ZKML Benchmarking Project Report

Introduction

As machine learning gains global attention, ensuring the integrity and authenticity of machine learning models, particularly in Machine Learning as a Service (MLaaS), has become crucial. Zero-knowledge Machine Learning (zkML), and specifically zk-SNARK, emerges as a key technology in bridging the trust gap, ensuring both computational integrity and data privacy. This project aims to develop a zkML benchmarking tool to guide developers through the complexities of the zkML landscape, focusing on the inference phase of ML models due to the high computational demands of the training phase.

Current Landscape

Many research works and industrial frameworks have been proposed to convert an ML model into an arithmetic circuit. The landscape is categorized into several key areas:

- Model-to-Proof Compilers: Convert conventional ML model formats (e.g. ONNX, Keras, etc) into verifiable computational circuits
  - **EZKL:** A library and command-line tool for converting ONNX models into zk-SNARK circuits.
  - **Orion:** Generates Validity ML enabling verification of inference by leveraging Cario and ONNX's capabilities.
  - **keras2circom&Circomlib-ml:** Python tools transpiling Keras models into Circom circuits.
  - **LinearA:** A framework bridging Apache TVM to arithmetic circuits in ZKP systems.
  - **Daniel Kang's zkml:** Constructs proofs for ML model execution written in Halo2.
- zkML-Specific Proving Systems: Generate efficient verification for ML models with circuit-unfriendly operations
  - **zkCNN:** A novel approach for verifying convolutional neural networks.
  - **Zator:** Utilizes recursive zk-SNARKs for deep neural network verification.
  - **ZeroGravity:** Verifies pre-trained, public WNN inferences with private inputs.
- Hardware Acceleration: Build specialized hardware to support proof generation
  - **Supranational:** GPU acceleration solutions.
  - **Accseal:** ASIC chips for ZKP hardware acceleration.
  - **Icicle:** CUDA implementation for general functions in zero-knowledge proof.
- Application: Design zkML use cases
  - **Worldcoin:** Integrates zkML for privacy-preserving proof of personhood protocols.
  - **ZKaptcha:** Enhances captcha services by analyzing user behaviors through zkML.
  - **ZKaggle:** A Bounty platform for hosting, verifying, and paying out bounties
  - **RockyBot:** On-chain verifiable ML trading bot.

Proving System

Generalized proving system like Plonkish, STARK, R1CS and GKR are the backbone of the aforementioned zkML frameworks and crucial for realizing zkML. However, the naive conversion of an ML model into an arithmetic circuit is not always feasible. For example, the advancement of verifiable ML is hindered by non-arithmetic operations, notably activitaion functions such as ReLU, sigmoid,and tanh. Plonkish-based systems, such as Halo2 and Plonky2, are popular due to their table-style arithmetization schemes that handle neural network non-linearities efficiently through lookup arguments. Yet, these lookups come with notable prover memory consumption costs. In terms of performance, other systems like R1CS excel in small proof sizes, and GKR-based systems seem best suited for large models. Despite the introduction of various proving systems in recent years, each excelling in specific applications and advancements in non-trusted-setup, proving efficiency, and deterministic proof size, almost all systems face accuracy loss during quantization.

Taken together, it's essential to highlight the main challenges of compiling neural networks to ZKP systems:

1. **Floating Point in ZKP:** Neural networks are often trained using floating point numbers. In zkML, this brings about the challenge of quantizing these numbers into fixed-point representations, without significant accuracy loss.
2. **Compatibility:** ZKP systems aren't inherently compatible with the complex operations (e.g. activitation function, matrix multiplication, etc) commonly used in neural networks.
3. **Performance:** Crafting ZKP for ML models is a delicate act of balancing various trade-offs. Researchers must consider memory usage, proof size, and prover time to ensure optimal performance.

Therefore, some progress has been made in designing zkML-Specific proving systems to optimize the proof for the advanced ML models, including zkCNN, vCNN, and pvCNN. As these names suggest, they are optimized for CNN models, and thus can only be applied to certain CV tasks, such as MNIST or CIFAR-10.

In summary, no exisitng proving systems sufficiently addresses these challenges across various ML tasks, highlighting the need for comprehensive benchmarking in the zkML ecosystem.

Shortlisted Project

Due to time constraints, we benchmark a shortlisted selection of zkML projects to compare aspects such as proof generation time, prover memory usage, and framework compatibility. This selection does not represent the entire landscape but aims to exemplify current trends through a subset of frameworks. We plan to update the benchmark results in response to emerging frameworks.

A basic information of shortlisted frameworks is provided as follows.

Name	Model Format	Star	Proof System	Link
EZKL	ONNX	713	Halo 2 *	GitHub Repo
Orion	ONNX	132	ZK-STARK	GitHub Repo
DDKang ZKML	TFLite	314	Halo 2	GitHub Repo
keras2circom	tf.keras	69	R1CS Groth16	GitHub Repo
opML	ONNX (WiP)*	65	Fraud Proof***	GitHub Repo

Halo2*: EZKL customs halo2 circuits through aggregation proofs and recursion, and optimizes the conversion using fusion and abstraction.

ONNX(WiP)**: Work in Progress

Fraud Proof***: Instead a traditional zk proofs to prove the validity of computation, opML provides an any-trust guarantee using the fraud proof system that any honest validator can force opML to behave correctly

Benchmark Methodology

The methodology adopted for benchmarking zkML frameworks is designed to systematically evaluate and compare the capabilities and performance of various zk proof systems for the verifiable inference. This evaluation focuses on aspects crucial for practical deployment in MLaaS scenarios, such as proof generation time, memory usage, and system compatibility.

Selection of zkML Frameworks

The selected frameworks for benchmarking include:

- **EZKL (Halo 2)**
- **Orion (ZK-STARK)**
- **DDKang ZKML (Halo 2)**
- **keras2circom (R1CS Groth16)**
- **opML (Fraud Proof)**

These frameworks have been chosen based on their popularity (indicated by GitHub stars), the proof system they utilize, and their support for different ML model formats. This diverse selection ensures a comprehensive analysis across various zk proof systems.

Key Metrics for Evaluation

The primary metrics for benchmarking include:

1. **Proof Generation Time:** The duration taken by each framework to generate a proof. This metric is crucial for assessing the efficiency of the proof system.
2. **Maximum Prover Memory Usage:** The peak memory usage during the proof generation process, indicating the resource intensity of the system.
3. **Framework Compatibility:** Evaluation of each framework's compatibility with different machine learning model formats and operating systems.

Planned Benchmarking Tasks

Our benchmarking methodology includes tasks on two computer vision (CV) datasets - MNIST and CIFAR10 - to evaluate the frameworks under different levels of complexity:

1. **MNIST Dataset:**
    - Simplicity of the Task: MNIST, being a dataset of handwritten digits, represents a less complex task, suitable for evaluating the basic capabilities of zkML frameworks.
    - Framework Assessment: We will observe how each framework handles relatively simple image data and whether it can maintain accuracy and efficiency.

2. **CIFAR10 Dataset:**
    - Increased Complexity: CIFAR10, with its more complex image data (like animals, vehicles, etc.), increases the challenge for zkML frameworks.
    - Parameter Variation: We will test the frameworks on this dataset with an increasing number of parameters and layers, pushing the boundaries of each framework's capacity.
    - Accuracy Loss Measurement: If applicable, we will measure and compare the accuracy loss across different frameworks, providing insight into their robustness and fidelity in more complex tasks.

Benchmarking Process

The benchmarking process for evaluating zkML frameworks is structured to provide a comprehensive analysis of their performance across various metrics. Here’s an expanded breakdown of the steps involved:

1. **Circuit Design for MLPs:** For each zkML framework, we design circuits representing Multi-Layer Perceptrons (MLPs), focusing on structures typically used in machine learning models. These circuits are tailored to each framework's specifications and capabilities.

2. **Uniform Testing Conditions:** We employ the same benchmark suites across all frameworks to ensure consistent and fair testing conditions. These suites include MLP architectures with varying parameters, FLOPs, and number of layers, providing a uniform basis for comparison.

3. **Encoding MLPs:** Each framework's unique approach to encoding MLPs is critically examined. This involves understanding how each proof system translates machine learning operations into its respective arithmetization scheme.

4. **Exclusion of Pre-Processing Steps:** Our measurements concentrate exclusively on the proof generation phase, deliberately omitting pre-processing or witness generation steps to maintain a focused evaluation of proof generation efficiency.

5. **Modifying MLP Structures:** To assess scalability and robustness, we systematically modify the MLP structures within each framework by increasing the number of parameters and layers. This alteration simulates more complex ML models and highlights each framework’s adaptability and performance under escalated complexity.

6. **Comparative Performance Analysis:** We compare the performance of each framework under modified conditions. This includes analyzing how changes in MLP structures impact key metrics like proof generation time and memory usage.

7. **Highlight Differences:** Clear distinctions in performance and capabilities across various proving systems are marked. This includes detailed information on how each system responds to different benchmarking tasks and conditions, including those with modified MLP structures.

8. **List Pros and Cons:** A comprehensive list of the advantages and disadvantages for each framework and its associated proving system is provided. This helps in understanding the suitability of each framework for specific types of ML models, particularly as complexity increases.

9. **Framework-Specific Feature Evaluation:** Any unique or specialized features of each framework, such as specialized circuit designs or optimizations for particular types of ML models, are evaluated.

This helps in understanding how these features contribute to the overall performance and utility of the framework.

Through this detailed benchmarking process, we aim to provide a nuanced understanding of each zkML framework’s capabilities, especially in handling increasingly complex machine learning models. This approach will guide users in selecting the most suitable framework for their specific requirements in the realm of zkML.

Other Consideration

In addition to the primary metrics for evaluating zkML frameworks, several other considerations play a crucial role in our comprehensive benchmarking process. These considerations include the analysis of related benchmarking works, support from the community, the alignment of zkML frameworks with their underlying proving systems and the fairness in comparative analysis.

ZK Score

The ZK Score, as discussed by Ingonyama, is a significant metric in evaluating the efficiency of zk proof systems, particularly from a hardware acceleration perspective. It measures the throughput of modular multiplications per watt, offering a standard for comparing hardware efficiencies. While ZK Score provides valuable insights, especially in the context of hardware acceleration projects, its applicability to zkML frameworks needs careful consideration.

- **Applicability Limitations:** ZK Score predominantly focuses on the hardware aspect and may not fully encapsulate the software or algorithmic efficiencies inherent in zkML frameworks. Thus, while useful, it might not be entirely suitable for a comprehensive assessment of all aspects of zkML frameworks.

Support and Community Involvement

- **Well-Organized Committee:** The level of support and development from a well-organized committee or community behind a zkML framework is crucial. Active community involvement often leads to more robust and user-friendly frameworks.
- **Documentation Quality:** High-quality, comprehensive documentation is essential for the usability and accessibility of zkML frameworks. It significantly impacts the ease with which developers can adopt and implement these technologies.

Alignment with Proving Systems

- **Backbone Proving System Compatibility:** A key consideration is how well a zkML framework aligns with its underlying proving system. This includes examining if the framework leverages the full capabilities of the proving system or if any specific optimizations improve performance.
- **Framework-Specific Optimizations:** Any optimizations or modifications made to the proving systems to enhance their suitability for ML tasks are critically evaluated. This includes adaptations for handling complex operations typical in ML models, like matrix multiplications or activation functions.

Fairness in Comparative Analysis

Given the diversity in the underlying mechanics of these proving systems, a direct comparison across different systems may not always present a fair assessment. Each proving system is designed with specific goals and trade-offs in mind, making them suitable for different types of tasks and applications.

Limitations and Future Work

While the methodology is comprehensive, it focuses on the prover side of the zk proof systems, leaving aspects like verifier runtime and proof size for future analysis. Furthermore, the evolving nature of zkML technologies means continuous updates and refinements to the benchmarking process will be necessary.

Conclusion

These additional considerations ensure that our benchmarking methodology is well-rounded and covers aspects beyond basic performance metrics. By analyzing factors like other benchmark works, special features, community support, and alignment with proving systems, we aim to provide a more nuanced and holistic view of the zkML landscape. This comprehensive approach is crucial for understanding the full spectrum of capabilities and limitations of various zkML frameworks and guiding users in selecting the most appropriate tools for their specific needs.