

NOYA: An Innovative ZKML Implementation

Omar Daouk

Hadi Esna

Pablo Albani

Abstract

NOYA utilizes Machine Learning (ML) models with Zero-Knowledge Machine Learning (ZKML) to predict yields across different chains, protocols, and assets while minimizing the required trust in strategy execution. Integrating ML and ZK allows for the development of safer, more transparent, and more efficient Decentralized Finance (DeFi) protocols.

Introduction

The DeFi landscape has seen explosive growth since 2020, pushing cryptocurrency users to look for ways to maximize yield on their investments. As the DeFi ecosystem continues to expand and mature, complexities surrounding yield farming, liquidity provision, and cross-chain transactions have increased significantly.

Traditional yield farming aggregators [1] usually operate on one chain and protocol. However, NOYA is a non-custodial, permissionless protocol that offers users risk-adjusted yields across different chains, protocols, and assets.

NOYA uses sophisticated artificial intelligence (AI) to predict yields and Zero-Knowledge Machine Learning (ZKML) [2][3] for safe strategy execution.

Architecture

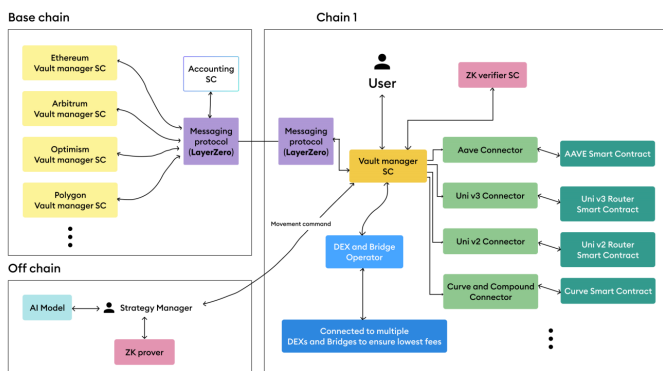


Figure 1: NOYA's architecture **

NOYA uses a base chain to handle the accounting of the users' shares. NOYA "vault manager" and "accounting" smart contracts are connected via a secure messaging protocol, LayerZero. This approach allows NOYA to securely use DeFi platforms on any chain the vault is deployed on. The architecture of NOYA can be divided into two sections, On-chain and Off-chain.

On-chain

- Accounting Smart Contracts:

On the base chain, a smart contract operates as the accountant of the system. The basechain is the operational backbone, managing user shares, deposits, and withdrawals. For instance, when a deposit is made on any other chain, such as Ethereum or Arbitrum, it relays a message via a secure messaging protocol to the base chain, updating the user's account. This ensures users have a comprehensive and precise overview of their investments across all chains.

The tokenized deposits are a modified version of ERC4626. While we retain the same computational framework of the standard, we've adapted the token handling capability to accommodate multiple tokens across multiple chains – an aspect that ERC4626 does not support.

- Omnichain Vault managers:

These are another set of smart contracts that allow the strategy manager to deposit and manage positions on various protocols while keeping the funds safe. These smart contracts are responsible for receiving users' funds, verifying the actions of the strategy manager, and preventing any fraudulent actions that could be done.

Users can deposit their assets on any chain that suits their preferences. For instance, if a user deposit 10,000 USDT or ETH on Ethereum, the smart contract that receives this transaction needs to report it to the base chain, which is responsible for the accounting process.

The following formula determines the calculation of deposits:

$$sharesReceived = assetsDeposited * totalSupply * totalAssets$$

** Higher resolution image in the Annex

And withdrawals are calculated using the following:

$$assetsReceived = sharesBurnt * totalAssets / totalSupply$$

- Connectors:

For each protocol, we employ connectors that allow for secure integration with NOYA. These connectors work as intermediates, connecting them with different protocols while maintaining the integrity and security of transactions.

The connector interface consists of the following key functions:

- **Deposit:** This enables strategy managers to allocate funds to a specific protocol.
- **Withdraw:** This function allows the retrieval of allocated funds from a protocol.
- **Claim:** This action enables claiming rewards or yields generated by the deposited funds.
- **Manage:** This allows strategy managers to adjust strategies per market dynamics and optimize returns.
- **TVL:** This offers real-time tracking of the total value of assets locked within each protocol.

These features, designed within the connector interface, enable a high degree of flexibility while maintaining a solid security framework.

- Bridge and DEX interface:

The protocol minimizes fees and optimizes routing by using different DEXs and bridges. These smart contracts provide a universal interface for bridging and exchanging tokens.

We've implemented rigorous security measures to mitigate inherent risks associated with potential DEX manipulation or fraudulent strategy managers. Specifically, for each action that interacts with DEXs, we compare the output amount with multiple trusted price sources. This includes price oracle networks like Chainlink and sizable liquidity pools on Uniswap. Cross-verifying with these external, reliable sources ensures the system is not exposed to price manipulation attacks.

- Messaging interface:

Keeping the system's integrity requires a Secure Messaging Protocol to ensure the seamless connectivity of smart contracts across multiple chains. For example, when a user initiates a deposit or requests a withdrawal, the base chain is promptly notified to keep an accurate record of the event.

Similarly, when a strategy manager attempts to transfer funds from one chain to another or when we accumulate rewards from an investment, the base chain is notified to update the accounting system. This continuous flow of information helps maintain an up-to-date record of all transactions, ensuring the system's integrity and fostering a high degree of transparency.



Figure 2: NOYA's Messaging Protocol **

NOYA utilizes the "LayerZero" protocol as its core messaging protocol. The protocol's design incorporates two essential roles: the oracle and the relay.

In terms of the oracle, we plan to employ the Polyhedra implementation. This decision is primarily due to Polyhedra's use of advanced technology for creating zero-knowledge (ZK) proofs of the message. This enhancement allows NOYA messages to be transferred to the destination chain faster, cheaper, and more secure.

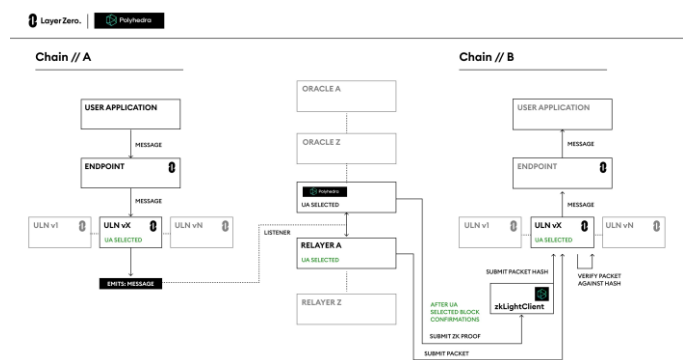


Figure 3: Layerzero and Polyhedra's architecture **

- ZKML verifier:

The application of Zero-Knowledge (ZK) technology allows smart contracts to perform on-chain verification of the outputs generated by the Machine Learning (ML) model and the following decision-making process. The implementation of the ML algorithms is carried out by using the EZKL library [4] and using halo2 as the proving backend.

- "Movement" command:

We have designed the "Movement" command in a way that abstracts various blockchain actions into universal interfaces that the vault smart contract can execute.

Once the strategy manager formulates the Movement command, NOYA's smart contracts will handle the execution. Based on the security setting of the vault, including movement cooldown, security oracle, and on-chain checks, the strategy smart contract decides whether to move the funds.

Off-chain

- Strategy manager:

The strategy manager is an off-chain entity that monitors the network state constantly and sends a transaction to the vault manager smart contract whenever a profitable opportunity arises.

- AI models and Strategy logic:

These models are designed to help the strategy manager to find profitable opportunities in DeFi. They are comprised of yield prediction, price prediction, and asset routing models (finding the best route to transfer assets).

- ZK prover:

The zk prover is responsible for receiving the model, output value, input value, and weights and creating a proof of the calculation. The strategy manager can later use this proof to prove to the smart contract that their decision is the model's output.

Actions in the system

Within the design, multiple actions take place:

1. **User Interactions:** These include actions initiated by users, such as depositing tokens on any supported chain, withdrawing tokens, or viewing their shares and associated value in real time.
2. **Strategy Manager Actions:** Activities primarily associated with strategy managers, such as creating "Movement" commands for transactions, managing user funds across different chains, and reviewing transaction logs for better decision-making.
3. **DAO Operations:** Within the architecture, the role of the admin is to maintain the smart contracts. This includes adding and removing protocol integrations, incorporating or disconnecting bridges or DEXs, and overall system upkeep. It's important to note that admin actions are subject to a delay within the smart

contract framework. Once an administrative decision is made to change system parameters, it must first be committed. The smart contract will then permit its execution after a specified waiting period of 14 days. This latency period provides a buffer for security and ensures careful consideration of any decision made. In the future, this administrative role will be transferred to a Decentralized Autonomous Organization (DAO). Consequently, the community will assume the authority to vote on system actions, further democratizing the decision-making process and fostering greater transparency and community involvement.

4. **Security Failsafes:** A crucial category comprising all the actions dedicated to maintaining system integrity and user safety, such as fraud detection, transaction verification, and enforcement of multiple layers of security. These smart contracts are responsible for halting the system in unexpected situations like hacks, failures, chain halts, etc. In the future, this administrative role will be transferred to a SubDAO.

Security

Smart Contract Security: This section includes rules for users and strategy managers when interacting with the system. Some of the primary rules include:

- *Dex and bridge interactions are limited to vault settings:* Each vault has its own security settings enforced with every transaction. This measure helps mitigate the risk of high slippage fees in Dex.
- *Delayed admin actions:* All admin actions, adding a new chain to trusted chains or a new protocol integration, are delayed, giving users a window to withdraw their assets if they disagree with the new decision.
- *Delayed and capped deposits and withdrawals:* Due to our omnichain architecture, we must use messaging protocols, which inherently carry risks. Deposits and withdrawals are therefore stalled to provide a window to react to fraudulent messages and chain failures.

ZKML - Verifier of the Strategy Execution: Our Machine Learning (ML) approach dictates asset movements by predicting future yields. However, smart contracts can't know whether the movement command is the decision of the model or not. Thus, we use Zero-Knowledge (ZK) scalability to verify the ML model on-chain and keep the model weights private. For asset movements, the strategy manager needs to provide a valid proof.

Risk

Additionally, we have developed a risk framework based on multiple characteristics and features that enables us to

assess and categorize risk for various chains, protocols, and tokens.

The risk framework involves the analysis of the smart contracts involved in the strategy, the quality and result of the external audits performed by the protocol, and the evaluation of the tokens that take part, both as invested assets or as generated rewards.

This feature allows users to farm pools that align with their risk-reward preferences.

Machine Learning Model: The Strategy Powerhouse

NOYA employs Machine Learning (ML) models to manage its operations, using advanced algorithms to predict and optimize yield across different platforms. Using advanced ML techniques, NOYA can leverage large data sets and dynamically react to market conditions, ensuring users receive optimal investment returns.

One of the key components of NOYA's ML strategy is the use of LightGBM [5], a gradient-boosting framework that employs decision tree-based algorithms. The model is designed to predict the behavior of yield for a specific chain, protocol, pool, and time frame.

Gradient boosting is a powerful technique used extensively in the field of predictive modeling for tabular data classification. It builds models from weak predictors in an iterative way, which makes it more accurate and reliable. Unlike other tree-based algorithms, LightGBM grows the tree leaf-wise (best-first) instead of level-wise, which often results in better accuracy,

Simple decision trees suffer from high variance and can easily overfit the training data, making their predictions unreliable for new data. Bagging models, such as Random Forest, reduce this variance by averaging the predictions of many independent trees. However, they often suffer from high bias as they fail to capture complex data patterns fully.

On the other hand, Gradient Boosting Models (GBM) effectively reduce bias and variance by combining weak learners with strong ones, thereby improving the model's performance. Each successive tree in the model is designed to learn from its predecessor's mistakes, minimizing residual error over iterations. Unlike bagging models where trees are grown independently, it grows trees sequentially, which allows for the optimization of an arbitrary differentiable loss function. Therefore, they can model complex non-linear relationships and provide superior predictive performance.

GBMs construct each decision tree to correct the residuals (the differences between the observed and predicted values) of the previous tree. This can be represented by the formula:

$$F_m(x) = F_{m-1}(x) + \text{learning_rate} * h(x; a_m)$$

Where $F_m(x)$ is the prediction of the m-th tree, $h(x; a_m)$ is the m-th decision tree, and learning rate is a parameter to shrink the contribution of each tree.

Getting into LightGBM specifically, we can see that the classical implementation of gradient boosting can be viewed in terms of gradient descent, where the direction of travel is provided by the negative gradient of the cost function, and the line search provides the step size. In our case, instead of just computing the gradient and following it, it uses the second-order derivative to gather more information to make a better approximation about the direction of the maximum decrease (in the loss function) and the step size. So we update our point $x^{(i)}$, at every iteration as:

$$x^{(i+1)} = x^{(i)} - \frac{\nabla f(x^i)}{\text{Hess}(x^i)}$$

Where "Hess" refers to the Hessian or the second-order derivative of the loss function.

To implement this, we need to use newton boosting, which requires the implementation of Taylor expansion to rewrite the loss function around the current estimate in terms of the gradient and Hessian, which can also then be further simplified into the summation of the leaf values of the tree:

$$L(f_m) = \sum_{j=1}^{T_m} [G_{jm} W_{jm} + 1/2 H_{jm} W_{jm}^2] + \Omega(f_m)$$

This model also applies various regularization techniques to avoid overfitting; since L1 regularization in trees is applied to leaf scores rather than directly to features, as in regression, it reduces the depth of trees. This will reduce the impact of less-predictive features, but it isn't as severe as in regression, where L1 regularization (Lasso) can set the contribution of features to zero. L1 and L2 are used together in this model, similar to Elasticnet. The L1 regularization will punish the less-predictive features, and L2 (Ridge) is used to further punish large leaf scores without significantly impacting the less-predictive features. With this, the loss function with respect to the Tth tree can be expressed as:

$$L(f_m) = \sum_{j=1}^{T_m} [G_{jm} W_{jm} + 1/2 H_{jm} W_{jm}^2] + \gamma T_m + 1/2 \lambda \sum_{j=1}^T W_{jm}^2$$

Now that it has a way to measure the loss, it can build every tree, optimizing one level at a time. This can be done by expressing the gain of a potential split in terms of the equation above. Thus, we are using the loss function to construct the tree controlled by the model complexity:

$$Gain = 1/2 \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

Where GL and HL are the sum of the gradient and the hessian values on the leaf values on the left-hand side of the split, and GR and HR are the sum of the gradient and the hessian values on the leaf values on the right-hand side of the split.

For classification problems, especially those involving tabular data, LightGBM can often outperform Deep Neural Networks (DNNs). DNNs have shown exceptional performance on tasks involving high-dimensional data, such as image and speech recognition. Still, they often struggle with tabular data due to its structured and often low-dimensional nature.

NOYA's model, in contrast, can effectively capture the interactions and non-linear relationships between features in tabular data. Furthermore, it is less computationally expensive and quicker to train than DNNs. It also offers better interpretability, which is crucial in many applications where understanding the model's decisions is as important as the accuracy of its predictions.

Model Training and Validation

Data from pools, TVLs, volumes, prices, yields, and all other data used in the models are primarily collected directly from each chain, using Full and Archive nodes. The information for some smaller protocols is currently not being collected from the blockchain. Nevertheless, the pipelines for doing so are currently under development.

Different time windows were defined to calculate relevant features, such as moving averages, standard deviations, and quantiles. The data were then standardized and used to create a training dataset where for every pool at every period, all data points could be aggregated for the same chain, protocol, and tokens, reconstructing the same variables in an aggregated way.

To ensure optimal performance and predictability, NOYA employs 10-fold cross-validation for hyperparameter optimization [6]. Cross-validation is a resampling procedure used to evaluate ML models on a limited data sample. The primary purpose is to gauge how the statistical analysis results will generalize to an independent data set. In the context of NOYA, it ensures that the model's predictions are as accurate as possible, improving the reliability of the yield optimization strategies.

Moreover, NOYA utilizes Bayesian optimization for hyperparameter search. This advanced optimization algorithm constructs a posterior distribution of functions that best describes the function you want to optimize. In Bayesian Optimization, the hyperparameters that are put forward for evaluation by the objective function are selected by applying a criterion to the surrogate function. This criterion is defined by a selection function using a metric called Expected Improvement, which is given by the following equation:

$$EI_y(x) = \frac{\gamma y l(x) - l(x) \int_{-\infty}^y p(y) dy}{\gamma l(x) + (1 - \gamma) g(x)} \left(\gamma + \frac{g(x)}{l(x)} (1 - \gamma) \right)^{-1}$$

This allows it to balance between exploiting known information and exploring new possibilities. High values of Expected Improvement correspond to points where the surrogate function predicts a high accuracy, while they also relate to points where the prediction uncertainty is high.

The main yield model at NOYA is a Binary Classification Model that aims to predict whether a pool's yield will remain at or above its current level for a specific time frame. The model uses more than 200 variables from multiple sources, like blockchain transactional data, Prices, Bridges, Stablecoin Inflows/Outflows, and other relevant data, making it a robust predictive tool.

NOYA's model uses weights to calculate predictions for every potential pool the strategy might switch to at any given time. By factoring in these predictions along with other variables such as bridging fees, swaps costs, and gas, it measures the time to recover these costs. It allows the strategy to make informed decisions about whether to switch pools or rebalance.

Through comprehensive backtesting over more than one year of data, NOYA consistently optimizes its strategies. The goal is always to maximize yield, ensuring the model remains accurate and efficient in predicting high-yield opportunities across multiple chains, protocols, and assets.

Leveraging Machine Learning for Yield Optimization in Specific Protocols

NOYA not only capitalizes on high-yield opportunities across diverse chains, protocols, and assets, but it also employs specialized models to optimize positions within each protocol on an individual basis.

Specifically, for Uniswap V3 [7], it utilizes a dedicated model tailored to each chain, pool, and fee tier [8]. This model's primary function is to optimize the range of concentrated liquidity to achieve the highest capital utilization and performance. The underlying mechanism of this model is a multinomial classification system. In this supervised learning approach, the profitability of providing liquidity to each tick is evaluated, and consequently, the optimal range to maximize this profitability is determined. In parallel, NOYA is also developing a similar model geared towards the management of the provision of leveraged liquidity based on lending protocols that allow the deposit of yield-generating tokens as collateral. This model will further enhance NOYA's capabilities to optimize yields and maximize investment returns.

Moreover, the tokenization of deposits expands our possibilities, allowing for the collateralization of these tokens. Leveraging the tokenized deposits can further our reach and capabilities, enabling us to create Leveraged Omnichain Tokenized Models. We can develop a leveraged Uniswap LP or a leveraged Omnichain Uniswap Stablecoin Strategy by securely locking these tokenized deposits.

Finally, we are exploring the possibility of hedging our Positions using a combination of financial derivatives like Options and Perpetuals, allowing us to obtain a Delta Neutral Yield even for non-stable pools or other strategies that involve non-pegged and highly volatile assets.

EZKL: A Catalyst for Enhanced Trust and Transparency in NOYA's Operations

NOYA utilizes the Zero-Knowledge Machine Learning library EZKL to help minimize the required trust in its strategy execution.

EZKL is a cutting-edge tool engineered for implementing deep learning models and other computational graphs in a zk-snark or Zero-Knowledge Machine Learning (ZKML). This technology not only increases the scalability of the data but also preserves the confidentiality of the computational graph.

At the moment of writing, EZKL does not support machine learning models based on decision trees or trained using bagging or gradient boosting of these algorithms. In the

future, EZKL is expected to be compatible with our model and NOYA might even utilize even more advanced models that are compatible currently with EZKL. However, to circumvent this for now, we trained an alternative Fully Connected Linear Neural Network to develop a ZKML implementation.

In the backend, EZKL employs Halo2 as a proof system. Halo2 is an efficient and flexible proof system that doesn't require a trusted setup. It's prime feature is the ability to create recursive proofs, which are essential for scaling and maintaining blockchain applications. The use of Halo2 helps maintain the robustness and reliability of the proof system in EZKL.

Through the strategic deployment of EZKL, NOYA reinforces its commitment to transparency, allowing users to verify that the same model used in production is also used for backtesting.

Halo2, the proving system of NOYA

The Halo2 framework developed by EF provides the capability to establish and interlink gates to execute arithmetic calculations. In a typical plonk setup, you have access to an addition gate and a multiplication gate, both featuring two inputs and a single output. These gates can be freely positioned and interconnected based on your needs. You can reveal or hide the wire values per your preference during the proof generation process. Halo2 uses PLONKish arithmetization, and In PLONK-style arithmetization, gates are represented as a table structure. The column of this table can be:

1. Constant columns for placing constants.
2. Instant columns for inserting public inputs.
3. Advice columns, designed for storing private inputs, are often called witnesses.
4. Selector columns intended for positioning selectors.

For every row in this table, we impose a set of gate constraints that embody the computation in the form of gates.

In the context of Halo2, there are three fundamental elements we can utilize to construct our gates:

1. Custom Gate: This stipulates an expression that must be satisfied by the interplay between the rows and columns within a gate.
2. Lookup Gate: This component Is a helpful tool to optimize some calculations that are costly in custom gates, such as range checks and bitwise operations.
3. Copy Constraint: This ensures that two linked cells have identical values.

Custom gates of NOYA ZKML

NOYA represents floating point numbers as fixed-point numbers. For example, the number 0.3 on a scale of 8 (one of the settings of EZKL) will be represented as 77 ($0.3 * 2^8$).

To Prove $\text{ReLU}(x) = y$, NOYA prefills a table with the inputs and outputs (e.g. (-2, 0), (-1, 0), (0, 0),...) and proves that (x, y) exists in that table.

There are alternatives to this approach (e.g., performing a bit decomposition before creating the table. But overall, this is more efficient.

To prove $y = Ax + b$, NOYA uses a multiplication gate and performs a shift on the output.

To prove the dot product of 2 vectors, NOYA uses an accumulated polynomial argument composed of the custom gate (with constraints $a_i * b_i + m_{i-1} = m_i$)

a0	a1	m	s_dot
-----	-----	-----	-----
a_i	b_i	m_{i-1}	s_dot
		m_i	

With the same technique, we can prove the matrix to vector multiplication. NOYA uses an "init" polynomial argument composed of the custom gate (with constraints $a_i * b_i = m_i$) This might be changed in the future with improvements in EZKL and general proving systems.

a0	a1	m	s_dot
-----	-----	-----	-----
a_i	b_i	_	s_dot
		m_i	

Proving benchmark

Creating ZK proofs is a computationally heavy job. It requires a considerable amount of memory and CPU power to calculate. The latest technologies have pushed the boundaries, but we still have to spend a lot of computational resources to create proofs. For a better understanding of the costs in the system, we've performed multiple benchmarking tests. During this process, we've created over 2000 proofs. We measured and averaged time spent creating proofs and memory usage during proof generation

to understand the system better. All of these proofs have been executed on the following server:

CPU: Intel(R) Xeon(R) W-2295 CPU @ 3.00GHz (36 cores)

Memory: 528 GB DDR4

Benchmark 1: Logrows (k) (time and consumed memory)

The logrows argument is the base 2 logarithm of the number of rows that are in the halo2 circuit. This plays an important role in how large the model can be. We can put more gates in the table with more rows and prove a bigger computation.

The higher Logrows mean less work for the verifier (smaller proofs and cheaper verification) and more work for the prover (more time and memory-consuming computation) and vice versa.

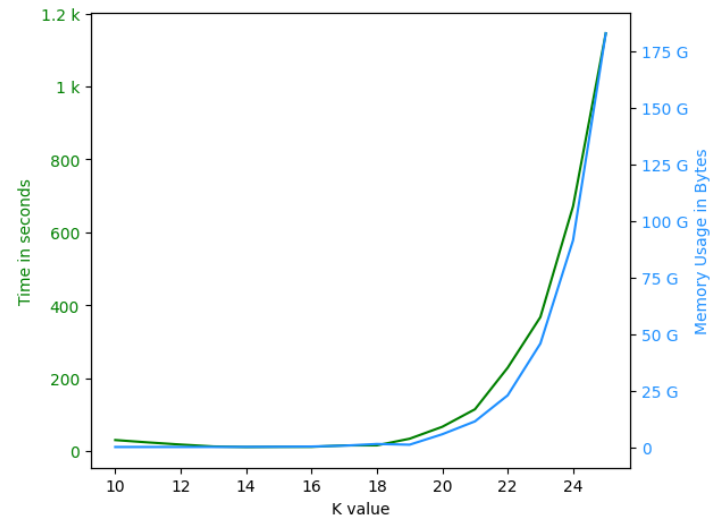


Figure 4: Variation of Time and Memory Usage as K-value changes

Benchmark 2: Error with scale

EZKL quantizes a floating point value to a fixed point value by multiplying by 2^{scale} and rounding. This rounding creates an error in the calculation. In this benchmark, we calculated the error. As shown in the chart, as the K increases, the error amount decreases because the gap between numbers is decreasing.

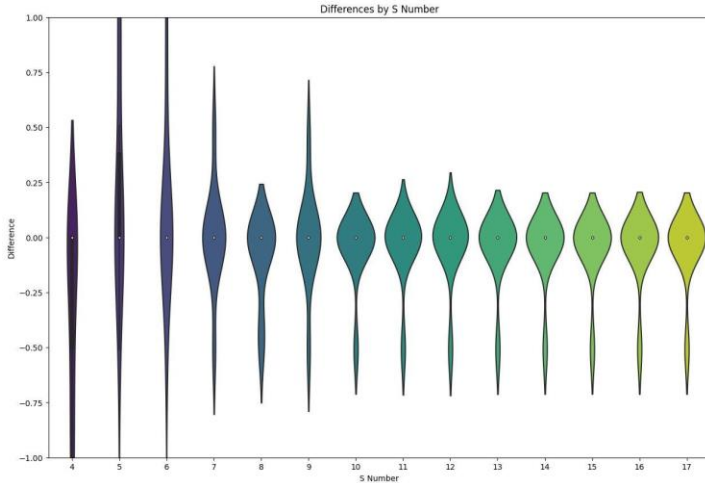


Figure 5: Boxplot of Rounding Error at different K values used for Scaling

Benchmark 3: Number of Neurons by Layer (time and consumed memory)

For this benchmark, the architecture of the neural network was fixed to a 10-layer fully connected feed-forward network with ReLu activation, and the only parameter that was changed was the number of units (neurons) that conform to every layer of the network to measure the changes in time consumed and the Memory Usage.

Looking at different network architectures, it's clear how adding wider layers to a fixed 10 Layer model impacts Time and Size in memory, having even a linear growth with a greater slope above the 500 neurons per layer. It's needless to say that we need bigger models for more advanced ML applications.

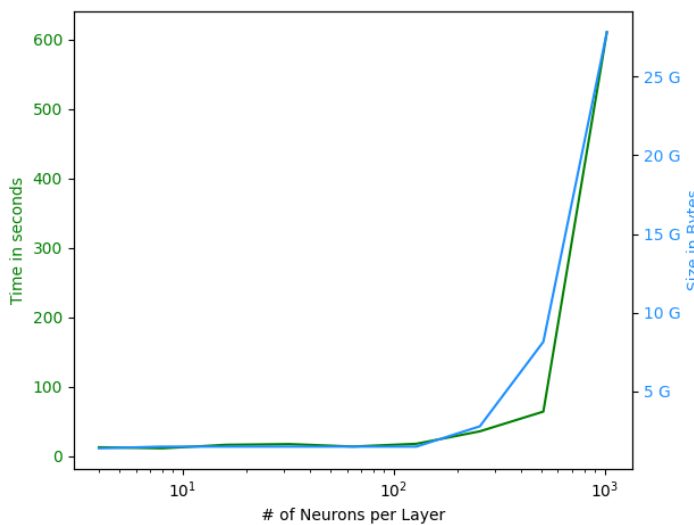


Figure 6: Variation of Time and Memory Usage per number of Neuron in each Layer of the NN, in logarithmic scale

Benchmark 4: Number of layers

In this benchmark, the neural network's architecture was fixed to 256 neurons per layer in a fully connected feed-forward network with ReLu activation. Only the number of total fully connected layers was changed to see how it impacted the Time consumed and the Memory Usage.

It can be seen that the growth in both Time and Memory is linear to the number of Layers added to the architecture.

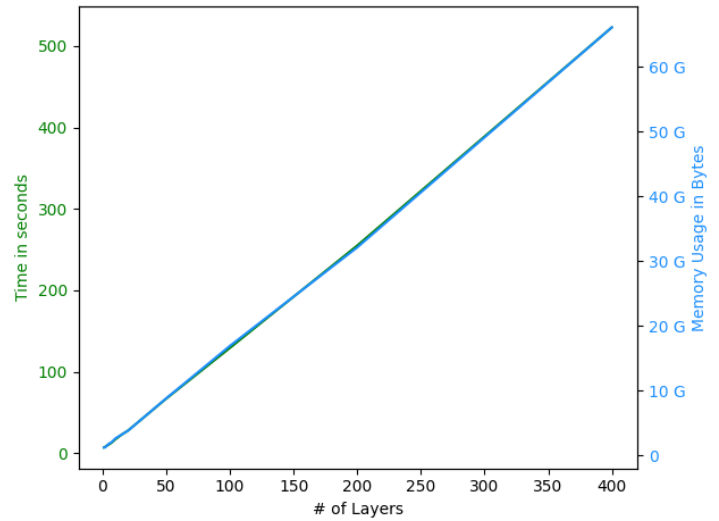


Figure 7: Variation of Time and Memory Usage per Layers in the NN

Roadmap and Possibilities

Competing AI Models: Towards a Single Verifier (Aggregator)

As more advanced AI models become increasingly prevalent, NOYA envisions becoming a marketplace where different AI models compete for liquidity. Leveraging the trustless backtesting capability offered by ZKML, these models can provide known returns and secure future executions facilitated by proofs of model execution. This approach represents a significant step in ensuring safe and efficient strategy execution in the DeFi landscape.

Aggregation is a feature of Halo2 that allows NOYA to create aggregated (recursive) proofs. This approach opens the door to optimizing the verification furthermore. By using this feature, we can reduce the cost of verification by batching multiple proofs together. We already have this implemented and are now using it to simultaneously prove multiple strategies and movements.

Leveraging Transformers for Enhanced Analysis

As the model can be trained with more data and specifically decoding intricate blockchain contracts, labeling diverse addresses, tracing yield and price movements instigated by whales, and discerning specific kinds of address behaviors, we aim to apply a Time Series optimized transformer [9] as our primary predictive model. This technique draws from our previous experience devising an internal banking institution model. Nevertheless, a more extensive and comprehensive data set is required to yield accurate and reliable results due to the vast parameters and the intricate behaviors we aim to explain.

The research into cutting-edge transformers for tabular time-series data suggests that the most viable model would be the Bidirectional Encoder Representations from Transformers (BERT) model, a product of Google's research and development that offers considerable advantages in encoding complex datasets.

To understand our data adequately, we propose an innovative approach: the encoding representation of protocol, tokens, and chains in every pool. Each of these factors would be encoded with aggregated data specific to them. This method transforms our data into a form that can be more effectively processed by the BERT model [10].

The BERT model utilizes two primary components in processing: the encoder and the decoder. The encoder reads and interprets the input data, while the decoder uses this interpretation to produce a response or output.

The numerical variables currently employed will be incorporated into the encoder-decoder structure of the transformer. This approach will enable us to process our data more precisely and efficiently.

The proposed method, therefore, not only leverages the power of the BERT model but also presents a novel way to adapt our data to the model's specifications.

Innovating with Autonomous Agents: An Evolution of Learning Models

Another strategic transition evaluated is to switch from the conventionally used Supervised Learning models, which rely heavily on pre-existing outputs for making predictions, to a more sophisticated approach characterized by Deep Reinforcement Learning (DRL) models [11, 12, 13].

DRL is a specialized form of Machine Learning that trains an agent to make decisions through interaction with its environment, rewarding or penalizing actions based on the results they generate. It involves learning from positive and

negative reinforcement and is widely recognized for its ability to learn optimal strategies in complex, uncertain environments.

In the context of finance and blockchain, DRL models have considerable potential. These models can create trading strategies, optimize portfolio management, and facilitate autonomous decision-making in decentralized finance (DeFi) environments. Several established agents can be adapted to form the basis of these models, such as Q-learning or Actor-Critic models, distinguished by their ability to balance exploration and exploitation effectively.

In a typical DRL agent, decisions are made based on the state of the environment. The agent then receives feedback in the form of rewards and updates its policy based on this feedback. Over time, the agent learns to optimize actions to achieve the maximum cumulative reward.

A DRL agent can be represented by the equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Where s represents the current state, a represents the action taken, r is the immediate reward, s' is the new state after acting a , and γ is the discount factor, which determines the present value of future rewards.

Compared to supervised models, DRL agents have several advantages. They can adapt and evolve in response to environmental changes, making them more robust to market conditions or blockchain protocol shifts. Moreover, they can learn to perform complex tasks and develop innovative strategies without explicit programming. However, developing these agents also presents several challenges, primarily from their dependence on high-quality, representative data and the difficulty of defining appropriate reward functions.

Crowdsourcing Model Optimization through Competitions

In line with our commitment to harnessing collective intelligence, we plan to make our data and problems accessible to a dedicated Machine Learning community such as Kaggle. This initiative aims to stimulate competition among ML enthusiasts to design and run models that optimize Yield, akin to the strategy adopted by several Trading Hedge Funds in the stock markets over recent years, which is now gaining momentum in the Crypto space. If successful, this could pave the way for a revenue-sharing model, where community members are incentivized to continually refine and improve our models, ensuring the delivery of optimum results.

Trustless Strategy Execution: The Challenge of Data Provenance

Ensuring data provenance presents a significant challenge, particularly within Zero-Knowledge Machine Learning (ZKML) context. Executing a trustless strategy is directly linked to the reliability of the data provided by oracles. Several innovative solutions have emerged, including ZK Oracles and Optimistic Oracles. Utilizing oracles like Hyper Oracle and Axiom, NOYA aims to guarantee data provenance and pioneer truly trustless and efficient strategy executors. An example application for this would be fully trustless automated liquidity provisioning for Uniswap V3.

Conclusion

NOYA is an Omnichain Yield Aggregator that aggregates yield for different assets, protocols, and chains. It uses Layerzero and Polyhedra for secure messaging and bridge and DEX aggregators for optimal asset routing. By utilizing EZKL, NOYA creates proofs for their strategies and even aggregates them justifying several movements and securing multiple vaults simultaneously. Going forward, NOYA is looking to build more accurate models with advanced architectures and use safer model inputs for a more robust architecture.

References

01. Simon Cousaert, Jiahua Xu, Toshiko Matsui. (2021). SoK: Yield Aggregators in DeFi. IEEE International Conference on Blockchain and Cryptocurrency.
02. Dcbuilder.eth (2023) An introduction to zero-knowledge machine learning (ZKML).
<https://worldcoin.org/blog/engineering/intro-to-ZKML>
03. Zhang, Y. (2020, November). Zero-Knowledge Proofs for Machine Learning. In Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice
<https://dl.acm.org/doi/10.1145/3411501.3418608>
04. Dante (alexander-camuto), Jason (jasonmorton). (2023) Easy Zero-Knowledge Inference
<https://github.com/zkonduit/ezkl>
05. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017, August). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).
06. S Loesch, N Hindman, MB Richardson, J Wu, XY Chen, H Zhang, LD Xiong (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. Journal of Electronic Science and Technology
07. H Adams, N Zinsmeister, M Salem (2021) Uniswap v3 Core.
<https://berkeley-defi.github.io/assets/material/Uniswap%20v3%20Core.pdf>
08. M Neuder, R Rao, DJ Moroz, DC Parkes (2021). Strategic liquidity provision in uniswap v3. arXiv preprint arXiv:2106.12033 <https://arxiv.org/pdf/2106.12033>
09. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).
10. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805
11. Farzan Soleymani, Eric Paquet. (2020). Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath. Expert Systems with Applications.
12. Yue Deng, Feng Bao, Youyong Kong, Zhiqian Ren, Qionghai Dai. (2016). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS
13. Yengyao Jiang, Dixing Xu, Jinjun Liang. (2017). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. Deep Portfolio Management
14. Nont Kanungsukkasem, Teerapong Leelanupab. (2019). Financial Latent Dirichlet Allocation (FinLDA): Feature Extraction in Text and Data Mining for Financial Time Series Prediction. IEEE Access.
15. L Zhou, K Qin, A Cully, B Livshits (2021). On the just-in-time discovery of profit-generating transactions in defi protocols. IEEE Symposium
16. H Teng, W Tian, H Wang, Z Yang (2022) Applications of the Decentralized Finance (DeFi) on the Ethereum. IEEE Asia-Pacific
17. Laura Alessandretti, Anne Kandler, Luca Maria Aiello, Andrea Baronchelli. (2018). Anticipating Cryptocurrency Prices Using Machine Learning. Complexity.
18. Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, Pierre-Alain Muller. (2019). Deep learning for time series classification: a review. Data Mining and Knowledge Discovery

Annexes

Figure 1: NOYA's architecture

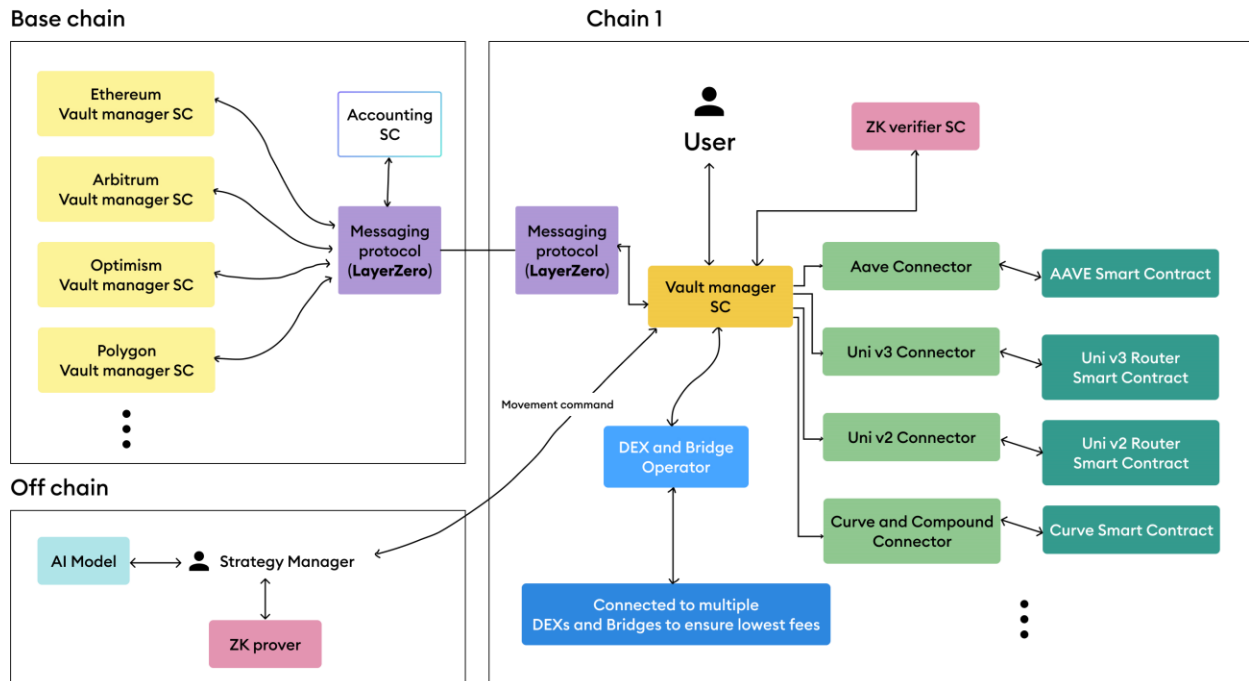
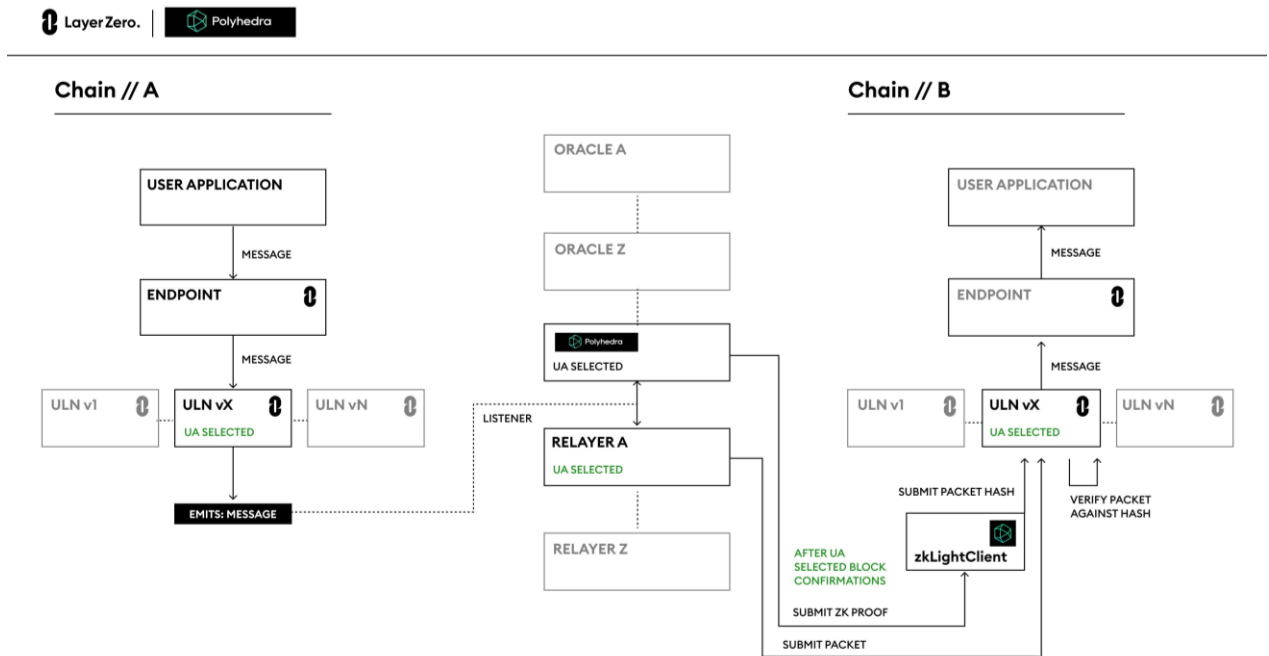


Figure 2: NOYA's Messaging Protocol



Figure 3: Layerzero and Polyhedra's architecture



KNML Full Backtesting Results

Benchmark 1: Logrows - k (time and consumed memory)

K Value	Time (sec)	Size in Memory (Gb)
10	29,91	0,17
11	23,45	0,17
12	17,54	0,18
13	12,32	0,19
14	10,88	0,22
15	11,33	0,27
16	11,33	0,38
17	15,19	0,76
18	15,39	1,50
19	33,53	1,18
20	66,28	5,80
21	114,40	11,49
22	228,32	22,98
23	367,24	45,85
24	671,35	91,48
25	1145,78	182,86

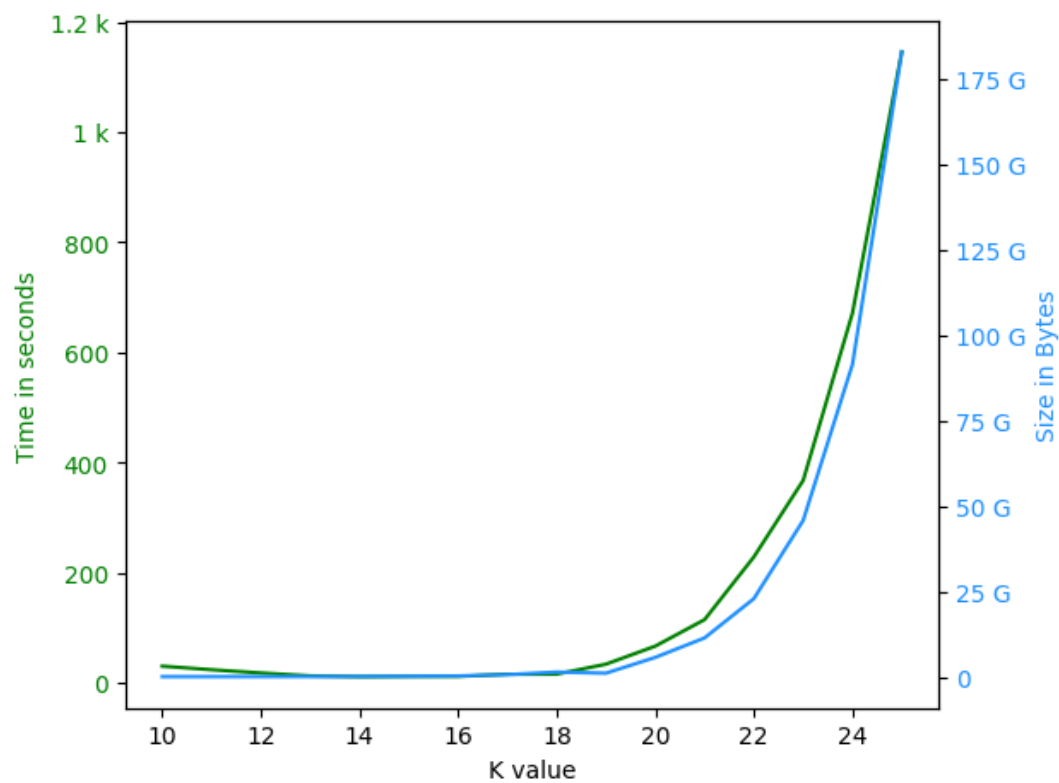


Figure 8:: Variation of Time and Memory Usage as K-value changes

Benchmark 3: Number of Neurons by Layer (time and consumed memory)

# Neurons	Time (sec)	Size in Memory (Gb)	Parameters
4	12,78	1,40	1.481
8	11,27	1,50	3.305
16	16,29	1,50	10.245
32	17,36	1,50	17.185
64	13,81	1,50	57.369
128	17,78	1,50	218.573
256	35,73	2,79	379.777
512	64,13	8,16	1.398.017
1024	610,25	27,81	5.496.577

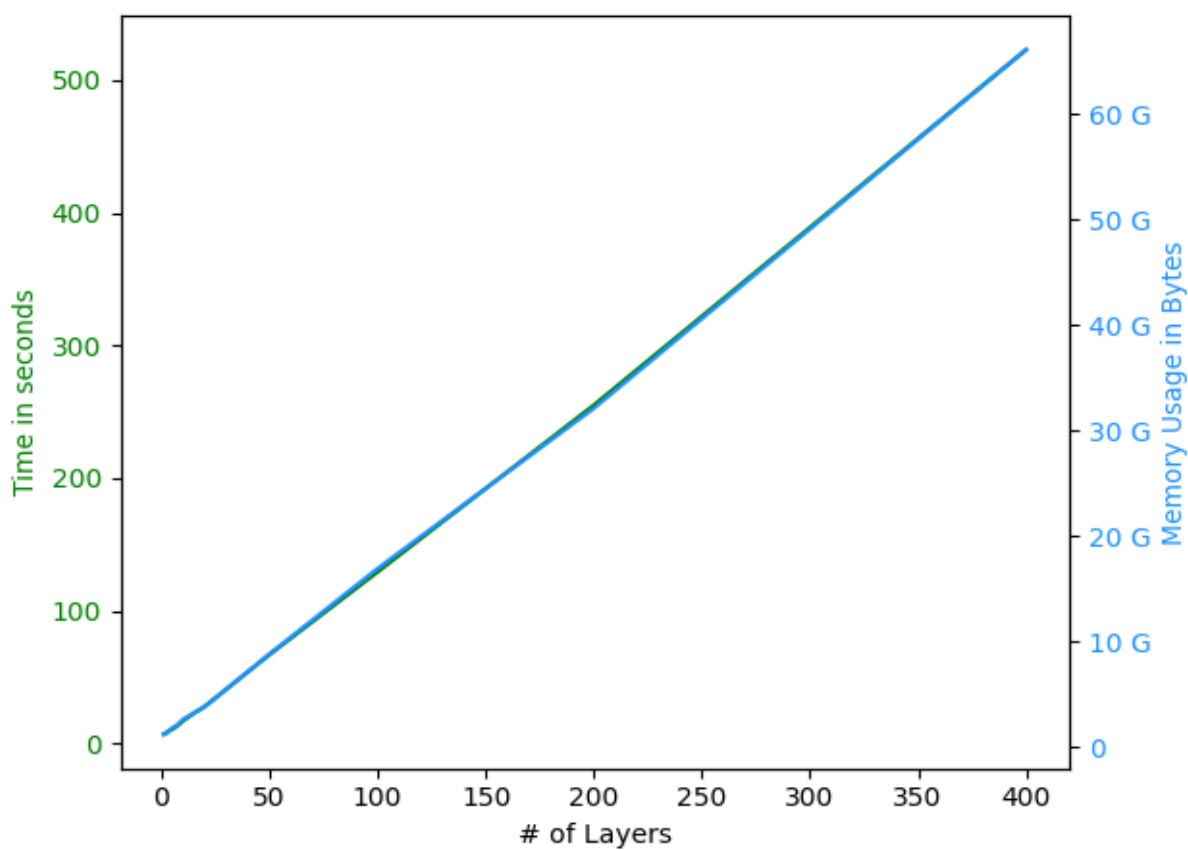


Figure 9: Variation of Time and Memory Usage per Layers in the NN

Benchmark 4: Number of layers

# Layers	Time (sec)	Size in Memory (Gb)	Parameters
1	7,00	1,20	19.201
2	7,80	1,20	68.097
3	9,00	1,40	116.993
5	10,70	1,80	193.585
7	12,80	2,00	270.177
10	17,00	2,60	424.449
15	22,80	3,20	657.217
20	27,80	3,80	889.985
50	67,00	8,80	2.131.521
100	129,30	16,90	4.187.041
200	255,00	32,10	8.354.561
400	523,00	66,10	16.693.521

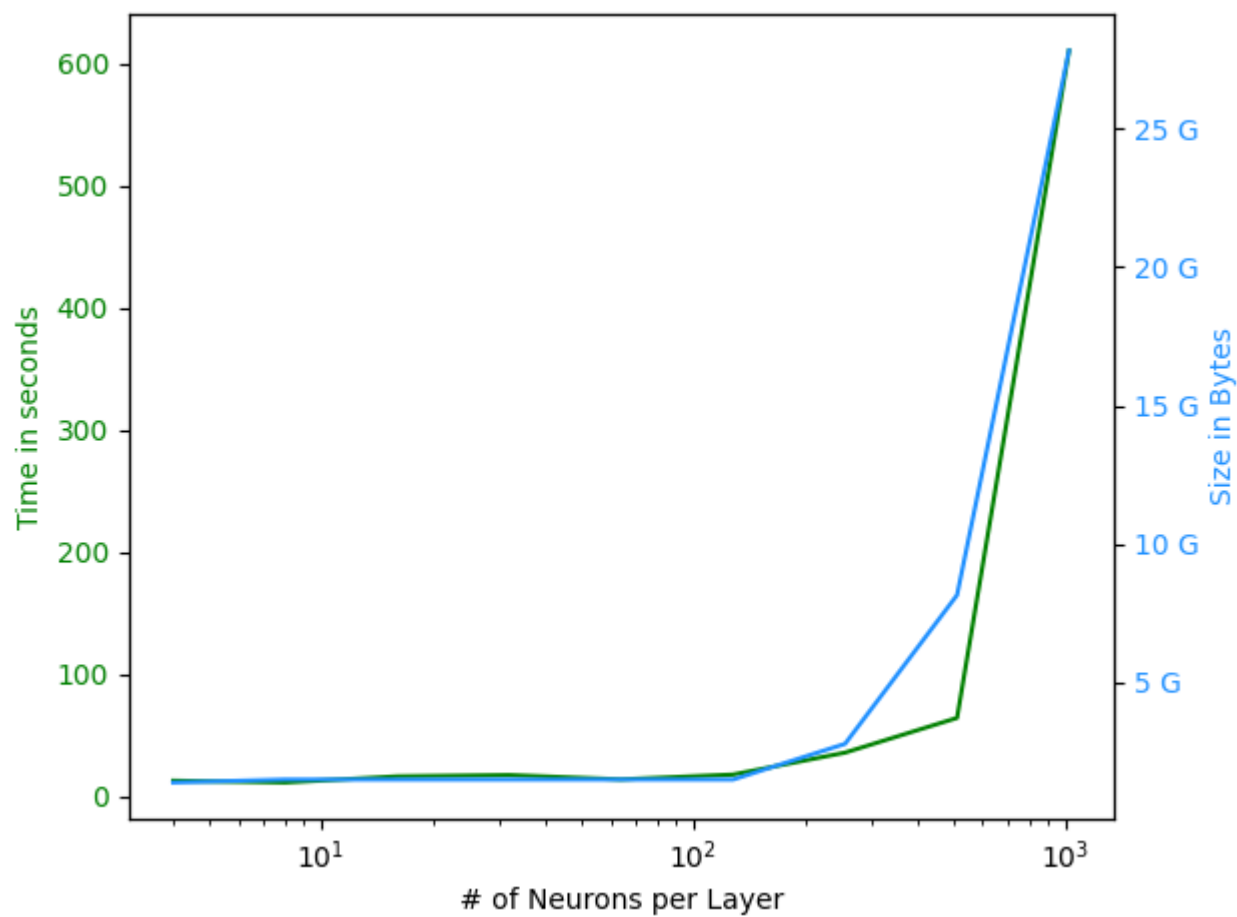


Figure 10: Variation of Time and Memory Usage per number of Neuron in each Layer of the NN, in logarithmic scale

Multiple Strategies Backtesting Results

USD Stables – Low Risk

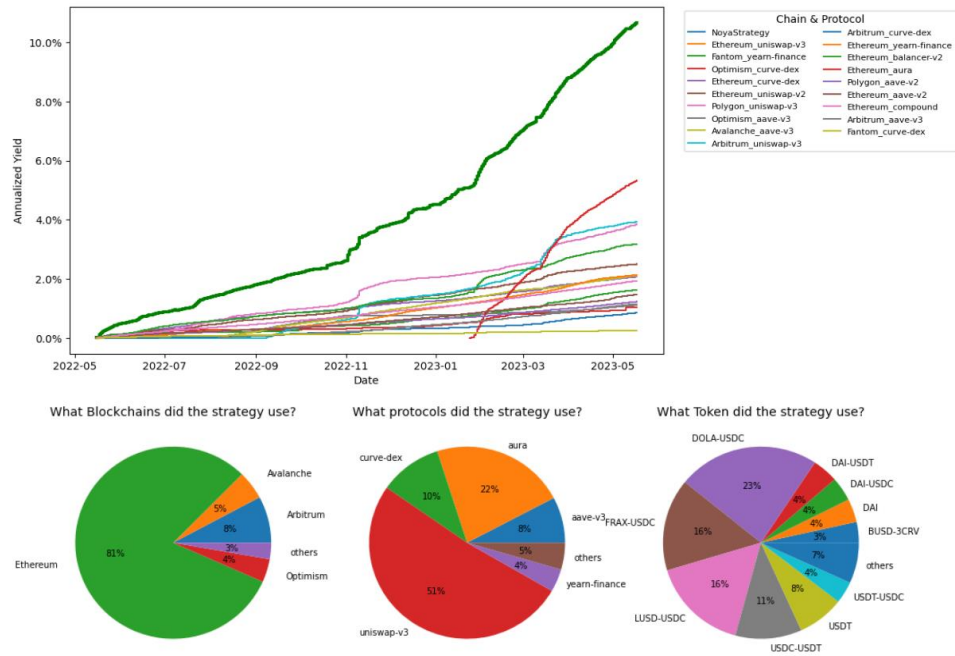


Figure 11: One-year backtesting for NOYA's "USD Stables – Low Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.

USD Stables – Medium Risk

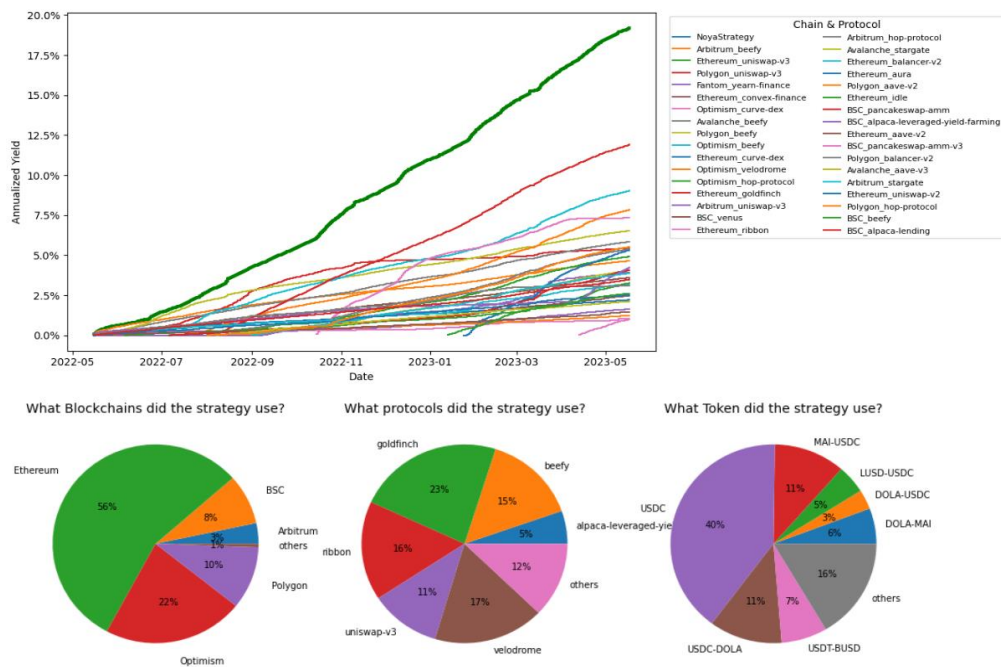


Figure 12: One-year backtesting for NOYA's "USD Stables – Medium Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.

USD Stables – High Risk

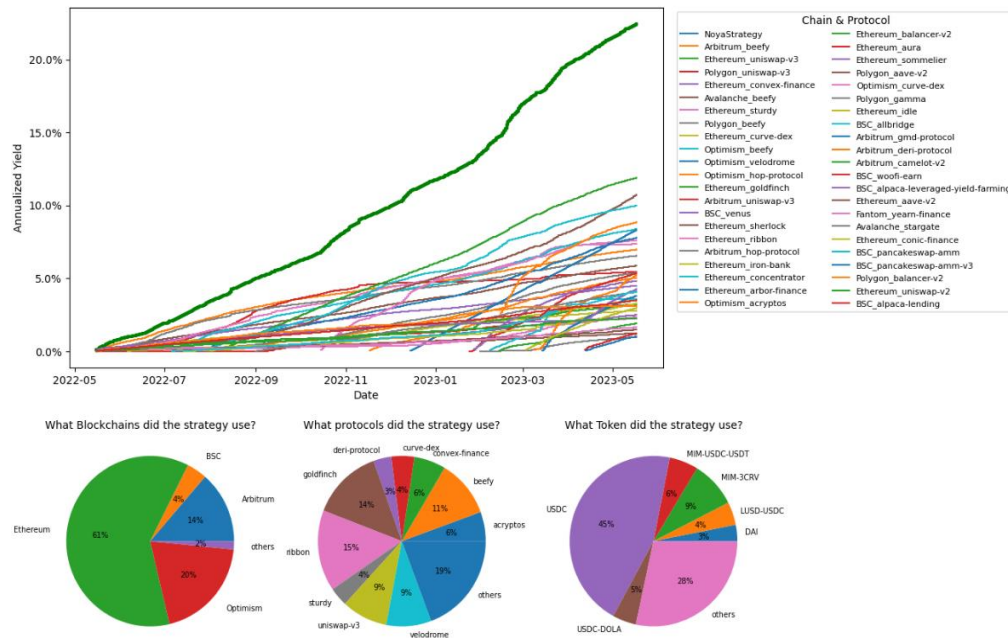


Figure 13: One-year backtesting for NOYA's "USD Stables – High Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.

ETH Stables – Low Risk

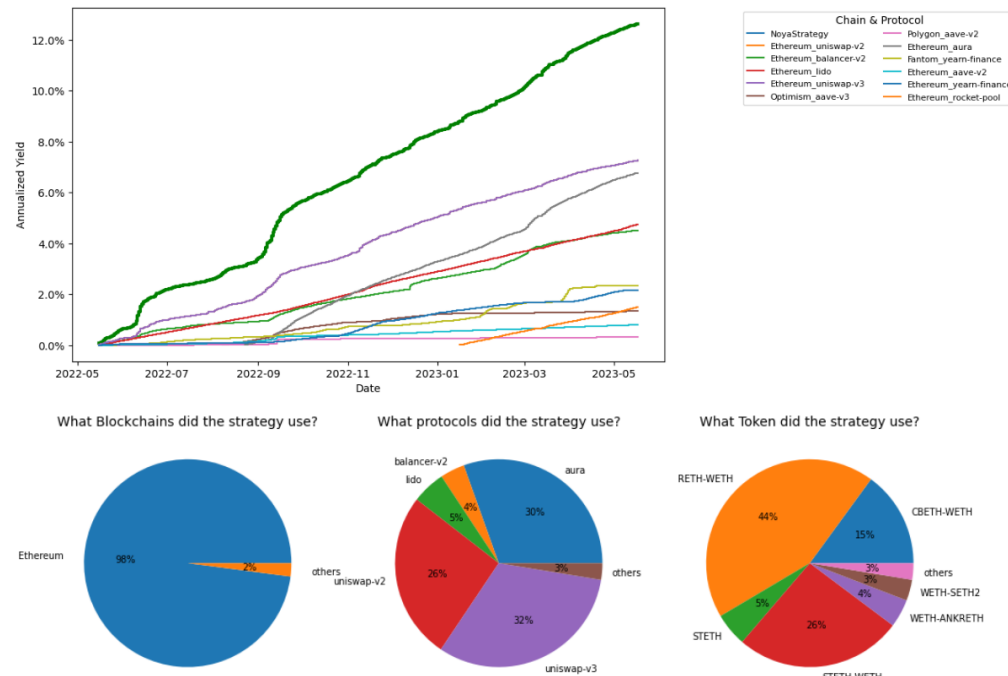


Figure 14: One-year backtesting for NOYA's "ETH Stables – Low Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.

ETH Stables – Medium Risk

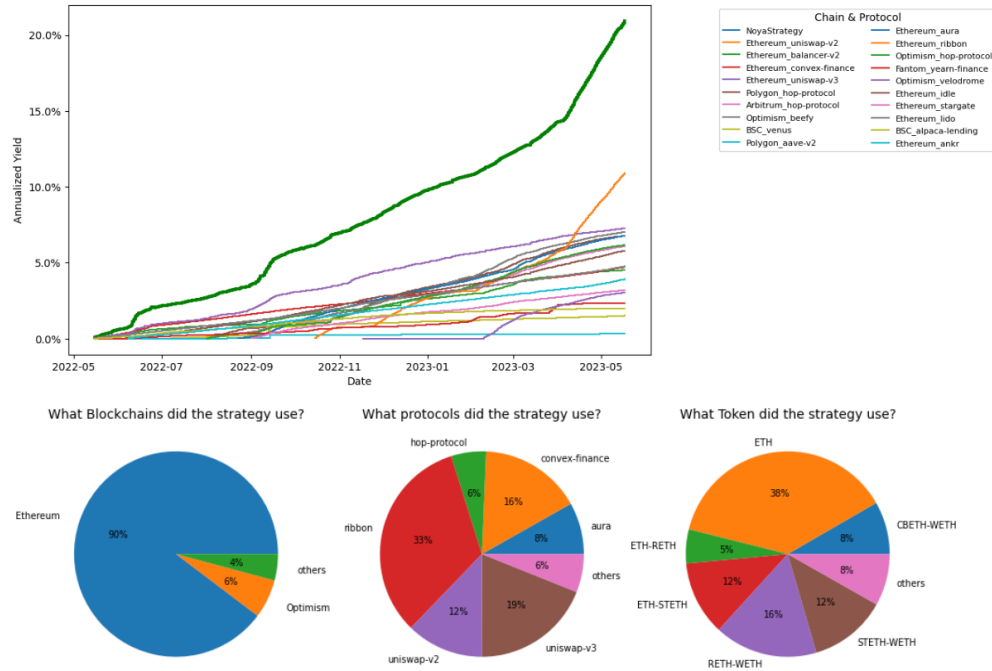


Figure 15: One-year backtesting for NOYA's "ETH Stables – Medium Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.

ETH Stables – High Risk

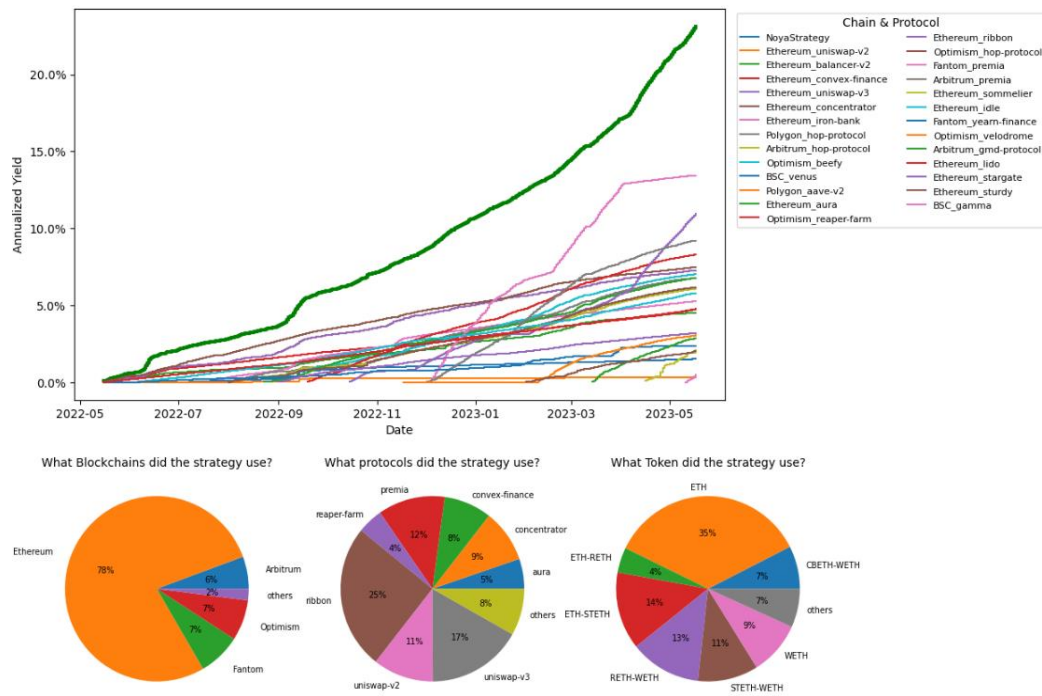


Figure 16: One-year backtesting for NOYA's "ETH Stables – High Risk" Pool using LightGBM model trained on historical data. The simulation includes costs of swaps, bridges and gas paid. The protocols chosen in the legends are just for illustrative purposes and will vary based on market conditions.