Agatha: Smart Contract for DNN Computation

Zihan Zheng
University of Science and Technology
of China
Hefei, China
zzh1996@mail.ustc.edu.cn

Peichen Xie
Peking University
Beijing, China
xpc@pku.edu.cn

Xian Zhang
Microsoft Research
Beijing, China
zhxian@microsoft.com

Shuo Chen
Microsoft Research
Beijing, China
shuochen@microsoft.com

Yang Chen
Microsoft Research
Beijing, China
yachen@microsoft.com

Xiaobing Guo
Microsoft Research
Beijing, China
xiaobing.guo@microsoft.com

Guangzhong Sun
University of Science and Technology
of China
Hefei, China
gzsun@ustc.edu.cn

Guangyu Sun
Peking University
Beijing, China
gsun@pku.edu.cn

Lidong Zhou
Microsoft Research
Beijing, China
lidongz@microsoft.com

Abstract

Smart contract is one of the core features of Ethereum and has inspired many blockchain descendants. Since its advent, the verification paradigm of smart contract has been improving toward high scalability. It shifts from the expensive on-chain verification to the orchestration of off-chain VM (virtual machine) execution and on-chain arbitration with the pinpoint protocol. The representative projects are TrueBit, Arbitrum, YODA, ACE, and Optimism. Consequently, verification of more and more complicated computations for smart contract is achieved, such as the aggregated execution of DeFi transactions. Inspired by visionaries in academia and industry, we consider the DNN (deep neural network) computation to be promising but on the next level of complexity for the verification paradigm of smart contract. Unfortunately, even for the state-of-the-art verification paradigm, off-chain VM execution of DNN computation has an orders-of-magnitude slowdown compared to the native off-chain execution.

To enable the native off-chain execution of verifiable DNN computation, we present *Agatha* system, which solves the significant challenges of *misalignment* and *inconsistency*: (1) Native DNN computation has a graph-based computation paradigm misaligned with previous VM-based execution and arbitration; (2) Native DNN computation may be inconsistent cross platforms which invalidates the verification paradigm. In response, we propose the graph-based pinpoint protocol (GPP) which enables the pinpoint protocol on computational graphs, and bridges the native off-chain execution and the contract arbitration. We also develop a technique named Cross-evaluator Consistent Execution (XCE), which guarantees cross-platform consistency and forms the correctness foundation of GPP. We showcase Agatha for the DNN computation of popular models (MobileNet, ResNet50 and VGG16) on Ethereum. Agatha achieves a negligible on-chain overhead, and an off-chain execution overhead of 3.0%, which represents an off-chain latency reduction of at least 602× compared to the state-of-the-art verification paradigm.

1 Introduction

The Turing completeness of *smart contract* enables arbitrary computations to be executed and verified on blockchain nodes. It underpins the expressiveness of the decentralized computing paradigm. As a result, more and more complicated applications are emerging on the Ethereum platform [27], such as ERC-20 tokens [80], Cryptokitties [41], Zero-knowledge Proof verification [24] and Decentralized Finance (DeFi) [49].

However, current smart contracts are limited by low efficiency for complex applications, since a contract's computation is re-executed and verified on all Ethereum nodes (e.g., miners, full nodes). For example, a simple task of naïve matrix multiplication of 1000×1000 integers would cost over 3 billion gas (the unit of cost in Ethereum), which far exceeds the current Ethereum's block gaslimit (i.e., 15 million). Even if the task could span over 200 blocks to meet the gaslimit, it would result in an extremely low throughput of 4.2×10^{-4} tasks per second, given Ethereum's block interval of ~12 seconds. In comparison, even a low-end dual-core laptop can do this type of simple tasks with a latency of 0.39 seconds and a throughput of 2.5 tasks per second.

To improve contracts' efficiency for complex applications, one of the most promising solutions is to offload a task off-chain to a quorum of nodes and leave only a contract for arbitrating "fraud proofs" on-chain. A fraud proof, which is generated via an interactive *pinpoint protocol*, proves that the claimed result of a computation is fraudulent/incorrect. The arbitration process is designed to validate a fraud proof by executing only a few minor computation steps, making the on-chain cost extremely low. Existing technologies, such as Plasma [64], TrueBit [76], Arbitrum [44], YODA [19], ACE [82], Optimism [62], share this key idea. This opens up exciting opportunities for smart contracts to fulfill new scenarios in the next complexity level, such as the escrow contract, iterated hashing, aggregated execution of DeFi transactions, etc.

If the complexity moves up one more level, AI computations naturally become a fascinating paradigm that people want smart

contracts to support. Indeed, researchers in both the academia (e.g., YODA [19], TrueBit [76], ACE [82]) and the industry (e.g., Microsoft [36] and startups [18], [32], [46], [39]) have put forth the vision to fulfill AI computations with decentralized consensus. However, this is an uncharted territory, because contemporary AI computations, specifically DNN (deep neural network) computations, impose serious challenges for all existing technologies:

- *Overhead of on-chain execution.* Some proposed technologies [32, 36, 39, 46, 56, 70] either target small AI computations or only target the consensus among a small number of nodes. They do AI computations by the on-chain execution, thus cannot scale. If these approaches are used on Ethereum to run a DNN computation, the cost would be prohibitively high. For example, a single VGG16 [69] inference is estimated to consume ~ 90 billion gas, equivalent to running 4.3×10^6 ETH-transfer transactions[1] on the Ethereum platform. It is obviously impractical.

- *Overhead of off-chain execution.* Other technologies [19, 44, 64, 76, 82] orchestrate off-chain execution and contract arbitration, and thus significantly reduce the on-chain cost. However, to guarantee an alignment and the consistency between the off-chain execution and the contract arbitration, they all rely on restricted custom virtual machines (VMs) for the off-chain computations. For example, floating-point arithmetic and the multi-threaded execution, which commonly exist in DNN computations, are prohibited in these VMs. This may result in orders of magnitude slowdown compared to the native execution. It also significantly increases the burden for the verifiers and the pinpoint protocol.

Problem statement. The problem we consider in this paper is: *how to enable smart contract verification of DNN computations with low cost, both on-chain and off-chain.* Regarding the scale of the computations, we consider the popular DNN models, such as ResNet50 [38], VGG16 [69] and MobileNet [66].[2] Once the smart contract obtains this scale of DNN competence, one can imagine many types of real "smart" applications, such as:

- *Intelligent Automated Market Maker (AMM).* AMM such as Uniswap [49] has revolutionized the economic ecosystem of Ethereum. DNN can improve the liquidity and profit of the market maker [71, 72], which is also envisioned in the DeFi community [65].

- *Decentralized AI marketplace.* Decentralized exchanging of digital goods has been proposed recently for stronger fairness, compared to those centralized solutions [16, 22, 25]. There is a clear need to exchange AI models in the marketplace. Verification of DNN computations is essential to enable this scenario [46].

- *Blockchain-based Uber (BUber).* BUber has been extensively discussed to mitigate the fairness issues due to the opacity of the centralized intermediary [5, 75]. Scheduling algorithm with a DNN capability can further enhance BUber's efficiency [50].

- *Decentralized paper ballot counting.* Handwritten-signature recognition is used for paper ballot counting in national elections [60]. Allowing a decentralized smart contract to run the process may help increase the transparency and make the public more confident about the results [55, 67].

Agatha system. In response to this community vision, we develop a system named Agatha, which demonstrates the first practical contract verification for DNN computation on the public Ethereum. Agatha follows the existing off-chain execution approaches with contract arbitration [19, 64, 76, 82], so the on-chain overhead is greatly reduced. The main difference between Agatha and the previous approaches is about the off-chain computation, where Agatha enables the *native execution of DNN computation*, whereas others only support restricted VM-based execution. Therefore, Agatha significantly reduces the overhead of the off-chain execution, making the complex computation practical.

Technologies. Enabling the off-chain native execution of DNN computation for smart contract is our main contribution. It faces two significant challenges:

- *Misalignment of native execution and contract arbitration.* Although previous work, such as TrueBit [76], claims to achieve native off-chain execution for general-purpose computation, DNN computation is much different from their showcased applications. DNN computation is implicitly expressed as a computation graph of operations with rich toolchain support [7, 11, 58, 61]. In contrast, existing contract arbitration requires the computation to be expressed by serialized VM instructions. Therefore, considering hardware features used by conventional DNN computations, such as multi-threading, SIMD or even GPU instructions, it is an enormously complicated (and unnecessary) detour to transcode between DNN computations and serialized VM instructions and ensure their consistency.

- *Cross-platform inconsistency.* Native execution of DNN computation, especially on different hardware platforms, may lead to different execution results (floating-point vectors). This can be ascribed to various factors, such as imprecise approximations, different accumulation orders of floating-point numbers, etc. Different execution results or temporary variables during the execution can discredit the fraud proof of the quorum since every honest node can be a "fraud" due to execution on different platforms.

To solve the challenges, the key technologies of Agatha system are twofold:

- *Graph-based Pinpoint Protocol (GPP),* which is a protocol that can generate the "fraud proof" interactively and efficiently. Instead of representing DNN computation as executing VM instructions, GPP represents DNN computation as evaluating a graph of operations (e.g., Conv, Gemm) and further a graph of basic operations (e.g., fadd, fmul), where the evaluation results of basic operations can be efficiently arbitrated by our smart contract equipped with the floating-point arithmetic emulation. Moreover, the graph-based representation is highly compatible with conventional toolchains of DNN computation. By guaranteeing that evaluating the graph is consistent with the native execution, Agatha's pinpoint protocol is purely based on the graph evaluation, which circumvents the misalignment challenge. In addition, we introduce the *Two-phase Pinpoint* design, which optimizes the protocol to equip an orchestrated execution of both coarse-grained evaluator (native execution in the granularity of operation) and fine-grained evaluator (simulation in the granularity of basic operation).

- *Cross-evaluator consistent execution (XCE),* which guarantees the three consistencies between: (1) native execution and operation

---

evaluation, (2) operation evaluation and the evaluation of basic operation, and (3) basic-operation evaluation and the smart contract arbitration. To achieve every consistency, we have two steps in general: the first is to make the execution either compliant with the IEEE-754 standard of floating-point arithmetic or in integers; the second is to ensure a fixed order of floating-point sum/product. Following the workflow, we conduct a comprehensive investigation and solid tests to ensure consistency, which spans over hardware heterogeneity, compiler options, arithmetic libraries and so on. We formally prove that with XCE, the cross-platform consistency and the correctness of Agatha system are achieved.

In summary, GPP establishes the infrastructure that bridges the gap between the native DNN computation and the contract arbitration. And XCE forms the correctness foundation of GPP by ensuring the cross-evaluator consistency, which naturally leads to the cross-platform consistency.

Results. We showcase Agatha system for DNN inference using MobileNet [66], ResNet50 [38] and VGG16 [69], which are models widely used in the industry. The evaluations confirm the correctness of XCE, using unit tests and end-to-end tests, which include ∼7500 floating-point corner cases. We also measure the performance and the gas consumption of Agatha. Our off-chain native execution is observed to have an insignificant latency overhead (3.0% on average) compared to the original DNN computation. By contrast, previous VM-based off-chain execution has an average slowdown of 620×, which is 602× greater than ours. Regarding the on-chain cost: When there is no dispute, i.e., the normal case, the cost of Agatha is equivalent to ∼3 ETH-transfer transactions; When the two disputing parties fight all the way to the arbitration, i.e., the worst case, the cost is equivalent to ∼86 ETH-transfer transactions. These performance and cost numbers demonstrate Agatha's practicality for DNN computation, given both on-chain and off-chain overhead.

2 Smart Contract for Complex Computations

In this section, we give the background about smart contract for complex computation, which is the approach Agatha follows.

As mentioned in the introduction, our research is inspired by off-chain execution approaches with contract arbitration, such as Truebit [76], Arbitrum [44], YODA [19] and ACE [82]. Figure 1 shows the essence of the approach. The base is the smart contract, which runs on a peer-to-peer network of a huge number of nodes. Because the on-chain execution needs to always maintain a world-wide consensus, it cannot afford to run expensive computations. The basic idea of previous approaches is to have a small number of off-chain nodes (e.g., 10 independent parties or a dynamic quorum). The off-chain nodes can be elected either by network reputation or registration via deposit on the smart contract.

Figure 1 shows the steps in these approaches. For the initialization step ❶, the requestor sends the computation task ϕ and input data D to the off-chain nodes while optionally making corresponding commitment[3]. In step ❶, the first node that finishes computation, which is called the submitter in this paper, claims "$\mathcal{R} = \phi(\mathcal{D})$". This means that $\mathcal{R}$ is the result of computation ϕ on input $\mathcal{D}$. ϕ is too expensive for the smart contract to re-execute, so it

---

[3]The requestor implicitly exists as the transaction senders while the commitment is omitted in the aggregated execution of transactions [62].

is only submitted to the verifiers, which are the rest of the off-chain nodes. In step ❷, every verifier independently validates the claim. Suppose the lower-left verifier declares that the claim is wrong, it starts a *pinpoint protocol*, involving the verifier itself, the submitter and the smart contract. This is shown in step ❸. The verifier tries to disprove the claim by pinpointing one concrete erroneous step in it, with the contract arbitrating in step ❹. Obviously, arbitrating about a single step is easy for the smart contract. In the end, if the submitter survives all challenges from verifiers for a pre-defined period T^v, "$\mathcal{R} = \phi(\mathcal{D})$" is accepted by the smart contract.
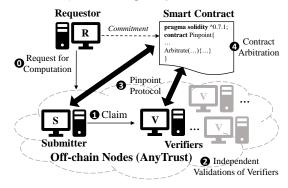

Figure 1: The essence of the previous approaches

There are various assumptions about the verifiers to guarantee security and liveness, such as AnyTrust [44], Financially Rational Players [76] and the hybrid Byzantine assumption [19, 82]. For simplicity, we use the AnyTrust assumption from Arbitrum [44].

The AnyTrust assumption. AnyTrust, rather than "majority trust", assumes that, for every claim, there is at least one honest node. Namely, either the submitter is honest, or at least one verifier is honest and will challenge within the pre-defined period. Suppose there are m verifiers, $m - 1$ of whom collude to stay silent about a submitter's wrong claim. Under AnyTrust, the only honest verifier will still succeed in disproving the wrong claim in front of the smart contract, and the wrong claim is thus rejected (formal definitions in Section 6). Like the previous work [44, 76], we also assume data availability (i.e., $\mathcal{D}$ and ϕ should be accessible to all verifiers) and anti-censorship (i.e., every verifier can always interact with the contract). They are solved by orthogonal countermeasures [48, 54].

The pinpoint protocol. The pinpoint protocol[4] also needs more explanations. The goal of the protocol is shown in Figure 2. The computation ϕ consists of a sequence of VM instructions (denoted as $\mathrm{VMI}_1, \mathrm{VMI}_2, \ldots, \mathrm{VMI}_n$) and the initial VM state is S_0. However, the verifier and the submitter get different states $S_n \neq S'_n$. The goal of the protocol is to pinpoint VMI_k, such that $S_{k-1} = S'_{k-1}$ but $S_k \neq S'_k$, and to send $(\mathrm{VMI}_k, S_{k-1}, S_k)$ to contract for arbitration. The submitter and the verifier are forced to get the same k with a challenge-response mechanism with a timeout penalty [22, 44, 76]. The pinpoint protocol is very efficient: for time complexity, the verifier and the submitter only need $O(\log n)$ rounds of challenge-response for both parties to commit to the number k, and the contract only needs a constant time to arbitrate the disagreement about VMI_k; regarding the space complexity, the state is structured as a

---

[4]Equivalent terminologies include *Bisection protocol* [44] and *Verification Game* [26, 76].

Figure 2: Pinpointing a disputed step in VM execution

Figure 3: The overview of Agatha system

Merkle tree (MT) [57], corresponding to a logarithmic-size message to the contract.

It is worth emphasizing that two conditions are needed for the pinpoint protocol to work: (1) The pinpoint mechanism must be able to locate the disputed computation step, and the contract must be able to arbitrate it efficiently. (2) For every step of the computation, there is only one correct output for a given input. Otherwise, the contract cannot verify its correctness. To achieve the two conditions, the computations handled by previous technologies are written as sequential VM instructions, which fits the instruction-register-memory paradigm. In addition, features such as multi-threading and floating-point are eliminated to guarantee the cross-platform consistency. For example, Arbitrum VM is based on EVM [29] while TrueBit leverages a restricted WebAssembly [34].

The overhead of off-chain computation. Besides the overhead of contract execution (i.e., on-chain part), the overhead of off-chain execution is also critical since the off-chain overhead determines the workload and throughput of off-chain nodes. For example, submitter or verifiers in TrueBit [76] earn bounties proportional to the length of VM instructions, which is the cost to post the computation task for the requestor; Arbitrum VM leverages the Intel SHA extension [35] to achieve a high off-chain throughput. Furthermore, in DeFi applications, the off-chain latency directly determines the efficiency of the market because of the off-chain racing counterparties (e.g., liquidity providers [37] and traders [83]).

3 Agatha Overview

The Agatha system consists of a smart contract deployed on Ethereum and several independent *Agatha clients*. For a DNN computation task, the Agatha system focuses on the verification process after one Agatha client submits a result of the task (specifically, the hash value of the result). If any client finds the result incorrect, and then wins the challenge-response dispute following the Agatha pinpoint protocol, the contract will reject the submitter's result. Otherwise, the submitter's result is accepted.

Similar to the previous technologies, Agatha has a workflow to reduce the on-chain overhead. However, the key difference is that Agatha enables the native DNN computation which reduces the off-chain overhead significantly. The design of the Agatha system is shown in Figure 3 and explained in this section.

3.1 Native execution

The core innovation of Agatha is enabling native execution in scalable contracts for DNN computation (see Table 1). There is a native evaluator E_N in each client. In Agatha, both submitter and verifier execute DNN computation with multi-threaded, highly
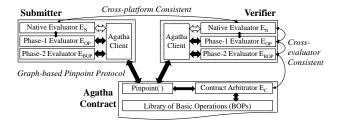
optimized and hardware-accelerated E_N, instead of a restricted single-threaded VM. Using E_N significantly reduces both the submitter's execution latency and the verifier's validation latency. For example, a multi-threaded E_N only takes 0.076 seconds to run a VGG16 inference (see Section 7.3); an ideal VM (simulated by a single-threaded, highly optimized native evaluator) takes 0.439 seconds, 5.78 times more than the E_N; restricted VMs such as Arbitrum VM are typically 800 times slower than the ideal VM. If E_N is further accelerated by GPUs, its performance improvement over restricted VMs can be three orders of magnitude.

Although some previous VM-based schemes have imagined combining native execution, VM-based pinpoint protocol and arbitration [76], they provide no detail. We consider this highly difficult, because of the paradigm misalignment. To enable pinpoint protocol and contract arbitration, the VM paradigm takes a sequence of VM instructions and an initial state as inputs, and outputs a final state. However, a multi-threaded E_N takes a function and some data as inputs, and outputs the result. If we want to bridge them, there are two potential routes. First, if we define ϕ as VM instructions, we need to transcode the serialized VM instructions to a native multi-threaded program and ensure their consistency. Second, if we define ϕ as a function, the verifier and the submitter must have consensus on how ϕ is represented by a specific sequence of VM instructions. However, both routes are hugely unnecessary detours. As we will explain next, the graph-based paradigm is a far more direct representation of DNN computation.

3.2 Unified graph-based paradigm

Agatha forgoes the VM-based paradigm, which is a wrong abstraction for DNN computation. A DNN, as a function, says nothing about instructions, registers, memory, etc. Alternatively, following the mainstream paradigms [7, 11, 61], we express a DNN computation as a computation graph consisting of *operations*. A computation graph is a directed acyclic graph with operations as its nodes and tensors as its edges. For example, Figure 4 shows the computation

Steps	Previous	Hypothetical	Agatha
Submitter's execution	VM	Native	Native
Verifiers' validation	VM	Native	Native
Pinpoint protocol	VM	VM	Semi-native
Contract arbitration	VMI	VMI	BOP

Table 1: Comparison of the previous work, a hypothetical method and Agatha

of ResNet50 [38], and the internal computations of Conv, Relu and Gemm are shown in the figure. An operation is defined mathematically using *basic operations*, such as summation, max and multiplication. In the rest of the paper, we use the terminology "basic operation", or *BOP*, to refer to an individual computation of numbers. The terminology "operation" or *OP*, without a preceding "basic", means the computation of tensors.
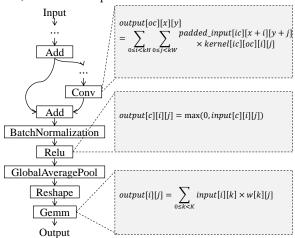

Figure 4: The computation graph of ResNet50

Using the graph representation provides a unified paradigm from native execution to contract arbitration. It also has two main advantages. First, we can leverage existing high-performance graph evaluators, such as ONNX Runtime [58], TensorFlow [11] and PyTorch [61], as our E_N. Second, as we will show next, we can design a simple and efficient pinpoint protocol for DNN computation.

Each Agatha client leverages a phase-1 evaluator E_{OP} and a phase-2 evaluator E_{BOP}, used for our two-phase pinpoint protocol. In phase 1, the DNN computation in dispute invokes E_{OP} to compute the results of the operations, and then the Agatha contract can locate an operation in dispute. In phase 2, the operation in dispute further invokes E_{BOP} to generate the results of the basic operations, and thus the contract can locate a basic operation in dispute. After pinpointing, the contract performs the arbitration by calculating the basic operation on Ethereum with its on-chain evaluator E_C.

The simplicity comes from the graph representation. For this protocol, the arbitration contract only needs to understand integer and floating-point arithmetic, but not other complicated machinery; the client only needs to understand computational graphs, but not sophisticated VM instructions and states.

The efficiency benefits from both native execution and the nature of DNN computation. E_{OP} can invoke the high-performance E_N, and E_{BOP} can make arbitration simple. Imagine two alternative pinpoint protocols. The first one, without phase 1, uses E_{BOP} to generate internal intermediate results of the whole graph, and lets the contract locate a basic operation in dispute. This is too inefficient for a DNN computation, which usually has billions of basic operations. The second one, without phase 2, demands the contract to directly locate an operation in dispute, and then arbitrate the operation by re-executing it on-chain. This is not practical for DNN computations, whose operations are too complex to implement and execute in smart contracts.

3.3 Consistency

Forgoing the VM-based paradigm brings a new issue about verification – inconsistency. Using the graph-based paradigm, we notice that operations in DNN computation are only defined as mathematical formulas, but the details of the implementations still have flexibility, such as the precision and the order of internal basic operations. Consequently, with the same input, the output of DNN computation, usually a floating-point vector representing confidence values, is not strictly consistent. For example, we test one Gemm operation with the same inputs on different software/hardware platforms. The results are shown in Table 2. The tests in Table 2(a) run the Gemm operation with different E_N on an i7-8700 processor with 12 logical cores. "Serialization" refers to the straightforward for-loop implementation of matrix multiplication, similar to our E_{BOP}. Others are popular software for DNN computation. We can see that their outputs are different from each other (i.e., with different hash values). More interestingly, we test PyTorch and find that even the same software on different hardware yields different results, as shown in Table 2(b).

(a) Different software		(b) Different hardware	
Software	**Result Hash**	**Hardware**	**Result Hash**
NumPy	068754...	4 vCPU (i7-8700)	fd8858...
ONNX Runtime	c2baad...	8 vCPU (i7-8700)	c53a15...
PyTorch	630fa0...	12 vCPU (i7-8700)	630fa0...
TensorFlow	50bd3a...	16 vCPU (Skylake)	068754...
Serialization	c415be...	16 vCPU (Broadwell)	f71df1...

Table 2: The inconsistency among the results of the same Gemm operation on different platforms

With this issue in mind, we can revisit the pinpointing and arbitration mechanism and understand two major challenges about verification – cross-platform consistency and cross-evaluator consistency. Without cross-platform consistency, the verifiers are unable to tell if a submitted claim is correct by re-executing it. On smart contracts, the equality of two large data objects is checked by comparing their secure-hash values or Merkle tree roots. Equality means an exact match – even one-bit difference will fail the equality test. Even if someone would invent an equality test that could tolerate a degree of imprecision, it would not satisfy the requirement, as we observe that the imprecision of some intermediate operations cannot be bounded. For example, the reciprocal of a small value is very sensitive to its imprecision. Therefore, we must tackle the challenge directly, so that every E_N can make results consistent down to every bit.

Cross-evaluator consistency guarantees the correctness of honest clients in our pinpoint protocol and naturally leads to the cross-platform consistency (details in Section 6). Specifically, for a computational graph, E_N and E_{OP} should output a consistent result; for an operation, E_{OP} and E_{BOP} should output a consistent result; for a basic operation, E_{BOP} and E_C should output a consistent result. Like cross-platform consistency, our goal is to identify and eliminate all sources of inconsistency among the four evaluators.

4 Graph-Based Pinpoint Protocol

As mentioned earlier, pinpoint protocol is the bridge between native evaluator E_N and contract arbitrator E_C. To align E_N and E_C, Agatha does not design the pinpoint protocol based on VM, which is not aligned with the intrinsic nature of DNN computation. Instead, we design a graph-based pinpoint protocol (GPP). GPP includes the following three components and defines how to use them for the pinpoint protocol: (1) Phase-1 evaluator E_{OP} which aligns with E_N and executes in the granularity of operation; (2) Phase-2 evaluator E_{BOP} which aligns with both E_C and E_{OP}, and executes in the granularity of basic operation (BOP), along with the setup tools for E_{BOP}; (3) an enhanced evaluator E_C which can arbitrate the floating-point arithmetic in BOPs.

Next, we begin with explaining the components for E_{BOP}, because E_{BOP} plays a central role in GPP. It not only determines the BOPs that E_C needs to support, but also is the basis of E_{OP}.

4.1 Enabling graph-based pinpoint with E_{BOP}

Circuit generation. Representing a DNN computation as a computation graph can be considered in two levels: the operation (i.e., OP) layer and the basic operation (i.e., BOP) layer, as discussed earlier. The first-level computation graph is already generated explicitly [7], similar to the one in Figure 4. However, we need the ability to generate the second-level computation graph (i.e., circuit). Table 3 shows the BOPs we support. They are in four categories: floating-point arithmetic, integer arithmetic, assignment and typecasting. We have confirmed that every operation in our runtime can be expressed as a circuit with these BOPs.

Categories	Basic Operations
Floating-point Arithmetic	f32_add, f32_sub, f32_mul, f32_div, f32_min, f32_max, f32_sqrt, f32_round, f32_floor
Integer Arithmetic	i32_add, i32_sub, i32_mul
Assignment	= (the operand type can be f32, i32 or u8)
Type Casting	f32_to_u8, u8_to_f32, u8_to_i32, i32_to_f32, f32_to_i32

Table 3: Eighteen basic operations (BOPs) in this work

We implement a rule-based circuit generator for conventional OPs in DNN computation. The generator handles the operations with different shapes and other parameters. We have confirmed that the generated circuits and the operations are *logically* identical, but Section 5 will discuss the inconsistency situations when *concretely* evaluating them. Figure 5 shows a tiny example of the generated circuit for an integer matrix multiplication (i.e., MatMulInteger). Figure 5(a) shows the multiplication, and Figure 5(b) shows the circuit. The circuit is a directed acyclic graph (DAG). The vertices of the graph are BOPs (e.g., i32_add and i32_mul, shown as "+" and "×"). Two special vertices, shown as "in" and "out", are the source and the sink of the graph. The directional wires represent the variables of the computations, including the input variables (i.e., $in_1 \sim in_6$), intermediate variables (i.e., $v_1 \sim v_4$), and the output variables (i.e., out_1, out_2). We also use v_5 and v_6 as the aliases of out_1, out_2, respectively.

Since DNN computation is already expressed as a graph of operations, with the circuit generator, we can represent the whole DNN computation as a giant circuit, which involves billions of BOPs.
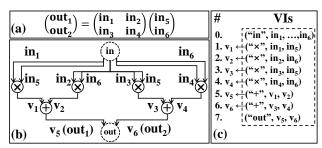

Figure 5: A circuit of matrix multiplication: (a) the formula, (b) the circuit, (c) serialized Vertices with Inputs (VIs).

Circuit serialization. Because the computation graph $G = (V, E)$ is a DAG, a topological sorting (TS) [43] can be used to serialize it, which has a complexity of $O(|V| + |E|)$. Figure 5(c) shows the serialization result of the graph in Figure 5(b). Every item in the sequence, except the first and the last, is denoted by its BOP and the variables on its incoming edges. We call each of these items a *"vertex with inputs"*, or *VI*.

A VI has two forms: the symbolic form and the concrete form. In the symbolic form, referred to as VI^S, every variable is assigned a unique name by the circuit generator, such as in_3 and v_4. For example, ("×", in_1, in_5) can be a VI^S. In the concrete form, referred to as VI^C, every variable is replaced by its actual value. For example, ("×", 1, 5) can be a VI^C.

Therefore, the pinpoint problem can be defined using VI^S and VI^C as follows. *Prerequisite*: the submitter and the verifiers agree on the same VI^S-sequence and all input values. *Problem*: the submitter and a verifier get different VI^C-sequences, so the smart contract needs to determine who is incorrect (note that it does not imply that the other is correct).

Circuit evaluator E_{BOP}. E_{BOP} produces the VI^C-sequence based on the VI^S-sequence and the concrete values of the inputs. It sequentially outputs every VI^C using the corresponding VI^S. Let's assume $(in_1, ..., in_6) = (1, ..., 6)$ in Figure 5(a). First, VI_0^C is initiated as ("in", 1, ..., 6). Then, the evaluations for VI_1^C, ..., VI_6^C are performed, based on VI_1^S, ..., VI_6^S. The results of these VI^Cs are stored. For example, VI_5^C refers to v_1 and v_2, so the evaluator loads the output of VI_1^C (i.e., $v_1 = 5$) and VI_2^C (i.e., $v_2 = 12$), respectively. The process continues until the last VI^C, i.e., $VI_7^C = ($"out", 17, 39$)$ is produced.

Commitment scheme. With the setup work of circuit generation, serialization and the evaluator, when a verifier disagrees about the computation result of the submitter, the pinpoint protocol begins. In this subsection, we assume the whole DNN computation is expressed as a circuit. Then, our pinpoint protocol can be based on a commitment scheme [14] leveraging Merkle trees (MTs) and a smart contract. The protocol allows the submitter and the verifier to locate their leftmost divergent VI^C. The complexity is $O(\log n)$, where n is the length of the VI sequence. The commitment scheme guarantees the non-repudiation of the disputing parties during the process. The protocol includes one preparation step and three interaction steps, as intuitively shown in Figure 6 (formal settings in Section 6) :

- *Prepare.* Before the pinpoint, with the VI^S-sequence as leaves, both the submitter and the verifier generate and reach a consensus on

MT^S. They also agree on the input values (i.e., VI_0^C). For example, the consensus can be denoted as a unanimous signature to the contract on the root of MT^S and VI_0^C. Then, the submitter submits the output of the circuit (e.g., VI_7^C) to the contract.

- *Locate the leftmost divergent VI^C*. The verifier challenges the submitter by invoking the contract API if the verifier disagrees with the output. The submitter is first required to generate MT^C from the VI^C-sequence. Then, according to the on-chain challenges of the verifier, the submitter responds to the contract with a complete Merkle path traversing from the treetop to the leaf. The verifier selects the path based on the local MT^C in order to find the leftmost VI^C divergent from the submitter's. During the process, the contract keeps validating if the Merkle path of the submitter is correct and follows verifier's challenges. Thus, the submitter cannot cheat to regenerate an MT^C while the verifier cannot deny the challenge requests sent to the contract.[5] Suppose that the VI_5^C is located, which means that VI^Cs before the VI_5^C are consistent.[6] By referring to the definition of VI_5^C, the submitter and the verifier dispute either about the concrete input (i.e., the value of v_1 or v_2) or about the BOP (i.e., "+").

- *Upload the MT^S path.* If the disagreement is about the BOP, the submitter is required to upload a Merkle path of VI_5^S to the contract. The contract verifies the path and checks if the BOP in VI_5^S matches the submitter's. If both pass, the verifier is marked "incorrect" by the contract.

- *Upload the MT^C path.* If the disagreement is about the input (e.g., v_2), the submitter is required to upload: (1) an MT^S path of VI_5^S to prove that the output of VI_2^C provides the disputed input; (2) an MT^C path of VI_2^C to prove that submitter's value for v_2 is correct. If the contract validates the submitter's uploaded paths and that the output of VI_2^C is submitter's v_2, the verifier is marked as "incorrect" by the contract. Otherwise, the submitter is marked "incorrect".
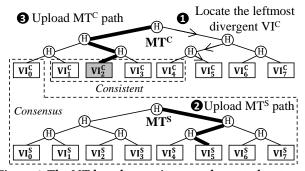

Figure 6: The MT-based commitment scheme to locate and arbitrate the divergent VI.

In the description above, for simplicity, the Merkle trees are shown as binary trees. In fact, they can be k-ary. Choosing the k is a trade-off between the tree generation time, the number of interaction rounds and the gas consumption. In Agatha, we choose $k = 32$. The details will be provided in Section 7.

---

[5]If the submitter aborts the process, the verifier invokes the timeout function of the contract to mark the submitter as "incorrect".

[6]If the submitter and the verifier locate the leftmost divergent VI^C as VI_0^C, our protocol still works since VI_0^C has a unanimous commitment.

Up to this point, we have explained how to pinpoint a disputed BOP in a DNN computation, assuming the protocol can process the entire computation graph. However, this is difficult in reality. We will show in Section 7 that the number of basic operations can be as large as 30 billion, the memory size to include these BOPs is over 600 GB. If we assume the number of OPs in the first-layer is n_1 and each OP has n_2 BOPs, the space complexity would be $O(n_1 \cdot n_2)$ either for generating or for evaluating the graph. To reduce the significant overhead, we propose two-phase pinpoint protocol, which introduces the Phase-1 evaluator.

4.2 Introducing E_{OP} for two-phase pinpoint

To make pinpoint protocol practical, the pinpointing process needs to be done in two phases. Phase-1 identifies the disputed OP with another highly efficient evaluator E_{OP}. As shown in Figure 7(a), the leaves of the Merkle tree for Phase-1 are OPs, so the size of the tree (i.e., P1_MT^C) is much smaller. The procedure in Phase-1 is as what we described above, with two important details to note: (1) the VI represents an OP (e.g., (MatMulInteger, $\vec{in}$)) not a BOP (e.g., "×", in_1, in_5). Variables $\vec{in}$ can be tensors with arbitrary shape and data type. The "BOP" field of VI in the previous section is replaced as one of operations (e.g., Conv) that takes input variables with certain dimensions. In our protocol, this OP is represented by the Merkle root of MT^S of the operation's symbolic circuit (i.e., P2_MT^S), rather than the string "MatMulInteger", so it is a well-defined concrete computation. (2) Different from evaluating a circuit, E_{OP} leverages E_N to evaluate every operation with native performance and computes based on those operation results, which is a feature of current DNN tool-chain [58].
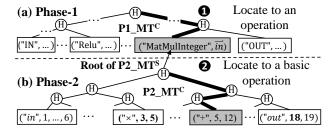

Figure 7: Two-phase pinpointing: (a) Phase-1 (b) Phase-2

Once the disputed operation is pinpointed in Phase-1, the submitter and the verifier start Phase-2, which is within the Merkle tree P2_MT^C in Figure 7(b). Generating P2_MT^C and P2_MT^S is on-demand, so the space complexity of the protocol is reduced to $O(n_1 + n_2)$. In the end, a disputed BOP is submitted to the contract for arbitration.

4.3 Contract arbitrator E_C

We implement the contract arbitrator E_C for all the BOPs in Table 3 via the integer-based emulation, which is fully compliant with the current Ethereum contract standard. In other words, E_C is aligned with E_{BOP} with the granularity of BOP. The implementation carefully complies with IEEE-754 and demonstrates a negligible on-chain overhead for arbitration (see Section 7).

5 Ensuring Consistency

Implementing the Agatha system requires several kinds of consistency as introduced in Section 3.3. In this section, we systematically explain the methods to eliminate all sources of inconsistency, from basic operations to DNN computation, so that we can ensure both cross-evaluator consistency and cross-platform consistency from bottom up.

5.1 Basic operations ($E_C - E_{BOP}$ consistency)

Both E_{BOP} and E_C evaluate basic operations. Among basic operations, integer operations are unambiguous; the main pitfall is due to floating-point operations.

Although the IEEE-754 standard strictly defines floating-point operations, it requires a thorough study to make E_{BOP} and E_C compliant to IEEE-754. For example, we must avoid implementing the minimum and maximum operations of two floating-point numbers by comparison (e.g., using C++ `std::max` in E_{BOP}), which violates the IEEE-754 standard to tackle corner cases such as signed zeros and NaNs. Non-standard mathematical functions, such as x86's `RCP` and `RSQRT`, should never be used.

5.2 Operations ($E_{BOP} - E_{OP}$ consistency)

Both E_{OP} and E_{BOP} evaluate operations. E_{BOP} is based on serialization of basic operations and requires the result consistent with E_{OP}. E_{OP} evaluates an operation by invoking E_N, so it also requires cross-platform consistency. We have conducted a comprehensive study to make Agatha correct and efficient for evaluating operations.

5.2.1 Inconsistent operations

To identify cross-platform and cross-evaluator inconsistent operations, we have systematically reviewed the implementations of operations on the software/hardware platforms listed in Table 2 (covering different software, different processor micro-architectures, and various numbers of logical cores in the same processor), and investigated all the operations defined in ONNX [7] operator set version 7 – a total number of 100. We identify the inconsistent operations by analyzing their two essential sources:

1. Non-associativity of floating-point operations
2. Standard-incompliant floating-point optimizations

Non-associativity. Floating-point operations, as defined in the IEEE-754 standard [1], are not associative [33, 59, 81]. Thus, any computation that consists of multiple floating-point operations must specify the order of the operations, which is expressed as a computation graph, a.k.a. a *circuit*. Otherwise, with different circuit representations, a computation that is consistent in real numbers becomes inconsistent in floating-point numbers.

Unfortunately, an operation only has a mathematical definition, which implies an ambiguous circuit representation. We observe that an operation may be interpreted into different circuit representations on different software/hardware platforms, for performance reasons. For example, Figure 8 illustrates three possible circuits for a ReduceSum operation that aggregates eight `float64` numbers: Circuit (a) shows the straightforward for-loop implementation; (b) shows the implementation using 128-bit SIMD instructions (e.g.

SSE); (c) shows the implementation using 256-bit SIMD instructions (e.g. AVX). Besides the SIMD width, we notice that the circuit may also depend on the number of threads, the cache size, and the summation algorithm.
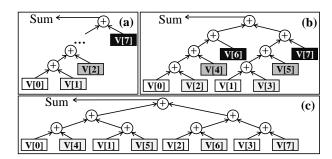

Figure 8: The circuits with different summation orders: (a) sequential, (b) an order observed in an implantation in SSE (128-bit SIMD), (c) an order observed in AVX (256-bit SIMD). Elements of the same sub-figure with the same grey-scale are packed into one register.

Standard-incompliant optimizations. An operation may be inconsistent even if it has an unambiguous circuit representation. We show that the inconsistency comes from three categories of standard-incompliant optimizations.

First, using imprecise implementations. Although the value of a mathematical function on floating-point numbers is well-defined according to the IEEE-754 standard, some software enables approximate algorithms or specific hardware instructions to accelerate the mathematical computation at the cost of precision. For example, the Eigen library, which is used by TensorFlow and ONNX Runtime, is configured with "EIGEN_FAST_MATH=1" by default, so it uses imprecise implementations rather than its standard-compliant implementations.

Second, value-changing transformations. Some software applies distributive laws and substitutes division with reciprocal multiplication in order to accelerate operations (such as BatchNormalization). However, this transforms the operation's existing circuit representation and may change the result [1].

Third, improper compilation. The computation can be non-standard due to improper compilation options. For example, "--ffast-math" option in GCC [9] enables unsafe compilation optimizations and violate the IEEE standard even if the software avoids imprecise implementations or value-changing transformations.

Based on our broad investigation about the operations and the platforms, we summarize all the inconsistent operations in Table 4. Remarkably, inconsistent operations due to non-associativity exist in almost every DNN computation. Gemm and Conv, shown in Figure 4, are two such pervasively used operations. Thus, it is important to handle the inconsistency issue caused by floating-point non-associativity.

5.2.2 Addressing non-associativity for consistent operations

Disabling optimizations incompliant with standard makes many operations consistent. The other inconsistent operations are due to the non-associativity of floating-point operations. These operations are usually the most computationally intensive part in a DNN, so we

Essential source	Operations
Non-associativity	AveragePool, Conv, ConvTranspose, Gemm, GlobalAveragePool,GlobalLpPool, GRU, LogSoftmax, LpNormalization, LpPool, LRN, LSTM, MatMul, Mean, ReduceL1, ReduceL2, ReduceLogSum, ReduceLogSumExp, ReduceMean, ReduceProd, ReduceSum, ReduceSumSquare, RNN, Softmax, Sum
Standard-incompliant optimizations	***Square root***: BatchNormalization, GlobalLpPool, InstanceNormalization, LpNormalization, LpPool, ReduceL2, Sqrt; ***Max/Min***: Clip, GlobalMaxPool, HardSigmoid, Max, MaxPool, MaxRoiPool, Min, ReduceMax, ReduceMin, Relu; ***Transformation***: BatchNormalization, InstanceNormalization

Table 4: Inconsistent operations in ONNX Opset Version 7

	Precision	Performance	Usability
Sequencing	★★	★	★★
Customized sequencing	★★	★★	★
ReproBLAS	★★★	★	★
Fixed-point arithmetic	★	★★★	★★★

Table 5: Different methods to tackle floating-point non-associativity

must consider performance factors when we analyze the following ways to to tackle these operations.

The most straightforward method is *sequencing*, i.e., assigning a particular circuit representation to an operation by applying the left associativity for every basic operation. With the unambiguous circuit, E_{OP} and E_{BOP} have the same behavior and naturally output the same result. However, sequencing loses the benefit of parallel execution and hardware SIMD instructions.

A variant of sequencing, *customized sequencing*, stipulates that the order of basic operations is the order running on a specific hardware platform, for both E_{OP} and E_{BOP}. In this way, on the platform we can leverage the benefit of parallelism. For example, we can specify the order of summation for a ReduceSum operation to comply with an eight-core AVX-enabled CPU, like Figure 8(c). However, to guarantee cross-platform consistency, the computation on other hardware platforms must comply with the customized circuit. We consider this a difficult task for many reasons, such as the size of the circuit, participant's consensus and software implementation.

Another possibility is using data types that have the associative property. For example, ReproBLAS [20] is a linear algebra library in which the floating-point numbers are represented by "binned types" and the result of summation is reproducible and independent of the order of summation. ReproBLAS has the potential to eliminate the inconsistency from non-associativity. However, it is currently incomplete to support DNN operations and multi-thread execution, and too complicated for on-chain arbitration.

Fixed-point arithmetic is a simple and efficient solution. With the associative property of fixed-point arithmetic, we can customize the optimal computation order on different platforms while achieving consistency. E_{OP} can perform parallel computing, so the performance improvement is significant compared to sequencing. Another advantage is that many existing libraries and hardware platforms support high-performance fixed-point arithmetic.

The comparison of the above methods is shown in Table 5. We adopt both sequencing and fixed-point arithmetic for Agatha. Specifically, in the context of DNN computation, using fixed-point

arithmetic (or integer arithmetic[7]) is called "quantization", which has only an insignificant loss on the DNN model's accuracy [42, 73]. In practice, if someone wants the insignificant accuracy loss even smaller (e.g., from 1% to 0.1%), one can use partial quantization. For example, most matrix multiplication-based operations (such as Gemm, Conv, RNN, LSTM) are quantized, but others still take floating-point numbers [58]. Considering this accuracy-performance tradeoff, Agatha adopts fixed-point arithmetic for quantization operation and sequences others, faithfully following the DNN model description.

5.3 Graphs ($E_{OP} - E_N$ consistency)

Both E_N and E_{OP} evaluate the whole computational graph of a DNN. We have ensured that the operations are consistent, so it is easy to ensure the whole graph's consistency. The main pitfall here is the graph optimization in E_N. Some software's default graph optimization level (such as ONNX Runtime's) makes graph optimization so deep that the graph transformation changes the floating-point values. Setting a safer level resolves the issue.

6 Security Analysis

In this section, we provide the security analysis of our Agatha system with formal definitions and proofs.

6.1 Definitions

The security definitions in this paper are very similar to those of Arbitrum [44]. The main difference is that we introduce the native DNN computation in the AnyTrust assumption and use GPP instead of VM-based pinpoint protocol. Besides, we also introduce the definitions for the evaluators used in Agatha system:

AnyTrust assumption. There are a quorum of off-chain nodes $\{N_1, \ldots, N_m\}$ with their corresponding platforms $\{P_1, \ldots, P_m\}$ for DNN computation, smart contract C as the arbitrator, a native DNN computation ϕ^n, native evaluator E_N, input data $\mathcal{D}$, and a graph-based pinpoint protocol Π. We use $N_s \in \{N_1, \ldots, N_m\}$ to denote the submitter. AnyTrust assumption holds if for every claim of N_s that "$\mathcal{R}_s = E_N(P_s, \mathcal{D}, \phi^n)$" sent to other nodes in $\{N_1, \ldots, N_m\}$ with corresponding commitment to contract C, either N_s *honestly participates* in Π or there exists a verifier $N_v \in \{N_1, \ldots, N_m\}$ ($v \neq s$) to correctly compute $\mathcal{R}_v = E_N(P_v, \mathcal{D}, \phi^n)$ and to *honestly participate* in Π if $\mathcal{R}_v \neq \mathcal{R}_s$.

Evaluators in Agatha system. There are four evaluators (E_N, E_{OP}, E_{BOP} and E_C):

- *For native execution, E_N* is the evaluator for native DNN computation ϕ^n as the mapping: $P_* \times \mathcal{D} \times \phi^n \mapsto \mathcal{R}_*$, where P_* denotes the platform to run E_N.
- *For Phase-1 pinpoint protocol, ϕ^n* is conventionally expressed as a sequence of operations (OPs): $\left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\}$. ϕ_i^{op} is executed by E_{OP} as the mapping: $P_* \times \mathcal{D}_i^{op} \times \phi_i^{op} \mapsto \mathcal{R}_i^{op}$, where $\mathcal{D}_i^{op}$ and $\mathcal{R}_i^{op}$ denote the input and output of ϕ_i^{op}, respectively. D_i^{op} comes from either input $\mathcal{D}$ or previous output $\mathcal{R}_j^{op}$ ($j < i$). E_{OP} can also calculate the intermediate result for OPs: $P_* \times \mathcal{D} \times \left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_i^{op}\right\} \mapsto \mathcal{R}_i^{op}$.

---

[7]Fixed-point arithmetic is essentially based on integer arithmetic.

- For Phase-2 pinpoint protocol, ϕ_i^{op} can be further serialized as a sequence of basic operations (BOPs): $\left\{\phi_{i,1}^{bop}, \phi_{i,2}^{bop}, \ldots, \phi_{i,l_i}^{bop}\right\}$ (e.g., by topological sorting). E_{BOP} is the mapping: $P_* \times \mathcal{D}_{i,j}^{bop} \times \phi_{i,j}^{bop} \mapsto \mathcal{R}_{i,j}^{bop}$, where $\mathcal{D}_{i,j}^{bop}$ and $\mathcal{R}_{i,j}^{op}$ denote the input and output of $\phi_{i,j}^{bop}$, respectively. Similar to E_{OP}, E_{BOP} can also evaluate the intermediate result: $P_* \times \mathcal{D}_i^{op} \times \left\{\phi_{i,1}^{bop}, \phi_{i,2}^{bop}, \ldots, \phi_{i,j}^{bop}\right\} \mapsto \mathcal{R}_{i,j}^{bop}$.

- For contract arbitration, E_C evaluate contract arbitration ϕ^c as the mapping: $P_*^E \times \mathcal{D}^{bop} \times \phi^c \mapsto \mathcal{R}^{bop}$, where P_*^E denotes the platform to run Ethereum client while $\mathcal{D}^{bop}$ and $\mathcal{R}^{bop}$ denote the input and output for ϕ^c, respectively. Obviously, E_C is cross-platform consistent, namely $E_C(P_*^E, \mathcal{D}^{bop}, \phi^c) = E_C(P_*'^E, \mathcal{D}^{bop}, \phi^c)$ for arbitrary platforms P_*^E and $P_*'^E$ with Ethereum clients.

Graph-based pinpoint protocol (GPP) Π. Π is encoded in contract C and triggered by the claim of submitter N_s. Π provides two exclusive APIs $\left\{\pi^S, \pi^V\right\}$ for the challenge of verifier N_v ($v \neq s$) and the response of submitter N_s, respectively. Besides the Merkle tree verification logic, four parameters are pre-determined: (1) a timeout period $T^v = O(|\phi^n|)$ which is proportional to the latency of $E_N(P_v, \mathcal{D}, \phi^n)$) and specifies the maximum interval for the first verifier to invoke π^V; (2) a timeout period $T^{op} = O(|\phi^{op}|)$ which is proportional to the latency of $E_{OP}(P_v, \mathcal{D}, \left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\})$ for Phase-1 protocol and specifies the maximum interval between the verifier-submitter interactions (via π^S and π^V) during the pinpoint protocol; (3) $T^{bop} = O(|\phi^{bop}|)$ for evaluating Phase-2 protocol which is similar to T^{op}; (4) the termination conditions (either for the aborting case or the non-aborting case) for the pinpoint protocol, which includes the contract arbitrator E_C for ϕ^c (i.e., BOP).

N_s and N_v are considered *honestly participating* in Π if N_s and N_v always correctly interact with C within the time-out periods (i.e., T^v, T^{op} and T^{bop}) according to contract states and their evaluation results of ϕ^n, ϕ^{op}, ϕ^{bop} with E_N, E_{OP}, E_{BOP}, respectively. Once π^V is invoked, we assume that the count down of T^v is suspended until termination. Verifier N_v *fails in the pinpoint protocol* if either the time-out is reached or contract arbitration indicates N_s is right. Failed verifier can no longer invoke π^V for this claim. If all verifiers fail or there is no challenge within T^v, the contract *accepts* N_s's claim otherwise *rejects* the claim.

Following ACE [82], the core security properties of Agatha system is the correctness and the liveness:

Correctness of Agatha. Agatha's correctness is twofold: (1) The contract will not accept N_s's wrong claim given the AnyTrust assumption and our design of GPP and XCE; (2) N_s's correct claim will not be challenged by honest verifiers and will be accepted by the contract given malicious verifiers.

Liveness of Agatha. N_s's claim will be either accepted or rejected by the contract within a maximum period T_{max} even if N_s and $m-1$ verifiers maliciously invoke π^S and π^V, respectively.

XCE plays the key role to guarantee the correctness of GPP and the whole Agatha system. In particular, XCE guarantees the cross-evaluator consistency between E_N - E_{OP}, E_{OP} - E_{BOP} and E_{BOP} - E_C, which can further ensure the cross-platform consistency:

Cross-evaluator consistency of XCE. The consistency between three pairs of evaluators (i.e., E_N - E_{OP}, E_{OP} - E_{BOP} and E_{BOP} - E_C) are defined as follows:

- E_N - E_{OP}, which means $\forall \mathcal{D}$ and $\forall P_* \in \{P_1, \ldots, P_m\}$, the output of E_{OP} for the OP sequence is the same with E_N's output $\mathcal{R}_*$: $E_{OP}(P_*, \mathcal{D}, \left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\}) = E_N(P_*, \mathcal{D}, \phi^n)$.

- E_{OP} - E_{BOP}, which means $\forall \mathcal{D}_i^{op}$ and $\forall P_* \in \{P_1, \ldots, P_m\}$, the output of E_{BOP} for the BOP sequence and E_{OP}'s output are same: $E_{BOP}(P_*, \mathcal{D}_i^{op}, \left\{\phi_{i,1}^{bop}, \phi_{i,2}^{bop}, \ldots, \phi_{i,l_i}^{bop}\right\}) = E_{OP}(P_*, \mathcal{D}_i^{op}, \phi_i^{op})$.

- E_{BOP} - E_C, which means $\forall \mathcal{D}_{i,j}^{bop}$, $\forall P_* \in \{P_1, \ldots, P_m\}$ and $\forall P_*^E$ running Ethereum, the output of E_{BOP} for $\mathcal{D}_{i,j}^{bop}$ is the same with E_C's output: $E_{BOP}(P_*, \mathcal{D}_{i,j}^{bop}, \phi_{i,j}^{bop}) = E_C(P_*^E, \mathcal{D}_{i,j}^{bop}, \phi^c)$.

Cross-platform consistency. $\forall \mathcal{D}$ and $\forall P_i, P_j \in \{P_1, \ldots, P_m\}$, their outputs of E_N, E_{OP}, E_{BOP} are equal: (1) $E_N(P_i, \mathcal{D}, \phi^n) = E_N(P_j, \mathcal{D}, \phi^n)$; (2) $E_{OP}(P_i, \mathcal{D}, \left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\}) = E_{OP}(P_j, \mathcal{D}, \left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\})$; (3) $E_{BOP}(P_i, \mathcal{D}_i^{op}, \left\{\phi_{i,1}^{bop}, \phi_{i,2}^{bop}, \ldots, \phi_{i,l_i}^{bop}\right\}) = E_{BOP}(P_j, \mathcal{D}_i^{op}, \left\{\phi_{i,1}^{bop}, \phi_{i,2}^{bop}, \ldots, \phi_{i,l_i}^{bop}\right\})$.

6.2 Security proof

Lemma 6.1. *XCE guarantees the cross-platform consistency.*

PROOF. $\forall \mathcal{D}$ and $\forall P_i, P_j \in \{P_1, \ldots, P_m\}$, with the cross-platform consistency of E_C and the cross-evaluator consistency of E_C-E_{BOP}, we can get $E_{BOP}(P_i, \mathcal{D}^{bop}, \phi^{bop}) = E_{BOP}(P_j, \mathcal{D}^{bop}, \phi^{bop})$, namely E_{BOP} is cross-platform consistent. With the E_{OP} - E_{BOP} and E_{OP} - E_N consistency, we can further conclude $E_{OP}(P_i, \mathcal{D}_i^{op}, \phi_i^{op}) = E_{OP}(P_j, \mathcal{D}_i^{op}, \phi_i^{op})$ and $E_N(P_i, \mathcal{D}, \phi^n) = E_N(P_j, \mathcal{D}, \phi^n)$. □

Theorem 6.2 (The Correctness of Agatha). *The claim of N_s is accepted by contract C without challenges from honest verifiers if and only if $\mathcal{R}_s = E_N(P_*, \mathcal{D}, \phi^n)$.*

PROOF. If $\mathcal{R}_s = E_N(P_*, \mathcal{D}, \phi^n)$, according to Lemma 6.1 and Anytrust assumption, the honest verifiers will not challenge the claim since $\mathcal{R}_v = \mathcal{R}_s$.

If $\mathcal{R}_s \neq E_N(P_*, \mathcal{D}, \phi^n)$, we can prove that GPP will pinpoint to a ϕ^{bop} (unless N_s aborts midway), where $\mathcal{R}_s^{bop} \neq E_{BOP}(P_*, \mathcal{D}^{bop}, \phi^{bop})$. This inequality will be arbitrated by E_C and lead to the rejection of N_s's claim. If we assume that there is no such ϕ^{bop}, according to the E_{OP} - E_{BOP} and E_{OP} - E_N consistency, we can have $E_N(P_s, \mathcal{D}, \phi^n) = E_{BOP}(P_*, \mathcal{D}_i^{op}, \left\{\phi_{i,j}^{bop}\right\})$, where $\left\{\phi_{i,j}^{bop}\right\}$ denotes the BOP sequence for $\left\{\phi_1^{op}, \phi_2^{op}, \ldots, \phi_k^{op}\right\}$. Because $\mathcal{R}_s^{bop} = E_{BOP}(P_*, \mathcal{D}^{bop}, \phi^{bop})$ for every $\phi_{i,j}^{bop}$, we have $E_N(P_s, D, \phi^n) \triangleq \mathcal{R}_s = E_N(P_*, \mathcal{D}, \phi^n)$, contradicting to $\mathcal{R}_s \neq E_N(P_*, \mathcal{D}, \phi^n)$.

Similarly, we can also get that if $\mathcal{R}_s = E_N(P_*, \mathcal{D}, \phi^n)$, GPP will find $\mathcal{R}_v^{bop} \neq E_{BOP}(P_*, \mathcal{D}^{bop}, \phi^{bop})$ for the malicious verifiers' challenge. Therefore, all malicious verifiers will fail. □

Theorem 6.3 (The Liveness of Agatha). *Within maximum period $T_{max} = T^v + m \times (d(|\phi^{op}|) \times T^{op} + d(|\phi^{bop}|) \times T^{bop})$, where*

$d(|\phi^*|)$ is the depth of the k-ary Merkle tree representing ϕ^*, the claim of N_s will be either accepted or rejected.

PROOF. If N_s's claim is true and verifiers are all honest, the claim will be accepted within T^v. The worst case is that m-1 verifiers maliciously invoke π^V at T^v. For every challenge from malicious verifiers, the process takes at most $d(|\phi^{op}|) \times T^{op} + d(|\phi^{bop}|) \times T^{bop}$, which corresponds to $T_{max} = T^v + (m-1) \times (d(|\phi^{op}|) \times T^{op} + d(|\phi^{bop}|) \times T^{bop})$.

If N_s's claim is false, normally it will be rejected with honest verifier's challenge in $d(|\phi^{op}|) \times T^{op} + d(|\phi^{bop}|) \times T^{bop}$. There exists a worst case that the honest verifier and other malicious verifiers invoke π^V at T^v, which corresponds to the rejection of N_s's claim in $T_{max} = T^v + m \times (d(|\phi^{op}|) \times T^{op} + d(|\phi^{bop}|) \times T^{bop})$. □

7 Evaluations

In this section, we provide the details of our implementation and experiment setup. Then, we show the evaluations of Agatha's correctness and performance.

7.1 Implementation and experiment setup

Implementation. As shown in Figure 3, we implement the Agatha system with five steps: (1) With methods in Section 5, we implement XCE based on the ONNX Runtime [58], targeting three models (i.e., MobileNet, ResNet50, VGG16) and use XCE-enabled native evaluator E_N. (2) We integrate the rule-based generating and the topological serialization into our circuit generator with ~1700 lines of C++ code. We speed up the generator by reusing duplicate circuits, given the recurrence of operations. (3) We implement the logic of Merkle tree, the interface to the contract, Phase-1 and Phase-2 evaluators E_{OP} and E_{BOP}, and a command-line interface in the Agatha client, which can connect to Ethereum and run the pinpoint protocol automatically for the submitter and the verifier. We implement the Agatha client with ~4000 lines of Python code (testing logic included), which leverages the ONNX library [7] and Brownie framework [2]. The module for Merkle tree generating is highly optimized in C++ leveraging the XKCP library [3], which provides the Keccak-256 hashing algorithm. The leaves of the Merkle tree are fixed-length to enable packing before hashing. To balance the Merkle tree, we move the VI^C and VI^S of out/in to be under the Merkle root. We further compress the field of inputs in each VI with a Merkle tree structure. (4) We implement the Agatha contract in Solidity 0.7.1 [10] with ~500 lines of code, which specifies the logic for submitter-verifier interactions, Merkle path verification and the arbitration of BOP. The gas consumption is optimized in several ways, such as leveraging gas refunds [74] and using inline assembly in Solidity. (5) We implement the BOP contract library with ~1400 lines of Solidity code, in which the 32-bit floating-point arithmetic is adapted from the ABDK library [12].

Setup. For the off-chain nodes, we use Azure machines with 24 vCPUs (E5-2690 v3), which is the configuration for Section 7.3. We also use various machines of Table 2b to test cross-evaluator consistency defined in Section 6. For the nodes of Ethereum, we use Ganache [40] to simulate the public Ethereum (Istanbul version [28]), which reports the same gas consumption. And for benchmarks, we use the standard test images from ImageNet [21] and ONNX-format files

[6] of three quantized models (i.e., MobileNet, ResNet50, VGG16) following [77]. For the Merkle tree configuration, we set the branch size as 32 by default since it is optimal for the gas consumption (detailed in Figure 11).

7.2 Validation about consistency

As shown in Section 6, the correctness of Agatha system relies on the cross-evaluator consistency which are introduced in Section 5 and defined in Section 6. Besides our careful design and investigation, we also have conducted comprehensive tests to evaluate the cross-evaluator consistency. All cases are tested on the different hardware of Table 2(b): (1) For E_{BOP}-E_C consistency, we use random inputs and ~7500 floating-point corner cases as the unit tests for the E_{OP}-E_{BOP} consistency. The subnormal numbers, infinities, signed zeros and NaNs are all included to show consistency; (2) For E_{OP}-E_{BOP} consistency, we test more than 20 selected corner cases and more than 1000 random cases. All tests demonstrate consistency; (3) For E_N-E_{OP} consistency, the E_{OP} and E_N demonstrate consistent output hashes with all test images from ImageNet [21].

7.3 Performance evaluation

We first profile the three models used in this work. As shown in Table 6, we provide the storage size, the number of OPs and the number of BOPs in each model, respectively. We see that the model can be as large as 133 MB with 30 billion BOPs. In addition, the recurrence of OPs is common, so our deduplication (i.e., circuit reuse) can reduce the latency of the circuit generation substantially. Due to the page limit, we provide details of OPs (e.g., OP size distribution, circuit generation) in Appendix A.

Model	Storage	Before dedup		After dedup	
		#OP	#BOP	#OP	#BOP
MobileNet	3.8 MB	371	9.9×10^8	131	5.5×10^8
ResNet50	25 MB	387	8.1×10^9	97	3.9×10^9
VGG16	133 MB	186	3.0×10^{10}	75	2.2×10^{10}

Table 6: The number of OPs and BOPs in each model

The performance of Agatha evaluators. Figure 9 shows the latency of the native evaluator E_N for running each of the three models end-to-end, which is the average result of 10,000 tests. The latency of ours includes the inference time and the time to generate the output hash. Compared with the original evaluator (i.e., ONNX runtime), E_N introduces latency overheads as 6.6%, 1.4% and 1.3% for MobileNet, ResNet50 and VGG16 (the geometric mean is 3.0%), respectively. The overhead indicates the power of our design and the limited impact of turning off "--ffast-math" option.

We also test and estimate the latency for ideal VM and restricted VM, which denote the single-threaded native execution (the upper-bound performance of VM-based approaches) and the restricted VM of previous work, respectively. And for simplicity, we use Arbitrum VM as an example of the restricted VM which is based on EVM and actively being developed. Since EVM does not support floating-point arithmetic and has a gas limit, we use integer arithmetic for approximation and measure the speed of different instructions [63] to estimate EVM's lower-bound latency. We can see that E_N demonstrates an average speedup of 2.59× and 602× compared to ideal VM and restricted VM, respectively.
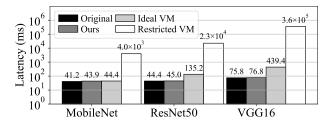
Figure 9: Inference latency of the original (i.e., baseline), our native evaluator, the ideal VM and the restricted VM

Figure 10 shows the local execution latency of Phase-1 and Phase-2 evaluators E_{OP} and E_{BOP}, respectively. We can see that the latency of E_{OP} is relatively small since we leverage the native execution of ONNX Runtime for evaluating every operation. To evaluate the E_{BOP} latency, for simplicity, we only showcase the largest operation in each model (e.g., ConvInteger in VGG16) which represents the worst-case latency. The worst-case Phase-2 latency depends on the operation size and E_{BOP} performance, which can be smaller or larger than Phase-1's. In addition, we also show the latency for one-phase pinpoint protocol where the whole DNN computation is expressed and evaluated in BOPs. We can find that the two-phase pinpoint greatly reduces the latency of the pinpoint protocol.
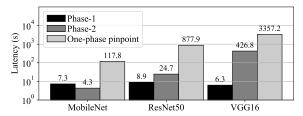

Figure 10: The Phase-1 and Phase-2 latency vs. latency of One-phase pinpoint

Gas consumptions. In Figure 11(a), we show the gas consumptions of the main functions in our contract. Since we have applied the inline mechanism, the gas consumptions are calculated by differential analysis. For the setup, the Agatha contract and the BOP library are deployed to Ethereum with $\sim 4.8 \times 10^6$ gas, which is one-time. For the normal case, the Submit (i.e., submitter's on-chain commitment) costs ~63,000 gas which corresponds to three ETH transfer transactions. For the pinpointing cost, we see that the gas consumed by BOPs is negligible compared to those of other functions. The maximal gas consumption for the whole pinpointing (i.e., Max total of VGG16) is about ~86 ETH transfer transactions.

Merkle tree branches. We also study the impact of the number of the Merkle tree branches on the performance, as shown in Figure 11(b). Due to the space constraint, we only show the case of the MobileNet, which is similar to those of ResNet50 and VGG16. When the branch size varies from 2 to 64, we see a decrease in the number of interaction rounds and the Merkle tree generating time, because the depth of Merkle tree decreases. Interestingly, the gas consumption for pinpointing is minimal when the branch size is 32, which is caused by two effects: (1) The number of rounds decreases with a larger branch size, which reduces the total gas of Challenge and Response in Figure 11(a). (2) The increase of branch size leads
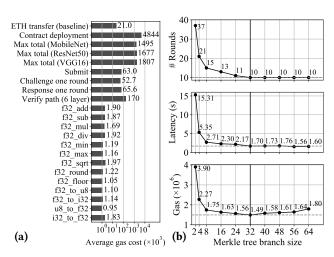

Figure 11: (a) The breakdown of gas consumption (b) The optimal choice of Merkle tree branch size for MobileNet

to more hashing workloads, which introduces the high overhead of Verify-Path in Figure 11(a).

8 Related Work

AI computation on blockchain. The vision of AI computation on blockchain is a subject mentioned by researchers and companies. In academia, Das *et al.* mention machine learning as a potential area for "Computationally Intensive Smart Contracts" [19]. And Teutsch *et al.* regard "Autonomous machine learning" as a promising application for TrueBit [76] while Wüst *et al.* figure out that machine learning is infeasible for current smart contracts [82]. Other researchers discuss the potential of enabling machine learning on a blockchain for smart home [70] and cooperative data-driven applications [56]. In industry, Microsoft proposes "Decentralized & Collaborative AI" on blockchains [36]. Several startups, including Cortex [18], SingularityNET [32], Algorithmia [46] and Oraclize [39], share this vision. However, the existing proposals either demonstrate tiny AI computations that achieve the consensus on a handful of nodes [18, 32, 36, 39, 46, 56, 70] or provide no details [19, 76, 82]. They are not scalable either for both on-chain or off-chain execution. It is impractical for Ethereum smart contract to use these approaches to run a real-world DNN computation.

Scalable smart contract. Due to the low scalability of blockchain [23, 53], there are currently two mainstream approaches to offload the computation from smart contract to off-chain nodes: either via the *validity proof* or via the *fraud proof*. The first approach relies on verifiable computation (VC) or Zero-knowledge Proof (ZKP) with representative projects as zkSync [47] and Aztec [51], which proves the integrity of computations with the proof about validity. And the contract only needs to verify the proof. However, current VC and ZKP use expensive cryptographic primitives [13, 15, 30, 68], which are currently impractical to support the complicated computations like DNN computations. For example, SafetyNets [31] demonstrates to interactively prove the AI computation of only a four-layer CNN. The state-of-the-art work, Slalom [77], has to additionally introduce

a trusted execution environment (TEE) for the DNN computation of MobileNet, ResNet50 and VGG16.

Therefore, the community also develops the second approach, i.e., the proof about fraud, for its potential to support large-scale applications. Representative technologies are Plasma [64], TrueBit[76], Arbitrum [44], YODA [19], ACE [82] and Optimism [62]. These technologies introduce a quorum of off-chain nodes with weak assumptions (e.g., AnyTrust), and benefit from the power of pinpoint protocol and the potential reuse of contract checkers [45, 52, 78]. This approach is a promising candidate solution to be a part of Ethereum 2.0 (i.e., the Optimistic Rollup [8]) and will be applied in Uniswap v3 [4]. It also inspires other applications such as the efficient 2PC protocol [84].

Referred delegation. Some early work investigates the situation where the arbitrator is not a smart contract [17, 26, 79]. In these work, two or more parties (with at least one honest party) are assumed to execute the same program. Then, the parties can challenge each other with a pinpoint style to locate the disputable step. The step is sent to a judge with weak computation power for arbitration. These technologies do not belong to scalable smart contract, but do share the pinpoint aspect, so are related to our work.

9 Conclusions

We introduce the Agatha system to make decentralized smart contracts capable of DNN computation, which is a goal envisioned by the decentralized computing community. Agatha follows the state-of-the-art verification paradigm that is based on pinpointing and arbitration. The existing technologies rely on VM execution, not supporting native execution, which would make DNN computation too slow to be practical. To enable native execution, we develop Graph-based Pinpoint Protocol (GPP) and Cross-evaluator Consistent Execution (XCE). Agatha is evaluated using popular DNN models, e.g., MobileNet, ResNet50 and VGG16. Compared to the baseline, Agatha's average overhead of off-chain execution is only 3.0%. By contrast, the slowdown would be at least 620× for the current VM-based technologies, assuming that these technologies could be built to handle industry-scale DNN models.

References

[1] 2019. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), 1–84. https://doi.org/10.1109/IEEESTD.2019.8766229

[2] 2021. Brownie: a Python-based development and testing framework for smart contracts targeting the Ethereum Virtual Machine. https://github.com/eth-brownie/brownie/.

[3] 2021. The eXtended Keccak Code Package. https://github.com/XKCP/XKCP.

[4] 2021. Introducing Uniswap V3. https://uniswap.org/blog/uniswap-v3/.

[5] 2021. La'Zooz. http://lazooz.org/.

[6] 2021. ONNX Model Zoo. https://github.com/onnx/models.

[7] 2021. Open Neural Network Exchange. https://github.com/onnx/onnx.

[8] 2021. Optimistic-Rollups for Ethereum. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/.

[9] 2021. *Semantics of Floating Point Math in GCC.* https://gcc.gnu.org/wiki/FloatingPointMath

[10] 2021. The Solidity Contract-Oriented Programming Language. https://github.com/ethereum/solidity.

[11] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16).* 265–283.

[12] ABDK. 2021. ABDK Libraries for Solidity. https://github.com/abdk-consulting/abdk-libraries-solidity.

[13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct non-interactive zero knowledge for a von Neumann architecture. In *Proceedings of the 23rd USENIX conference on Security Symposium.* 781–796.

[14] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences* 37, 2 (1988), 156–189.

[15] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP).* IEEE, 315–334.

[16] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 229–243.

[17] Ran Canetti, Ben Riva, and Guy N Rothblum. 2011. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM conference on Computer and communications security.* 445–454.

[18] Z Chen, W Wang, X Yan, and J Tian. 2018. Cortex-AI on blockchain: The decentralized AI autonomous system. *Cortex White Paper* (2018).

[19] Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. 2019. YODA: Enabling computationally intensive contracts on blockchains with Byzantine and Selfish nodes. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019.*

[20] James Demmel, Peter Ahrens, and Hong Diep Nguyen. 2016. *Efficient Reproducible Floating Point Summation and BLAS.* Technical Report UCB/EECS-2016-121. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-121.html

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 248–255.

[22] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 967–984.

[23] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 949–966.

[24] Jacob Eberhardt and Stefan Tai. 2018. ZoKrates-Scalable Privacy-Preserving Off-Chain Computations. In *IEEE International Conference on Blockchain. IEEE.*

[25] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. 2020. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security.* 543–557.

[26] Uriel Feige and Joe Kilian. 1997. Making games short. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* 506–516.

[27] Ethereum Foundation. 2021. Ethereum. https://www.ethereum.org/.

[28] Ethereum Foundation. 2021. The Ethereum Istanbul hard fork. https://eth.wiki/en/roadmap/istanbul.

[29] Ethereum Foundation. 2021. Ethereum Virtual Machine (EVM). https://ethereum.org/en/developers/docs/evm/.

[30] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* 2019 (2019), 953.

[31] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems* 30 (2017), 4672–4681.

[32] Ben Goertzel, Simone Giacomelli, David Hanson, Cassio Pennachin, and Marco Argentieri. 2017. SingularityNET: A decentralized, open market and inter-network for AIs. *Thoughts, Theories & Studies on Artificial Intelligence (AI). Research* (2017).

[33] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)* 23, 1 (1991), 5–48.

[34] W3C Community Group. 2021. WebAssembly. https://webassembly.org/.

[35] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, Jim Guilford, and Gil Wolrich. 2013. Intel sha extensions–new instructions supporting the secure hash algorithm on intel architecture processor. *Intel White Paper* (2013).

[36] Justin D Harris and Bo Waggoner. 2019. Decentralized and collaborative ai on blockchain. In *2019 IEEE International Conference on Blockchain (Blockchain).* IEEE, 368–375.

[37] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J Knottenbelt. 2019. Balance: Dynamic adjustment of cryptocurrency deposits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 1485–1502.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision.* Springer, 630–645.

[39] Joshua Herman. 2021. Make your Sklearn models accessible to Smart Contracts using Oracalize. https://medium.com/zitterbewegung/4d9593c80383.

[40] ConsenSys Software Inc. 2021. One Click Blockchain. https://www.trufflesuite.com/ganache.

[41] Dapper Labs Inc. 2018. CryptoKitties: Collect and breed digital cats! https://www.cryptokitties.co/.

[42] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[43] Arthur B Kahn. 1962. Topological sorting of large networks. *Commun. ACM* 5, 11 (1962), 558–562.

[44] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, 1353–1370.

[45] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. 2018. ZEUS: Analyzing Safety of Smart Contracts. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*.

[46] A Besir Kurtulmus and Kenny Daniel. 2018. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. *arXiv preprint arXiv:1802.10185* (2018).

[47] Matter Labs. 2021. zkSync. https://zksync.io/.

[48] Protocol Labs. 2021. IPFS. https://ipfs.io/.

[49] Uniswap Labs. 2021. Uniswap:Decentralized Trading Protocol. https://uniswap.org/.

[50] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. 2017. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, Vol. 34. 1–5.

[51] Spilsbury Holdings Ltd. 2020. Aztec. https://aztec.network/.

[52] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 254–269.

[53] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 455–471.

[54] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. *IACR Cryptol. ePrint Arch.* 2018 (2018), 236.

[55] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. 2017. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*. Springer, 357–375.

[56] Gihan J Mendis, Moein Sabounchi, Jin Wei, and Rigoberto Roche. 2018. Blockchain as a service: an autonomous, privacy preserving, decentralized architecture for deep learning. *arXiv preprint arXiv:1807.02515* (2018).

[57] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.

[58] Microsoft. 2021. ONNX Runtime. https://github.com/microsoft/onnxruntime.

[59] David Monniaux. 2008. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 30, 3 (2008), 1–41.

[60] California Secretary of State. 2021. Signature Verification, Ballot Processing, and Ballot Counting (Emergency Regulations). https://www.sos.ca.gov/administration/regulations/current-regulations/elections/signature-verification-ballot-processing-and-ballot-counting-emergency-regulations.

[61] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[62] Optimism PBC. 2021. Optimism. https://optimism.io/.

[63] Daniel Perez and Benjamin Livshits. 2020. Broken Metre: Attacking Resource Metering in EVM. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society.

[64] Joseph Poon and Vitalik Buterin. 2017. Plasma: Scalable autonomous smart contracts. *White paper* (2017), 1–47.

[65] Jesus Rodriguez. 2021. When DeFi Becomes Intelligent. https://www.coindesk.com/when-defi-becomes-intelligent.

[66] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[67] Mohamed Seifelnasr, Hisham S Galal, and Amr M Youssef. 2020. Scalable open-vote network on ethereum. In *International Conference on Financial Cryptography and Data Security*. Springer, 436–450.

[68] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.

[69] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015*.

[70] Kushal Singla, Joy Bose, and Sharvil Katariya. 2018. Machine learning for secure device personalization using blockchain. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 67–73.

[71] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market Making via Reinforcement Learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 434–442.

[72] Thomas Spooner and Rahul Savani. 2020. Robust Market Making via Adversarial Reinforcement Learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 4590–4596. https://doi.org/10.24963/ijcai.2020/633 Special Track on AI in FinTech.

[73] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[74] Wei Tang. July 2019. EIP-2200: Structured Definitions for Net Gas Metering. *Ethereum Improvement Proposals* no. 2200 (July 2019). [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-2200.

[75] Don Tapscott and Alex Tapscott. 2016. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin.

[76] Jason Teutsch and Christian Reitwießner. 2019. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756* (2019).

[77] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *International Conference on Learning Representations*.

[78] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 67–82.

[79] Jelle van den Hooff, M Frans Kaashoek, and Nickolai Zeldovich. 2014. Versum: Verifiable computations over large public logs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 1304–1316.

[80] Friedhelm Victor and Bianca Katharina Lüders. 2019. Measuring ethereum-based erc20 token networks. In *International Conference on Financial Cryptography and Data Security*. Springer, 113–129.

[81] Oreste Villa, Daniel Chavarria-Miranda, Vidhya Gurumoorthi, Andrés Márquez, and Sriram Krishnamoorthy. 2009. Effects of floating-point non-associativity on numerical computations on massively multithreaded systems. In *Proceedings of Cray User Group Meeting (CUG)*. 3.

[82] Karl Wüst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. 2020. ACE: Asynchronous and Concurrent Execution of Complex Smart Contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 587–600.

[83] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2020. High-Frequency Trading on Decentralized On-Chain Exchanges. *arXiv preprint arXiv:2009.14021* (2020).

[84] Ruiyu Zhu, Changchang Ding, and Yan Huang. 2019. Efficient publicly verifiable 2pc over a blockchain with applications to financially-secure computations. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 633–650.

A Understanding the Operations

For each model, we look into its operations. As shown in Figure 12, we present the size (i.e., the number of BOPs) of every operation in a small-to-large order, with the size axis in the logarithmic scale. We can see that the operation sizes are not uniformly distributed. Some operations (e.g., ConvInteger) can be several magnitudes larger than the smaller ones (e.g., Add).
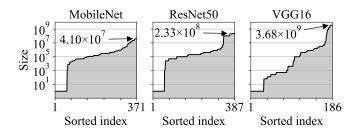

Figure 12: The distribution of operation sizes

In Figure 13, we illustrate the relation between the operation sizes and the latency of generating operation's MT^S. The points in the figure correspond to all the operations in the three models. We can see that, for a large operation, the latency is proportional to its size. For a small operation, the latency is about a small constant, mainly due to the setup time.
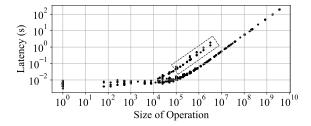

Figure 13: The latency to generate the circuit of operations

Interestingly, we discover that the latency is not only related to the operation sizes, but also the operation types, attributes, and input/output shapes. For example, we highlight the branch which is caused by the longer generation latency of shape broadcasting. Generating all the $P2\_MT^S$ and $P1\_MT^S$ trees for MobileNet, ResNet50, VGG16 take 58s, 444s and 1642s, respectively. Note that this latency is in the protocol's setup period.