Written by Conway (0x1cc) and Suning Yao (msfew)

TL;DR

Hyper Oracle's zkOracle enables trustless arbitrary compute, unlocking onchain AI and ML. We present opML, a flexible and performant approach until we launch zkML

.

- opML (Optimistic Machine Learning) ports AI model inference and training/fine-tuning onchain using an optimistic verification mechanism.
- opML provides ML proofs with low cost and high efficiency compared to zkML. The hardware requirements for opML are low: opML can run a large language model, e.g. a 7B-LLaMA (the model size is around 26GB) on a common PC without GPU, while zkML takes around 80 minutes to generate a proof of a 1M-nanoGPT model
- opML adopts a verification game (similar to Truebit and optimistic rollup systems) to guarantee decentralization and verifiability of the ML compute.
- The conclusion paragraph for this post was generated by Llama 2 in opML with 10 seconds of time for loading the model into memory and 2 minutes of time for output generation in the CPU without quantization (unoptimized).
- Hyper Oracle is the first open-source implementation of opML. You can use Stable Diffusion and LLaMA 2 in opML live here: https://ai.hyperoracle.io/. Try running a 13B Llama in opML now!

0. Onchain ML

Ethereum brings the possibility of onchain, verifiable computing. However, due to the network's resource overhead and performance limitations, it is not yet possible to support fully onchain ML or AI computing on mainnet.

Implementing onchain ML or AI can give machine learning computation all the advantages of current onchain computing, including:

- Fairness
- Verifiability
- Computational Validity
- Transparency
- Decentralization

Both opML and zkML are methods for verifiably proving that a specific model has been used to generate a particular output. This is useful for the following reasons:

- Many ML models are closed-source, meaning there is a lack of transparency regarding aspects like parameters and model size.
- Even if a model is open-source, there's no concrete evidence to confirm that the specified model was actually used for a given output, rather than, for instance, arbitrarily generating the results.
- With an increase in artificially generated content, a method to 100% prove that an output is valid can be useful. Consider proving that an image is the work of an AI model as opposed to an artist.

1. zkML

a) Introduction

Just as a ZK Rollup's offchain computation is written as arithmetic circuits to generate a succinct zk proof of the computation, and thereafter the proof is verified onchain, machine learning models can be "circuitized" and proved on Ethereum.

In this way, we can use zkML to provide onchain inference of machine learning models.

[

截屏2023-07-09_20.19.57

2704×1520 156 KB

](https://ethresear.ch/uploads/default/original/2X/7/763b8e7e1388f1104086ad6278bac84bfe472b1f.jpeg)

b) Implementation

For the implementation of zkML, there are usually two approaches:

- Specific ZK circuit: Implementing different specific ZK circuits for different models (thus meeting the different needs of each use case, such as floating point precision)
- General zkML Runtime: Adding ZK to ML models with a common runtime (just like zkEVM for EVM programs)

However, the two solutions above still have to overcome some difficulties due to the nature of ZK.

c) Challenge

The biggest challenge of any zk system is the proving overhead (1,000x compute overhead).

This means that no matter how fast the evolution of zk proof systems or hardware/software/model-specific optimizations are, zkML performance will always be worse than normal ML inference, with inevitable proving overhead in terms of latency and cost.

In addition, there are several other issues that need to be addressed and optimized in zkML: quantization, memory usage, circuit size limit, etc.

Currently, only small models can be implemented as zkML. For larger models, such as GPT3.5, it is currently not possible due to the above limitations. Therefore, we need another solution to implement practical onchain ML for large models now.

2. opML

a) Introduction

ZK Rollups use zk proofs to prove the validity of computation, while Optimistic Rollups use fraud proofs to secure the computation.

An optimistic verification approach can be used for onchain ML technology in addition to the verification of rollup computation.

By comparison, zkML and opML secure onchain ML with different approaches:

- zkML: performs ML inference computation + generates ZK proofs for each ML inference
- opML: performs ML inference computation + re-executes a segment of ML inference computation onchain when the computation is challenged

opML solves the computationally expensive problem of zkML by generating proofs only when necessary.

[

截屏2023-07-09_20.45.54

2704×1520 129 KB

](https://ethresear.ch/uploads/default/original/2X/8/8781593fe12ec61d161280d018861e3cf38ed0c8.jpeg)

b) Implementation Overview

For now, opML is very rare on the market (the only open source project is OPML-Labs/opml: OPML: OPtimistic Machine Learning on Blockchain (github.com)). For an opML implementation, two main components are needed:

- Compiler: allows ML models to be interpreted onchain for fault proof challenges
- Mechanism design: ensures a reasonable challenge period to ensure that the whole optimistic system is economically reasonable and the challenge can be censorship resistant

Compared to financial applications (e.g. rollup transfers or cross-chain applications), security for the computation of an ML model is less demanding. Such computation in ML may be inherently stochastic and not deterministic under some conditions. For a combination of reasons, we can reconsider the security requirements for the performance of onchain ML, but note that opML is still verifiable and trustless.

For opML the challenge period can be shorter, because it is not a rollup like Optimism, which involves a lot of financial operations and maintains a public ledger. When optimized, the challenge period can be like a sovereign rollup of a few minutes or even a few seconds. Overall, opML's challenge period is likely to be less than 10 minutes, rather than the typical 7 days of OP Rollup.

c) Single-Phase Verification Game: Onchain ML Model in EVM

The one-phase pinpoint protocol works similarly to referred delegation of computation (RDoC), where 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 (smart contract on blockchain) for arbitration.

In one-phase opML:

- We build a virtual machine (VM) for offchain execution and onchain arbitration, ensuring its equivalence between the offchain VM and the onchain VM implemented in a smart contract.
- To ensure the efficiency of AI model inference in the VM, we have implemented a lightweight DNN library specifically designed for this purpose instead of relying on popular ML frameworks like Tensorflow or PyTorch. Additionally, a script is provided that can convert Tensorflow and PyTorch models to this lightweight library.
- The cross-compilation technology has been applied to compile the AI model inference code into the VM program instructions.
- The VM image is managed with a Merkle tree. Only the Merkle root stands for the VM state will be uploaded to the onchain smart contract.
- The bisection protocol will help to locate the dispute step, then the step will be sent to the arbitration contract on the blockchain.

Performance

: We have tested a basic AI model (a DNN model for MNIST classification) on a PC. With our unoptimized implementation, we are able to complete the DNN inference within one second in the VM, and the entire challenge process can be completed within one minute in a local Ethereum test environment.

d) Multi-Phase Verification Game: Unlock Full Potential with Acceleration

Limitations of One-Phase Pinpoint Protocol

The one-phase verification game has a critical drawback: all computation must be executed within the Virtual Machine (VM), preventing us from leveraging the full potential of GPU/TPU acceleration or parallel processing.

Consequently, this restriction severely hampers the efficiency of large model inference, which also aligns with the current limitation of the referred delegation of the computation (RDoC) protocol.

Transitioning to a Multi-Phase Protocol

To address the constraints imposed by the one-phase protocol and ensure that opML can achieve performance levels comparable to the native environment, we propose an extension to a multi-phase protocol. With this approach, we only need to conduct the computation in the VM in the final phase, resembling the single-phase protocol. For other phases, we have the flexibility to perform computations that lead to state transitions in the native environment, leveraging the capabilities of CPU, GPU, TPU, or even parallel processing. By reducing the reliance on the VM, we significantly minimize overhead. This results in a remarkable enhancement in the execution performance of opML, almost akin to that of the native environment.

The following figure demonstrates a verification game consisting of two phases (k = 2). In phase-2, the process resembles that of a single-phase verification game, where each state transition corresponds to a single VM micro-instruction that changes the VM state. In phase-1, the state transition corresponds to a "Large Instruction" encompassing multiple micro-instructions that change the computation context.

The submitter and verifier will first start the verification game in phase-1 using a bisection protocol to locate the dispute step on a "Large Instruction". This step will be sent to the next phase, phase-2. Phase-2 works like the single-phase verification game. The bisection protocol in phase-2 will help to locate the dispute step on a VM micro-instruction. This step will be sent

to the arbitration contract on the blockchain.

To ensure the integrity and security of the transition to the next phase, we rely on the Merkle tree. This operation involves extracting a Merkle sub-tree from a higher-level phase, thus guaranteeing the seamless continuation of the verification process.

[

Untitled

651×585 18.1 KB

](https://ethresear.ch/uploads/default/original/2X/c/ca7fbe3ad65f6c19a6f77f83181760d259403403.png)

Multi-Phase opML

In this demonstration, we present a two-phase opML approach, as utilized in the LLaMA model:

- The computation process of machine learning, specifically Deep Neural Networks (DNN), can be represented as a computation graph denoted as G

. This graph consists of various computation nodes, capable of storing intermediate computation results.

- DNN inference is essentially a computation process on the aforementioned computation graph. The entire graph can be considered as the inference state (computation context in phase-1). As each node is computed, the results are stored within that node, thereby advancing the computation graph to its next state.
- Therefore, we can first conduct the verification game on the computation graph (at phase-1). On the phase-1 verification game, the computation on nodes of the graph can be conducted in a native environment using multi-thread CPU or GPU. The bisection protocol will help to locate the dispute node, and the computation of this node will be sent to the next phase (phase-2) bisection protocol.
- In phase-2 bisection, we transform the computation of a single node into Virtual Machine (VM) instructions, similar to what is done in the single-phase protocol.

It is worth noting that we anticipate introducing a multi-phase opML approach (comprising more than two phases) when the computation on a single node within the computation graph remains computationally complex. This extension will further enhance the overall efficiency and effectiveness of the verification process.

Performance Improvement

Here, we present a concise discussion and analysis of our proposed multi-phase verification framework.

Suppose there are n

nodes in the DNN computation graph, and each node needs to take m

VM micro-instructions to complete the calculation. Assuming the speedup ratio on the calculation on each node using GPU or parallel computing is \alpha

. This ratio represents the acceleration achieved through GPU or parallel computing and can reach significant values, often ranging from tens to even hundreds of times faster than VM execution.

Based on these considerations, we draw the following conclusions:

1. Two-phase opML outperforms single-phase opML, achieving a computation speedup of \alpha

times. The utilization of multi-phase verification enables us to take advantage of the accelerated computation capabilities offered by GPU or parallel processing, leading to substantial gains in overall performance.

1. Two-phase opML reduces the space complexity of the Merkle tree. When comparing the space complexity of the Merkle trees, we find that in two-phase opML, the size is O(m + n)

, whereas in single-phase opML, the space complexity is significantly larger at O(mn)

. The reduction in space complexity of the Merkle tree further highlights the efficiency and scalability of the multi-phase design.

In summary, the multi-phase verification framework presents a remarkable performance improvement, ensuring more efficient and expedited computations, particularly when leveraging the speedup capabilities of GPU or parallel processing. Additionally, the reduced Merkle tree size adds to the system’s effectiveness and scalability, making multi-phase opML a compelling choice for various applications.

Performance

: We have tested a 7B-LLaMA (the model size is around 26GB) on a common PC without GPU. Using our unoptimized implementation, the entire challenge process can be completed within 20 minutes in a local Ethereum test environment. (We can definitely make it faster using GPU and some optimization technologies of the fraud proof system!)

e) Consistency and Determinism

In opML, ensuring the consistency of ML results is of paramount importance.

During the native execution of DNN computations, especially across various hardware platforms, differences in execution results may arise due to the characteristics of floating-point numbers. For instance, parallel computations involving floating-point numbers, such as (a + b) + c

versus a + (b + c)

, often yield non-identical outcomes due to rounding errors. Factors like programming language, compiler version, and operating system can also influence floating-point computations, causing further inconsistencies in ML results.

To tackle these challenges and guarantee the consistency of opML, we employ two key approaches:

1. Fixed-point arithmetic, also known as quantization technology, is adopted. This technique enables us to represent and perform computations using fixed precision rather than floating-point numbers. By doing so, we mitigate the effects of floating-point rounding errors, leading to more reliable and consistent results.
2. We leverage software-based floating-point libraries that are designed to function consistently across different platforms. These libraries ensure cross-platform consistency and determinism of the ML results, regardless of the underlying hardware or software configurations.

By combining fixed-point arithmetic and software-based floating-point libraries, we establish a robust foundation for achieving consistent and reliable ML results within the opML framework. This harmonization of techniques enables us to overcome the inherent challenges posed by floating-point variations and platform disparities, ultimately enhancing the integrity and dependability of opML computations.

3. opML v.s. zkML

a) Overview

opML

zkML

model size

any size (available for extremely large models)

small/limited (due to the cost of ZKP generation)

validity proof

fraud proof

zero-knowledge proof

requirement

Any PC with CPU/GPU

large memory for ZK circuit

cost

low (inference and training can be conducted in native environment)

extremely high (generating a ZKP for ML inference is extremely high)

Finality

delay for challenge period

No delays (wait for ZKP generation)

security

crypto-economic incentives for security

cryptographic security

*: Within the current opML framework, our primary focus lies on the inference of ML models, allowing for efficient and secure model computations. However, it is essential to highlight that our framework also supports the training process, making it a versatile solution for various machine learning tasks.

b) Performance & Security

Both zkML and opML present distinct advantages and disadvant The main difference is flexibility (performance) and security.

- zkML allows for on-chain ML execution with the level of highest security, but it is limited by proof generation time and other constraints.
- opML facilitates on-chain ML execution with enhanced flexibility and performance, but is not secured by mathematics and cryptography like ZK.

The security model of opML is similar to the current optimistic rollup systems. With the fraud proof system, opML can provide an any-trust guarantee: any one honest validator can force opML to behave correctly.

Suppose there exists only one honest node, and all other nodes are malicious. When a malicious node provides an incorrect result on chain, the honest node will always check the result and find it incorrect. At that time, the honest node can challenge the malicious one. The dispute game (verification game) can force the honest node and the malicious node to pinpoint one concrete erroneous step. This step will be sent to the arbitration contract on the blockchain, and will find out the result provided by the malicious node is incorrect. Finally, the malicious node will be penalized and the honest node can get a corresponding reward.

[

截屏2023-09-14_18.45.15

2696×1514 116 KB

](https://ethresear.ch/uploads/default/original/2X/0/08ae3e800f6b110ef4f53d1bbdcf77d82def6d9c.jpeg)

c) Cost of Onchain ML

In terms of cost, opML has a great advantage over zkML due to the huge cost for generating ZKPs.

Proof Generation Time

The proof time of zkML is long, and significantly grows with increasing model parameters. But for opML, computing the fraud proof can be conducted in the nearly-native environment in a short time.

[

Proof Time for zkML from ModulusLabs

877×625 86.8 KB

](https://ethresear.ch/uploads/default/original/2X/8/87facb6a5e4d36c2f97427bfeb90e0b792521d46.png)

Proof Time for zkML from ModulusLabs

Here is an example of a significantly long proving time of zkML. It runs a nanoGPT model inference with 1 million parameters inside a zkML framework and takes around 80 minutes of proving time.

Memory Consumption

The memory consumption of zkML is extremely huge compared to the native execution. But opML can be conducted in a nearly-native environment with nearly-native memory consumption. Specifically, opML can handle a 7B-LLaMA model (the model size is around 26GB) within 32GB memory. But the memory consumption for the circuit in zkML for a 7B-LLaMA model may be on the order of TB or even PB levels, making it totally impractical. Indeed, currently, no one can handle a 7B-LLaMA model in zkML.

[

Untitled 2

926×624 106 KB

](https://ethresear.ch/uploads/default/original/2X/c/c91c8c40bfd451e7356940a115ee8aa139b0a6d6.png)

Memory consumption for zkML from ModulusLabs

d) Finality

We can define the finalized point of zkML and opML as follows:

- zkML: zk proof of ML inference is generated (and verified)
- opML: challenge period of ML inference has ended

For the finality of the two, we can compare and conclude that: though opML needs to wait for the challenge period, zkML also needs to wait for the generation of the ZKP.

When the proving time of zkML is longer than the challenge period of opML (it happens when a ML model is large), opML can reach finality even faster than zkML!

When the ML model is large, opML is all you need!

[

截屏2023-09-14_18.45.32

2696×1514 73.1 KB

](https://ethresear.ch/uploads/default/original/2X/d/d85e3f4a997bf294b61bf0c0b84ef701090a3ce7.jpeg)

Further Optimization with ZKFP (ZK Fraud Proof)

Furthermore, we can utilize multiple checkpoints and multi-step ZK verifier to provide azk-fraud proof to speed up the finality of opML.

Assuming that there are N

steps in the opML fraud proof VM. Then the number of interactions would be O(\log_2(N))

. One optimization approach to reduce the number of interactions is to use a multi-step on-chain arbitrator, instead of only one-step on-chain arbitrator.

This can be done by using a zkVM, e.g., zkWASM, RISC Zero, Cairo VM, which can provide zk-proof for the multi-step execution. Suppose the zkVM can generate a zk-proof for m

steps, then the number of interactions can be reduced from O(\log_2(N))

to O(\log_2(N/m))

. Besides, instead of using a bisection protocol with only 1 checkpoint, we can provide multiple checkpoints (for example k

(k >1

) checkpoints) in the dispute game (verification game). At this time, we can further reduce the number of interactions from O(\log_2(N/m))

to O(\log_{k+1}(N/m))

. With these optimizations, we can greatly speed up the finality of opML, and it is possible to achieve almost instant confirmation!

For our previous work, we collaborated with EthStorage and Delphinus Lab onzkGo (a minimum modified Go compiler to produce Wasm code compatible with ZK prover). As a result of these modifications, the execution of the compiled L2 Geth Wasm code can now be run using zkWASM. This can be a solid foundation for ZKFP.

5. Conclusion

The following is a conclusion generated by LLaMA 2 in opML.

Loading the model into memory took 10 seconds, and generating output 2 minutes with CPU without quantization (unoptimized).

In conclusion, the optimistic approach to onchain AI and machine learning offers significant advantages over existing methods. By embracing this novel approach, we can harness the power of AI while ensuring the integrity and security of the underlying blockchain system. opML, in particular, has shown impressive results in terms of efficiency, scalability, and decentralization, making it an exciting development in the field of AI and blockchain technology. As the use cases for onchain AI expand, it is essential that we continue to explore innovative solutions like opML, which can help us unlock the full potential of these technologies while maintaining the highest standards of transparency, security, and ethical responsibility.

6. Acknowledgement

Thanks to Will Villanueva and Luke Pearson for their insightful feedback and review!

About Hyper Oracle

Hyper Oracle is a programmable zkOracle protocol that makes smart contracts smarter with arbitrary compute and richer data sources. Hyper Oracle offers full security and decentralization for trustless automation and onchain AI/ML so builders can easily create next-gen dApps.

hyperoracle.io | x.com/hyperoracle | github.com/hyperoracle