a computer.

Let us consider a situation that $r \gg c$, i.e., the payment of the requester should be significantly larger than the cost of computing. That can be observed from the real-world blockchain such as Ethereum, where users have to pay pretty considerable premiums for on-chain computation resources. The requester in our case should be fine with such a surcharge as well, because his/her purpose is to let the correct computing result shown in the blockchain to further facilitate his/her business, which is much more valuable than the tiny cost of computing. So we will ignore c in the remaining of this paper.

Security goal. Next, we specify the basic security requirements for outsourcing the expensive and randomized computation to off-chain services in game-theoretic setting.

Detailedly, we consider that a requester outsources the execution of a computation task to n workers denoted by $W_1, \ldots, W_n$. Our protocol and incentive design essentially instantiate a game Γ joined by these workers, as the workers have different utilities under different joint strategies. For a work W_k, we let its strategy set denoted by $\mathbf{S}_k$. Also, there is always a strategy of "always sending the true result", denoted by σ_t, in per each worker's strategy set.

Our security requires that the joint strategy $\vec{\sigma} = \{\sigma_t, \ldots, \sigma_t\}$ is the only one that survives iterated elimination of weakly dominated strategies (IEWDS). The intuition of such an IEWDS refining equilibrium is quite clear: although sending the correct result might not always bring better utilities, when other workers do not send the correct result, but there is always no harm for a worker to send the correct one, and there even is a situation where sending the correct result brings the dominating utility. So at such a refined equilibrium, everyone worker will apply the strategy of "always sending the true result". When this security requirement is satisfied, we can say our protocol realizes a *practical mechanism* $(\Gamma, \vec{\sigma} = \{\sigma_t, \ldots, \sigma_t\})$, as it can realize the desired Nash equilibrium $\vec{\sigma}$ surviving IEWDS in the game Γ.

IV. Protocol: two independent workers

In this section, we consider a fundamental scenario where the complex and randomized computation task is outsourced to two independent workers. Also, the absence of a trusted third-party (TTP) arbiter is taken into consideration. We observe that the state-of-the-art incentive mechanism will suffer from deviation by free-riding due to the absence of TTP arbiter, which can be intuitively understood as that a worker always prefers to copy and paste the result reported by the other worker. We present our protocol to address this critical issue, and analyze the security in game-theoretic setting.

Intuitions. Our basic idea is to outsource the computations of a smart contract to two $independent$ rational workers, such

[2]After a smart contract is deployed in the blockchain, it will have a unique blockchain address, such that one can point it out through that address.

[3]In practice, the incentive mechanism of smart contract is much more detailed, c.f. the gas mechanism of Ethereum for details.

that the workers can perform the computation *off-chain* and the blockchain nodes therefore get rid of doing these heavy work. At the same time, a simple incentive mechanism should be hosted by the blockchain to incentivize the workers submit the correct output (and prevent false output as well).

A naive solution may allow the two workers submit their computing results to the blockchain anytime, which implies someone can submit earlier, and the other one can report the computing result later. Unfortunately, the blockchain is transparent, so the latter worker can free-ride by copying the first submission without doing any computation; worse still, the second worker will submit the same, even if knowing the earlier submission is wrong.

Intuitively, the major challenge is that the free-riding problem discourages the two workers to compute the result independently. To solve this critical issue, our idea is to ensure that the two workers reveal their computing results to the blockchain simultaneously. For the purpose, our protocol involves a *commit* phase, and a *reveal* phase to let the results "simultaneously" posted on the blockchain (or once committing a result, one cannot change the value even if revealed later). Such that, the incentive mechanism can ensure: 1) rational workers always submit their results "simultaneously", otherwise there will be significant penalty; 2) rational workers have no interest to deviate by committing a false result.

Protocol details. Our protocol can be described as follows:

- TaskPublish. *The requester announces the computing task via the blockchain.*

  When the requester R would like to run intensive computations atop the blockchain, she sends a smart contract that clearly specifies the task and the incentive mechanism. The computing task can be viewed as a randomized program denoted by $P(\vec{x};\vec{r})$, where $\vec{x}$ is the inputs and also stored on the chain, and $\vec{r}$ is the random coin which will be announced by the requester in the next phase.

- TaskPrepare. *All participants fund enough deposits, and the random coin is released if necessary.*
  The budget (i.e., the promised rewards, denoted by r) of the requester will be deposited to the same contract. Then, a worker[4] that is interested in the task should send its deposit, denoted by d.
  Once both two workers, denoted by W_i, W_j, send their deposits. Random coin can be released, denoted by $\vec{r}$, if necessary. In practice, this coin can be randomly picked by the requester, or be derived by a future block [20].
  After all these are completed, the protocol can move on. Remark that the blockchain addresses of W_i, W_j are recorded by the contract, such that the contract can require the workers to authenticate their transactions by signing under the corresponding secret keys later.

- Commit. *The workers send the commitments encapsulating their computing results, respectively.*

  A worker, for example W_i, figures out a result, denoted by $\vec{y}_i$ for the task. If the result is correctly compute, we can expect $\vec{y}_i$ same to $\vec{y} = P(\vec{x};\vec{r})$. Then, W_i will send the commitment of $\vec{y}_i$, denoted by c_i, to the smart contract. The commitment c_i is gotten via a standard commitment scheme[5], namely, $c_i = \mathsf{Commit}(\vec{y}_i, , K_i)$, where K_i is a randomly chosen secret of W_i until to open the commitment. For the other worker W_j, it also commits its result to the blockchain, which can be done through the same commitment scheme.
  Assuming that all computations and commitments can be done within a-priori delay Δ_c (in unit of block number), Δ_c can be pre-specified by the contract. If any worker misses the time to commit, its deposit is taken away.
  We also remark that the execution of the outsourced computation is done *off-chain*, which will hurt the consensus of the underlying blockchain.

- Reveal. *The workers open the commitment to reveal their computing results, respectively.*
  When both workers commit results, a worker, e.g. W_i can reveal the result $\vec{y}_i$ to the blockchain. If the blockchain receives K_i, it computes $\mathsf{Open}(c_i, K_i)$ to get $\vec{y}_i$. If the Open algorithm outputs $fail$, W_i will lose all its deposit. Also, supposing that reveal should be done within a-priori countdown timer Δ_o, a countdown timer can be instantiated by the contract, if any worker fails to reveal in time, its deposit will not be returned.

- Reward. *The blockchain rewards/punishes the workers according to the results they revealed.*

  Once both the commitments are open or the countdown timer for revealing stage expires, the smart contract will check the equality of result(s), and make rewards or penalties, according to the pre-defined incentive clauses. More detailedly, the clauses can be abstracted as:
  - if any worker fails to commit or de-commit or submit a result in the common-knowledge set $\mathbf{E}_{ck}$, it loses all deposits, and the other party gets only half of its deposit refunded (if it submits a result);[6]
  - if $\vec{r}_i = \vec{r}_j$, the two workers will split the reward, i.e., W_i gets $r/2$, and W_j gets $r/2$;[7]
  - if $\vec{r}_i \neq \vec{r}_j$, both the two workers will lose half of their

[4]We remark that we ignore how to select two workers out of all qualified services. Any proper worker selection method should be compatible here, for example, we can either let the requester randomly choose, or let the workers bid, or based on the reputation score (if there is).

[5]Remark that the commitment scheme itself is not necessarily non-malleable, as the transactions encapsulating commitments are accepted by the contract only if they are validly signed by workers.

[6]In practice, we may have valid heuristics to efficiently check weather an output is in $\mathbf{E}_{ck}$. For example, the false ones in $\mathbf{E}_{ck}$ usually have small entropy (except some cases like outputting inputs), while the output of programs is unpredictable in our setting, which infers much more entropy. Such that the requester can provide a list of random coins and then let the workers to compute the program for several times under different random coins, which will allow the smart contract to easily filter out low-entropy results as they are suspiciously chosen from the set of $\mathbf{E}_{ck}$.

[7]One may also wonder a trivial strategy that both of the workers output some junks in the set of $\mathbf{E}_{ck}$. But these have been captured by the contract, and are seen as failures of submitting.

TABLE I
THE GAME OF TWO WORKERS IN NORMAL FORM

Utility of W_i,W_j (W_i \ W_j)	σ_e	σ_t	σ_f
σ_e	$-d, -d$	$-d, -d/2$	$-d, -d/2$
σ_t	$-d/2, -d$	$r/2, r/2$	$-d/2, -d/2$
σ_f	$-d/2, -d$	$-d/2, -d/2$	$-d/2, -d/2$

deposits, i.e., W_i gets $-d/2$, and W_j gets $-d/2$.

Intuitively, the blockchain can only know the following information: 1) the equality of two results (if there are two); 2) anyone who fails to submit a result. And our incentive mechanism are trying to make best usage of these information to deter the submission of incorrect results.

Security analysis (sketch). Here we briefly discuss the security of the above protocol in game-theoretic setting. Let us firstly check all possible pure strategies of each worker. Considering that the workers are non-colluding, there are three pure strategies for each worker in general:

- "do not commit a result in the Commit phase, or do not open the commitment in the Decommit phase, or submit a result in $\mathbf{E}_{ck}$", and we denote this strategy as σ_e;
- "reveal the correct result to the blockchain", and this strategy is denoted by σ_t;
- "reveal some false output chosen out of $\mathbf{E}_{ck}$", and let this pure strategy denoted by σ_f.

Since we consider a program having large output space, where the possible correct results are placed in an unpredictable way. So when two non-colluding workers are submitting two junks out of $\mathbf{E}_{ck}$ without coordination, their best way is to sampling random junks, as the unpredictable output makes them have no advantage other than guessing. Also, for large output space, *there is no chance for such two junks to be the same*, i.e., the contract can always find two different submissions, if both workers makes junks out of $\mathbf{E}_{ck}$.[7]

Now we are ready to show the security of our protocol. In Table.I, the utilities of two workers are shown for different joint strategies, as the incentive clauses are clearly defined. We can leverage IEWDS method to check our game:

- For any worker, the pure strategy of playing σ_e is strictly dominated (i.e. worse utility for sure), so σ_e is "deleted" for both workers, i.e., both workers will at least submit;
- For any worker, the pure strategy of playing σ_f is weakly dominated (i.e. not better utility for sure), so σ_f is "deleted" for both workers, i.e., sending the true result is at least not worse than sending random junk.

So the only left IEWDS refining Nash equilibrium is the joint strategy $\{\sigma_t, \sigma_t\}$, i.e., "both workers send the correct result", and we can claim our protocol realizes a *practical mechanism* $(\Gamma, \{\sigma_t, \sigma_t\})$, which clearly realizes our security goal.

V. CONCRETE INSTANTIATIONS

Here we will present two concrete machine learning (ML) related examples, and show the subtleties to apply our protocol for either a deterministic computation intensive program (e.g. test a machine learning classifier) or a randomized program (e.g. train a machine learning classifier).

Evaluating ML classifier. A machine learning classifier is a trained model that can predicate a data sample to which category. Informally, a ML classifier can be abstracted as a deterministic function represented as $\vec{y} = f(\vec{x}; \vec{w})$, where $\vec{w}$ is the parameters of the classification model, $\vec{x}$ is a batch of input data, and $\vec{y}$, as the output, corresponds to a string representing the predicated category for per each test data sample. The performance of a ML classifier can be evaluated via many well-known methods, one of which is to feed the classifier a test set of some data samples that have known categories.

Consider the application that one would like to purchase a good trained models via blockchain, e.g., pay only when the precision is above 90%. The smart contract can properly wrap our protocol, and evaluate the performance of the submitted ML classifier and then make the proper payment. Note that we need to make the output of the computation unpredictable to the workers (which is significant to prevent workers obtain advantages other than the common-knowledge set $\mathbf{E}_{ck}$, so they can only guess when output junks out of $\mathbf{E}_{ck}$). Detailedly, when the purchaser is requesting an ML classifier (or the parameters $\vec{w}$), a smart contract is published to announce the purchase with a clearly defined incentive mechanism that follows our design rationale. Since the quality of the model will be justified by the testing data, to make sure the testing result is unpredictable to the workers, the requester then randomly chooses some subset $\vec{x}$ from his test set. Now the parameters and the testing data are both present, the workers are incentivized to check the precision of the returned model.

The payment condition in the smart contract will be expecting the bitstring indicating which test data is correctly classified. If both workers return the same indicating string, then the contract computes the precision and then decides the payment. This step is simply to ensure the unpredictability of the output (the precision of the model), a simpler solution could simply let each worker to return the predicating precision as a real number with several bits of precision to increase the uncertainty without requiring the contract to compute the precision from the indicating string.

The above case shows the subtlety to use our protocol for deterministic programs: we have to make sure the outsourced program can "extract entropy" from the input, otherwise, the predictable output might raise security concerns. Besides this key observation, this particular use-case points out a promising way to decentralizing the marketplace of machine learning models, where ML models can be bought/sold by fine-grained pay-per-performance rules.

Training ML classifier. The training of ML classifier is often a randomized and complex program (e.g., random forest, deep learning, etc.). Usually, we can view the randomized training of a machine learning classifier as a function $\vec{w} = F(\vec{x}; \vec{r})$, where $\vec{x}$ is the input training data along with their ground truth categories, $\vec{r}$ corresponds the random coins used in the

execution, and the output $\vec{w}$ is the classifier's parameters that capture the knowledge learned from $\vec{x}$. For example, in deep learning, the random coins represent the randomness used in each stochastic gradient descent (SGD) iteration.

Consider another interesting application of crowdsourcing computing resources for an ML training on blockchain. Our protocol can also be leveraged to execute ML classifier training tasks over the blockchain, with only few modifications to the training programs. For example, when someone is trying to train a ML classifier over a large scale of data set, s/he can publish a smart contract to announce the program of training $F(\cdot;\cdot)$, the training set $\vec{x}$, the random coin $\vec{r}$, and an incentive mechanism. Here only the digest of the training data needs to be uploaded on chain, while the actual data can be stored in some cloud storage. The workers should check the validity of the training data by comparing the digest first.

One other practical challenge is: the training program might need an unpredictable amount of random coins, but the input randomness $\vec{r}$ has fixed-length. Technically, to make a training program compatible with our protocol, we need keep an internal counter ctr in the program. When the training program is using a random coin (for example, a new random coin used for a new SGD iteration), it will compute and use $h(\vec{r}||ctr)$ as the coin, and then the counter ctr will increase by one. Note that $h(\cdot)$ is usually modeled as a random function. This trick enables the training program to use a considerable amount of public random coins, although the randomness source is just one fixed-length input, i.e., $\vec{r}$. More importantly, this technique ensures that there is only one training result, although the process of training is randomized. Taking our protocol, external computation services will perform the above "randomized" training process, and report the correct classifier parameters. This promising example captures many interesting applications that are driven by large-scale data.

The above concrete example clarifies the key technique to use our protocol for randomized programs, that is to modify the program a little bit such that it can extract randomness from a coin given by the requester. Thus, many interesting decentralized applications can be imagined, for example, a social computing marketplace atop the blockchain.[8]

VI. Extension: n workers ($|\text{coalition}| \leq n-1$)

The protocol for two workers rely on a key assumption that they are independent, i.e., non-colluding. Now we are ready to present our protocol in a more general scenario where n workers instead of two workers can hired to compute the outsourced computations. More generally, we will allow the n workers to collude, as long as the size of their coalitions is up to $n-1$ (inclusively). Detailedly, in such a coalition, the colluding workers there can make a strategy pre-agreed by all of coalition members to deviate the game. If the coalition strategy is not weakly dominated by "always sending the correct", the mechanism fails. Otherwise, the mechanism is still a $(n-1)$-resilient practical mechanism $(\Gamma, \vec{\sigma} = \{t, \ldots, t\})$ that can tolerate these coalitions. In this section, we will show the existence of such a $(n-1)$-resilient practical mechanism. Remark that this more general conclusion essentially captures the situation of two independent workers (i.e. 1-resilient practical mechanism).

Protocol brief. The protocol for n workers also has TaskPublish, TaskPrepare, Commit, Reveal and Reward phases, which are similar to the protocol for two workers. Therefore we only focus on the incentive mechanism for n workers, and present the clauses only:

- if any worker fails to commit or de-commit or submit a result in the common-knowledge set $\mathbf{E}_{ck}$, it loses all deposits, and all the other workers who submit will only get a refund of half deposit;
- if n workers successfully report the same computing result, the n workers will share the reward equally, i.e., each worker gets r/n;
- if n workers submit different results, i.e., there is as least one submission differs from others, all workers will lose half of their deposits;

The above incentive mechanism for n workers is a straightforward extension of the one for two workers. Again, the blockchain can only know the equality of reported results, along with who fails to submit. Our incentive mechanism is designed to use these little information to deter false results.

Security analysis (sketch). Here we briefly discuss the security of the protocol for n workers. For a coalition C of size up to k $(1 \leq k \leq n-1)$, it roughly has the following strategies:

- let all player in it play e, i.e., the coalition will submit nothing, denoted by σ_1;
- let all members play t, i.e., the coalition will submit all correct results, denoted by σ_2;
- let all members "submit the same pre-agreed randomly chosen junk", denoted by σ_3;
- let p members send the correct results, and the other $l = k-p$ members send nothing, and the family of strategies can be denoted by $\boldsymbol{\sigma}_4$;
- let q members send the random junks, and the other $l = k-q$ members send nothing, and we denote the family of strategies by $\boldsymbol{\sigma}_5$;
- let p members send correct results, and the other $q = k-p$ members send junks, denoted by $\boldsymbol{\sigma}_6$;
- let q members to send junks, some p members send correct results, and the remaining $l = k-p-q$ members send nothing, and this strategy family denoted by $\boldsymbol{\sigma}_7$.

Notice that a coalition's strategy can be enforced for all its member workers. For example, the same randomly chosen junk can be agreed by all its members. But for another worker who is out of the coalition C, it cannot know what the random junk agreed within C. When C is making a decision on what a strategy to play, its utility can be discussed by three situations. In each of the situation, the out-of-coalition worker(s) might play different strategies:

[8]We remark that the crowd-shared data, as the input of social computing, can also be collected via the blockcahin [3].

TABLE II
THE UTILITY OF AN ARBITRARY COALITION C ($|C| \leq n-1$) IN THE GAME OF n WORKERS

Utility of C: Others' strategy / Strategy of C	Situation **I**	Situation **II**	Situation **III**	Note
σ_1	$-kd$	$-kd$	$-kd$	k is the size of coalition C
σ_2	$-kd/2$	kr/n	$-kd/2$	
σ_3	$-kd/2$	$-kd/2$	$-kd/2$	
$\boldsymbol{\sigma}_4$	$-pd/2-ld$	$-pd/2-ld$	$-pd/2-ld$	$p+l=k$, $p\in\mathbb{N}^+$, $l\in\mathbb{N}^+$
$\boldsymbol{\sigma}_5$	$-qd/2-ld$	$-qd/2-ld$	$-qd/2-ld$	$q+l=k$, $q\in\mathbb{N}^+$, $l\in\mathbb{N}^+$
$\boldsymbol{\sigma}_6$	$-(p+q)d/2$	$-(p+q)d/2$	$-(p+q)d/2$	$p+q=k$, $p\in\mathbb{N}^+$, $q\in\mathbb{N}^+$
$\boldsymbol{\sigma}_7$	$-(p+q)d/2-ld$	$-(p+q)d/2-ld$	$-(p+q)d/2-ld$	$p+q+l=k$, $p\in\mathbb{N}^+$, $q\in\mathbb{N}^+$, $l\in\mathbb{N}^+$

- Situation **I**: there is at least one out-of-coalition worker submits nothing (including to submit a result in $\mathbf{E}_{ck}$);
- Situation **II**: there is at least one out-of-coalition worker submits a randomly chosen junk (out of $\mathbf{E}_{ck}$);
- Situation **III**: all out-of-coalition worker(s) submit the correct result.

When the coalition C applies different strategy in different situations, the utility of this coalition can be derived and shown in Table.II. Essentially, for any coalition formed by up to $n-1$ workers, its utility can be represented in a similar table. After performing IEWDS, the only survived joint strategy of all coalitions is "to send the correct results". In details, the iterative deletion of the weakly dominated strategies can be discussed as follows:

- For any coalition, the pure strategy σ_1 is strictly dominated, so σ_1 is deleted, also, situation **I** can be removed, as a consequence of the previous deletion;
- For any coalition, the pure strategies in $\boldsymbol{\sigma}_4$, $\boldsymbol{\sigma}_5$ and $\boldsymbol{\sigma}_7$ are strictly dominated, and therefore these strategies can be removed one by one;
- For any coalition, the pure strategies in $\boldsymbol{\sigma}_6$ are weakly dominated, and therefore can be removed, also after this deletion, the situation **II** disappears.

Clearly, for any coalition C ($1 \leq |C| \leq n-1$), the only left pure strategy is to "send all correct result(s)" after IEWDS. Actually, the game even can guarantee that no matter how a coalition plays, no single worker in that coalition can get better utility than "sending the true result". Here we omit the trivial steps of proving this conclusion. As such, all n workers will submit the correct results, even if we admit that these workers can collude to make up any coalitions size up to $n-1$. In sum, we can realize a practical mechanism $(\Gamma, \{\sigma_t, \ldots, \sigma_t\})$ which is $(n-1)$ resilient.

VII. CONCLUSION

We introduce a very simple game theory perspective to overcome the inherent limitations of public blockchains, such that the chain can "audit" the execution of a large class of computationally intensive programs (including most machine learning tasks per se). We envision such a solution as a stepping stone towards more general future work to enable any computational intensive program atop the public chain.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.
[3] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *IEEE ICDCS '18*, pp. 853–865.
[4] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE S&P 2014*, pp. 475–490.
[5] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings ACM CCS '15*, pp. 706–719.
[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.
[7] J. Plantin. How outsourcing can help you build a better startup. [Online]. Available: https://medium.com/@JackPlantin/how-outsourcing-can-help-you-build-a-better-startup-5d04516fe71a
[8] A. Kosba, A. Miller, E. Shi *et al.*, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE S&P 2016*, pp. 839–858.
[9] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM CCS 2016*, pp. 270–282.
[10] J. Van Bulck, M. Minkin, O. Weisse *et al.*, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Security Symposium 2018*.
[11] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, pp. 465–482.
[12] E. Ben-Sasson, A. Chiesa, D. Genkin *et al.*, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, pp. 90–108.
[13] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://eprint.iacr.org/2016/086.
[14] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proceedings ACM CCS '17*, pp. 211–227.
[15] V. Pham, M. H. R. Khouzani, and C. Cid, "Optimal contracts for outsourced computation," in *International Conference on Decision and Game Theory for Security*. Springer, 2014, pp. 79–98.
[16] R. Nix and M. Kantarcioglu, "Contractual agreement design for enforcing honesty in cloud outsourcing," in *International Conference on Decision and Game Theory for Security*. Springer, 2012, pp. 296–308.
[17] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya, "Incentivizing outsourced computation," in *Proceedings ACM NetEcon '08*, pp. 85–90.
[18] A. Kupcu, "Incentivized outsourced computation resistant to malicious contractors," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 633–649, Nov 2017.
[19] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. EUROCRYPT 2015*. Springer, pp. 281–310.
[20] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *Proc. IEEE S&P 2015*, pp. 287–304.