Dynamic Encryption Windows: Optimal Efficiency in Delay-based Weight Copying Mitigation (Working Paper)

Ren Labs

September 28, 2024

Abstract

We propose a tradeoff-optimized approach to delay-based weight-copying mitigation mechanisms (Commit Reveal Opentensor [2024a]), based on a dynamic encryption window. The mechanism deterministically guarantees weight copying is less profitable than delegating to an average validator, rendering it irrational, while granularly minimizing the information-delay tradeoff. This is accomplished by adapting encryption duration dynamically to the subnets consensus-context, without requiring individual implementation for subnets. We discuss the landscape of existing solutions and their tradeoffs, present a theoretical mechanism, formalize it, and share our implementation strategy. Finally, backtest results demonstrate the effectiveness of our approach.

1 Introduction

Weight copying has been a core inefficiency since genesis of the first weight-based decentralized algorithmic incentive protocol Bittensor [2021], equally present in the Commune [2023]. Validators are incentivized to submit weights to the chain that are in consensus with the weights of other validators. This consensus-incentive is a key ingredient to Yuma Consensus, which creates Stakeholder incentive compatibility, allowing the market-mechanism to function while following arbitrary objectives.

Weights are a highly expressive format to represent the results of validators local evaluations of miner performance against the applied scoring function. The network (i.e. subnet) collects local weights to compute consensus producing the global chain weights, in order to adjust and execute the distribution of new coin emissions every epoch (number of blocks).

Due to the transparent nature of a blockchain, weights submitted during the collection period of an epoch are public and the consensus-weights of the last executed epoch are part of the historical chain state. Weight copying is the act of submitting weights that were copied from either (1) an individual validator or (2) the consensus, without performing the costly computational work of evaluating miner performance directly.

Since copying in-consensus weights is strengthening the consensus without distorting the incentive distribution over miners, the network can operate functionally and efficiently up to a certain extent of weight-copying popularity. For this reason, it took well over a year before it entered the public consciousness and protocol developers started taking it serious.

Weight copying has many facets and depending on the subnet context can be both feature and bug.

On the positive side, it can allow continued functioning of subnets with emissions too low to cover costs of the average validator, as long as at least one real validator is active. During rough times of prolonged market recession this can allow subnets to stay afloat that without weight-copying would not be economically feasible, by trading off decentralization / robustness to a variable extent. Miner costs reduce alongside, due to less validation queries. This makes the protocol as a whole more adaptive and resilient to its environment. Further, weight copying accelerates consensus convergence and generally lends more control to honest and consistent validators.

On the negative side, weight-copier validators can obtain the reward-optimal weight on every subnet consistently, which is virtually impossible for real validators and hence can outperform the stake APR of real validators by potentially significant margins. This comes at the consequence of making copier validators the optimal choice for those delegating their stake in the quest of maximizing staking rewards, over time concentrating a growing proportion of the total stake to weight-copiers. This increases centralization and undermines the robustness guarantees of Yuma Consensus, further its an incentive-trajectory pointing towards eventual decay of a decentralized, qualitative and robust consensus.

It has also emerged as a significant source of community unrest, as loyal and honest community members are spending many working hours and thousands of dollars monthly to operate real validation infrastructure just to get outperformed by dishonest copiers who operate fully automated and on close-to-zero cost. This strong sense of unfairness is a cancer to the communities happiness and focus.
Further, weight copying also poses damage to the utility output of those subnets that depend on validators not only for miner evaluation but utility creation, e.g. in synthetic data generation where validators generate an essential element like the question to the miners response.

It is clear that weight copying has to be taken serious and demands a technical solution. However, if weight copying is mitigated or utilized as an economic tool has to be a local decision for every subnet.

1.1 Landscape of Solutions

Existing approaches to mitigate weight copying fall into two categories: onchain weight-information-delay schemes (Commit Reveal, Liquid Alpha) and offchain provable computation (proof of weights).

Commit Reveal (v2) Opentensor [2024a] can be implemented by subnets, requiring validators to submit a hash of their weights (commit), and after a set number of blocks (subnet parameter) submit their weights (reveal) which has to match the committed hash. This hidden period is normally set to expand over multiple epochs, meaning also consensus and emissions are delayed until reveal. This hidden window creates a delay between the last public weights and the weights required for the current epoch.
Meaning, if there is enough weight-distribution-delta contained in the commit-reveal interval, the copier-weight is far enough out-of-consensus to cause a consensus penalty high enough to make weight copying irrational. Weight delta entirely depends on subnet miner churn and incentive distribution variability, which are both strong markers of a healthy and functional incentive landscape.

The delay-window can be adjusted by the subnet owner based on their historical analysis and context. The optimal delay-length is the precise minimum number of blocks it takes for enough weight-delta to accumulate to make weight copying irrational. This length changes every delay-window, and will average very differently for different subnets. Subnet owners can take a data-driven guess for setting the delay parameter, but inevitably are going face varying degrees of inefficiency, with too short or too long delays happening constantly.

Liquid Alpha Opentensor [2024b] is a complementary feature to Commit Reveal, by intelligently amplifying the sensitivity of validator EMA bond calculation to delta in the weight distribution. This increases the consensus penalty and consequently lowers the necessary delay-length to achieve copier-irrationality. Further, liquid alpha increases weight-delta itself by making new miner discovery and aggressive bonding more profitable. This is achieved by automatically adjusting the *alpha* (α) value between a lower and upper bound (subnet parameter) in the EMA bond calculation personalized for each miner based on the structure (spreads) of the incentive distribution and the miners position within it, exploiting the latent potentials to set a high α-value. Liquid Alpha is a powerful complementary to any delay-based approach.

Proof of Weights AI [2024]. by Inference Labs is a comprehensive approach that contrary to Commit Reveal is attempting to solve the problem at its core. Delay-based schemes are only addressing onchain copying but remain vulnerable to offchain collusion. The fundamental challenge is verifying that the real validator scoring computation has been performed to obtain the weights. As long as this cannot be deterministically proven, there will remain some room for weight-copying variants.
Proof of weights works by converting the validation scoring function into a provable zero-knowledge circuit with a subnet-wide verification key. Running the circuit to score miners and compute weights produces a proof, which can be verified onchain through the verification key. Copying weights without a valid proof is now useless, because invalid proofs can be penalized to the extent that anything but running the real scoring circuit is a loosing strategy.

Unfortunately converting the full validation scoring function into a end-to-end provable zero-knowledge circuit is very challenging and in many cases virtually impossible with present tools. For instance, many subnets utilize large models in combination with several other computationally complex and nondeterministic components and converting

that into a provable zero-knowledge circuit is a whole R&D venture in itself.
However, Inference Labs is actively working on only proving a part of the scoring function (that converts inputs to weight vectors), such that guessing the inputs is harder than running the real validation. Although the authenticity of the inputs cannot be verified so can be manipulated to an extent. They are working on a peer prediction method randomly testing validators inputs to mitigate that. There is a lot of open work in this area.

1.2 Tradeoffs

Commit Reveal + Liquid Alpha mitigates onchain weight copying through information-delay, but as a consequence also delays all processes dependent on the information. This includes consensus, emission distribution and deregistrations. Moreover, it delays the feedback signal for miners which is crucial for their optimization process.
It bottlenecks the subnets dynamism and slows the speed of information-propagation. Due to the limiting effects on decentralized coordination capacity, the impacts on utility-output over time could be noticeable.

Commit Reveal increases the burden on subnet owners, as it has to be implemented independently for each subnet, requiring modification of the validator code, increasing the systems complexity. The owner has to spend cognition on setting the delay parameter intelligently and be ready to adjust it based on changes in the subnet landscape.
Commit Reveal only addresses onchain weight copying, but remains vulnerable to offchain collusion.

Due to the potential significance of the information-delay tradeoff, the inefficiency of a static delay adjusted manually by the subnet owner, while the optimal length changes dynamically, is the primary weaknesses of the approach.

Proof of Weights addresses the problem at its core but comes at the cost of significant technical overhead. En-to-end proving is not feasible in cases where subnets utilize complex architectures in their scoring function. In the cases where implementation is feasible, there is still a computational overhead.
Since this approach is still being developed and tradeoffs depend on the details, the insight is limited.

While proof of weights is a comprehensive solution to weight copying in theory, it is apparent in practice there are many tradeoffs involved and in order to make it feasible subnets with complex scoring you have to sacrifice the comprehensiveness and robustness.

1.3 Ideal Delay-based Solution

We believe the delay-based paradigm is most likely to yield a widely applicable method with attractive tradeoffs in the near future. The ideal delay-based solution would possess 3 key characteristics:

1. Dynamically length-minimized delay while deterministically guaranteeing a set constraint of weight-copying rationality is met

2. Can be turned on and off by subnet parameter setting, without individual implementation per subnet required

3. Implementation is trustless, compatible with chain decentralization and sufficiently low-complexity

If a delay-based approach would exhibit all those characteristics, its tradeoff profile would be substantially more attractive than Commit Reveal and could deliver tangible benefits to both Bittensor and Commune.

2 Mechanism

The core concept of the mechanism is delaying the onchain availability of weight information while measuring how much consensus penalty the delay carries for copying the last public weights, in order to dynamically minimize its length while satisfying a set constraint of weight-copying relative profitability.

Each encryption window keeps cumulative state of simulated copier dividends $D_c(e)$ and average delegation dividends $D_d(e)$ for N stake at each contained epoch e.

We define copier weights as the last public consensus weights Q and simulate copier dividends $D_c(e)$ by first appending the weight vector Q to the weight matrix of real validators W_e: $\begin{bmatrix} W_e \\ Q \end{bmatrix} = M_e$, using N copier stake, to then compute consensus $M_e \rightarrow C_e$ and retrieve dividends for the simulated copier UID $D_c(e)$. We calculate the average delegation dividends $D_d(e)$ by dividing the epoch's sum of dividends by the sum of active stake, scale the result by N stake and substract the minimum taker fee:
$D_d(e) = \left(\frac{N \cdot \sum \text{Dividends}_e}{\sum \text{Active Stake}}\right) \cdot (1 - minTakerFee)$ We set N as 5% of the activate stake.

We define **full copier-irrationality** as the threshold where weight copying is less profitable than delegating to an average validator with minimum taker fee $D_d(e) < D_c(e)$ while assuming zero operational cost for the copier and delegator.

We see no conceivable scenario where a further reduction of copier profitability than below average delegation dividends would be necessary to mitigate weight copying, but many scenarios in which less reduction would be desirable in order to 1) reduce the average delay-length or 2) give a margin of rationality for the weight copying strategy, i.e. to leverage its advantages (section 1) while still ensuring lower profitability than running an average honest validator.

For this purpose we introduce the *copier_margin* parameter $p \in [0, 1]$: $D_c(e) < (1 + p) \cdot D_d(e)$, to give subnets control over the tradeoff between mitigating weight copying and delaying epochs (backtests in sector N), allowing to set a rationality margin for weight copying as a strategy.

For each epoch e within the encryption window we check if the constraint of $D_c(e) < (1 + p) \cdot D_d(e)$ is satisfied to time decryption and execution of pending epochs,

resetting the state variables to zero $e \leftarrow 0$, $D_c(e) \leftarrow 0$, $D_d(e) \leftarrow 0$ and opening a new encryption window.

Since the delay-length fully depends on the delta accrued between the hidden epochs consensus weights and last public epochs consensus weights Q, periods of staleness with little incentive distribution variability could lead to excessive delays. In order to prevent this, we introduce a max_delay $Z \in \mathbb{N}^+ \cup \{\text{off}\}$ (epochs) parameter, giving subnets the option to set a maximum delay length.

The $copier\_margin$ & max_delay parameters provide tools to subnet owners to control all elements of the mechanism for which considerations depend on their local context, while shielding them against all other complexity.

Mechanism Adaptation for Optimal Efficiency at cost of constraint guarantee local to each encryption window:
In the above described mechanism exists a degree of inefficiency which is necessary to guarantee the copier relative profitability constraint local to each encryption window. Between the preliminary epoch e_{n-1} and final epoch e_n of each encryption window $E = \{e_1, e_2, \ldots, e_n\}$ could be a larger consensus penalty for the copier D_c than necessary to satisfy the constraint $D_c(e) < (1 + p) \cdot D_d(e)$, producing an excess copier-penalty. After nulling D_c and D_d uppon decryption, that excess is lost from measurement, yet for continuously operating validators it is just as relevant to their consideration of weight copying rationality. This means across multiple encryption windows a potentially significant quantity of copier-penalty excess accumulates, which if applied to the measurement could reduce the average delay length by meaningful margins.

We can fix this inefficiency by instead of nulling both D_c and D_d, we substract $D_d(e)$ with applied $copier\_margin$ p from $D_c(e)$ to take the absolute value of the result, add an epsilon to adjust for the constraint checking $<$, not $=$ and overwrite D_c to start the next encryption window $D_d \leftarrow |D_c(e) - (1 + p)D_d(e)| + \epsilon$, while setting $D_c \leftarrow 0$. This way, the copier-penalty excess is fully transferred to the next encryption window and hence reduces the average delay length.

This comes at the tradeoff of losing the constraint satisfaction guarantee local to encryption windows, while keeping it across multiple encryption windows, which is what matters in practice. However, new weight copiers joining the network may be able to operate profitably for a limited period based on excess copier penalty accumulated from the previous window(s).

We run a comprehensive backtest comparing the average delay-length averaged over $0 \leq p \leq 0.09$ for each major Bittensor subnet in Section 4, showing that this adaptation can significantly reduce the delay. We conclude the above explained tradeoff is worthwhile and are hence picking this version for our implementation.

2.1 Definitions

Let:

- $E = \{e_1, e_2, \ldots, e_n\}$ be the set of epochs, where each $e \in E$
- W_e be the matrix of real validator weights at epoch e
- Q be the vector of consensus weights of the last public epoch (copier weights), constant for the duration of the encryption window
- $M_e = \begin{bmatrix} W_e \\ Q \end{bmatrix}$ be the simulated weight matrix after appending the copier weight vector Q to the real weight matrix W_e
- C_e be the consensus state at epoch e
- minTakerFee $\in \mathbb{R}_0^+$ be the current blockchain parameter value for the minimum taker fee
- *copier_margin* $p \in [0, 1]$ be the allowed relative profit % margin above full irrationality for the weight copying strategy
- $N \in \mathbb{R}^+$ be the measured stake amount
- *max_delay* $\in \mathbb{N}^+ \cup \{\text{off}\}$ be the maximum encryption window length (epochs) parameter set by the subnet owner

2.2 Functions

- sim_consensus : $M_e \to C_e$
- copier_dividends : $N \times C_e \to \mathbb{R}_0^+$
- avg_delegation_dividends : $(N, \sum \text{Dividends}_e, \text{Active Stake}, \text{minTakerFee}) \to \mathbb{R}_0^+$

2.3 State Variables

- $D_c(e)$: Cumulative copier dividends, defined as:

$$D_c(e) = \sum_{i=1}^{e} \text{copier\_dividends}(N, Q, C_i)$$

  representing the sum of copier dividends from the first epoch (e_1) up to the current epoch e for N Stake

- $D_d(e)$: Cumulative average delegation dividends, defined as:

$$D_d(e) = \sum_{i=1}^{e} \text{avg\_delegation\_dividends}(N, C_i, minTakerFee)$$

  representing the sum of average delegation dividends from the first epoch (e_1) up to the current epoch e for N Stake

2.4 Objective

The mechanism minimizes the duration of the encryption window, while ensuring the constraint of relative copier profitability is always met with optional allowance of a $p\%$ relative profit margin $copier\_margin$ above full copier-irrationality:

$$\underbrace{D_c(e)}_{\substack{\text{cumulative}\\ \text{copier}\\ \text{dividends}}} < \underbrace{(1+p)}_{\substack{\text{copier}\\ \text{rationality}\\ \text{margin}}} \cdot \underbrace{D_d(e)}_{\substack{\text{cumulative}\\ \text{avg delegation}\\ \text{dividends}}} \tag{1}$$

where:

- $D_c(e)$ is the cumulative copier dividends at epoch e
- $D_d(e)$ is the cumulative average delegation dividends at epoch e
- p is the allowed profit % $copier\_margin$ of rationality above full copier-irrationality

2.5 Algorithm

We now formalize the dynamic encryption window algorithm:

Algorithm 1 Dynamic Encryption Window

Inputs:
Q ▷ Vector of consensus weights of the last public epoch (copier weights)
W_e ▷ Matrix of real validator weights for each epoch e
$N \in \mathbb{R}^+$ ▷ Measured stake amount
$\text{minTakerFee} \in \mathbb{R}_0^+$ ▷ Minimum taker fee (blockchain parameter)
$copier\_margin\ p \in [0, 1]$ ▷ allowed % profit margin above full copier-irrationality
$max\_delay \in \mathbb{N}^+ \cup \{\text{off}\}$ ▷ Maximum delay length (epochs)
procedure INITIALIZE
    $D_d(e) \leftarrow |D_c(e) - (1+p)D_d(e)| + \epsilon$
    $e \leftarrow 0$
    $D_c(0) \leftarrow 0$
    Open encryption window
end procedure
function PROCESSNEWEPOCH(e)
    $M_e \leftarrow \begin{bmatrix} W_e \\ Q \end{bmatrix}$
    $C_e \leftarrow \text{sim\_consensus}(M_e)$
    $d_c \leftarrow \text{copier\_dividends}(N, C_e)$
    $d_d \leftarrow \text{avg\_delegate\_dividends}(N, C_e, \text{minTakerFee})$
    $D_c(e) \leftarrow D_c(e-1) + d_c$
    $D_d(e) \leftarrow D_d(e-1) + d_d$
    if $D_c(e) < (1+p) \cdot D_d(e)$ **then**
        Close encryption window
        Execute all pending epochs
        INITIALIZE
    end if
end function
Main:
INITIALIZE
On event **NewEpoch**:
    $e \leftarrow e + 1$
    if $max\_delay \neq \text{off} \wedge e > max\_delay$ **then**
        Close encryption window
        Execute all pending epochs
        INITIALIZE
    PROCESSNEWEPOCH(e)

3 Implementation

Apparent Challenges

Measuring Carried Consensus Penalty To measure the carried consensus penalty, we must compute on the weights without revealing their information. This necessitates an encryption-decryption scheme, as the weight information must reach the chain nodes, rather than a commit-reveal scheme where only a hash of the weights is submitted.

Implementation at Chain Level In order to allow application to subnets by setting a parameter, it has to be implemented on the blockchain. Currently both the Bittensor and Commune Blockchain run on proof of authority mode, in which the authority nodes should perform the mechanism process. We discuss implementation with full chain decentralization later in this section.

Trustlessness To maintain the trustless characteristic of the system:

- It must be provable that weights were not tampered with.
- Decryption should occur in an off-chain, isolated cryptographic environment.
- This environment should make it extremely challenging for machine operators to extract weights before the encryption window decrypts publicly.
- Subnets' public decryption keys should be regularly rotated and evenly distributed to prevent node centralization.

Strategy

The high-level process of our implementation strategy is illustrated in Figure 1 and detailed below:

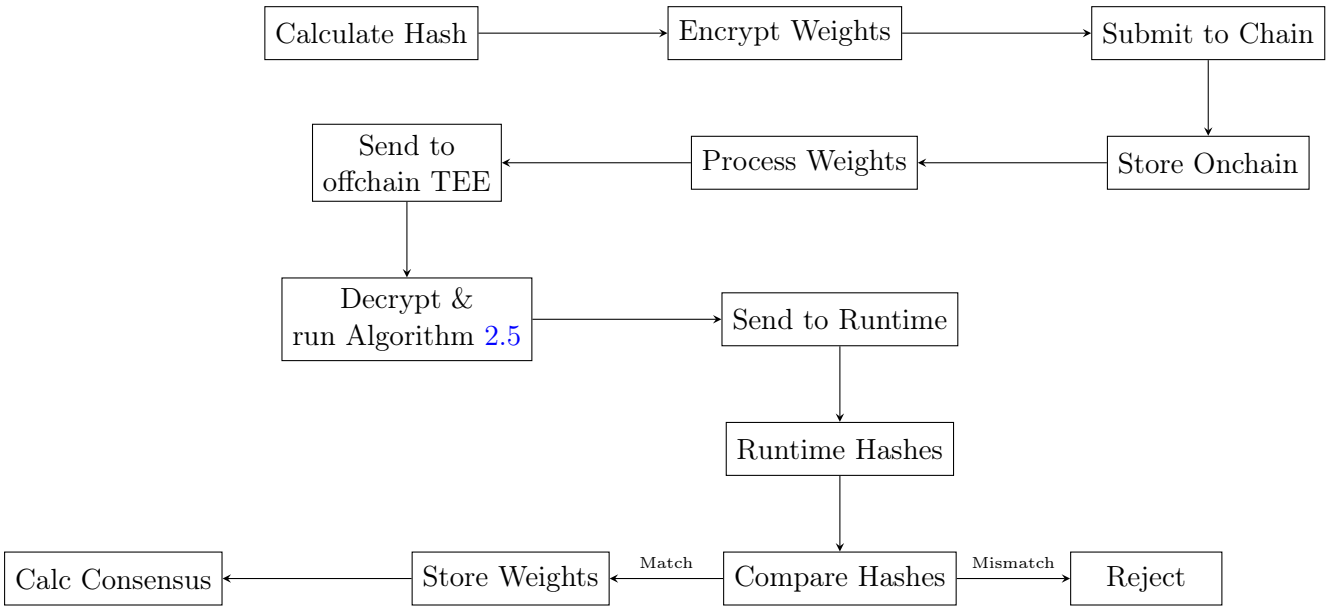

Figure 1: High-level process flowchart

1. **Encryption of Validator Weights:** The weight setting function calculates the hash of decrypted weights, then encrypts validator weights using asymmetric encryption before submitting both hash and encrypted weights to the blockchain. The encryption scheme incorporates randomization to prevent weight copiers from building rainbow tables.

2. **Submission and Storage of Encrypted Weights:** Encrypted weights and weights hashes are stored on-chain via extrinsic call

3. **Decryption and Processing by Nodes:** Specific nodes, designated as weight decryption nodes, are authorized to decrypt and process the weight data offchain. These nodes perform the following tasks:

   (a) Decrypt the weight data for subnets they are assigned to.

   (b) Perform consensus calculations using the decrypted weights.

   (c) Run the DEW Algorithm 2.5 to determine if the relative copier profitability constraint is satisfied.

   (d) If satisfied, submit the decrypted weights back to the runtime.

4. **Runtime Handling of Decrypted Weights:** When decrypted weights are submitted back to the runtime:

   (a) The runtime verifies weight authenticity by comparing hashes of the validator's original weights and the offchain worker's submitted weights, preventing tampering by the decryption node.

   (b) For every epoch with decrypted weights now available, consensus is calculated and distributed, saving the decrypted weights to a runtime storage.

   (c) The system may reassign decryption responsibilities to different nodes periodically, in a round robin like fashion.

This structure ensures even distribution of weights across decryption nodes and prevents tampering through hash comparison.

3.1 Compatability with PoS Chain Decentralization

The high-level methodology of our implementation strategy is designating an arbitrarily large or small set of nodes with the ability and responsibility to decrypt weights in an off-chain Trusted Execution Environment TEE (enclave). This approach provides strong protection against weight inspection by the machine owner while verifying on-chain that weights have not been tampered with. This role could be perfectly executed by subnet owners, which is compatible with full chain decentralization.

4 Backtests and Analysis

We run the Dynamic Encryption Window Algorithm 2.5 on historic Bittensor subnet metagraph data to collect statistics and conduct analysis, to provide insight into the mechanisms behavior in practice for different subnet-types, under different parameter setting.
We test on metagraph data between block 3,600,000 and block 3,744,000 (400 tempos)
We apply liquid alpha with default parameters for all tests.

We start by visualizing the optimal delay length for the only subnet, at time of testing, actively applying commit-reveal (SN12).

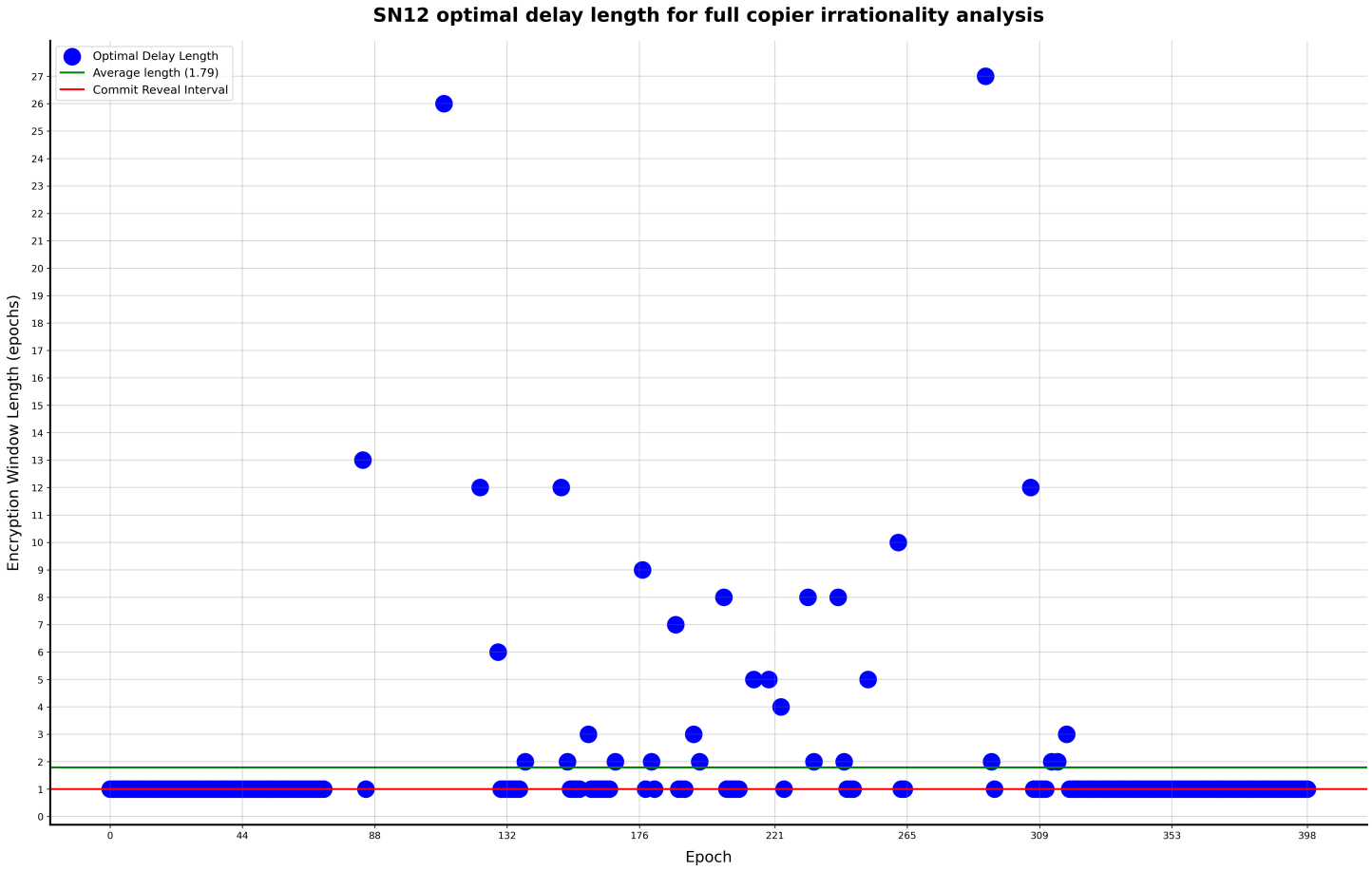

Figure 2: Display of the minimum delay-length (optimal) for full copier irrationality (p=0), in comparison to the Commit-Reveal delay-length on Bittensor SN12 over 306 epochs

The plot shows that Commit-Reveal is often too short to achieve copier-irrationality. The average optimal length is 1.79 epochs, while the Commit-Reveal interval is set at 1 epoch. However, while inefficient Commit-Reveal is well compatible with the SN12 landscape and is still going to yield good results in mitigating weight copying.

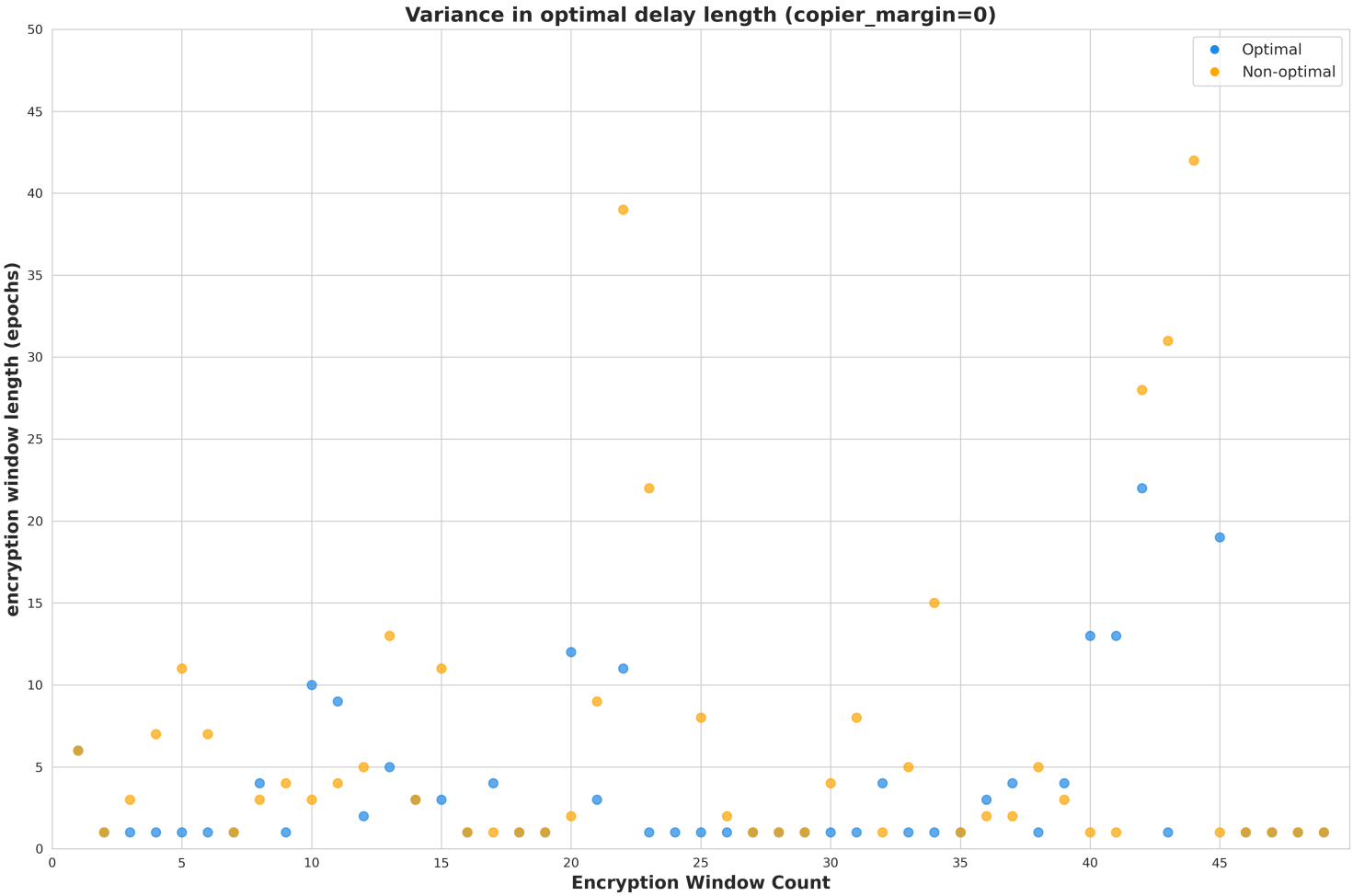

Figure 3: Displays the variance in optimal delay length on SN9 for both the optimal and non-optimal (local constraint guarantee) mechanism 2.5

The high volatility in optimal delay length highlights the benefits of a dynamic delay, as any deviation from the optimal length causes either not accomplishing sufficient copier penalty or paying the redundant cost of delaying longer than necessary.

Next we analyse the influence of the $copier\_margin$ p parameter on the average delay length on 14 major Bittensor subnets.

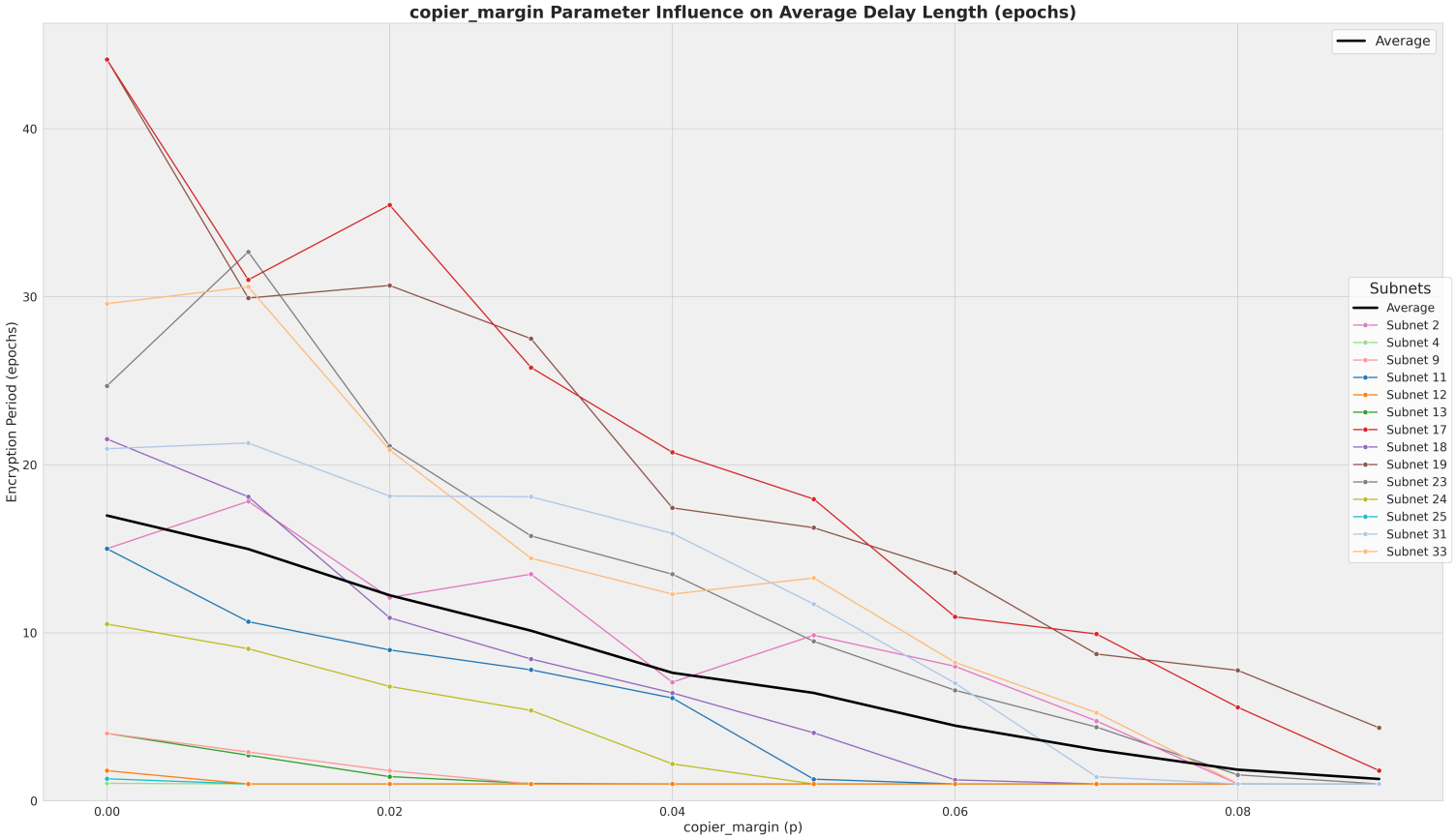

Figure 4: Displays the average delay-length under different $copier\_margin$ (p) parameter setting, using the dynamic encryption window mechanism. Assuming 9% delegation fee.

$p = 0$ is full copier-irrationality and $p = 0.09$ is the maximum possible $copier\_margin$ while ensuring weight copying is not more profitable than running an average validator.

The results show that most subnets have to accept a very long delay in order to achieve full copier-irrationality (p=0), but by allowing a profit margin for the weight copying strategy they can drastically reduce the average delay-length to acceptable values while still ensuring weight-copying is less profitable than honest validation.

This indicates that the right $copier\_margin$ parameter is unique to each subnet and depends on its tendencies for incentive-distribution-variability and local context.

We now analyze how copier consensus penalty is usually distributed across epochs within encryption windows, by calculating the gini coefficient over the portions of the total copier consensus penalty across epochs contained in each encryption window.

Figure 5: Displays the smoothed gini coefficient for each encryption window in the analysed data of major Bittensor subnets

High gini coefficient means that consensus penalty for the copier is distributed unevenly across the epochs contained in the encryption window, meaning with gini coefficent 0.5 dependent on the length likely one epoch was responsible for the vast majority of incentive-distribution-delta, while the distribution was largely stale during the rest of the epochs.

As expected, subnets with large average encryption lengths (evident based on their low count of windows relative to other subnets over the same timespan) are dense in high gini coefficients (i.e. SN19, SN18, SN17, SN33), indicating that those subnets incentive-distribution has too little variability for delay-based methods to yield good results, while every once in a while a larger change occurs in a short amount of time which then produces sufficient consensus penalty for the copier to decrypt the window.

Further on more moderate subnets, such as SN9, SN12, SN13 we can see phases of low consistent variability (high gini coefficient) come in clusters which aligns with the reality of different human entities competing with pauses in deploying advancements or scaling registrations.

We now analyze the efficiency differences between the suboptimal mechanism with constraint satisfaction guarantee local to the encryption window, and the optimal mechanism with only a global cross-encryption window constraint satisfaction guarantee.

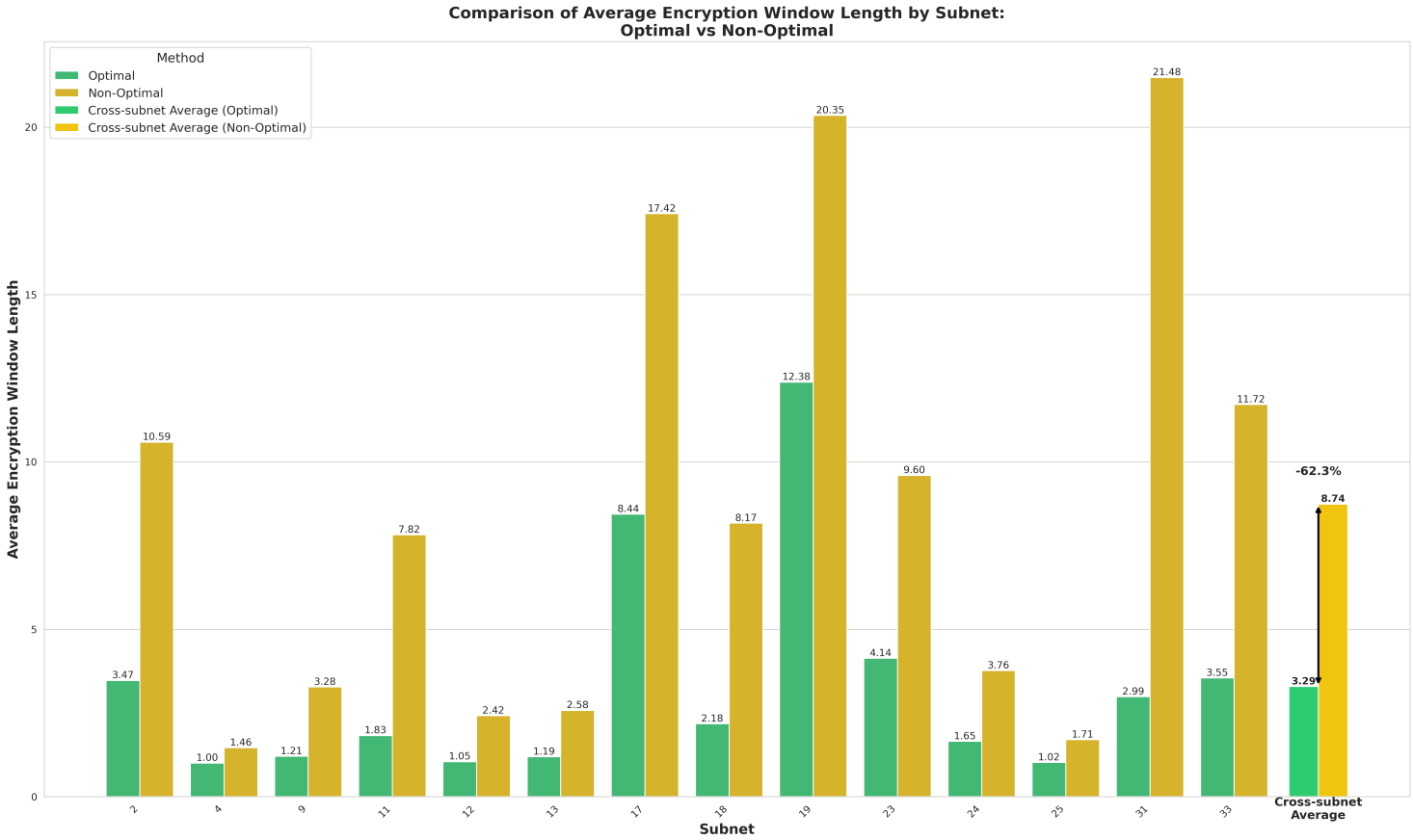

Figure 6: Displays the comparison in average encryption length across subnets between the Optimal less strict, and non-optimal fully strict mechanism detailed in Section 2

The difference in efficiency on average across all averages is a surprisingly significant -62.3% in average encryption length. This exceeded our expectations but does make sense when considering that incentive-distribution-variability is usually triggered by changes in the landscape, such as a large miner changing their model across all keys. Those changes can be so large in a single epoch, that they produce a large excess in weight-distribution-delta above what would be needed to produce sufficient copier consensus penalty to satisfy the mechanism constraint. The optimal mechanism adds the excess to the next window, while the non-optimal disregards it.

This backtest makes it clear the tradeoff of loosing the constraint satisfaction guarantee local to encryption windows is worthwhile to take with no apparent exceptions.

In the final backtest, we conduct the same analysis on the impact of liquid alpha on the average delay length, under use of the optimal mechanism version. Note that all prior backtests were applying liquid alpha with default parameters.

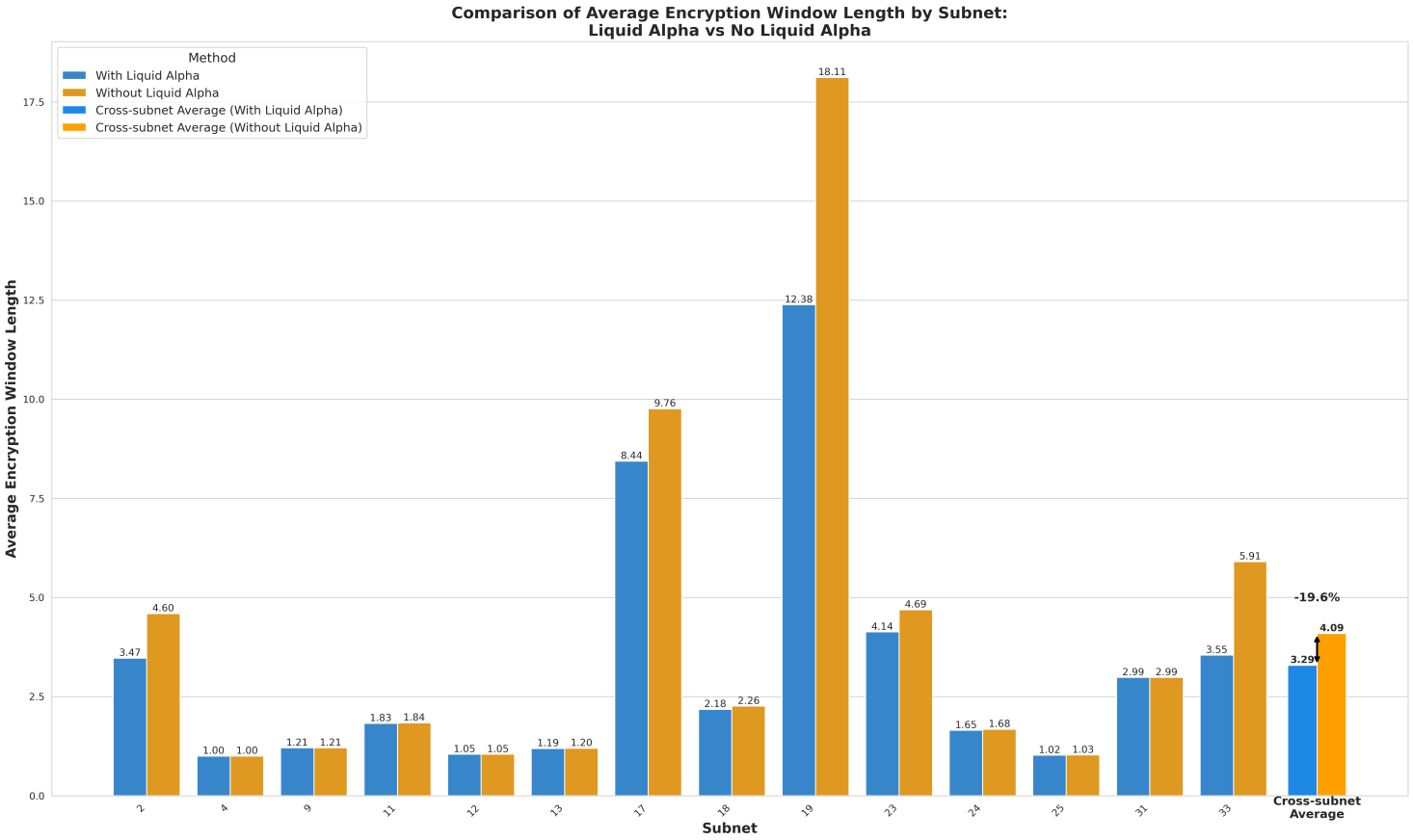

Figure 7: Displays the impact on average encryption length across subnets of applying liquid alpha with default parameters, using the optimal mechanism

Liquid alpha shows a significant reduction of -19.6% in average delay and is evident to be a great addition to any delay-based approach.

5 Closing Comments and future work

The delay-based paradigm is only one way of addressing weight copying, and fails to address offchain sharing of weights. However, the most crucial problem of weight copiers outperforming honest validators is solved, as obtaining the reward-optimal weight offchain consistently is virtually impossible. We expect the popularity of the copying strategy and the proportion of total stake allocated to it to decrease substantially. Child keys are further contributing to this, by giving validators an option to redelegate their stake on subnets they dont want to validate, no longer forcing them to copy weights.

We strongly believe all this is sufficient to lessen the weight copying issue to the degree, that hopefully it can disappear from the daily discussions and concerns of the community, freeing their working memory to focus on more productive things.

This paper is our take on the endgame for delay-based weight copying mitigation and we consider the problem reduced to insignificance, but we don't think this is the end for mitigating weight copying in general. The research by Inference Labs on proof of weights is promising to actually guarantee real validation work was performed within the constraints set by the subnet owner, which will be a significant advancement beyond just the issue of weight copying.

Future delay-based Work In order to reduce the miner feedback-delay tradeoff further, subnet validators could standardize always sending the score back to miners directly such that only feedback on their performance relative to other miners is delayed, not the performance against the scoring function.

The bond penalty factor (beta β) can be boosted directly, to further reduce the needed delay. Since bond penalty is only applied to above consensus weights (that were clipped), an amplification would have to also be applied to below-consensus weights to mirror the above-consensus penalty minus the already intrinsic penalty of having a lower bond in an higher incentive miner. Increasing bond penalty however comes at the tradeoff of increasing penalty on honest validators on subnets with scoring variance due to i.e. probabilistic components in the scoring function. Adding a boosted beta subnet parameter, granting the option to subnets where benefits outweight costs, could be a reasonable step in enhancing the owners toolset to regulate the subnet landscape, and support delay-based weight copying mitigation.

References

Omron AI. Proof of weights. https://docs.omron.ai/proof-of-weights, 2024. URL https://docs.omron.ai/proof-of-weights.

Bittensor. Bittensor: A peer-to-peer intelligence market. https://bittensor.com/whitepaper, 2021. Accessed: 2024-03-15.

Opentensor. Weight copying in bittensor (working paper). Technical report, Opentensor, May 2024a. URL https://github.com/opentensor/developer-docs/blob/main/static/papers/BT_Weight_Copier-29May2024.pdf. Published on May 29, 2024.

Opentensor. Amplifying the weight-copying penalty in bittensor (working paper). Technical report, Opentensor, July 2024b. URL https://github.com/opentensor/developer-docs/blob/main/static/papers/BT-Consensus-based-Weights.pdf. Published on July 12, 2024.