Intro

We're glad to announce a project byTACEO, aiming to build ONNX frontend for zkLLVM to create zkML circuits for effectively proving large ML models to Ethereum, like GPT-2 or BERT. The team of researchers and developers is leveraging =nil; toolchain to bring an ONNX extension to the zkLLVM circuit compiler. The plan is to introduce a smooth pipeline for circuit definition over existing and tested ML models, empowering decentralized applications with ML-enhanced scenarios.

As well as =nil; team, TACEO folks see their mission in making cryptographic operations practical, providing developers with the essential tools to create reliable and scalable decentralized applications. The tool they're building now would be a valuable addition to the zk ecosystem. In this blog post, we'll cover the main idea and summarize the steps of the project.

TACEO is an R&D cryptography company born out of collaborative efforts at the University of Graz of Technology in Austria. The company founders originated innovative work in symmetric primitives for zero-knowledge proofs (Poseidon, Reinforced Concrete), multiparty computation (LowMC, MiMC), homomorphic encryption (Rasta, Pasta), and post-quantum signatures (SPHINCS+ and Picnic).

Why does the industry need zkML, and what's the core challenge

An increasing number of use cases now demand the capabilities of Machine Learning to make meticulously calculated decisions amidst vast data. We understand that ML could make a significant impact on applications in DeFi, NFTs, healthcare solutions, secure identification, and many others. Most of these would work with sensitive data, and even if the data itself could be encrypted before the model trains on it, there's still a critical trust question. We'd need to know if the model was trained correctly and if it provides the correct output based on the input we gave it. All of this can be verified if an application can prove each execution of the ML model with a zkProof.

It's only a question of time that ML should enter the realm of decentralized applications. Many forefront developer teams in the industry are working now to implement provable ML (zkML). However, this task is of high technical complexity: while zkML is a hot topic, only adding sophisticated computationally intensive ML models like GPT-2 or BERT would have a real impact. With the current approach to zk (among other things - with manual circuit definition) this is a dead end for zkML.

Today, proving even simple computational logic requires teams to manually write circuits for months. Describing one ML model to compile a circuit may be doable, time-wise, but compiling a new circuit for each new model or an updated version would have to start from scratch, and in this case, the time to complete the task would be ridiculously long with a manual approach.

Also, applying zkDSL to rewrite ML models does not always seem possible because most zkDSLs are not general-purpose coding languages, and may not have an expression for specific ML operations.

Even if all of this could be solved, this pipeline would require training of the model to get production-level performance, while we'd rather take an existing trained model - and anyone would, to save time and production resources.

So, the key challenge is: how can we work with existing trained ML models while keeping a trustless approach to their execution?

Solution & Components: zkLLVM circuit compiler

The goal is to minimize trust and bring ML models to a provable framework, which requires zk proving, which, in turn, requires creating a circuit for further proof generation over model execution. And this is the tricky part.

Previously we've introduced zkLLVM circuit compiler to prove computations in C++ or Rust - mainstream development languages that are more common for wider developers and require less effort to get correct computations code than any custom zkDSL.

To prove Neural networks, one would need to add another standard to this list - ONNX (Open Neural Network Exchange), the most popular standard to express Neural networks. This becomes possible due to the specifics of LLVM, the framework behind =nil; zkLLVM compiler and TACEO team leverages this toolchain to build their target pipeline (we'll summarize the project steps later in this post).

With the goal to compile ONNX models to circuits through zkLLVM, and then use the circuits to prove MLs to Ethereum or other L1s/L2s, we're aiming to introduce a zkLLVM extension that will help us bring provable large-sized ML models to Ethereum, and this will ease the way for experienced ML developers to contribute to Web3 without having to learn zk or any other custom programming language.

zkLLVM Documentation

Solution & Components: Placeholder

Since ML computations have their specificities, we would need a proof system adjustable to its specific needs - like =nil; Placeholder, a modular PLONK with IVC, and custom gates.

Placeholder's modularity allows to recompile the proof system setup to reuse any model in different applications with different security assumptions. Also, Placeholder offers flexibility in inner arguments for different models depending on block size and total repetition parameter.

In the case of zkML, developers would benefit from using the Flexible Gates technique allowing customization of compilation strategies. With this solution, we can reuse a circuit definition for different ML primitives, like ReLU, GeMM, maxPool, etc.; a new primitive can be built without any additional overhead.

We're actively working on Placeholder now. Read more about it in our blog post.

Project design: ONNX-MLIR-zkLLVM

Here's the vision of the project our teams agreed upon.

Below we provide a short explanatory summary of the project pipeline and encourage you to read the TACEO blog post on this subject to learn more technical details.

- ONNX

ONNX (Open Neural Network Exchange) is an open standard allowing a common exchange and file format of trained neural networks. TACEO team will add ONNX as a front-end for the zkLLVM compiler infrastructure.

- ONNX-MLIR

MLIR (also a project based on the LLVM framework, just like zkLLVM) is an extensible compiler infrastructure that allows the creation of domain-specific compilers. ONNX-MLIR defines Passes and Dialects that serve as a bridge between ONNX and LLVM-IR. Additionally, it enables the execution of analytics and optimizations passes on the IR.

- LLVM-IR

During the lowering step to LLVM-IR, TACEO team will rewrite certain sub-optimal operations with the Pass infrastructure for better performance. "Naively" lowering certain operations (especially matrix multiplications) results in sub-optimal LLVM-IR. Therefore, they will create Passes to rewrite those parts to enable the use of custom gates, leading to smaller circuits.

- zkLLVM assigner

zkLLVM assigner parses the LLVM-IR and assigns values to the circuit. TACEO team will extend the functionality of the assigner to support the custom gates created in the previous step. Additionally, they will add fixed-point operations (for approximating floating-point arithmetic) as those are heavily used for machine learning and are currently unsupported.

- Placeholder proof system

The proof system outputs the proof when all previous steps are done.

With procedural circuit definition over existing & trained ML models as a target, we believe this is a firm step toward new scenarios for trustless, secure, and scalable decentralized applications. Enabling zkLLVM to compile ONNX-based circuits would open the gates for many developers to leverage ML models in the Web3 world, building more complex use cases for the world of decentralized applications - in a trustless manner.

We'll keep an eye on the progress of TACEO team and their project. Keep in touch.