Lesson 11

Today's topics

- Sumcheck protocol
- GKR
- IVC
- Folding Schemes
- Federated Machine Learning
- Aligned Layer

Sumcheck protocol

The sumcheck protocol was an early ('90s) idea which is finding new applications.
It is an interactive process between prover and verifier, to allow the prover to show they know the sum of a multivariate polynomial.
It is interesting in the oracle aspects of the protocol, and the fact that the verifier only needs to know the polynomial at the very end.

In addition to the sumcheck proving the sum, we can also use it to check that a vector is all zero elements, so it provides a useful tool

Process

From Paper

The prover wants to prove that she knows H the sum of a multivariate polynomial g

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_v \in \{0,1\}} g(b_1, \dots, b_v).$$

We assume here that the verifier has oracle access to g, i.e. can evaluate $g(r1, \dots, rv)$ for a randomly chosen vector $(r1, \dots, rv) \in Fv$ with a single query to an oracle

The process has v rounds, each round is linked to the previous, that is we say that if the 1st round is correct, then we will accept the next one etc.

Description of Sum-Check Protocol.

- Fix an $H \in \mathbb{F}$.
- In the first round, $\mathcal{P}$ sends the univariate polynomial

$$g_1(X_1) := \sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v).$$

  $\mathcal{V}$ checks that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, and that $H = g_1(0) + g_1(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$, and sends r_1 to $\mathcal{P}$.
- In the jth round, for $1 < j < v$, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial

$$g_j(X_j) = \sum_{(x_{j+1},\ldots,x_v)\in\{0,1\}^{v-j}} g(r_1, \ldots, r_{j-1}, X_j, x_{j+1}, \ldots, x_v).$$

  $\mathcal{V}$ checks that g_j is a univariate polynomial of degree at most $\deg_j(g)$, and that $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_j \in \mathbb{F}$, and sends r_j to $\mathcal{P}$.
- In Round v, $\mathcal{P}$ sends the univariate polynomial

$$g_v(X_v) = g(r_1, \ldots, r_{v-1}, X_v)$$

  to $\mathcal{V}$. $\mathcal{V}$ checks that g_v is a univariate polynomial of degree at most $\deg_v(g)$, rejecting if not, and also checks that $g_{v-1}(r_{v-1}) = g_v(0) + g_v(1)$.
- $\mathcal{V}$ chooses a random element $r_v \in \mathbb{F}$ and evaluates $g(r_1, \ldots, r_v)$ with a single oracle query to g. $\mathcal{V}$ checks that $g_v(r_v) = g(r_1, \ldots, r_v)$, rejecting if not.
- If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ halts and accepts.

Cost of the protocol

There is one round in the sum-check protocol for each of the v variables of g. The total communication is

$\sum_{i=1}^{v} deg_i(g) + 1$

$= v + \sum_{i=1}^{v} deg_i(g)$ field elements.

In particular, if $deg_i(g) = O(1)$ for all j, then the communication cost is $O(v)$ field elements. The running time of the verifier over the entire execution of the protocol is proportional to the total communication, plus the cost of a single oracle query to g to compute $g(r1, \ldots, rv)$.

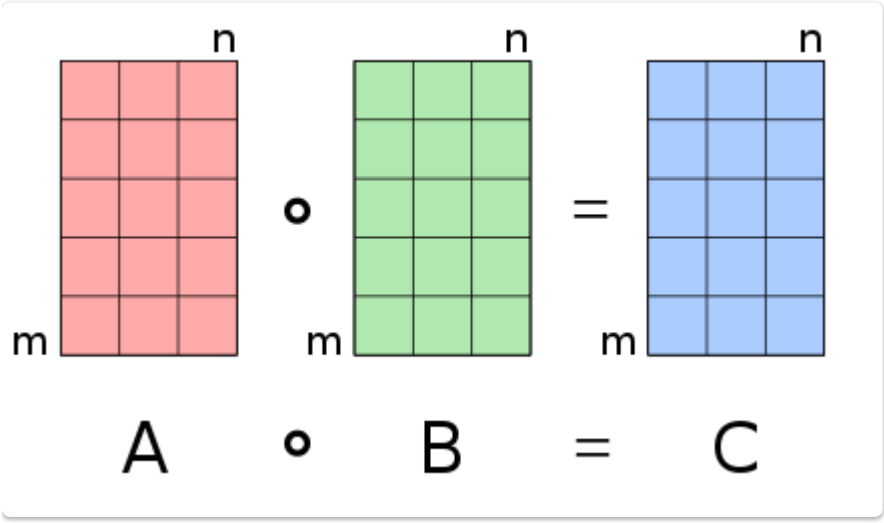
Resources

A detailed lecture from Alessandro Chiesa

Hadamard Product

See Definition

The hadamard product of 2 matrices of the same dimensions is a binary operation that returns a matrix of the multiplied corresponding elements.



$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 8 & -2 \end{bmatrix} \circ \begin{bmatrix} 3 & 1 & 4 \\ 7 & 9 & 5 \end{bmatrix} = \begin{bmatrix} 2 \times 3 & 3 \times 1 & 1 \times 4 \\ 0 \times 7 & 8 \times 9 & -2 \times 5 \end{bmatrix} = \begin{bmatrix} 6 & 3 & 4 \\ 0 & 72 & -10 \end{bmatrix}$$

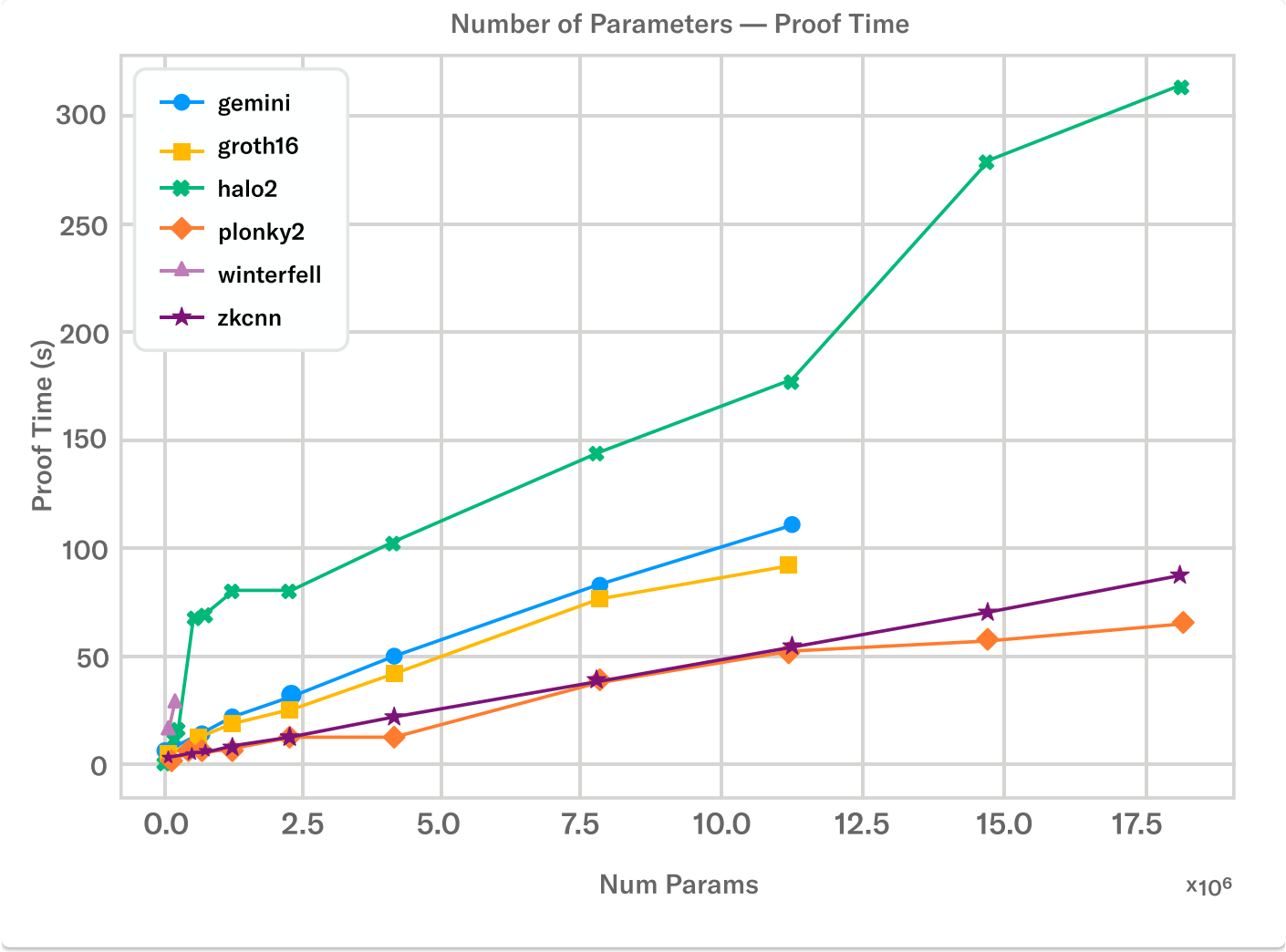
This is different from the normal matrix product.

GKR Introduction

See Video, screenshots are taken from the video.

Modulus believe that zkML is such a specialised computational regime, that it requires a special prover which is GKR.

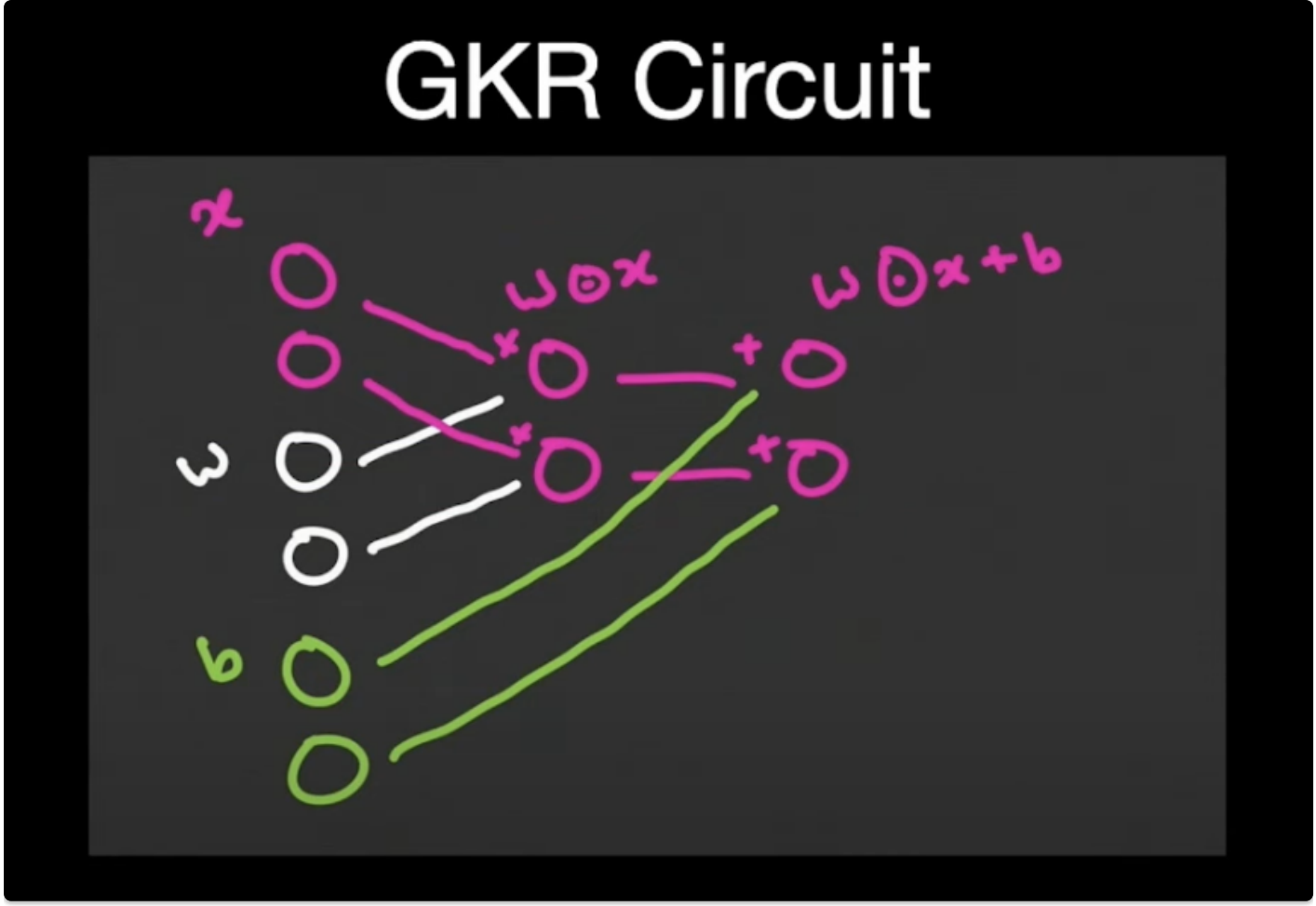
Proving Task/Regime	Proof System
Constraint/circuit flexibility; ease of development	UltraPlonk/Halo2, Circom/Groth16
Virtual machine program trace	AIR + STARKs
Proof Recursion	FRI + Plonky2
VDFs, IVC	Nova + other folding schemes
Machine Learning Inference	**GKR**



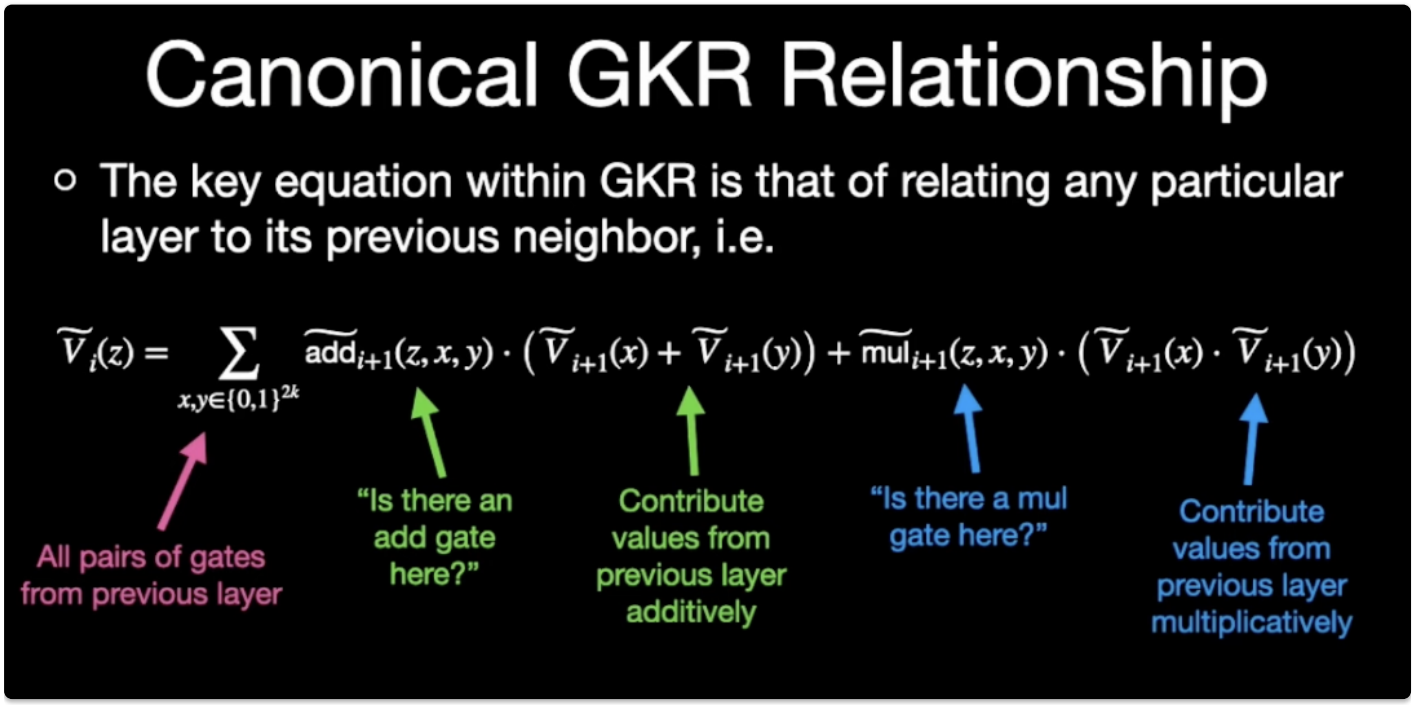
GKR was originally developed in 2008 in [this paper]

GKR is an interactive proof composed of layers, where the sumcheck protocol is used as we move between the layers.

When building a GKR circuit, we can represent the layers as multilinear polynomials.
A multilinear polynomial is a multivariate polynomial that is linear in each of the variables.


GKR Circuit
x
w
b
w⊙x
w⊙x+b

When building the relationships between the layers we use the concept of selectors (these are widely used in arithmetic circuits).
The selector is used to indicate whether we have an addition gate or a multiplication gate. It is a binary value that allows us to switch between the types of gates.



Sumcheck

$$\sum_{b_1,\ldots,b_k \in \{0,1\}^k} f(b_1, \ldots, b_k)$$

b_1 gets "bound" to a random challenge r_1

Summation over only $b_2, \ldots, b_n$

$$\sum_{b_2,\ldots,b_k} f(r_1, b_2, \ldots, b_k)$$

Each of the b_is are sequentially bound to r_is

Summation halves in size each round

$$\sum_{b_{i+1},\ldots,b_k} f(r_1, \ldots, r_i, b_{i+1}, \ldots, b_k)$$

Final round "oracle query"

$$f(r_1, \ldots, r_k)$$

Sumcheck Application

$$\widetilde{V}_i(z) = \sum_{x,y \in \{0,1\}^{2k}} \widetilde{\text{add}}_{i+1}(z,x,y) \cdot \left(\widetilde{V}_{i+1}(x) + \widetilde{V}_{i+1}(y)\right) + \widetilde{\text{mul}}_{i+1}(z,x,y) \cdot \left(\widetilde{V}_{i+1}(x) \cdot \widetilde{V}_{i+1}(y)\right)$$

- The verifier is able to reduce the above claim to an oracle query to the function within the summation:

$$\widetilde{\text{add}}_{i+1}(z,u,v) \cdot \left(\widetilde{V}_{i+1}(u) + \widetilde{V}_{i+1}(v)\right) + \widetilde{\text{mul}}_{i+1}(z,u,v) \cdot \left(\widetilde{V}_{i+1}(u) \cdot \widetilde{V}_{i+1}(v)\right)$$

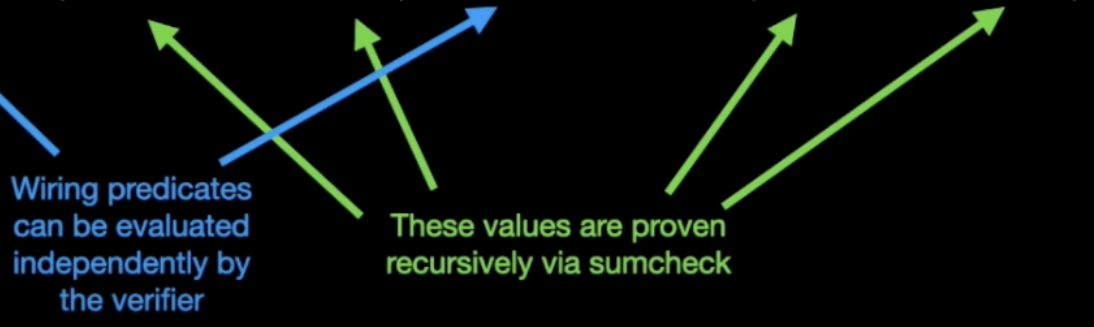
Random "left hand side" value from the previous layer

Random "right hand side" value from the previous layer

Sumcheck Application

How does the verifier actually check the oracle query?

$$\widetilde{\text{add}}_{i+1}(z,u,v)\cdot\left(\widetilde{V}_{i+1}(u)+\widetilde{V}_{i+1}(v)\right)+\widetilde{\text{mul}}_{i+1}(z,u,v)\cdot\left(\widetilde{V}_{i+1}(u)\cdot\widetilde{V}_{i+1}(v)\right)$$



Sumcheck —> GKR Recap

- The prover claims to the verifier that the output of a circuit has some value $\widetilde{V}_0(r_1,\ldots,r_k)=y$.
- The verifier reduces this claim to an equivalent claim on the next circuit layer $\widetilde{V}_1(r'_1,\ldots,r'_k)=y'$ via sumcheck.
- This process repeats until the verifier is left with a claim on the input layer $\widetilde{V}_d(r_1^{(d)},\ldots,r_k^{(d)})=y^{(k)}$, which is verified via a polynomial commitment opening.

"~~I am~~ GKR is speed"

- Sumcheck works purely over field elements, and is *extremely parallelizable* on the prover end.
- The prover performs a polynomial commitment on only the circuit's *input layer*.
- Time-optimal sumcheck protocols exist for common linear operations (e.g. matrix multiplication)
- (Parallel) proving can be *faster* than (sequential) computing!!

Structured Circuits

- As described in [Tha13], GKR heavily rewards *structured* circuits.
- Such structure predicates that the outputs of a circuit layer (e.g. at index i) only depend on inputs from the previous layer which are a direct function of i.
- The verifier is able to evaluate the wiring predicate of such a circuit in logarithmic time!
- GKR supports dataparallel operations natively.

Remainder

See article for further details and sign up form for their 4 week crash-course on Remainder.

GKR Resources

Thaler Book Section 4.6
ZK Study Club video (audio is poor)
Article about GKR
Video from ZKProof

IVC

Incrementally verifiable computation

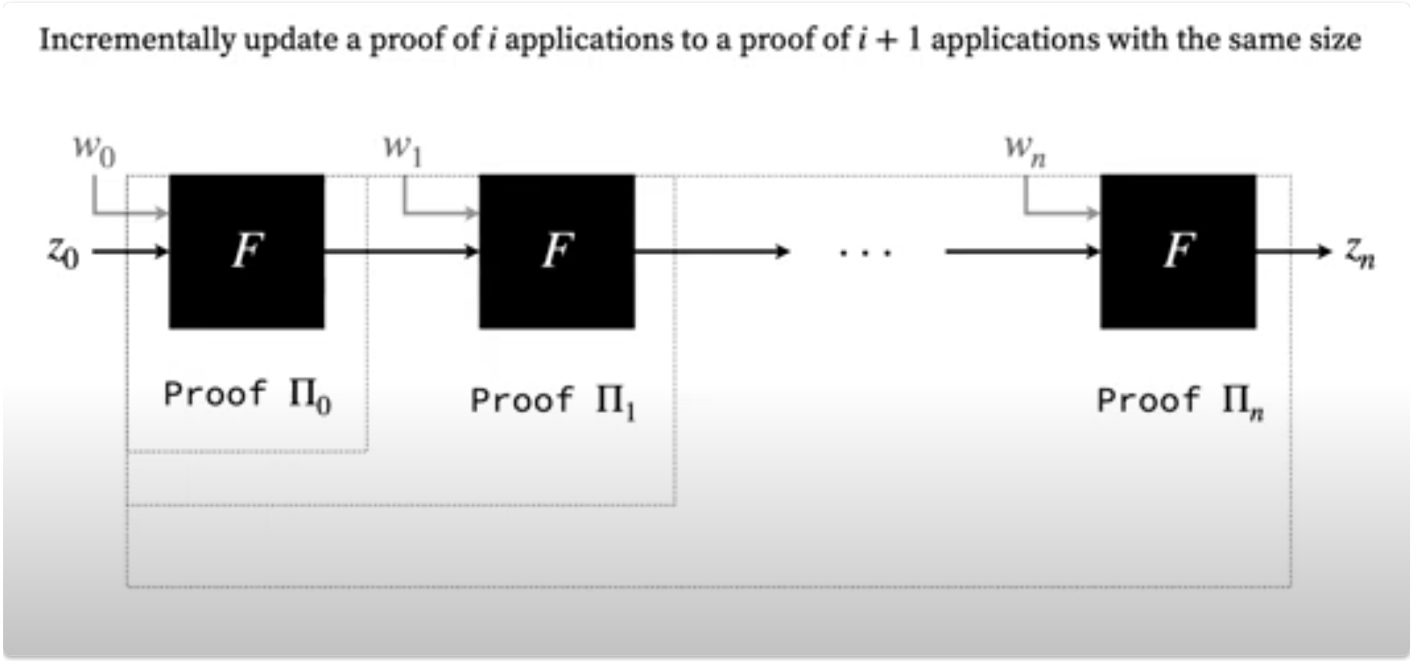
Valiant produced an influential paper in 2008 which introduced a composable proof system :
" there is an algorithm for merging two proofs of length *k* into a proof of the conjunction of the original two theorems in time polynomial in *k*, yielding a proof of length *exactly k*."
"... showing that knowledge can be “traded” for time and space efficiency in non interactive proof systems."

Folding Schemes introduction

With earlier SNARKS if we wanted to use recursion, we needed to have a verifier as part of our circuit, a complex and potentially non optimal technique.

Halo improved on this by allowing some of the verification algorithm to be taken out of the circuit.

Folding schemes take this much further, with virtually all of the verification outside of the circuit.
Folding schemes are often used for IVC



Folding Scheme definition

From the Nova paper

A folding scheme, a weaker, simpler, and more efficiently realizable primitive, which reduces the task of checking two instances in some relation to the task of checking a single instance.

The prover and verifier hold two N-sized NP instances, and the prover in addition holds purported witnesses for both instances. The protocol enables the prover and the verifier to output a single N-sized NP instance, which we refer to as a folded instance.

Furthermore, the prover privately outputs a purported witness to the folded instance using purported witnesses for the original instances. Informally, a folding scheme guarantees that the folded instance is satisfiable only if the original instances are satisfiable.

Several existing techniques exhibit the two-to-one reduction pattern of folding schemes. Examples include the sumcheck protocol and

the split-and-fold techniques in inner product arguments

A general approach involves a combination of the instances, and commitments to the 'error' or 'cross product' terms that this produces.

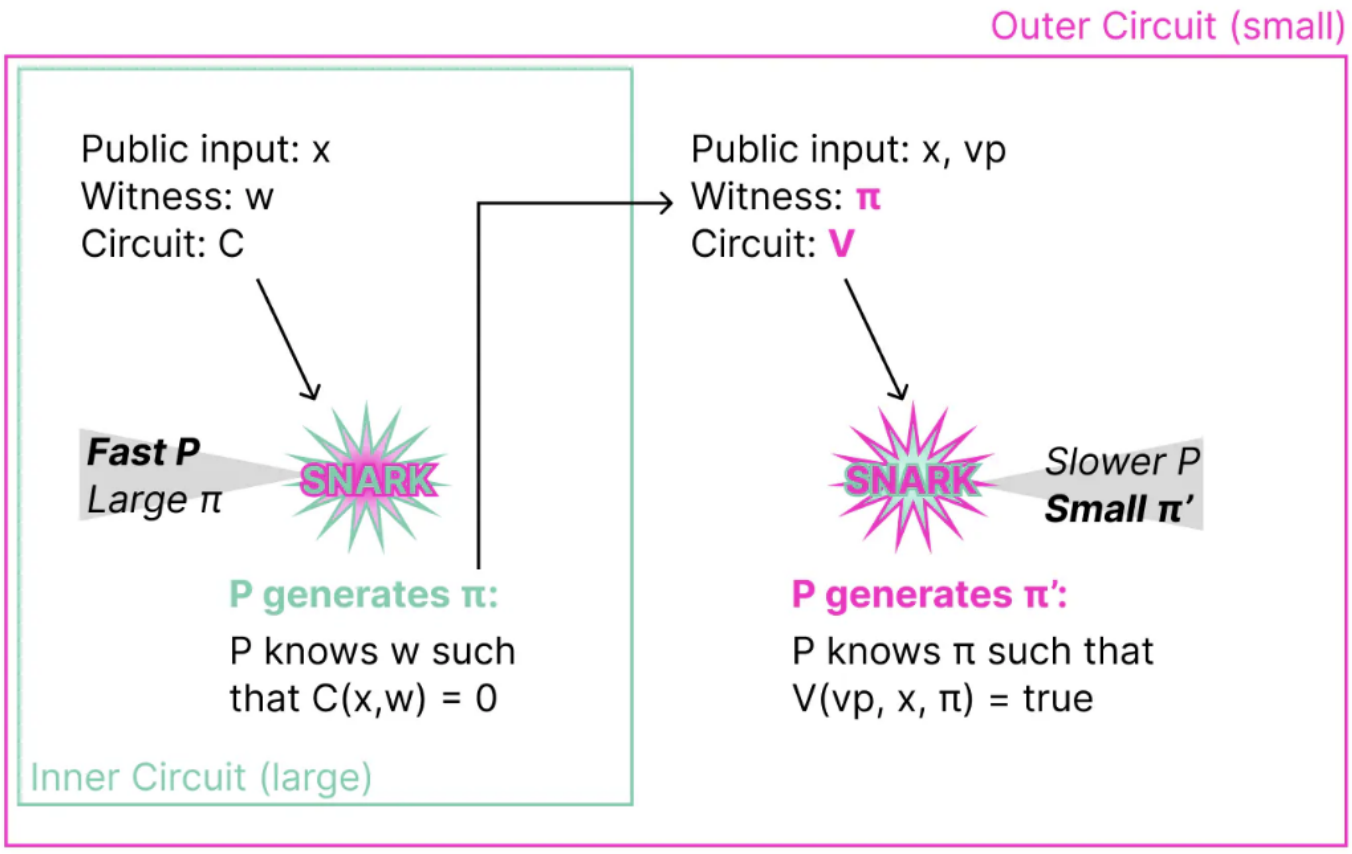
Recursion and aggregation

From Taiko article

Circuits - inner and outer

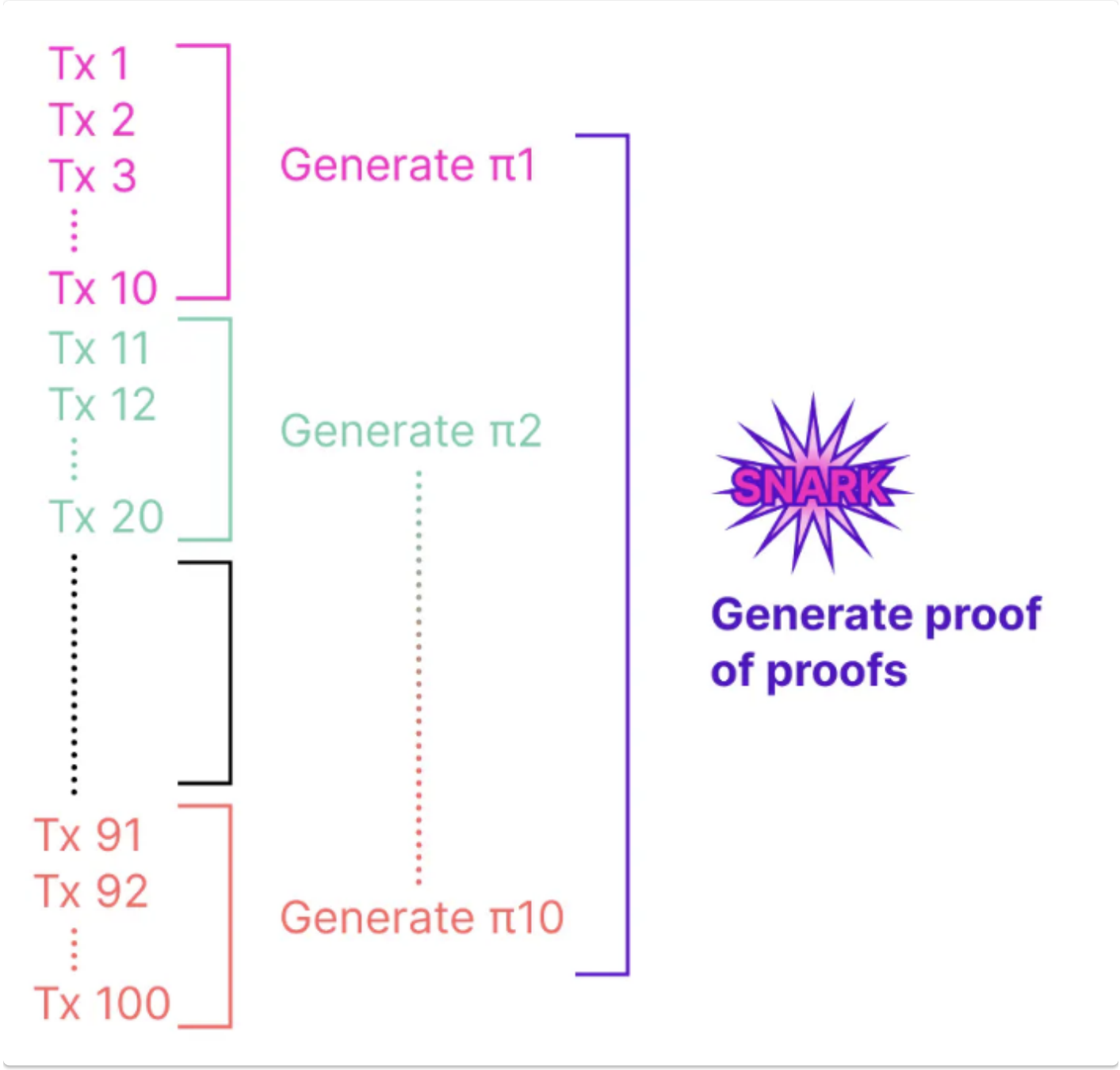
1. Inner Circuit (large): Prover proves that they know the witness. At this stage, proof generation is fast, but the proof is large (except for the case when the proof size is a constant);
2. Outer Circuit (small): Prover proves that they know the proof. At this stage, proof generation is slower (but it's not crucial as in most cases the circuit is much smaller than the first), but the proof is small.

P – Prover
V - Verifier
π – proof (SNARK)
vp – public parameters for V

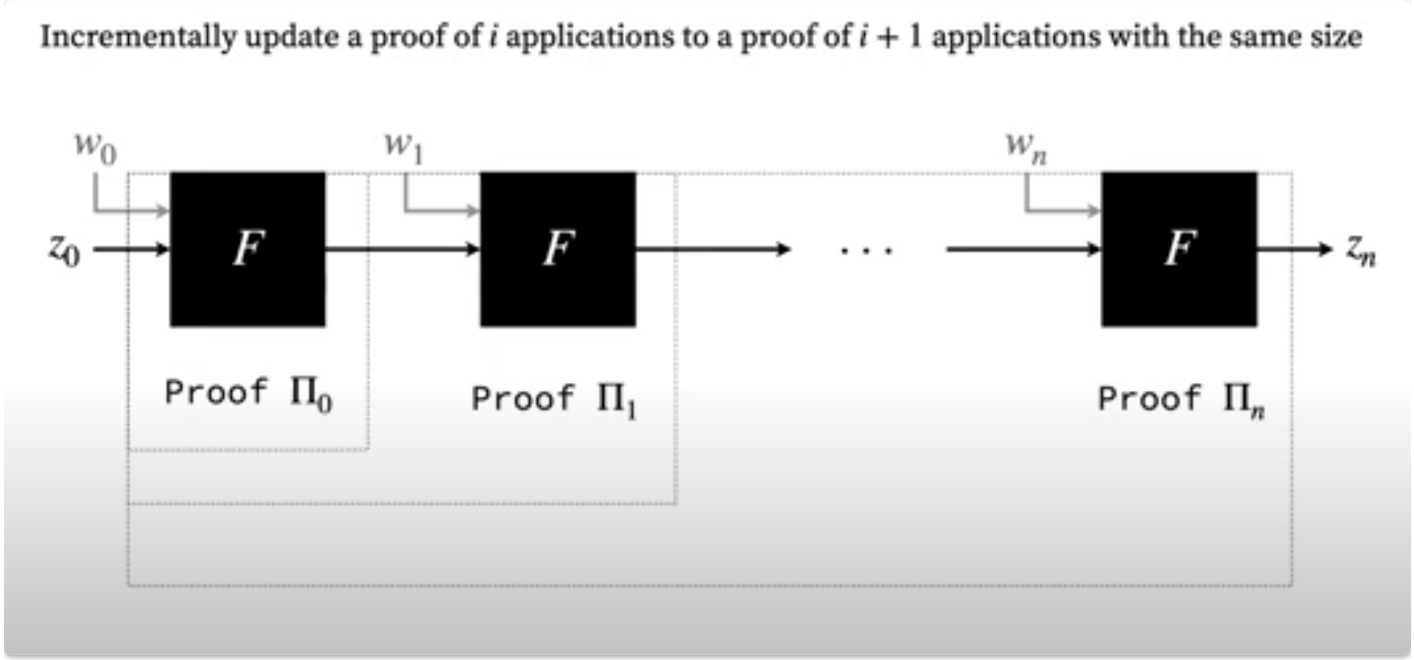


Proof aggregation

For example, as done in zk rollups



IVC - incrementally update proofs

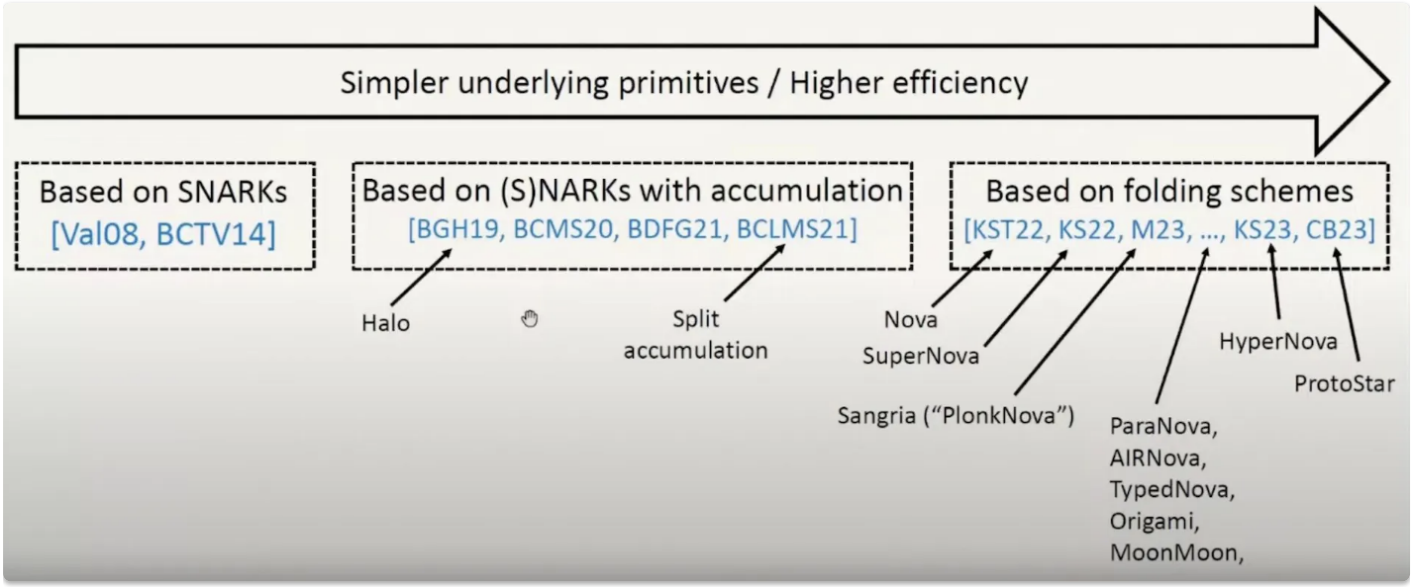


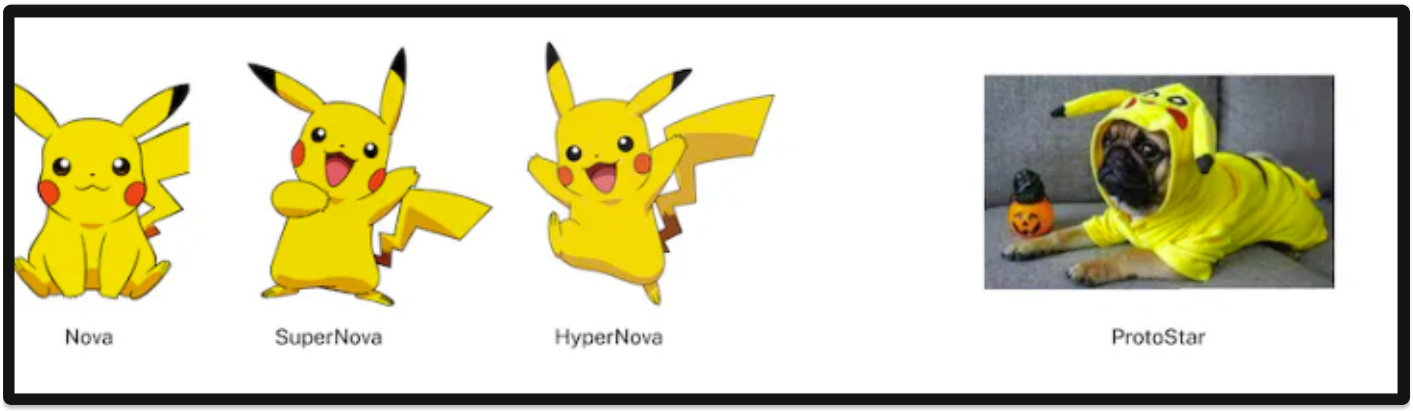
Folding instead of recursion

Advantages of folding for a function F that we are iterating over.

- Doesn't perform any polynomial divisions or multiplications (ex., via FFTs that require much memory and are computationally heavy);
- Works with *any* type of elliptic curves (ex., secp, secq, Pasta, etc.);
- F is specified with R1CS;
- No trusted setup;
- Compared to the recursion overhead, folding is expected to be 5-50x faster than Groth16, PLONK, Halo, etc. Prover time is dominated by two multi-exponentiations of size O(C), where C = |*F*|;
- Verifier Circuit is const-size (two scalar multiplications);
- Proof size is O(log C) group elements (few KBs).

Main Projects





NOVA

See Paper

See article for an in depth description.

See Video from Justin Drake

If our constraints were all linear, folding would be simpler, however R1CS is not linear.
Nova introduced the idea of 'relaxed' R1CS, by adding additional parameters to the instance.
With this new form, we can take linear

combinations of the instances, to allow them to be folded together.

Sangria

See paper
Nova allowed folding for R1CS arithmetisation, Sangria provides folding for PLONK. Again this requires us to 'relax' the gate constraints by adding extra terms. Fortunately copy constraints do not need to be changed.

Protostar

See paper
This relies on accumulation , a simple yet powerful primitive that enables incrementally verifiable computation (IVC) without the need for recursive SNARKs

The verifier is expressed as a series of equations (more formally, an *algebraic* check). These folding schemes are based on the techniques of Nova and Sangria and introduce optimisations to avoid expensive commitments to cross-terms (these are the terms that are handled by the 'relaxed' format of

arithmetisation), especially when dealing with high degree gates.

Supernova

This improves on Nova in the case where we are encoding steps of a VM, originally the prover would end up paying for operations that were supported , even if they were not being invoked. In Supernova we can reduce this to only paying for the operations that are invoked.

Hypernova

See paper
"A distinguishing aspect of HyperNova is that the prover’s cost at each step is dominated by a single multi-scalar multiplication (MSM) of size equal to the number of variables in the constraint system, which is optimal when using an MSM-based commitment scheme."
This is Nova using ccs and also using a VDF

Protogalaxy

See Paper
This builds on ideas from ProtoStar to create a folding scheme where the recursive verifier's marginal work, beyond linearly combining

> witness commitments, consists only of a logarithmic number of field operations and a constant number of hashes.

Resources

See awesome repo

Federated Machine Learning

See this tutorial

See paper
See this guide
See this interactive introduction

Similar to zkML in its desire for privacy, federated machine learning uses different techniques.
A typical approach would be that a number of users have data on which they would like to collaboratively train a model while still keeping their data private.
To achieve this secure multi party computation or FHE can be used.

Typical steps are

1. After initialization, the global model parameters are transferred to local clients in each iteration
2. Local clients update their local model parameters
3. The global server gathers local model parameters to update global parameters

4. These steps are repeated until local and global parameters converge

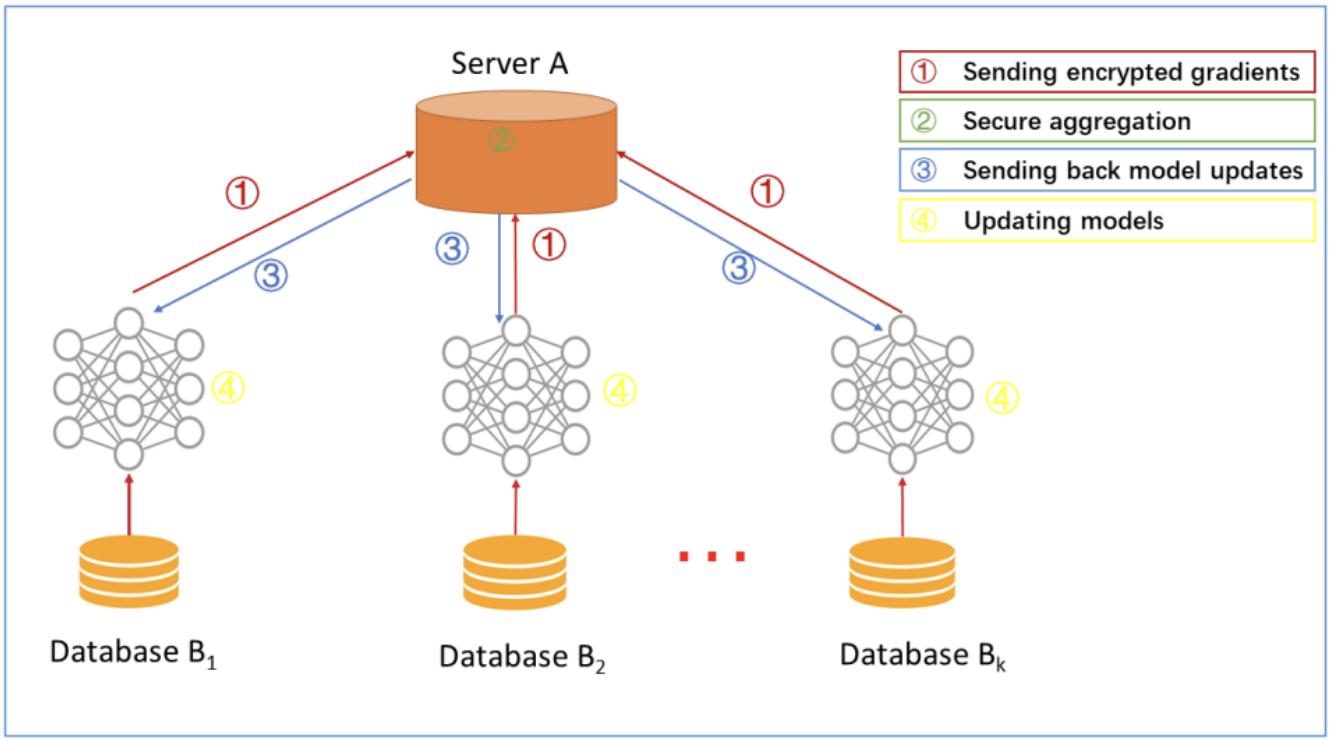


Fig. 3. Architecture for a horizontal federated learning system

In a typical federated training scenario, we are dealing with potentially a very large population of user devices, only a fraction of which may be available for training at a given point in time. This is the case, for example, when the client devices are mobile phones that participate in training only when plugged into a power source, off a metered network, and otherwise idle.

This paper
gives an practical method for federated learning of deep networks.

Types of Algorithms

Federated stochastic gradient descent (FedSGD)

In FedSGD, the central model is distributed to the clients, and each client computes the gradients using local data. These gradients are then passed to the central server, which aggregates the gradients in proportion to the number of samples present on each client to calculate the gradient descent step.

Federated averaging (FedAvg)

Federated averaging is an extension of the FedSGD algorithm.

Clients can perform more than one local gradient descent update. Instead of sharing the gradients with the central server, weights tuned on the local model are shared. Finally, the server aggregates the clients' weights (model parameters).

Federated learning with dynamic regularization (FedDyn)

Regularization in traditional machine learning methods aims to add a penalty to the loss function to improve generalization. In federated learning, the global loss must be computed

based on local losses generated from heterogeneous devices.

Due to the heterogeneity of clients, minimizing global loss is different than minimizing local losses. Therefore, FedDyn method aims to generate the regularization term for local losses by adapting to the data statistics, such as the amount of data or communication cost. This modification of local losses through dynamic regularization enables local losses to converge to the global loss.

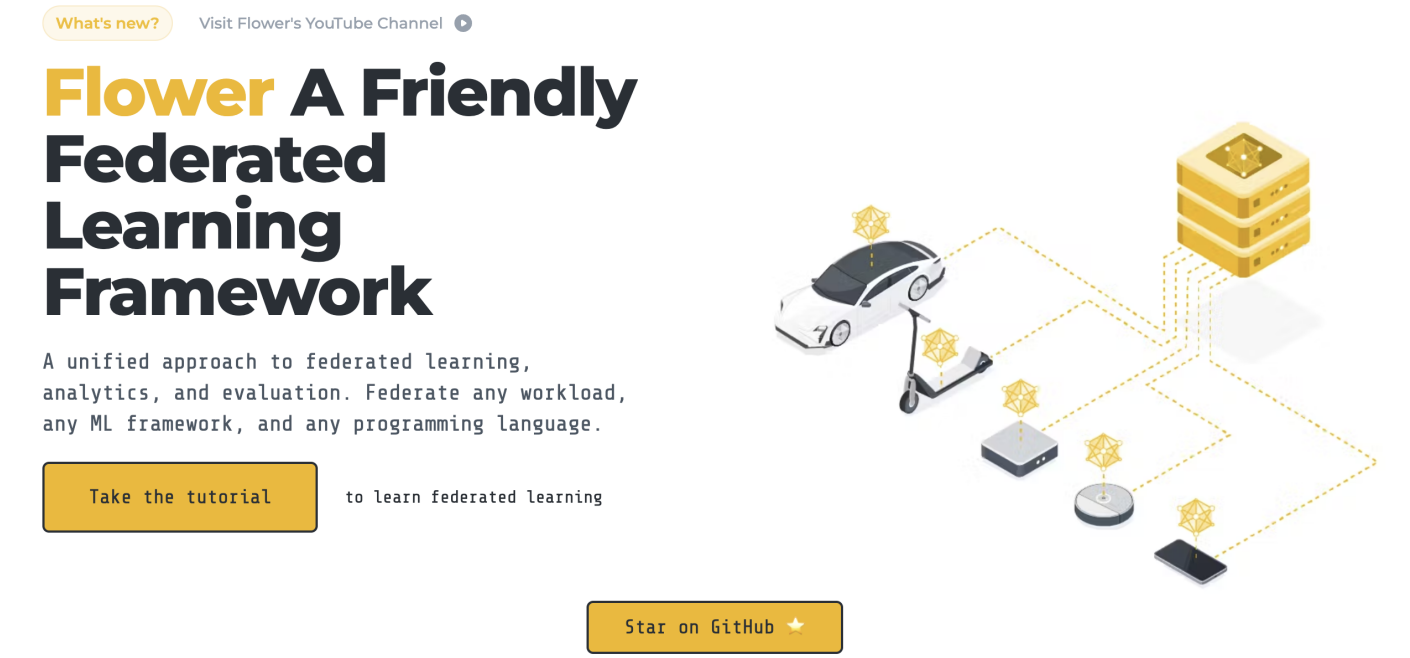
Federated Learning with Tensor Flow

See Docs

Tutorial

See instructions

Flower.ai



See Docs

See Blog

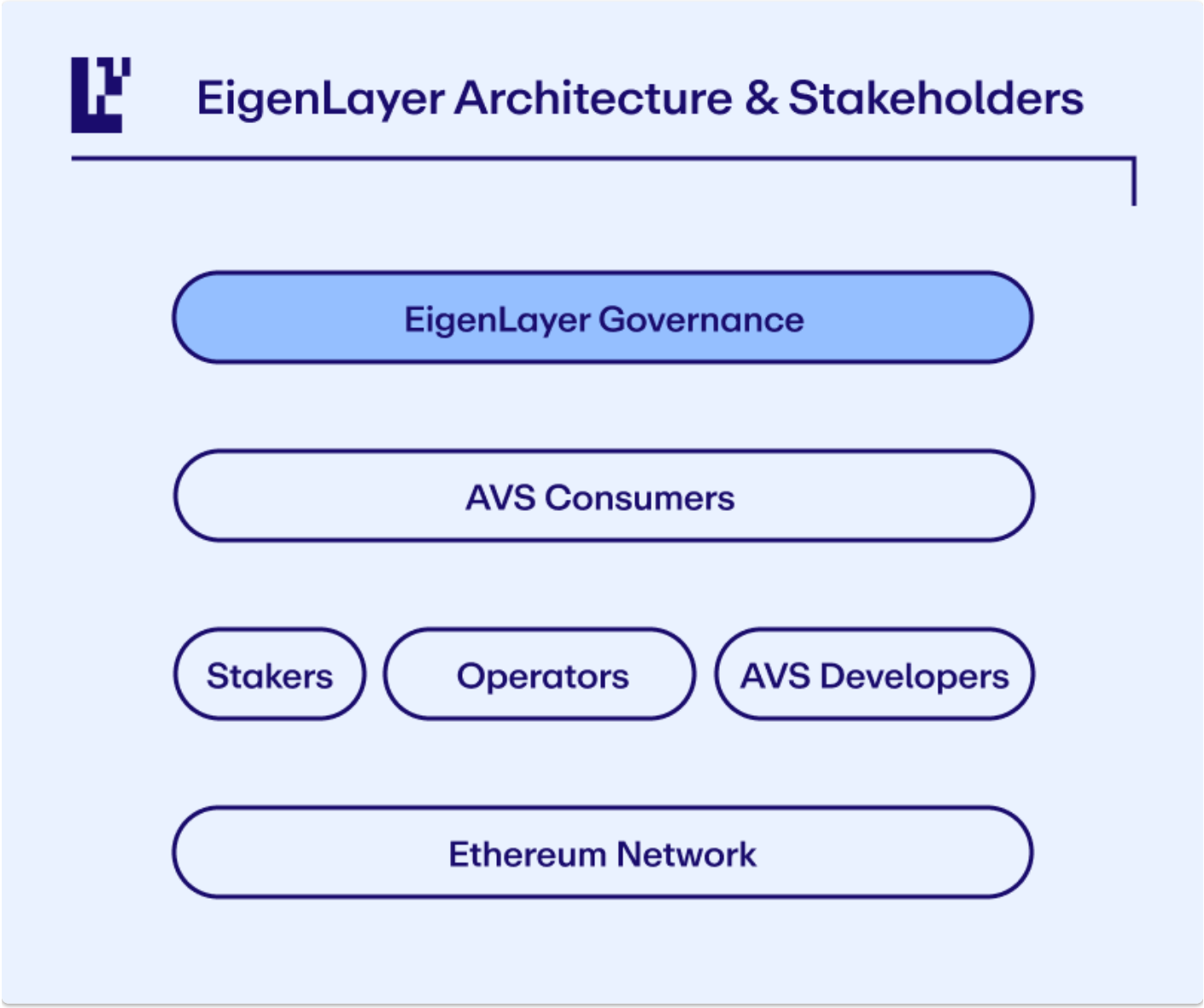
See Example Projects

See Repo

EigenLayer introduction

See Docs

EigenLayer is a protocol built on Ethereum that introduces restaking, users that stake ETH natively or with a liquid staking token (LST) can opt-in to EigenLayer smart contracts to restake their ETH or LST and extend cryptoeconomic security to additional applications on the network to earn additional rewards.

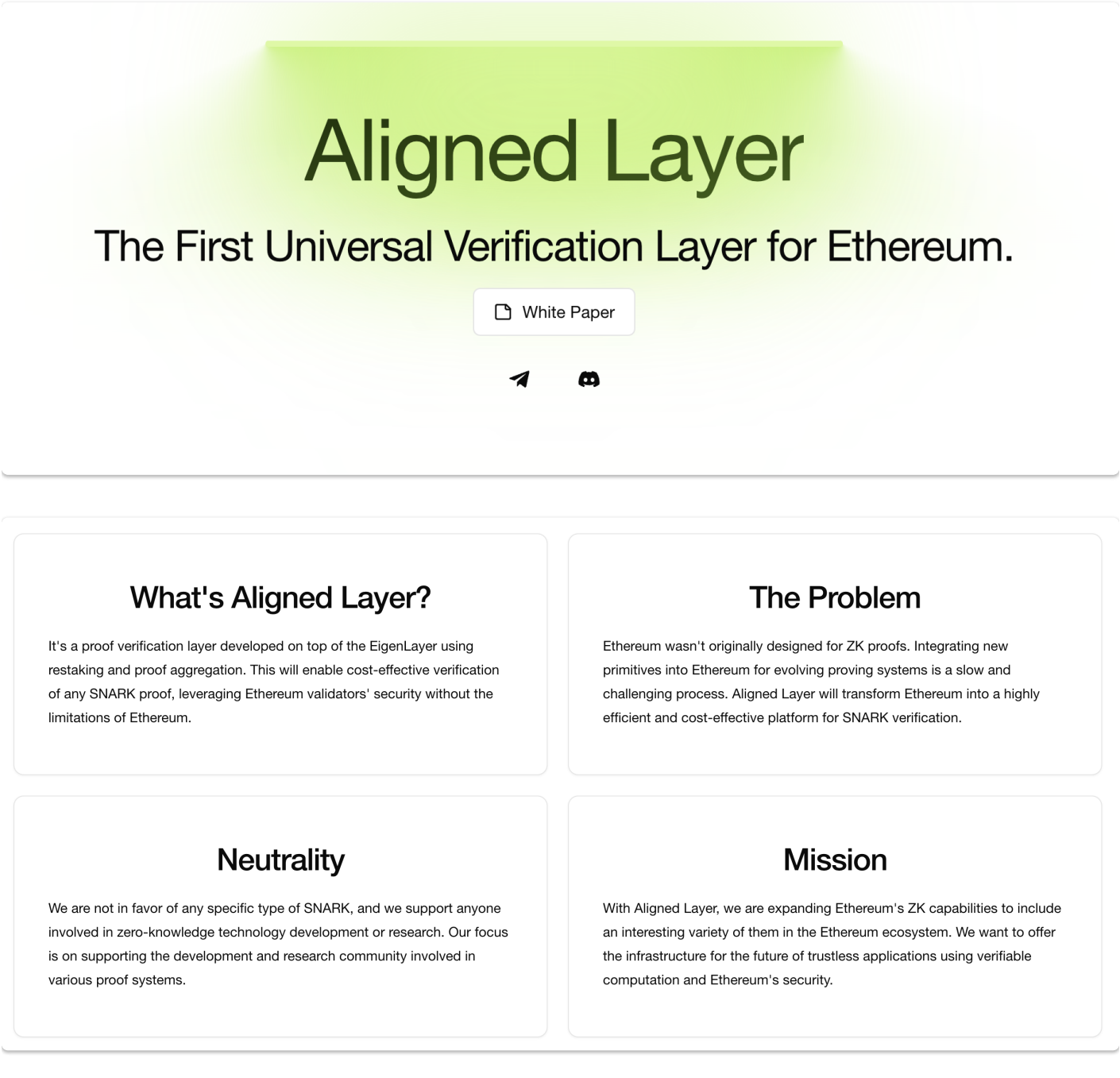


- Restaking enables stakers to restake their Native ETH or Liquid Staking Tokens (LST) to provide greater security for services in the EigenLayer ecosystem, known as Actively Validated Services (AVSs).
- Operators are entities that help run AVS software built on EigenLayer. They register in EigenLayer and allow stakers to delegate to them, then opt in to provide various services (AVSs) built on top of EigenLayer.
- Delegation is the process where stakers delegate their staked ETH to operators or run validation services themselves, effectively becoming an operator. This process involves a double opt-in between both parties, ensuring mutual agreement. Restakers retain agency over their stake and choose which AVSs they opt-in to validate for.
- Actively Validated Services (AVSs) are services built on the EigenLayer protocol that leverage Ethereum's shared security.
    - Operators perform validation tasks for AVSs, contributing to the security and

integrity of the network.

- AVSs deliver services to users (AVS Consumers) and the broader Web3 ecosystem.

Aligned Layer



See white paper

Aligned Layer will be a layer for verification and aggregation on top of Eigen Layer.
This will enable cost-effective verification of any SNARK proof, leveraging Ethereum validators' security.
The layer will be SNARK agnostic.

The verification layer will be a decentralised network of verifiers secured by restaking and proof of aggregation.

Aligned Layer rents the security offered by Ethereum to verify zero knowledge proofs. Several verifiers can be deployed to check any type of proof quickly.

Components

• Aligned Layer: It receives proofs from different proof systems, verifies them, sends the final result to Ethereum, and posts the data into a DA layer.
• Data Availability Layer (DA): provides storage for the different proofs.
• General Prover/Verifier: Every several days, takes the proofs from the DA layer and generates
a proof of the verification of all the proofs. The general prover can be based on the SP1, Risc0 or Nexus's virtual machine, which are virtual machines that can prove general Rust code. The proof of verification of the proofs is done using the corresponding verifier codes in Rust.

Diagrams from the white paper

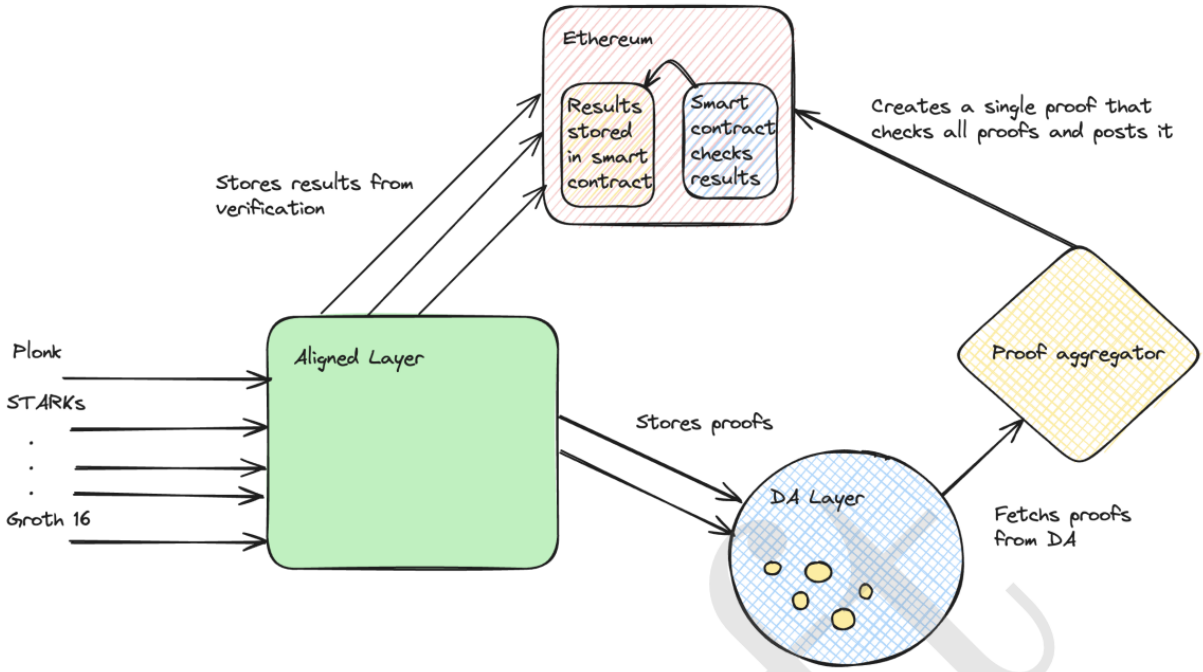
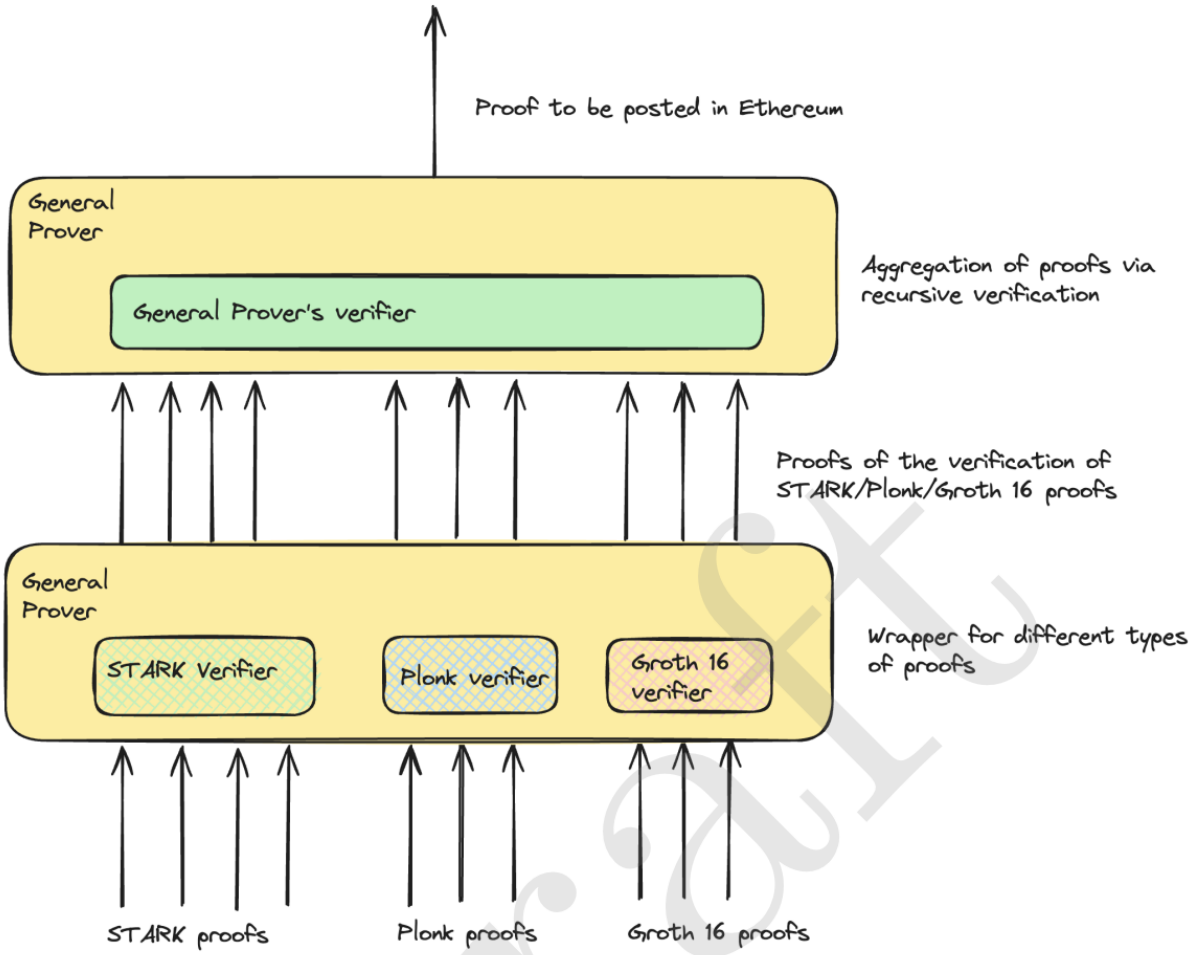


Figure 1: Core components



System	Groth 16[9]	Plonk+KZG[10]	STARKs[8]	HyperPlonk[11]	Binius[12]
Proof size	Constant	Constant	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\sqrt{n})$
Verification time	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell)$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\ell)$	$\mathcal{O}(\sqrt{n})$
Arithmetization	R1CS	Plonkish	AIR	Plonkish	Plonkish
Polynomials	univariate	univariate	univariate	multivariate	multivariate
Field	Curve-specific	Curve-specific	small fields	Curve-specific	Binary
Post quantum?	No	No	Yes	No	Yes
Prover time	$\mathcal{O}(n \log n)$[13]	$\mathcal{O}(n \log n)$[11]	$\mathcal{O}(n \log n)$[11]	$\mathcal{O}(n)$[11]	$\mathcal{O}(n)$[11]

Table 2: Comparison proof systems. ℓ is the size of the public inputs and n the length of the program/number of gates

Proposed Use cases

- Soft finality for Rollups and Appchains.
- Fast bridging.
- New settlement layers (use Aligned + EigenDA) for Rollups and Intent based systems
- P2P protocols based on SNARKs such as payment systems and social networks.
- Alternative L1s interoperable with Ethereum
- Verifiable Machine Learning
- Cheap verification and interoperability for Identity Protocols
- ZK Oracles
- New credential protocols such as zkTLS based systems.
- ZK Coprocessor
- Encrypted Mempools using SNARKs to show the correctness of the encryption.
- Protocols against misinformation and fake

news.

- On-chain gaming

Timescale

2024

April - private testnet

May - public testnet

September - mainnet launch