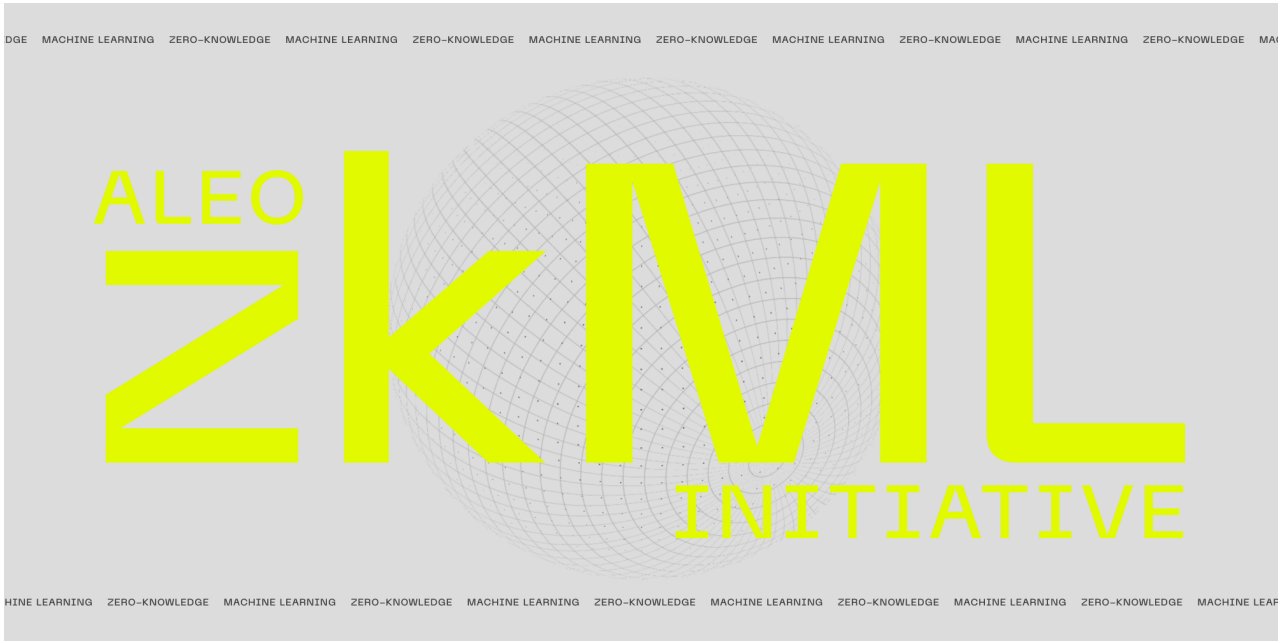


zkML Initiative - Developer Guidelines

Categories

Season 1 of the zkML Initiative will be focused on:

[zkML Algorithms]

Create useful zkML algorithms using the Leo language and/or Aleo instructions.

[Python Developer Tooling]

Create the ability to use zkML algorithms directly from Python.

zkML Algorithms

Under the hood, the Leo Language and Aleo Instructions create a zero-knowledge proof which proves a computation was done correctly while only revealing which inputs and outputs the prover decides to make public.

Generally, when machine learning algorithms are run, these inputs and outputs are entirely public, or it is impossible to get any assurances about the model and data. However, using the proofs which Leo/Aleo Instructions create, machine learning algorithms can be run while protecting private data.

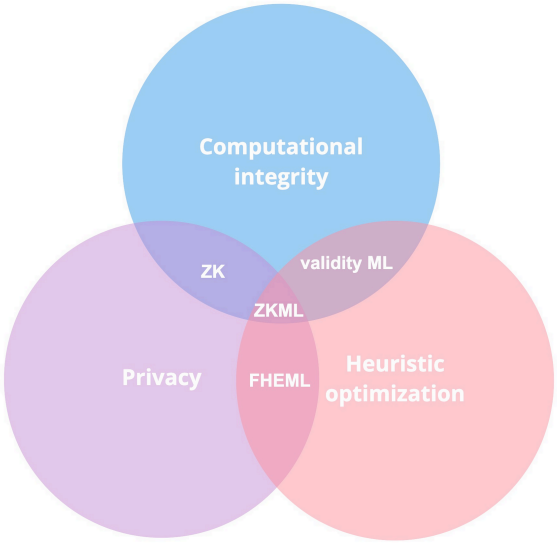
In this category of the zkML Initiative, the goal will be to build common ML algorithms in zero-knowledge using Leo or Aleo instructions.

Such algorithms include:

Linear Regression	Decision Trees	Neural Network Layers	XGBoost / AdaBoost	K-Means / KNN

Submission Guidelines

- a Github repository with the code you wrote
- a small demonstration of how your code works - this could be in the form of a small web app, a CLI tool, or a small video that shows how it works
- a README with:
    - instructions on producing reproducible results
    - if a working example was not obtained, please explain the limitations and critical research results that prevented a working example
- a short writeup on privacy, usability, and correctness (you can include this in the README or separately)
    - What is the privacy impact of the implementation of this algorithm?
    - How does it preserve the privacy of its users?
    - How usable would this be to machine learning developers and practitioners?
    - What's a hypothetical scenario that this model could be used in?
    - Does the model produce accurate results?
- a submission form will be provided on May 12th



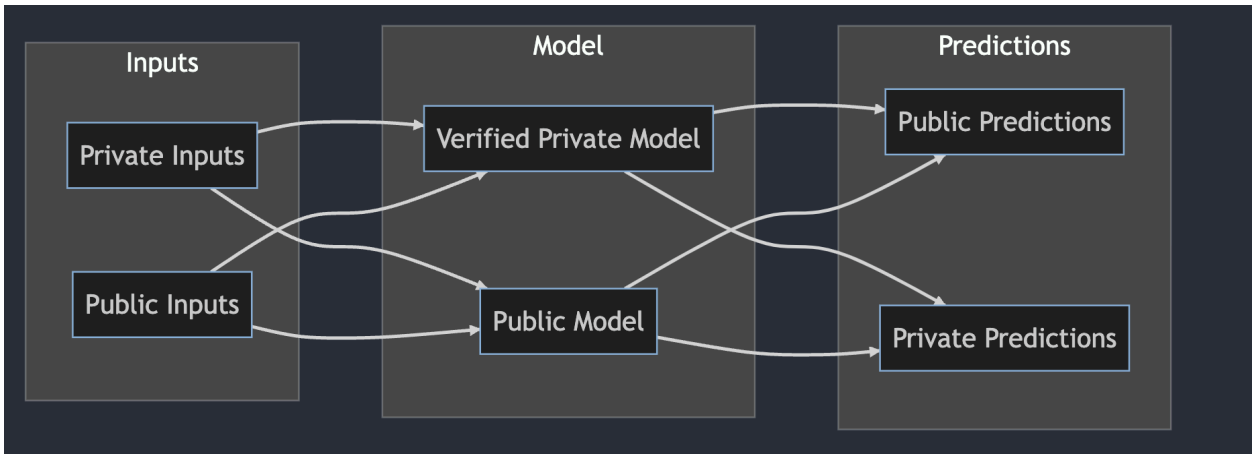
Source: Worldcoin

Philosophical Considerations

When building a private machine learning model, one has to question what is being protected and why, prior to building the model itself. Some of these considerations include:

- Is the privacy of the inputs being protected to protect user data?
- Are the predictions sensitive? Perhaps we want to make a proof that a prediction satisfies certain conditions but we don't want to expose the prediction itself.
- Is the model itself private or public? Why would either choice be made (maybe we let the user run the model on their own device so their data isn't exposed to an external server, yet we want to verify they ran the correct model).

The entire possibility space might be seen on a graph like this:

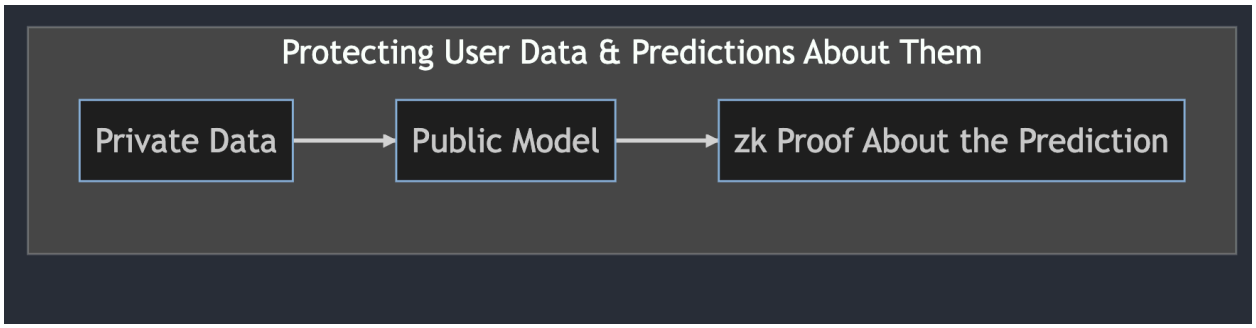


Example 1: Protecting User Data

Say that you're simply measuring via a classification model whether or not someone's provided data that places them in a specific class. The model might look like this:

Example 2: Protecting User & Their Prediction

Perhaps one would like to go a step further and protect predictions about a user. Say there's a model trying to predict the risk category for a loan, but exposing that category would expose undesirable data about the user. It is possible that the Snark provides verification that the user is within a certain range of acceptable risk without exposing the exact category that they're in.



It's up to YOU!

What needs to be protected and why are important considerations as the model itself, and some might say, even more important of a consideration.

Python Developer Tooling

The machine learning world works largely in Python so making zero-knowledge machine learning available directly in Python would be a big win for the world of zero-knowledge machine learning. In this category the aim is to build some introductory tools to help developers working in Python do their work directly in Python!

Tooling Examples

Type Translation	**Running Leo from Python**	**Wrapping Rust in Python**
Most of the types in zero-knowledge proving are integer types while most types in ML are floating point. Translating these types properly so that both the ML results and the correctness of zero-knowledge proofs would help elevate the world of zkML.	The build here would essentially be Leo programs built and run through a set of command line tools. Being able to run zero-knowledge programs directly from Python enables future developers to work seamlessly with their existing ML tools.	Underlying Leo is the Rust language. Much of the lifting in Leo's program compilation and chain interaction happens in Rust. Rust and Python have many good foreign function interface (FFI) tools that help bridge the ability to execute these core algorithms in Python.

Submission Guidelines

- a Github repository with the code you wrote
- a small demonstration of how your code works - this could be in the form of a small web app, a CLI tool, or a small video that shows how it works
- a README with:
    - instructions on producing reproducible results
    - if a working example was not obtained, please explain the limitations and critical research results that prevented a working example
- a short writeup on impact, usability, and extensibility (you can include this in the README or separately)
    - How does this change the landscape of python ZkML tooling?
    - How usable would this be to machine learning developers and practitioners?
    - Can this submission be extended in the future to include a broader set of tools and features?
- a submission form will be provided on May 12th

Judging

Judging Panel

- Mike Turner - Protocol Engineer
- Victor Sint Nicholaas - Protocol Engineer
- Frank Chen - Product Manager
- Konstantin - zkML Grantee
- Ambassador #1
- Ambassador #2

Voting Mechanism

Weighted voting. Each judge receives 10 votes per prompt category (Algorithms / Python Developer Tooling) with 20 votes total, and they can allocate as many or as little votes to each project by category. Votes will be totaled at the end by category to determine ranked winners. Ties will be broken with another round of weighted voting.

Criteria

Engineering	How clean is the code? How efficiently does the program(s) run? How well is the tooling implemented?
Functionality	What can you accomplish by using the algorithm/tooling?
Foundations	Is this something that people would find valuable or useful?
User Experience	How easy is it for people to use the algorithm/tooling?
Bonus - UI	Do you have a user interface that people can interact with? (This is not required but could be nice!)

Prizes

For each category, there will be a 1st, 2nd, and 3rd place prize for Aleo Credits.

Place	Prize
Algorithms 1st Place	80K Aleo Credits
Algorithms 2nd Place	40K Aleo Credits
Algorithms 3rd Place	20K Aleo Credits
Python Developer Tooling 1st Place	80K Aleo Credits
Python Developer Tooling 2nd Place	40K Aleo Credits
Python Developer Tooling 3rd Place	25K Aleo Credits

Terms of Use

Aleo’s zkML Initiative follows our general Grants Terms of Use.