Lesson 9

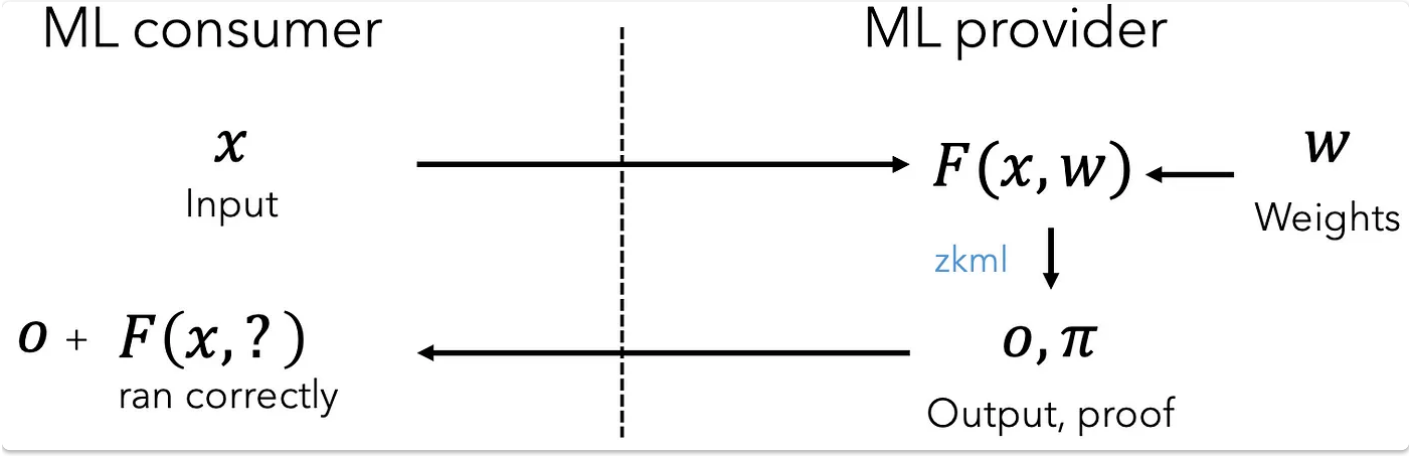
Today's topics

- Tensor Plonk
- Zero Gravity

Tensor Plonk

Motivation and zkML library

From article



The ML consumer has guarentees that the model computed correctly, but doesn't learn anything about the weights.

Using the library

```
cargo build --release
mkdir params_kzg

./target/release/time_circuit
```

```
examples/mnist/model.msgpack
examples/mnist/inp.msgpack kzg
```

This will construct the proof, where the proving key is generated during the process. Here, model.msgpack is the model weights and inp.msgpack is the input to the model. Some example logits of the model are

```
<snip>
final out[0] x: -5312 (-10.375)
final out[1] x: -8056 (-15.734375)
final out[2] x: -8186 (-15.98828125)
final out[3] x: -1669 (-3.259765625)
final out[4] x: -4260 (-8.3203125)
final out[5] x: 6614 (12.91796875)
final out[6] x: -5131 (-10.021484375)
final out[7] x: -6862 (-13.40234375)
final out[8] x: -3047 (-5.951171875)
final out[9] x: -805 (-1.572265625)
<snip>
```

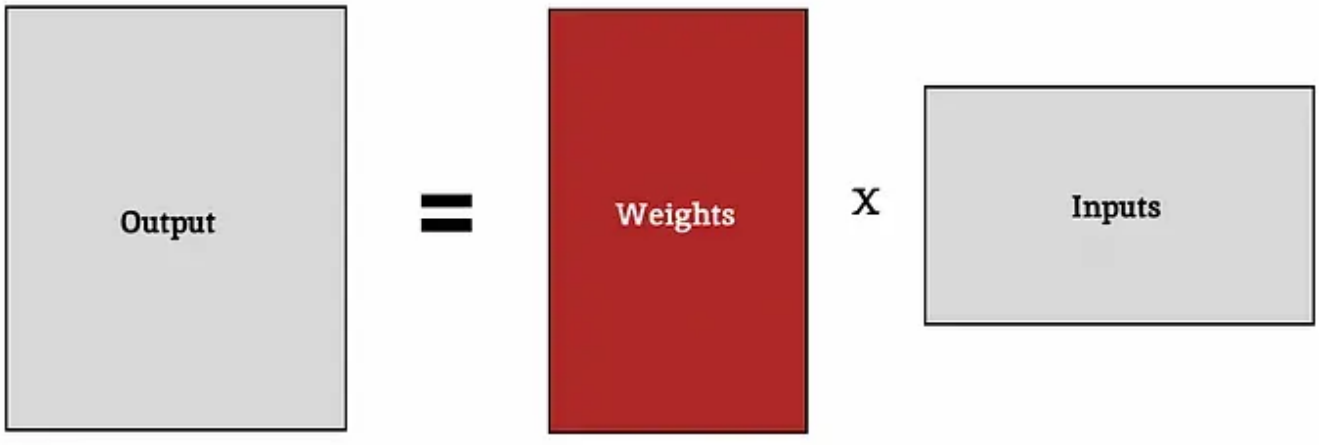
This is a classification task based on the standard images of handwritten numbers, the input was of the number 5, you can see in the outputs , output 5 is the largest.

The proof can be verified with

```
./target/release/verify_circuit
examples/mnist/config.msgpack vkey proof
public_vals kzg
```

See article

a “GPU” for ZKML.
From the article
ML models consist of *linear layers* interspersed with *non-linear layers*. The linear layers are operations like matrix multiplication. The non-linear layers are typically operations like the ReLU.



Typically if the weights are of size m x n, and the input is n x d, the runtime of performing the matrix multiplication is $O(m \text{ x } n \text{ x } d)$.

A common approach for dealing with non linearities is to use lookup tables such as Flookup or CQ

Tensor Plonk improvements

Work by Eagen & Gabizon called cqlin can prove the result of matrix multiplication in $O(n)$ time, if one of the matrices is fixed ahead of time.
This comes at a cost though, the verifier would need to do a pairing check per matrix multiplication. which would increase the proof size and verifier work by nearly 10x.
To solve this problem we can use a randomized check to verify all of the matrix multiplications within a model at once.

Lookup tables

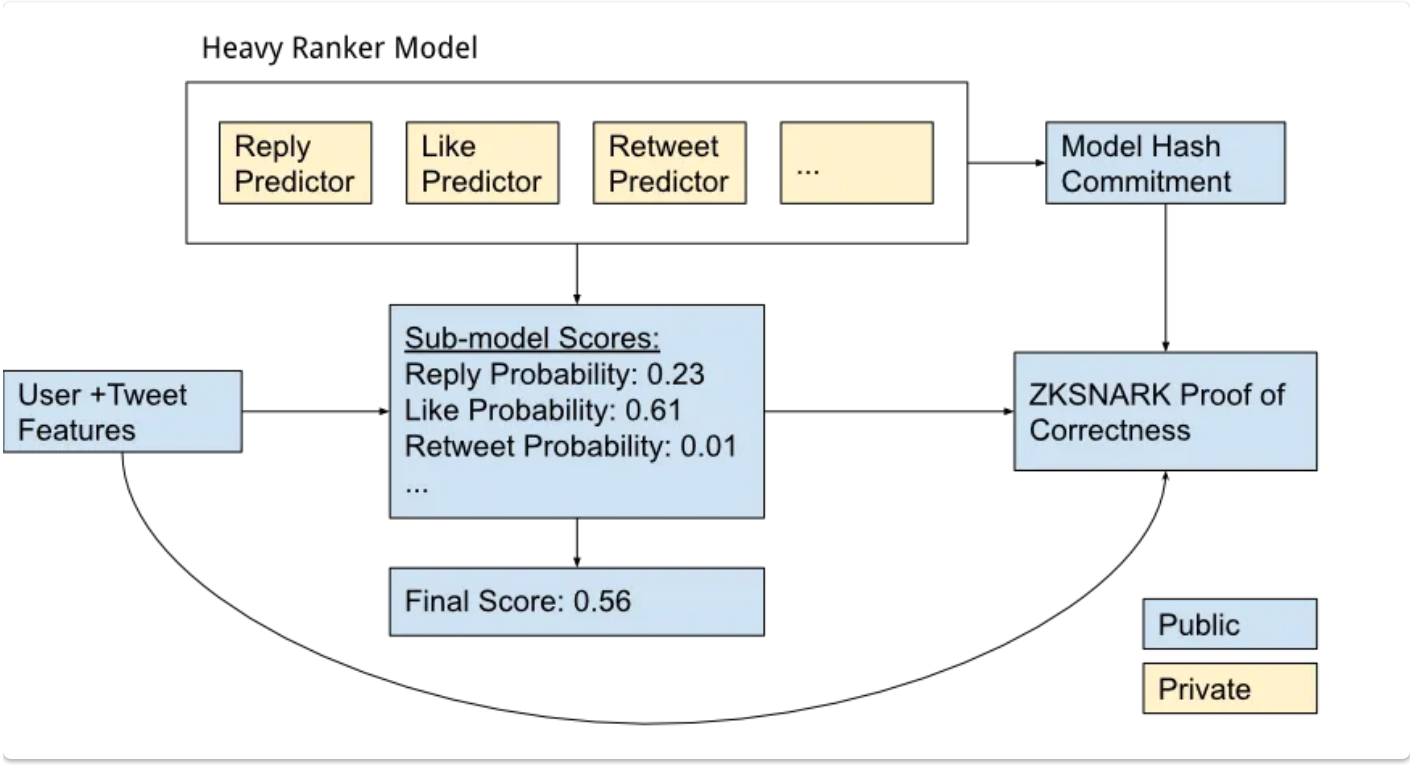
These are increadingly used in zk to reduce the cost of implementing some operations in circuits, for example simple range proofs.
The techniques used have improved, for example Caulk (video paper) and Cached Quotients (paper)
Previously the lookup table had to be the same as the size of the table, now we can have tables

that are independent of circuit size.
In Tensor Plonk we further optimise the tables by batching to reduce computation by 3x.

The final optimisation involves the commitment to the weights, instead of using a hash function, a KZG commitment of the weight vectors which can reduce the proving time by up to 10x.

Example with Twitter ranking

Twitter uses a ML model to select the tweets shown to you, this project uses zkML to confirm that this is running correctly.



Tensor Plonk provides improvements over other zkML frameworks

Proving the Twitter model currently takes 6 hours to prove for a single example using ezkl. Verifying the tweets published in one second (~6,000) would cost ~$88,704 on cloud compute hardware.

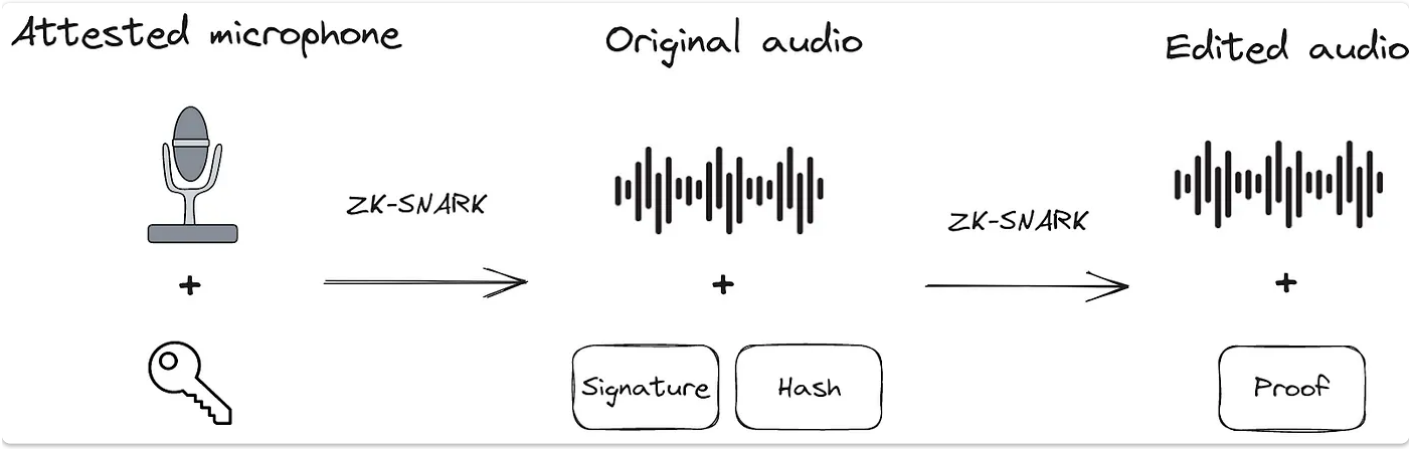
Using TensorPlonk, the proving cost would be ~$30.

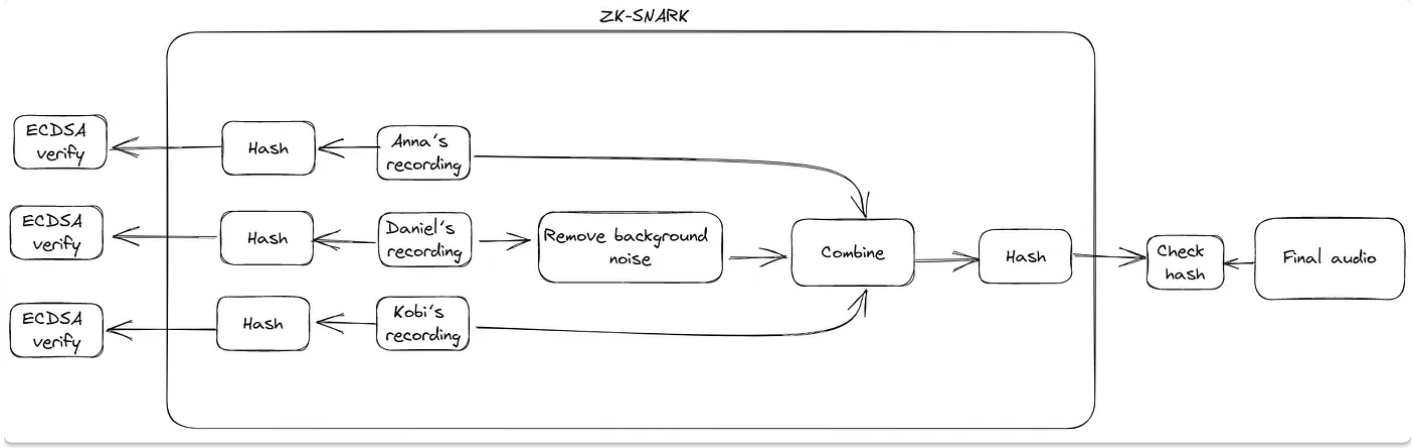
	TensorPlonk	ezkl (no hash)	ezkl (with hash)
Proving Time	6.7 seconds	1517.5 seconds	> 6 hours*
Verification Time	70 ms	250 ms	> 250 ms*
Proof Size	12.5 kb	88 kb	~800 kb*

Attested Audio

See Blog
and Podcast

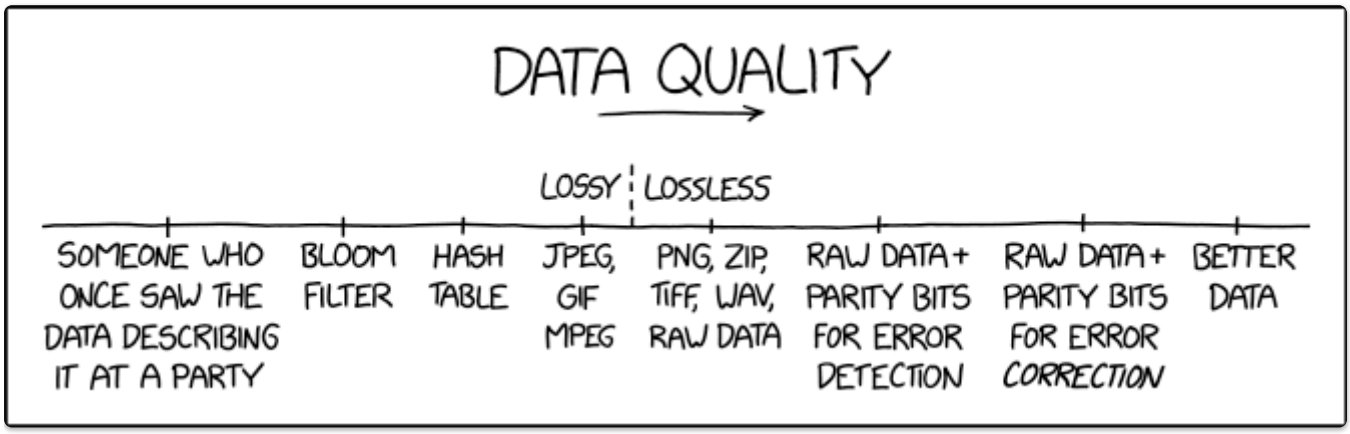
This was based on a project that generated answers to questions about zk using the voices of the presenters from the zk podcast.
The attested audio tried to show that a clip of audio recorded from the real presenters wasn't a generated clip.
In this example the real audio was edited to remove background noise, but SNARKs used to prove that this was being done correctly and was based on original audio.




ZK-SNARK
ECDSA verify
Hash
Anna's recording
ECDSA verify
Hash
Daniel's recording
Remove background noise
Combine
Hash
Check hash
Final audio
ECDSA verify
Hash
Kobi's recording

Bloom Filters

Oblig xkcd



A Bloom filter is a probabilistic, space-efficient data structure used for fast checks of set membership.
The Bloom filter principle: Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated.

Bloom filter answers are 'no' and 'maybe'
Bloom filters have the inverse behavior of caches

- bloom filter: miss == definitely not present, hit == probably present
- cache: miss == probably not present, hit == definitely present

The basic idea behind the Bloom filter is to hash each new element that goes into the data set, take certain bits from this hash, and then use those bits to fill in parts of a fixed-size bit array (e.g. set certain bits to 1). This bit array is called a bloom filter.

Later, when we want to check if an element is in the set, we simply hash the element and check that the right bits are 1 in the bloom filter. If at least one of the bits is 0, then the element definitely isn't in our data set! If all of the bits are 1, then the element might be in the data set, but we need to actually query the database to be sure. So we might have false positives, but we'll never have false negatives. This can greatly reduce the number of database queries we have to make.

For an interactive example, see

https://llimllib.github.io/bloomfilter-tutorial/

Zero Gravity

See Paper and introduction
The zero gravity project came first in the ZK Hack Lisbon in April since then it has received funding from PSE

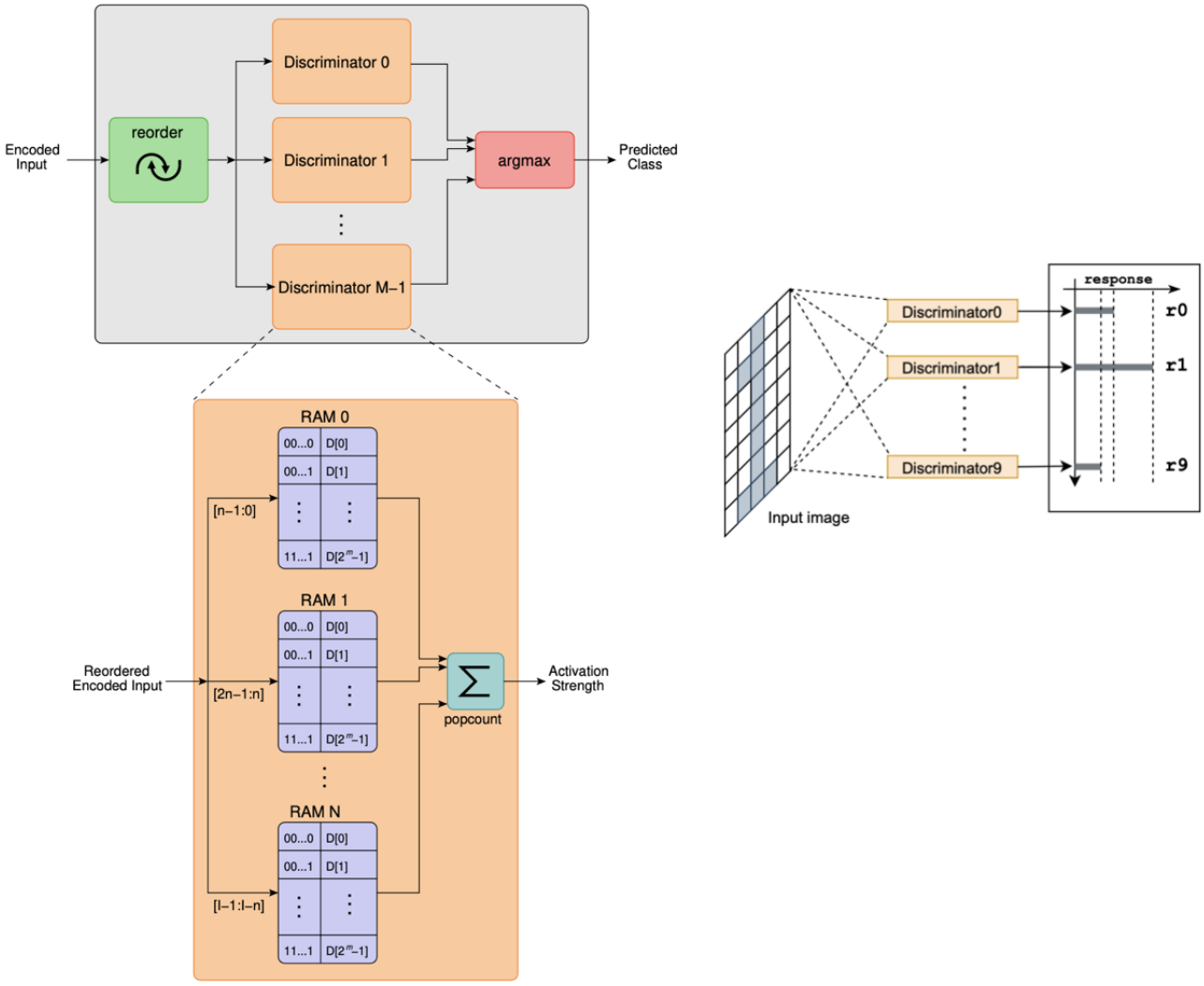
The article recognises the difficulties involved in reconciling the primitives used in zk and ML

"ZK operates at a fundamental level with modular arithmetic (i.e. with discrete values over a finite field) whereas neural nets and most machine learning models perform "smooth" operations on floating point numbers, called "weights". Existing approaches have attempted to bridge this divide by quantizing the weights

of a neural net, so that they can be represented as elements of the finite field. Care must be taken to avoid a "wrap-around" occurring in the (now, modular!) arithmetic of the quantized network, and weight quantization can only decrease model accuracy. "

Based on Weightless Neural Networks (WNNs) which use table lookups to perform inference. A WNN is entirely combinatorial. Its input is a bitstring (e.g. encoding an image), and their output is one of several predefined classes,

It learns from a dataset of (input, output) pairs by remembering observed bitstring patterns in a bunch of devices called RAM cells, grouped into "discriminators" that correspond to each output class. RAM cells are so called since they are really just big lookup tables, indexed by bitstring patterns, and storing a 1 when that pattern has been observed in an input string that is labeled with the class of this discriminator.
Bloom filters are to used to create a space efficient representation of the RAM cell.



An important point is that there is only one layer, which makes them less powerful than deep learning networks.

In Zero Gravity, the prover claims to know an input bitstring x such that the public model classifies it as class y

The input x can be treated as a private input, in which case the system is zero-knowledge: although inference does reveal something about x to the verifier (namely its corresponding

output class y), but this information is already contained in the statement being proved.

Comparison with DNNs

Recent WNN architectures have lower implementation cost than common neural networks, but are less accurate for common image recognition tasks.
A further drawback is that WNN architectures tend to have high memory requirements.

The project was further developed following a grant from the EF.
See blog

"This is a POC of an end-to-end platform for machine learning developers to seamlessly convert their TensorFlow Keras models into ZK-compatible versions. This all-in-one solution consists of three core components:"

- circomlib-ml: A comprehensive Circom library containing circuits that compute common layers in TensorFlow Keras.
- keras2circom: A user-friendly translator that converts ML models in Python into Circom circuits.

- ZKaggle: A decentralized bounty platform for hosting, verifying, and paying out bounties, similar to Kaggle, but with the added benefit of privacy preservation.

Halo2 implementation

See repo

You can use the WnnChip in your own circuits

See overview and example integration