PEARL // whitepaper

PEARL Research Labs

Abstract

This document describes the design of the PEARL network — a Proof-of-Useful-Work L1 protocol, where mining is produced natively from AI computations. The heart of the protocol is a new and efficient implementation of the core GPU operation (matrix-multiplication), allowing GPUs to implement proof-of-work as a *side-effect* of AI training and inference workloads (2-for-1). As such, the PEARL protocol intertwines *energy, data, and money* into a single *atomic* digital asset. This document outlines the protocol design, key implementation choices, and various economic aspects of the system.

Contents

1 Introduction

One of the biggest conceptual contributions of Bitcoin is turning *electricity into currency*: Bitcoin showed that scarce, verifiable energy can be transmuted into digital scarcity and credible neutrality. Alongside its sweeping success and adoption, Bitcoin mining taps to a niche, *artificial* computational task of performing random hashing, nowadays applicable only to specialized hardware (ASICs). By contrast, Artificial intelligence (AI) is projected to consume the vast majority of global electricity within a decade.[1] Indeed, it is increasingly clear that in the age of LLMs, the fundamental barrier of AI progress is neither models, algorithms nor hardware (GPUs), but the production and availability of *energy* for training and inference. Our central thesis is simple:

> *A permissionless monetary network that replaces Bitcoin's wasteful proof-of-work mining (artificial hashing) with the* native *operation underlying modern AI:* matrix multiplication *(MatMul). As such,* Pearl *turns* general compute *on commodity hardware (GPUs) into a monetary currency,* directly *leveraging AI growth to secure the network.*
>
> **Pearl *is the Bitcoin of the AI compute era.***

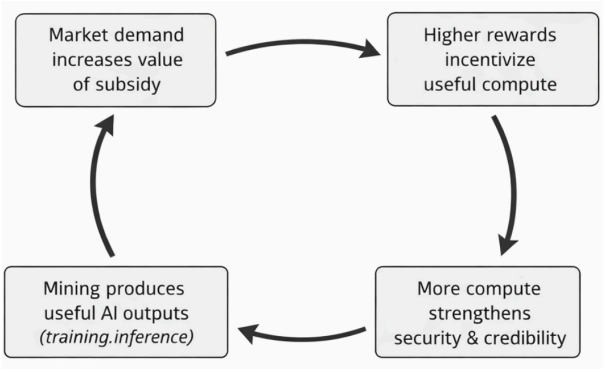
Motivation. AI is bottle-necked by the availability of energy. At the same time, networks such as Bitcoin require enormous amounts of energy and essentially are in direct competition with AI. Therefore, it is necessary to create an AI native Proof of Work blockchain which acts as a meter for the AI economy. Notably, such a meter cannot be tied to clicks or API calls, but rather GPU cycles. Pearl does exactly that: block rewards are generated in direct proportion to (verifiable) Matrix Multiplications, tying issuance to the core operation of GPUs that drives AI. As a result, with Pearl enabled, each kilowatt-hour spent on AI can *simultaneously* earn mining rewards and secure a decentralized network.

New unit economics for AI. Classical PoW monetizes *security*. Pearl monetizes *security and utility*, meaning that payout for security can be used to *subsidize useful work*. That is, miners can *use AI workloads (training and inference) for mining*, creating a *parallel revenue stream* from the exact same GPU cycles. The result is a virtuous loop:

Not only does the above improve the unit economics of AI companies, this dual-utility design also *increases the throughput of GPU providers*. Because Pearl mining is tiled, kernel-level, and parallel, it interleaves with normal AI computation with negligible overhead. Providers extract yield from idle fragments, pipeline stalls, and micro-batches, turning once-wasted headroom into block-eligible mining work.

[1] `https://economictimes.indiatimes.com/magazines/panache/former-google-ceo-eric-schmidt-sounds-alarm-on-ai-data-centers-soaring-power-demand-we-need-energy-in-all-forms/articleshow/121036712.cms`.

1. Market demand increases the value of the mining subsidy.
2. Higher reward value incentivizes the deployment of more useful compute into the network.
3. Greater compute power hardens the network's consensus security and its monetary credibility.
4. The mining process produces useful outputs (inference/training) that accrue real economic value.



Pearl *merges the world's two largest energy-consuming digital markets* (AI compute and cryptocurrency mining) into a single GPU-native operation. Practically, we integrate a new *MatMul mining kernel* into existing AI frameworks and runtimes. Our drop-in plugin implementation augments these calls with negligible additional operations to facilitate mining.

Towards a marketplace of useful compute. Pearl creates a correspondence between a real asset (GPU/TPU) performing a real action (matrix multiplications) and a token. Naturally, the flow is one-directional: useful compute allows one to mine tokens, but tokens are not automatically convertible into useful compute.

Pearl is suitable to close this loop. Indeed, Pearl attracts compute that does not necessarily have useful work; at the same time, demand for compute is constantly increasing, and a marketplace built on top of Pearl could match supply and demand. In the future, one can even envision native (on-chain) settlement of compute contracts: the chain can verify that a certain amount of computation happened on account of Alice asking Bob, and settle the payment – all more efficiently than existing contracting solutions. Having a marketplace for useful compute would allow to combine tokens mined from different tasks (such as MatMuls on different hardware and precisions) into a single token, using settlement value to calculate the fair exchange between hash rates. In addition, it would tie the value of the token with the value of useful compute on the network, giving participants access to a token linked to % of future useful compute streams.

1.1 Planned Upgrades

The core of the Pearl network is a new implementation of *exact* (INT) MatMul operation. As such it can be used as a drop-in replacement in applications that can be cast to this setting, e.g., inference. However, many modern AI workloads are not done natively in INT, but rather in low-precision *floating-point* formats (e.g., BF16 for training, and FP8 and FP4 for inference). The continued shift toward low-precision floating point is driven largely by throughput and I/O considerations: low-precision weights and activations can significantly increase *tokens-per-second* (TPS), a key KPI in many production systems (e.g., coding assistants). This speed/accuracy tradeoff is the essence of *model quantization*, and by now is a standard component of modern AI training and inference pipelines.

Extending the current PoUW scheme to *floating-point* datatypes is technically challenging.[2] Thus, in parallel to the implementation of Pearl, we have been developing a future PoUW protocol designed to address *both*

[2]Floating-point (FP) formats have a non-uniform dynamic range. As a result, even small perturbations to the input matrices (A, B) can produce disproportionately large changes in intermediate values, harming numerical stability and downstream accuracy

challenges simultaneously. This future version will not preserve *exact* matrix multiplication. Instead, it computes a *high-accuracy approximate* MatMul, just like *quantized multiplication works*. Thus, we view it as a plug-in quantization library that simultaneously provides a PoW certificate. The key idea is to leverage *native quantization noise* as the perturbation mechanism needed for PoW, while forcing the miner to commit to the underlying (pre-quantization) weight and activation matrices.

A formal specification and security/accuracy analysis of the new scheme will be made public for review in a future technical article and accompanying blog post. The purpose of this subsection is to prepare the Pearl community, *a priori*, for a future upgrade. This upgrade is intended to (i) enable on-boarding of training workloads, and (ii) support state-of-the-art low-precision inference workloads *without adaptation*. We will only propose upgrading the network after we complete extensive testing of both its accuracy and its security.

2 Blockchain Overview

Our system is PoUW blockchain built as a fork of the Bitcoin protocol, integrating the cryptographic proof of useful work mechanisms proposed to replace traditional hash-based PoW with our matrix multiplication variant. While maintaining the features of Bitcoin's security and consensus model, the blockchain introduces key adaptations to support the new proof of work protocol as well as other improvements.

At its core, our blockchain maintains a ledger of unspent transaction outputs (UTXOs), which represent coins available for spending. Transactions in our network consume existing UTXOs as inputs and create new ones as outputs, effectively transferring value. Each transaction is digitally signed using the sender's private key, ensuring authenticity and authorization. All nodes in the network propagate transactions and blocks using a gossip-like protocol over the P2P network, allowing for robust dissemination and fault tolerance.

The ledger is a linear sequence of blocks, each containing a batch of transactions, a timestamp, a reference to the previous block's hash, and a proof satsifying the proof-of-work condition. The PoW algorithm acts as a computational black-box, where given a block header, it requires finding a special input such that a particular condition is satisfied. The condition is adjusted as the competition for solving the black box varies, determining the block's difficulty. This target is recalibrated as the blockchain evolves to maintain consistent (expected) block time (see Section 5.4).

Nodes adhere to the "longest chain rule," which selects the chain with the greatest cumulative work (i.e., the most difficult chain) as the valid one. This rule ensures convergence and consistency in the presence of forks. When multiple chains exist temporarily (e.g., due to propagation delays), nodes continue mining on the one they see as the most difficult, and eventually all nodes converge on a single chain as it extends further. New blocks extend this chain, and only confirmed transactions within the longest chain are considered final.

To prevent double-spending and ensure ordering of transactions, our blockchain relies on the immutability of the blockchain enforced by proof of work mechanism and an economic incentive. Miners who create valid blocks are rewarded with newly minted Pearl coins (block subsidy) and transaction fees, providing both issuance and security. Because the proof of work is computationally costly and rewards are only granted for extending the valid chain, attackers would need to control the majority of the network's total "puzzle solving" power to subvert the system, which becomes economically and physically impractical at scale.

UTXO framework. The UTXO (Unspent Transaction Output) model is the accounting framework used to track the ownership and transfer of value. Unlike an account-based system that maintains balances per

address, the UTXO model defines coins as discrete chunks of value represented by outputs of transactions that have not yet been spent. Each UTXO is uniquely identified by the transaction in which it was created and the index of the output within that transaction.

A transaction consumes existing UTXOs as inputs and produces new outputs that become UTXOs themselves. Each input specifies a reference to a previous transaction's output and includes a cryptographic signature satisfying the conditions set in that output's locking script. This script typically requires a valid digital signature from the private key corresponding to a public address. When a transaction is validated, the node executes the unlocking script provided in the input together with the locking script of the referenced output to check whether the spending conditions are satisfied.

The structure of a transaction thus consists of one or more inputs, each referencing a previous UTXO, and one or more outputs, each specifying a value and a locking script. The sum of the input values must be greater than or equal to the sum of the output values; the difference, if any, is interpreted as a transaction fee and claimed by the miner who includes the transaction in a block.

The global UTXO set is maintained by each full node and represents the current state of spendable outputs. When a new transaction is received, a node checks that all referenced inputs exist in the UTXO set and are unspent, that the signatures are valid, and that no double-spending occurs. Once validated, the transaction updates the UTXO set by removing the consumed inputs and adding the new outputs.

3 PoUW Overview

At the core of our blockchain lies the PoUW protocol, which we describe in this section. Our objective is twofold: to design a PoW protocol that upholds the essential properties required for secure blockchain maintenance, while simultaneously computing a *useful* result, i.e., the product of two arbitrary matrices. Crucially, we demonstrate that this useful computation can be performed concurrently with the PoW mechanism, incurring negligible additional overhead.

PoW Protocols: A PoW protocol enables a party to generate a cryptographic proof that certifies the execution of a certain amount of *computational effort*. This concept underpins the security and fairness of blockchain systems by offering a decentralized and probabilistic mechanism for selecting the next block miner. Selection is distributed proportionally to the computational effort expended by participants. At a high level, let σ denote the current state of the chain (or a succinct digest thereof). A PoW protocol processes σ and yields a verifiable proof or identifier certifying that substantial computational work was performed. These proofs serve as *lottery tickets*, each with a small probability of winning, granting the right to mine the next block. Each valid proof is equally likely to win. To have the expected mining rate aligned with the computational effort invested, we need the number of such proofs a party can produce to be proportional to their computational power. To preserve this fairness, it is essential that the protocol enforces consistent computational cost across all participants, ensuring that each unit of work corresponds to a uniform chance of success.

Matrix Multiplication Algorithms: Matrix multiplication is a foundational operation in a wide range of computational workloads, particularly in the domain of machine learning. Both training and inference in modern ML models, ranging from deep neural networks to linear classifiers, rely extensively on repeated multiplication of matrices. These operations are computationally intensive, highly parallelizable, and occur

abundantly in large-scale deployments, making them a natural fit for underpinning a PoUW protocol. Over the years, a series of theoretical algorithms have been developed to asymptotically improve upon the naive $O(n^3)$ approach. However, in practical settings, especially within high-performance and hardware-accelerated environments, optimized variants of the naive algorithm are overwhelmingly favored. These implementations are better aligned with modern memory hierarchies and vectorized execution models, offering significant performance advantages despite their asymptotic inefficiency. Importantly, these algorithms exhibit highly predictable runtimes that are determined primarily by matrix dimensions, and not by the specific values of the entries.

Our PoUW protocol integrates these two components: proof-of-work generation and matrix multiplication. The protocol takes as input both the blockchain state σ, as required in a standard PoW setting, and a pair of matrices A, B, which serve as inputs to a matrix multiplication algorithm. As output, it yields the product $A \cdot B$, as well as a cryptographic proof that this result was obtained through the execution of a specific matrix multiplication algorithm, constituting a valid "lottery ticket" corresponding to the state σ.

A key design goal is that the total runtime of the PoUW protocol should closely match the runtime of the matrix multiplication itself, ensuring minimal overhead. Moreover, the cryptographic proof must certify not merely the correctness of the output $A \cdot B$, but also that the full matrix multiplication algorithm was faithfully executed. This constraint is crucial to prevent adversarial strategies. For instance, miners attempting to gain an advantage by multiplying trivial or degenerate matrices (e.g., all-zero inputs) in pursuit of a faster PoW. By binding the proof to the computational trace of a genuine algorithm, we ensure fairness across miners and align computational effort with meaningful output.

Privacy. Since the cryptographic proofs generated by the protocol may occasionally become public, namely, when a "winning ticket" is published as part of the blockchain, we must ensure that these proofs do not leak any information about the input matrices A and B. In many applications, such matrices may contain proprietary data, model weights, or sensitive user-derived information. Therefore, it is essential that the proof attests only to the correctness and integrity of the computation, without revealing any details of the inputs themselves. This necessitates the use of *zero-knowledge* techniques, preserving input confidentiality while enabling public validation.

3.1 High-Level Description

A central idea of our protocol is to introduce and eventually remove a carefully constructed form of *noise* in the matrix multiplication process, to preserve computational correctness while unifying computational hardness across all inputs. Specifically, we generate two noise matrices E and F, and add them to the input matrices A and B, respectively. The protocol then proceeds by computing the product of the perturbed matrices $(A + E)$ and $(B + F)$, and producing a proof attesting to the correctness of this computation.

The distribution of the noise matrices E and F needs to be carefully designed. On one hand, the product $(A+E)\cdot(B+F)$ should be as hard to compute as the product of two random matrices, so that an adversary cannot construct A and B in a way that simplifies the computation. This guarantees that the protocol maintains its computational hardness and that no miner can gain an unfair advantage by selecting degenerate or specially structured inputs.

On the other hand, we require that the original matrix product $A \cdot B$ can be efficiently and accurately recovered from the noised product $(A + E) \cdot (B + F)$. This recoverability condition is essential to ensure the *usefulness* of the computation: the final output must yield the intended matrix product, even though the proof was generated with respect to the noised version.

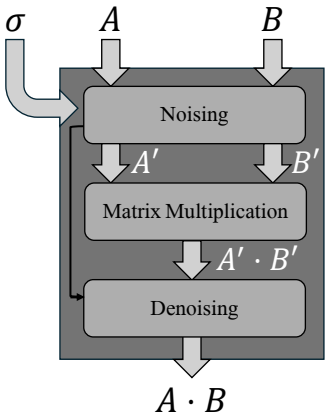


Figure 1: MatMul Framework.

Hence, the protocol, details of which will be elaborated in the subsequent sections, proceeds as follows:

- Given the inputs A, B, and the blockchain state σ, generate corresponding noise matrices E and F.
- Compute the product $(A+E)\cdot(B+F)$, and extract from the execution trace of the matrix multiplication algorithm a cryptographic proof that constitutes a valid proof of work.
- Recover the original product $A \cdot B$ from the result $(A + E) \cdot (B + F)$ via a quick post-processing step.

3.1.1 Low-Rank Noise Paradigm

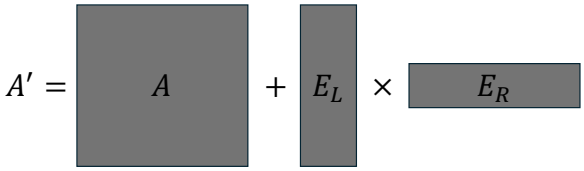


Figure 2: Noise Generation.

We construct E and F as random matrices of rank r, derived deterministically from a seed that is itself a cryptographic commitment to the inputs A, B, and the blockchain state σ. The use of low-rank matrices allows one to efficiently correct the noise. Given that E and F are rank-r matrices, the correction terms $E\cdot B$, $A \cdot F$, and $E \cdot F$ can each be computed using only $O(n^2 r)$ scalar multiplications. Consequently, once the noised product $(A + E) \cdot (B + F)$ is computed, the original product $A \cdot B$ can be recovered via the identity:

$$A \cdot B = (A + E)(B + F) - \big(E \cdot B + A \cdot F + E \cdot F\big).$$

As long as the rank r remains small relative to the matrix dimension n, the post-processing step is asymptotically negligible compared to the main multiplication, preserving the overall efficiency of the protocol. On the other hand, the correction identity is symmetric and can be evaluated in either direction. In particular, it also enables efficient computation of $(A+E)(B+F)$ from a known $A\cdot B$ and the low-rank terms $E\cdot B$, $A\cdot F$,

and $E \cdot F$. This symmetry poses a potential risk: it undermines the guarantee that miners actually performed the full matrix multiplication on the noised inputs, since one could simulate the output of $(A + E)(B + F)$ using only a precomputed $A \cdot B$ and the inexpensive low-rank corrections. To eliminate this shortcut and ensure that the prescribed computation is faithfully executed, our PoW mechanism requires not only knowledge of the resulting product $(A + E)(B + F)$, but also a proof that this product was obtained through a *direct execution* of a valid matrix multiplication algorithm. In essence, the miner must supply a verifiable trace of the actual computation performed on the noised matrices.

3.1.2 Matrix Multiplication with Trace

Nearly all matrix multiplications done in practice are executed using memory-optimized variants of the naive $O(n^3)$ algorithm. These implementations focus on improving cache locality and throughput, while preserving the algorithmic structure of classical matrix multiplication. Leveraging this, we require the miner to provide not only the final output of the matrix multiplication, but also a transcript of the intermediate values computed during the algorithm's execution. This transcript serves as a verifiable trace that the computation was performed *directly* and faithfully.

Let t be a fixed *block size*, and partition the input matrices A and B into non-overlapping $t \times t$ blocks.[3] Assume for simplicity that the matrix dimension n is divisible by t, so that each $n \times n$ matrix consists of $(n/t)^2$ blocks. We denote the block decomposition as:

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,k-1} \\ \vdots & \ddots & \vdots \\ A_{k-1,0} & \cdots & A_{k-1,k-1} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,k-1} \\ \vdots & \ddots & \vdots \\ B_{k-1,0} & \cdots & B_{k-1,k-1} \end{bmatrix},$$

where $k = n/t$ and each block $A_{i,j}, B_{i,j}$ is a t-by-t matrix.

The block-based naive matrix multiplication algorithm proceeds as follows: for each output block $C_{i,j}$ of the product matrix $C = A \cdot B$, initialize it to the zero matrix, and compute

$$C_{i,j} = \sum_{\ell=0}^{k-1} A_{i,\ell} \cdot B_{\ell,j}.$$

Each block product $A_{i,\ell} \cdot B_{\ell,j}$ is itself a $t \times t$ matrix multiplication and contributes to the transcript of intermediate computations. A truthful miner is required to compute all such block-level products and partial sums in their proof, we denote these $(n/t)^3$ intermediate values by

$$C_{i,j,k'} = \sum_{\ell=0}^{k'} A_{i,\ell} \cdot B_{\ell,j}.$$

We note that as long as $t \leq r$, the addition of low-rank noise to the full matrices A and B translates, at the block level, into the addition of full-rank noise, marginally, to each block $A_{i,\ell}$ and $B_{\ell,j}$. That is, although the noise matrices E and F are globally low-rank, their effect on individual $t \times t$ blocks appears indistinguishable from the addition of independent full-rank perturbations. This property plays a crucial role in both preserving the hardness of the computation at the block level, while the necessity to provide the entire transcript essentially forces a block-by-block computation.

[3] For simplicity, we first present the protocol under the assumption the blocks are square and of a fixed size. In the implemented protocol, detailed later, we allow variable and rectangular block sizes to allow hardware-friendly optimizations.

3.1.3 Proof of Work or Successful Mining

The miner maintains for each tile (i, j) a compact internal state

$$M_{i,j} \in \{0,1\}^{512},$$

initialized to zero. After each accumulation step, the current accumulator $C_{i,j,k}$ is compressed into a 32-bit value via an XOR over all tile entries. This value is mixed into $M_{i,j}$ using a lightweight rotation-and-XOR update rule.

After all chunks along the common dimension have been processed, the tile (i, j) is declared *opened* (i.e., it constitutes a valid Proof-of-Work) if

$$\text{BLAKE3}(\mathsf{seed}, M_{i,j}) < 2^{256-b},$$

where b is the global difficulty parameter and the hash output is interpreted as a 256-bit unsigned integer.

A successful mining proof therefore consists of commitments to A and B, The mining configuration and matrix dimensions, the indices (i, j) of an opened tile, and a proof that executing the prescribed noisy tiled matrix multiplication on inputs consistent with the commitments yields a tile whose accumulated hash state satisfies the above inequality.

3.1.4 Hashing and Commitments

To generate the noise matrices and later verify the correctness of a submitted block, our protocol relies on cryptographic *commitments* to the input matrices A and B. These commitments must satisfy the following properties:

- **Tile-level verifiability:** Given a commitment to a matrix, it must be possible to verify that the i-th/j-th strips presented by the prover indeed corresponds to the committed matrix. This can be efficiently implemented using standard cryptographic data structures such as *Merkle trees*, where each leaf encodes a tile and the root serves as the matrix commitment.
- **Binding:** It must be computationally infeasible to find two different matrices that result in the same commitment. This binding property ensures that, once the matrices A and B are committed, the noise generation process—being deterministically seeded by these commitments—introduces true unpredictability and entropy, making it impossible to tailor A and B to influence the noise retroactively.

To determine whether a given block multiplication qualifies as a valid "lottery ticket," a cryptographic hash function is applied to each intermediate product in the computation trace. The hash function h must satisfy two key properties:

- **Randomness extraction:** The hash function must behave as a randomness extractor on the noised block outputs. That is, even though $C_{i,j,k'}$ is deterministically derived from A, B, and the added noise, the unpredictability introduced by the noise should ensure that $C_{i,j,k'}$ is (approximately) uniformly distributed. This guarantees that each block product has an independent and fair chance of producing a winning hash. Standard cryptographic hash functions (e.g., SHA-256) are expected to satisfy this property under standard assumptions.

- **Low overhead:** Hash computation should be asymptotically and practically negligible relative to the cost of the corresponding matrix multiplication. Theoretically, practical $t \times t$ matrix multiplication requires $\Theta(t^3)$ operations, whereas hashing requires $O(t^2)$ time. For sufficiently large block sizes, this ensures that the hashing step does not become a bottleneck. In practice, we incorporate optimized implementations of the hash function to further minimize overhead.

The generation of the noise matrices as well as the intermediate hash tiles themselves are seeded with the commitment itself, implementing the Fiat-Shamir transform.

3.1.5 Privacy and Zero-Knowledge

A core distinction between standard Proof-of-Work protocols and Proof of Useful Work (PoUW) is the potential involvement of sensitive or proprietary data in the useful computation. In many practical settings, especially in machine learning or secure data processing, the input matrices A and B may encode private model weights or user data. Thus, it becomes essential to ensure that participating in the PoUW protocol does not reveal this information to other chain participants.

Our protocol begins to address this concern through the use of *commitments*. Since the noise matrices E and F are generated deterministically from a seed derived solely from the commitments to A, B, and the blockchain state σ, the full protocol can be executed and verified without ever revealing the actual contents of A and B. The commitments themselves are included in the proof, thereby allowing public verification of the computation trace while preserving the secrecy of the original inputs.

However, an important caveat arises: the winning (i, j) strips which determines whether the PoW condition is satisfied, is revealed as part of the proof. Since these strips directly depend on sub-matrices of A and B, it may leak partial information about their contents.

To address this, we incorporate a *zero-knowledge proof* into the protocol. This proof attests to the following statement:

> "There exist strips consistent with the commitments to A and B,
> such that their product yields an accumulated digest $M_{i,j}$ whose
> BLAKE3 hash equals a published value h."

The hash h is then verified in plaintext to ensure it meets the mining condition induced by chain state σ. This zero-knowledge component ensures that the verifier is convinced of the validity of the claimed multiplication and its consistency with the committed matrices, without learning anything beyond h. In doing so, we preserve both the verifiability and the privacy of the useful computation, aligning the protocol with modern requirements for secure decentralized computation.

On ASIC resistance. The protocol plausibly achieves ASIC-resistance by an economic argument. Since matrix multiplication is the native language of all modern AI accelerators (GPUs, TPUs), it is already the primary optimization target for the entire semiconductor industry. Additionally, an ASIC specifically designed to mine Pearl, will lose the ability to do useful work, and thereby will be overall less profitable.

4 PoUW Implementation Details

This section specifies the components that realize the prover (miner) and verifier.

4.1 Mining Configuration

Our protocol requires the miner to set the following parameters.

- **Common dimension k.** The dimension common to both matrices A, B.
- **Noise rank r.** The rank used for the noise factors that are added and later removed.
- **Tile shape (t_m, t_n).** A periodic partition of rows of A and columns of B. For exposition, we assume A is divided into disjoint blocks of t_m consecutive rows, and B into strips of contiguous t_n columns.
- **Difficulty target b.** A fractional number of bits that defines the logarithmic acceptance probability per `int8` multiply-add operation. Larger b yields fewer accepted tiles.
- **Matmul-accumulate type.** An identifier of the exact matmul algorithm being done. Initially, must be 0 denoting input matrices have $[-64, 64]$ entries and matmul-accumulate being done in a `int32` datatype.

4.2 MatMul Framework

Given matrices A, B to multiply, we first run our *commitment hash* with A, B, the mining configuration μ, and the blockchain state σ as its input; resulting in two 256-bits seeds s_A and s_B depending on A, B, μ, σ. We then *generate noise matrices* $E := E_L \cdot E_R,\ F := F_L \cdot F_R$ using s_A as the seed for generating E and s_B as the seed for generating F. We compute the noised matrices $A' := A + E,\ B' = B + F$. We then run a *MatMul algorithm* on A', B' that works tile-by-tile and checks a block-opening condition on each computed tile. Finally, we peel off the noise and return

$$A \cdot B = A' \cdot B' - (A \cdot F_L) \cdot F_R - E_L \cdot (E_R \cdot B'),$$

which we can compute quickly due to the shapes of E_L, E_R, F_L, F_R.

For protocols over 8-bit integers, we quantize both A, B to integers in $[-64, 64]$ and the noise matrices E, F are to the range $[-63, 63]$. That way, the noised matrix also fits in 8-bit integers without overflows.

Algorithm 1 Noisy MatMul Framework

Require: Matrices A, B, miner config μ, state σ
Ensure: Product $C = AB$, a list Blocks of zero or more opened-blocks

1: $(s_A, s_B) \leftarrow \textsc{CommitmentHash}(A, B, \mu, \sigma)$
2: $(E_L, E_R) \leftarrow \textsc{NoiseGeneration}(m, k; \text{key} = s_A)$
3: $(F_R^t, F_L^t) \leftarrow \textsc{NoiseGeneration}(n, k; \text{key} = s_B)$
4: $A' \leftarrow A + E_L E_R, \quad B' \leftarrow B + F_L F_R$
5: $C', \text{Blocks} \leftarrow \textsc{TiledMatMul}(A', B')$
6: $C \leftarrow C' - (AF_L)F_R - E_L(E_R B')$
7: **return** C, Blocks

4.3 Commitment Hash

The noise seeds s_A, s_B, also called the *commitment hashes*, depend on BLAKE3 hashes H_A, H_B of A, B respectively, on the mining configuration μ and the blockchain state σ.

H_A is computed as the BLAKE3 hash of the matrix A parsed as a row-major stream, keyed with μ and σ. H_B is computed likewise but with B parsed column-major. s_B is the hash of μ, σ, H_B while s_A is the hash of s_B, H_A.

The reasons for this choice of derivation of s_A, s_B are:

- H_A, H_B are keyed hashes of A and B to forbid preparing matrices with particularly advantageous hashes.
- A is parsed row-major while B column-major to reduce the size of the proof. A proof of block opening only involves revealing a few rows of A and columns of B. This is because BLAKE3 is a Merkle tree of the hashed data, thus only a Merkle proof for the elements involved in the computation of the winning tiles need to be revealed.
- The reason for asymmetry between the derivation of s_A, s_B in the protocol is that commonly in AI inference, the matrix B is known in advance. Hence, allowing F not depend on A allows the optimization of pre-noising B once per σ update.

Algorithm 2 Compute Commitment Hash

Require: Tensors A, B, key σ
Ensure: Commitment hashes of A and B
1: $\kappa \leftarrow \text{BLAKE3}(\sigma \| \mu)$
2: $H_A \leftarrow \text{BLAKE3}(\mathsf{Flatten}(A), \text{key}=\kappa)$
3: $H_B \leftarrow \text{BLAKE3}(\mathsf{Flatten}(B^T), \text{key}=\kappa)$
4: $s_B \leftarrow \text{BLAKE3}(\kappa \| H_B)$
5: $s_A \leftarrow \text{BLAKE3}(s_B \| H_A)$
6: **return** (s_A, s_B)

4.4 Noise Matrices Generation

Seeded with the commitment hashes s_A, s_B, we generate two low-rank noise matrices $E := E_L \cdot E_R$ and $F := F_L \cdot F_R$, where the rank of E (and F), and thus also the common dimension of E_L, E_R (and of F_L, F_R) is the chosen parameter r.

Each entry in the matrix E_L is drawn uniformly *over all signed* 6*-bit integers*, using BLAKE3 applied to a message containing the entry index and keyed with s_A as a pseudo-random generator (PRNG).

The matrix E_R, on the other hand, is a column-wise "selection" matrix in which every column has a single 1 and a single -1 in two uniformly random distinct positions, chosen by the same PRNG but with domain separation.

The matrices F_L, F_R are drawn likewise using the seed s_B but with F_L sharing the same distribution as E_R^t and F_R sharing the distribution of E_L^t.

This choice has the following features.

- E, F are of rank r, allowing for efficient peeling.
- Each entry of E and F has high entropy (6.7), forbidding significant non-useful speedups.

- Even with non-useful $A = B = 0$, computing all matmul tiles of $E \cdot F$ using identities such as

$$E \cdot F = (E_L E_R)(F_L F_R) = E_L(E_R F_L F_R) = E_L(E_R F_L) F_R,$$

  do not exhibit apparent speedups, as long as $k \leq O(r^2)$.

Algorithm 3 Noise Generation

Require: Dimensions m, k, rank r, key s
Ensure: Tensors E_L, E_R
1: $E_L \leftarrow \textsc{UniformMatrix}(s) \in [-32, 31]^{m \times r}$
2: $E_R \leftarrow \textsc{ChoiceMatrix}(s) \in [-1, 1]^{r \times k}$
3: **return** E_L, E_R

4.5 Tiled MatMul Algorithm

We execute the standard tiled matrix multiplication algorithm by partitioning the matrices A' and B' into tiles of size $t_m \times r$ and $r \times t_n$, respectively. We maintain an accumulator $C_{i,j}$ for each output tile, initialized to zero. For all tile indices $\ell = 0, \ldots, \lceil k/r \rceil - 1$, the accumulator is updated by adding the product of the corresponding input tiles:

$$C_{i,j}^{(\ell)} \leftarrow C_{i,j}^{(\ell-1)} + A'_{i,\ell} \cdot B'_{\ell,j}.$$

While the matrix product is computed for all tiles, the hash state updates are restricted to full-rank tiles, where $\ell < \lfloor k/r \rfloor$. For these steps, we compute $X_{i,j}^{(\ell)}$, the `int32` XOR of all $t_m t_n$ entries in the current accumulator $C_{i,j}^{(\ell)}$. This value updates a 512-bit state vector $M_{i,j}$, partitioned into 16 `int32` elements and initialized to zero:

$$M_u^{(\ell)} = \begin{cases} (M_u^{(\ell-1)} \lll 13) \oplus X_{i,j}^{(\ell)} & \text{if } u \equiv \ell \pmod{16} \\ M_u^{(\ell-1)} & \text{otherwise} \end{cases}$$

Upon completion, a block is effectively opened if the final hash state satisfies the difficulty condition:

$$\text{BLAKE3}(M^{(\lfloor k/r \rfloor - 1)}) \leq 2^{256-b} \cdot r \cdot t_m \cdot t_n,$$

where b is the blockchain difficulty target and the hash output is interpreted as a little-endian `uint256`.

Algorithm 4 Tiled MatMul with Accumulated Hash Check

Require: A', B'
Ensure: $C' = A' \cdot B'$ and list Blocks of opened blocks

Blocks $\leftarrow [\,]$
Initialize $C' \in \mathbb{Z}^{m \times n}$ to zeros
for $i = 0$ **to** m **with steps of** t_m **do**
  for $j = 0$ **to** n **with steps of** t_n **do**
    $h \leftarrow \min(t_m, m - i),\ w \leftarrow \min(t_n, n - j)$
    $C_{\text{blk}} \leftarrow 0_{h \times w};\ M \leftarrow 0^{16}$ ▷ Initialize with `int32` zeros
    for $\ell = 0$ **to** $\lfloor k/r \rfloor$ **do**
      $s \leftarrow \ell \cdot r;\ d \leftarrow \min(r, k - s)$
      $C_{\text{blk}} \mathrel{+}= A'_{i:i+h,\, s:s+d} \cdot B'_{s:s+d,\, j:j+w}$
      if $h{=}t_m$ **and** $w{=}t_n$ **and** $d{=}r$ **then**
        $X \leftarrow \bigoplus_{x \in C_{\text{blk}}} x$
        $M[\ell \bmod 16] \leftarrow (M[\ell \bmod 16] \lll 13) \oplus X$
      end if
    end for
    $C'_{i:i+h,\, j:j+w} \leftarrow C_{\text{blk}}$
    if $\text{BLAKE3}(M, \text{key} = s_A) \leq 2^{256-b} \cdot r \cdot t_m \cdot t_n$ **then**
      Append C_{blk} to Blocks
    end if
  end for
end for
return $(C'$, Blocks$)$

4.6 Block Opening Proof

A block opening proof provides the verifier with all the information needed to reconstruct the candidate tile and to check both the inner and outer hash predicates. The proof must be sufficient to validate that the rows and columns used in the reconstruction indeed belong to the committed matrices, and that the claimed tile location is correct. Concretely, the proof uses the property that BLAKE3 is a Merkle tree hash and contains:

- **Matrix commitments.** The BLAKE3 hashes H_A and H_B of matrices A and B.
- **Merkle authentication data.** For both A and B:
  - The leaf data covering the rows or columns used.
  - The indices of these leaves within the Merkle tree.
  - The Merkle paths needed to recompute H_A and H_B from the leaves.
- **Tile metadata.**
  - The row indices in A corresponding to the opened block.
  - The column indices in B corresponding to the opened block.
  - The depth index identifying the opened block.
- **Mining Configuration and Matrix shapes.** The matmul shape (m, n, k) of A,B as well as rank r and tile shapes t_m, t_n.

Verifier.

1. Validate the Merkle proofs for the provided rows of A and columns of B against H_A and H_B.
2. Derive noise seeds s_a, s_b from H_A, H_B the blockchain state μ and the mining config k, r, t_m, t_n.
3. Generate the noise matrices E_L, E_R, F_L, F_R from the noise seeds.
4. Reconstruct the candidate tile C_{tile} from the provided noised matrix strips, at the specified position and depth.
5. Test inner hash condition on C_{tile} with difficulty b_1 and the outer hash with difficulty b_2.

4.7 ZK-SNARK Block Opening Proof

The Merkle/fragment-based block-opening proof described above is conceptually simple but raised two main concerns: *size* and *privacy*. Specifically, the commitments are constructed as BLAKE3 Merkle roots over the rows (for A) and columns (for B). To open a block, one must include the corresponding leaf data from the relevant row and column strips, along with their authentication paths and indices.

A simple calculation shows that for two 16384×16384 matrices with tile size 16, the proof size approaches 0.5MB, and with a tile size of 64, it grows to nearly 2MB. Storing proofs of this magnitude in blockchain headers would significantly impact scalability. In terms of privacy, these row and column strips may contain personal or proprietary information that cannot be publicly revealed, making this approach unsuitable for AI or confidential workloads.

A zkSNARK-Based Solution. Both challenges described above can be addressed using a zero-knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK). A zkSNARK allows a prover to demonstrate that a computation was performed correctly—without revealing any information beyond the validity of the claim itself. For example, in our setting, it is desirable to attest that "the prover has (private) inputs that when given to the verifier described in previous section together with (public) chain state, yields a value M whose BLAKE3 hash is h" without revealing the inputs themselves. Verification is efficient, and the resulting proof is short (succinct).

This simultaneously compresses a multi-megabyte opening into a compact proof (kilobyte scale, depending on the backend) and preserves privacy for AI participants whose matrices contain proprietary or sensitive data.

Hash-based zkSNARKs. There exist various constructions of zkSNARKs based on different underlying cryptographic assumptions. We adopt constructions that rely *solely* on cryptographic hash functions. These schemes, commonly referred to as *hash-based zkSNARKs*, fit perfectly in our setting, as they offer several compelling advantages:

1. **Fast.** These zkSNARKs rely solely on lightweight cryptographic primitives (e.g., hash functions), avoiding expensive public-key or pairing-based operations. This results in exceptionally fast proof generation and verification.

2. **Post-quantum security.** Hash-based SNARKs are among the most efficient and *most secure* known approaches for achieving *post-quantum security*. Their security relies solely on the hardness of cryptographic hash functions, whereas other SNARK constructions typically depend on additional, stronger algebraic assumptions (e.g., knowledge-of-exponent or elliptic-curve pairing assumptions). As a result, hash-based SNARKs remain secure even against adversaries equipped with quantum capabilities.
3. **Transparent setup.** These schemes do not require a trusted-setup or shared randomness. Any verifier can independently validate a proof, and no trapdoors can be embedded in the system. This property greatly simplifies deployment and enhances user trust, as trusted setups are often viewed as a major security liability.

Implementation and optimizations. We adopt Plonky2,[4] a modern hash-based zkSNARK that achieves fast proof generation and verification while supporting efficient recursive composition of proofs, and zero-knowledge. As we mentioned, it requires no trusted setup and relies solely on well-understood cryptographic hash functions, ensuring transparency and post-quantum security. We adopt the following notable optimizations:

- **Direct arithmetic representation.** Instead of compiling the computation from a high-level programming language into arithmetic constraints, we directly express the desired verifier circuit in *Arithmetic Intermediate Representation (AIR)* form. While high-level languages improve developer ergonomics, a hand-crafted AIR allows fine-grained control over constraint structure, enabling to design a more efficient system.
- **Preprocessed columns.** While about 2/3 of the running time of the plaintext verifier is spent on deriving the noise E, F from s_A, s_B, the zk-prover and verifier can agree on the plaintext noise, rather than zk-proving the correct derivation of it. To this end, we extend Plonky2's AIR representation with *preprocessed columns* – allowing circuit reliance on public data agreed by both prover and verifier.
- **Recursion.** While hash-based zk-provers typically trade off proving time against proof size, recursion offers the best of both worlds: first prove the claim as fast as possible with only modest proof-size reduction, then recursively compress the proof aggressively while keeping the running time acceptable. We employ a 3-layered recursion, resulting in a final proof size below 60KB.

Data revealed by the zkSNARK The mining configuration r, k, t_m, t_n, matrix shapes m, n, position of opened tile i, j, and $H_A, H_B, \text{BLAKE3}(M, \text{key} = s_A)$.

4.8 Supported PoW Parameters

The supported PoW protocol parameters are chosen to depict AI-relevant GPU-compatible matrix multiplications. The parameters are restricted as follows.

- $m, n \leq 2^{24}$.
- The common dimension k must satisfy $16r \leq k \leq 4r^2$ for security of the protocol. It must also satisfy $k \leq 2^{16}$ to avoid matmul overflows, and $64 \mid k$ for commitment hash alignment.

[4] `https://github.com/0xPolygonZero/plonky2`.

- Each of the tile shapes t_m, t_n is specified as a 3-D arithmetic progression $(S_1, S_2, S_3, L_1, L_2, L_3)$

$$T = \Big\{ \ell_1 S_1 + \ell_2 S_2 + \ell_3 S_3 : \ 0 \le \ell_i < L_i \Big\},$$

  with $1 \le S_1 \mid S_2 \mid S_3$ and $1 \le L_1 L_2 L_3$ the size of the tiles. Each tile has the form $t+T = \{t+\ell : \ \ell \in T\}$. A tile partially outside of the m, n boundaries is illegible for proof of work.

- The rank r is allowed from the set $\{2^5, 2^6, \ldots, 2^{10}\}$. Lower or higher ranks are not expected to be an efficient miner choice.

- Let h be the size of row tiles and w the size of the column tiles. We require $h \cdot w \ge 32$ to ensure sufficient entropy in M. Furthermore, the verifier restricts $k(h+w) \le 2^{22}$.

5 Blockchain Technical Specifications

We embed our PoUW protocol into a blockchain protocol. Our blockchain forks Bitcoin with several adjustments, detailed below. Full protocol specifications are provided in the code: `https://github.com/pearl-foundation/pearl`.

5.1 Block Structure

Dynamic size proof of work. Unlike bitcoin, in which a 80-bytes block header is a self-contained proof of work, in our proposed POUW protocol, the proof encodes two optionally big matrices A, B as a zkSNARK. This necessitates some changes to the block header. We remove the nonce field, which serves as the proof of work in bitcoin. Instead, we supplement a block header with a variable-length message called 'block certificate', which serves as a validity proof of the block header. This block certificate is encoded as

$$\mathsf{certificate} \ := \ \mathsf{version} \parallel \mathsf{bytes}.$$

Separating the certificates keeps header format stable, yet allowing future upgrades to the certificate mechanism. We enforce a certificate size limit of 65KB.

Randomized, non-unique proofs. Unlike Bitcoin's POW witness, our *zero-knowledge* certificates are *inherently randomized*. From one valid witness it is straightforward to derive many distinct, equally valid proofs by resampling prover randomness, with no additional useful work. Consequently, the certificate bytes themselves *must not* determine a block's identity; doing so would make block IDs malleable at will by the miner after the fact.

Block identity. Like Bitcoin, block identity is the double sha-256 of the header data

$$\mathsf{hdr}^\star := \mathsf{version} \parallel \mathsf{prev\_hash} \parallel \mathsf{tx\_root} \parallel \mathsf{time} \parallel \mathsf{nBits} \parallel \mathsf{pouw\_meta},$$

serialized as 116 bytes. Here pouw_meta is a 32-bytes commitment (SHA-256) to the underlying PoUW witness – the public data revealed by the zkSNARK as detailed in Section 4.7. While several certificates may circulate for a single block, the pouw_meta field binds them to the same underlying useful work instance.

5.2 Addresses

Traditional cryptocurrency systems such as Bitcoin have accumulated multiple address formats over time, beginning with legacy pay-to-public-key-hash (P2PKH), followed by pay-to-script-hash (P2SH), and later SegWit variants. While each introduction was necessary for incremental upgrades, the coexistence of multiple formats today leads to fragmentation, implementation complexity, and a larger attack surface. Moreover, most legacy formats rely directly on ECDSA or Schnorr signatures, both of which face emerging risks in the presence of quantum adversaries.

Taproot, introduced via Bitcoin Improvement Proposal (BIP-341), consolidates the benefits of Schnorr signatures with a flexible script path structure. It enables uniform address representation, improved efficiency, better privacy, and a natural path toward quantum- and post-quantum–hardened upgrades in the future.

In the design of our system, we made the explicit choice to remove support for legacy address types (including ECDSA- and Schnorr-based formats) and to standardize exclusively on Taproot addresses. This decision was motivated by a combination of security, simplicity, and forward-compatibility considerations, as explained above. While this choice imposes the short-term cost of abandoning legacy address formats, it significantly strengthens the long-term resilience, security, and maintainability of the network.

Post-quantum readiness. To address the threat of quantum computing, we introduce support for the `OP_CHECKXMSSSIG` opcode (#222), which is functionally equivalent to `OP_CHECKSIG` but verifies signatures produced by the eXtended Merkle Signature Scheme (XMSS)—a hash-based signature algorithm believed to remain secure against quantum adversaries. Concretely, we use `SHAKE256` as the underlying hash function and parameterize the scheme to support approximately 256-bit security for 256-bit messages, with 32 distinct signatures available per key pair; the resulting signature size is 2340 bytes. While currently disabled in node consensus, this opcode is reserved for future activation should Schnorr signatures be deemed insecure. To prepare for this post-quantum regime, our wallet implementation includes a Taproot script path containing an XMSS public key. To separate Schnorr private keys from post-quantum keys, the wallet derives XMSS keys using purpose 222 along the path `m/222'/<coin_type>'/account'/branch/index`, reusing the same `account`, `branch`, and `index` as the corresponding Schnorr key.

Additionally, we implement the Pay-to-Merkle-Root (P2MR) output type proposed in BIP-360. P2MR functions similarly to Taproot but eliminates the quantum-vulnerable key-path spend, requiring all spends to occur via a script path and Merkle proof to minimize public key exposure.

5.3 VM and Script Extensions

Pearl employs a stack-based, non-Turing-complete virtual machine based on Bitcoin. Notably, we enables the opcode `OP_CAT`, which concatenates two stack items: given two byte strings a and b (with b on top of the stack), `OP_CAT` pops both items and pushes their concatenation:

$$\texttt{OP\_CAT}: \; a, b \mapsto a \parallel b.$$

To prevent excessive memory usage, `OP_CAT` is subject to strict resource bounds. The output size is capped at 520 bytes, and the operation incurs a dynamic cost of $\lceil \text{length}_{\text{out}}/64 \rceil$ against the tapscript budget, analogous to the fixed cost charged by `OP_CHECKSIG`.

5.4 Fees, Block Time, and Emissions

Fees. We adopt Bitcoin's first-price auction for transactions.

Block time. Bitcoin's 10-minute block interval was chosen conservatively in 2009 when network bandwidth, node performance, and global propagation latencies were dramatically worse than today. Modern networking conditions allow substantially faster block confirmation without compromising security or decentralization. We therefore adopt a 194 seconds (3:14) block interval, which improves user experience by reducing time-to-finality, accelerates on-chain activity and protocol responsiveness, and enhances economic throughput, while remaining comfortably within safe propagation margins for a globally distributed validator set.

Coin supply. We set a total of $S' = 2{,}100{,}000{,}000$ coins. The smallest supported unit is 10^{-8} of a coin, or a *grain*.

Emission curve. We adopt a smooth, polynomially decaying emission schedule designed to retain the desirable monetary properties of early Bitcoin while eliminating two known drawbacks: (i) the discontinuous "halving shocks" that create incentive cliffs and system-wide volatility, and (ii) an overly thin tail of long-term issuance, which can under-incentivize long-run security.

Let $t \in \{0, 1, 2, \dots\}$ denote the block height (i.e., the number of blocks since genesis). We target a *fat but finite* tail by choosing an emission rate that decays approximately like $1/t^2$, so that the remaining supply decays like $1/t$.[5]

We now normalize the curve so that (1) the basic unit of time is a single block, and (2) we match Bitcoin's approximate cumulative emissions after four years, i.e., 50%. With an expected block time of 194 seconds, four years correspond to $H = \lfloor 4 \cdot 365 \cdot 24 \cdot 60 \cdot 60/194 \rfloor = 650{,}226$ blocks. We use H as the characteristic time scale (in blocks) of the emission schedule.

Define the remaining supply *fraction* at block height t by

$$R(t) \;=\; \frac{H}{t+H},$$

so that $R(0) = 1$ and $R(t) \to 0$ as $t \to \infty$, and $R(H) = 1/2$. The corresponding cumulative allocation fraction is

$$A(t) \;=\; 1 - R(t) \;=\; \frac{t}{t+H}.$$

In terms of absolute units of supply, the cumulative allocation by block t is $S \cdot A(t)$.

The per-block emission is obtained by the discrete derivative of the cumulative curve, that is

$$E^*(t) \;=\; \Delta\big(S \cdot A(t)\big) \;=\; \frac{St}{t+H} - \frac{S(t-1)}{t-1+H} \;=\; \frac{SH}{(t+H)(t+H-1)},$$

with t starting at the first post-genesis block, i.e., $t = 1$. The actual emission is the floor of $E^*(t)$ in grains.

[5] For intuition, normalize the total supply to 1 and consider a continuous-time emission rate $e(t) = 1/t^2$ for $t \geq 1$. The cumulative allocation by time t is then

$$\int_{u=1}^{t} \frac{1}{u^2}\, du = \left[-\frac{1}{u}\right]_{u=1}^{t} = 1 - \frac{1}{t},$$

so the remaining supply fraction is $1 - (1 - 1/t) = 1/t$.

Difficulty adjustment. The difficulty adjustment algorithm used in Bitcoin is triggered once per $N = 2016$ blocks and is aimed at keeping average block time at 10 minutes. While it seems sufficiently responsive for Bitcoin, this algorithm was criticized for being either non-responsive or unstable in other blockchains.[6]

Aiming for responsivity and stability, we use the Weighted-Target Exponential Moving Average (WTEMA) algorithm.[7] Recall the difficulty of a block is determined by a `uint256 target`, which serves as the upper bound of a mining candidate's hash to be considered valid. In WTEMA-N rule, the target of the next block is determined by the target of the current block and its solvetime t, defined as the difference between the current and parent timestamps:

$$\mathsf{target}_{\mathrm{new}} = \mathsf{target}_{\mathrm{old}} + \left\lfloor \frac{\mathsf{target}_{\mathrm{old}} \cdot (t - T)}{N \cdot T} \right\rfloor.$$

Here, T is the intended solve time of 194 seconds (3:14 minutes), and $\tau = N \cdot T$ corresponds to a characteristic decay time of τ of 7 days. Initial difficulty is set low (nbits=0x1b00ffff) in order to (i) give early miners a fair opportunity to participate before significant AI enterprise adoption increases the network hashrate, and (ii) ensure the core team has minimal resources needed to support the massive research, development and infrastructure required to continually support state-of-the-art AI architectures and hardware.

Further constraints are imposed on timestamps to limit manipulations:

- **Monotonicity**: A block timestamp must be *later* than the timestamp of its parent block. Since timestamps are encoded in seconds, the time difference must be at least 1 second.
- **Reduced Future Time Limit**: In order to limit timestamp manipulation, a verifier must (temporarily) not accept a block whose UTC timestamp is 5 minutes or more into the future. In Bitcoin this threshold is 120 minutes. This change assumes a lenient clock-synchronization requirement.

Tie breaking. Unlike Bitcoin, where adjacent blocks typically have identical difficulty, WTEMA may assign a distinct difficulty to each block. If nodes were to select the heaviest chain exactly as in Bitcoin, then between two competing tips with the same parent they would systematically prefer the slightly harder block with the earlier timestamp. This creates an incentive for miners to backdate timestamps, pushing difficulty above the intended target interval T. To mitigate this, when two tips have nearly equal cumulative work, nodes should break ties by the order in which the tips were received, and revert to the heaviest-tip rule only when the cumulative work differs materially. Concretely, the work-difference threshold is $\min(\text{tip-work}_1, \text{tip-work}_2)/4$.

6 PoUW Econ-101

This section presents a simple econ-101-level model for understanding the expected prices and mining levels for a proof of useful work network like Pearl. The main point is the presentation of natural assumptions that imply that the price of a proof-of-useful-work token should naturally follow the growth of the target useful-work. In the case of Pearl, this includes global AI training and inference.

It should be clear that as a pure store-of-value token, the value of a proof-of-useful-work token is completely dependent on the expectation of its future adoption as a means of payment by a large enough set of users

[6] https://read.cash/@jtoomim/bch-protocol-upgrade-proposal-use-asert-as-the-new-daa-1d875696.
[7] https://github.com/zawy12/difficulty-algorithms/issues/76.

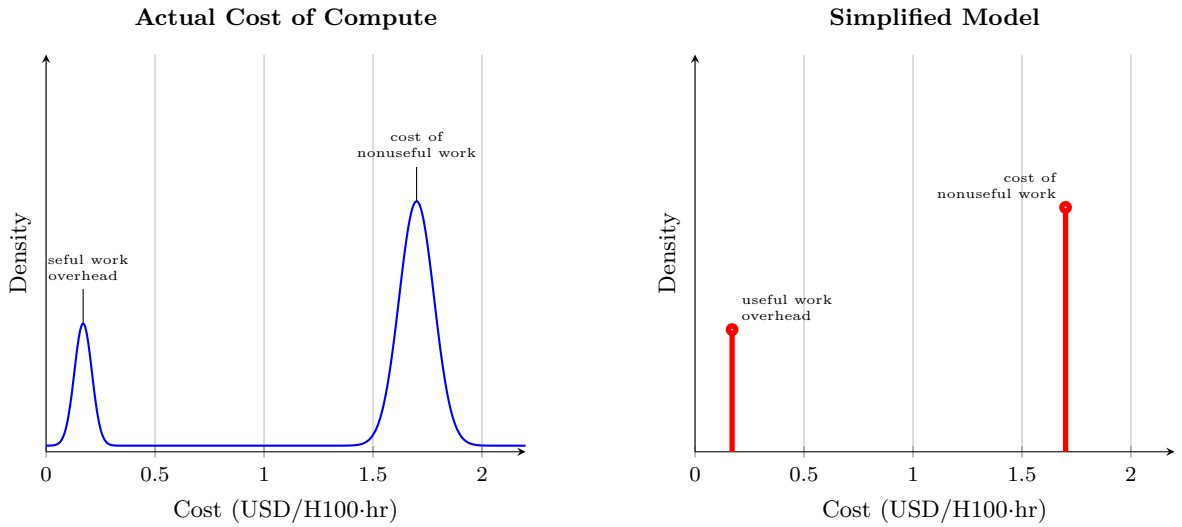
(at least with some non-zero probability). Presumably this will happen due to the advantages that it has over other store-of-value tokens such as Bitcoin. The case that this is so for the PEARL blockchain is an assumption (without which the value of the token and the amount of mining should clearly be zero.)

A more technical and detailed analysis (beyond econ-101) of the economy and value of PEARL will be made in a separate document to follow.

6.1 The Distribution of Costs of Work

Our starting point is that the mining power in the world that is potentially available for participation in the system at any given point in time is composed of processors that have "useful work" that can be combined with mining and those that do not have such useful work. Our basic assumption is that those processors have a cost of some amount C per unit of compute, whose units we will take in this note to be USD per the equivalent of 1 hour of an H100 GPU. The assumption is that there will be a significant difference between processors without useful work whose costs will be close to the current market price of compute (which in our example will be approximately the current value of about 1.7 USD per H100-hour), and processors with useful work who will only need to pay the overhead which is a small fraction of the total compute price (which in our example will be about 10% of that number).

The following graph on the left depicts a "histogram" of the costs of all available processors where the "bump" on the left are the costs of overhead of processors with useful work, and the bump on the right is those without useful work. For ease of analysis we will take a simplified model of costs as depicted by the figure on the left where all non-useful work has a fixed cost of C (in our example $C = 1.7$), and all useful work (i.e. the overhead on such work) has costs $\tau \cdot C$ (in our example, $\tau = 0.1$). This simplified model will make all analysis below easy, but its basic implications carry over to the more general model of costs, and in fact also to even more general, "general-equilibrium" type of models.



6.2 Minting, Security, and Block Rewards

We will assume that the networks allocates block rewards at some fixed rate that varies with time, i.e., that there is some fixed schedule of emissions $e(t)$ which denotes the instantaneous rate of emission of tokens at

time t. We will denote by $S = \int_0^\infty e(t)dt$ the fully diluted total supply of tokens. The units of S are "tokens" and the units of $e(t)$ are tokens per hour. We assume that at every time t the emissions are distributed between the miners at that time, proportionally to their mining power at that time. The total amount of mining power at a given point in time may be viewed as the "security" of the system at that time since an adversary will need a majority of that computing power in order to corrupt the network.

As our running example we will take the parameters of the PEARL system (see Section 5.4): a fully diluted supply of $S = 2.1 \cdot 10^9$ tokens with the emission rate schedule of $e(t) = S \cdot H/(H+t)^2$, where $H = 4\,years = 35040\,hours$ is the "first halving time", i.e., after H time, half of the tokens remain unallocated, i.e. $S/2 = \int_H^\infty e(t)dt$. With this schedule, the number of tokens that remain unallocated at any time T is given by $\int_T^\infty e(t)dt = S \cdot H/(H+T)$.

6.3 Instantaneous Equilibrium

The basic equilibrium idea is that the net cost of compute devoted to mining should be equal to the value of the minted tokens. If the former is larger than the latter then the miner loses money and should use its GPUs for something better. If the latter is greater than the former then miners make significant money and we would expect more miners with more compute power to enter into the mining market. We will say that we have an instantaneous equilibrium at time t if the following equation is satisfied.

$$\tau \cdot G(t) \cdot D(t) = p(t) \cdot e(t)$$

Here $G(t)$ is the computing power devoted to mining at time t (in GPUs), $D(t)$ is the cost of compute (units are \$ per GPU per hour), $p(t)$ is the token price (in \$ per token) in time t, and $e(t)$ is the emission rate at time t (in tokens per hour).

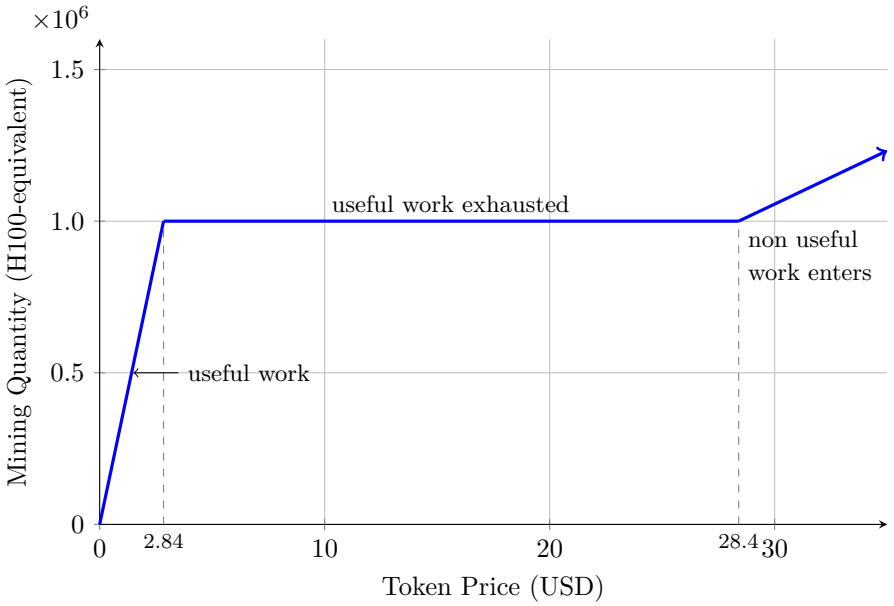
Before we continue our investigation under the assumption of "instantaneous equilibrium" let us pause for a second to look more closely at this assumption. Will the system necessarily reach this type of equilibrium? If the token price is too low relative to hash rate, $\tau \cdot G(t) \cdot D(t) > p(t) \cdot e(t)$ then it is hard to believe that miners will keep mining. After all, they can buy the token at market prices at a lower price than their compute costs for mining, thus shrinking the mining pool. On the other hand, it is easier to imagine scenarios where the token price is significantly higher than mandated by compute costs, $\tau \cdot G(t) \cdot D(t) < p(t) \cdot e(t)$. One may look at two different types of causes for such discrepancy:

1. **Success:** The system managed to recruit 100% of useful work so there are no new miners that can enter even though benefits>cost.
2. **Marketing Failure:** even though mining is beneficial, companies holding the useful work refuse to do so due to lack of trust, lack of awareness, regulation constraints, administrative overhead, or just plain delay or laziness.

It seems likely that on day 1 we will indeed have a large gap due to the second reason, since very few holders of useful work will be set up to mine PEARLs while producing their useful work. So, in such cases, what keeps the token price high? The answer is obviously future expectations. If a token holder today believes that the system will succeed and that the future will see miners flocking to the system then the token price will increase immediately to reflect the expected price in the future (with appropriate discounting). This increase in token price will provide net positive utility to each miner (since now the price of tokens obtained is strictly larger than the costs of mining) and will thus attract, with time, more miners. One can model such

scenarios more carefully, looking at the "future expectations multiplier" $F(f) = p(t) \cdot e(t)/(\tau \cdot G(t) \cdot D(t)) > 1$ which drives the increase in mining power.

The main point of this analysis is that it determines the momentarily relation between the token price and the mining power. The following graph depicts the hashing power (security) as a function of token price according to this instantaneous equilibrium equation, assuming the distribution of costs as above with a total of 1M GPUs of useful work and 2M GPUs of non useful work, on day 0, where $e(0) = 59931$ tokens/hour (as in the PEARL minting curve).



So we can see a linear relation between the mining quantity and the price that gets saturated when all useful work is used also for mining. During this stage, we are in an instantaneous equilibrium where just enough miners enter the market to equate mining costs and rewards. If the price rises further due to expectations of future growth, then new miners cannot enter the market since all useful work is exhausted (this is the first reason for not being in instantaneous equilibrium mentioned above) but miners start making higher and higher net utility as the rewards become significantly larger than the costs. In the third part of the graph, mining rewards are so high that even processors without useful work benefit by participating in mining.

Let us briefly pause to point out that in an instantaneous equilibrium, holding the price of compute, and the computing power devoted to mining, fixed, the price of the token is proportional to the overhead parameter τ—in particular, the price of the token becomes 0 if $\tau = 0$. This is an artifact of the fact that in this simple model, we are keeping the price of the useful work (i.e,. AI training or inference) fixed, as a first-order approximation. In the full-fledge general equilibrium model (to appear soon), we model the AI price as an endogenous variable (depending among other things on an AI demand curve); in equilibrium, the AI cost will decrease when computation overheads decrease—intuitively, as useful proof of work miners are also receiving tokens, they need to provide a *rebate* on the AI cost to stay competitive. This means that even if the computational overhead for useful mining decreases (even reaching 0), the *cost-overhead* τ always stays positive (and in fact grows with smaller computational overhead, as the rebate miners need to pay grows.) Let us also point out that the rebate provided by miners has an additional benefit: if the AI price decreases, the demand for it grows, which in turn leads to future growth in mining.

6.4 Token Price Appreciation

Assuming that we stay in equilibrium in the long term, we expect the token price to increase as more compute power enters the mining pool and as the emission rate decreases with time. To proceed from here, we will require some assumption on how much the computing power devoted to the mining pool grows with time:

Assumption: PEARL maintains the same fraction of world computing power as this computing power grows over time.

Before we proceed, let us take a moment to reflect on this assumption. The basic reason why we may expect this type of behavior is that everything in the model scales linearly with the absolute global computing power. In some sense, the fraction of world compute power that is willing to participate in PEARL mining (or in any other proof of useful work system) may be viewed as some "success" measure. While clearly that success level may vary over time according to the specifics of the system, its management, marketing, etc., on the average, once we have reached some kind of equilibrium, there is no reason to expect such "success" to increase or decrease.

If we further assume that the efficiency factor τ remains constant, then we we can use the equilibrium equation $\tau \cdot G(t) \cdot D(t) = p(t) \cdot e(t)$ at two different times $t_1 < t_2$ to calculate the token appreciation between these two points in time:

$$\rho_p = p(t_2)/p(t_1) = \rho_G \cdot \rho_D / \rho_e,$$

where $\rho_G = G(t_2)/G(t_1)$, $\rho_D = D(t_2)/D(t_1)$, and $\rho_e = e(t_2)/e(t_1)$.

ρ_e can be directly calculated from the emission curve: $\rho_e = e(t_2)/e(t_1) = ((H + t_2)/(H + t_1))^2$. E.g., in the first year, between $t_1 = 0$ and $t_2 = 8760$ hours, we have $1/\rho_p = (5/4)^2 \approx 1.56$, while in the tenth year, between $t_1 = 87600$ hours and $t_2 = 96360$ hours we will have $1/\rho_p = (15/14)^2 \approx 1.07$.

The main part of the expected increase in token price is our expectation of the growth of $\rho_G \cdot \rho_D$ which is the growth of the total cost of worldwide compute. We thus get that the expected increase in token price is just slightly super-linear in the total world computing costs, where the slight super-linearity is due to ρ_e that starts at a value of 56% increase per year and decreases as the system gets more mature. This may be viewed as implying that *the token value is "indexed" to the worldwide cost of useful computing* (and a bit more). Again, let us emphasize, that this is under the two assumption that (a) the system remains (approximately) in instantaneous equilibrium and (b) maintains its share of world computing power.

Assuming the continuation of current trends that (more or less) double computing power annually and halve costs bi-annually, then while we remain in instantaneous equilibrium, we should expect the token value to slightly more than double bi-annually. Of course, as discussed above, part of this expected future growth may already be incorporated in today's price.

7 Launch of PEARL

Today, April 27th, the node code becomes public and thereby the chain becomes live. Additionally, we make available a vLLM plugin that implements a new quantization mechanism which re-implements a layer in a DNN via our "two-for-one" scheme. Additionally, we publish a PEARL-certified version of Meta's Llama 3.3 70B, a state-of-the-art, open-weights large language models offering 70 billion parameters, optimized for high-performance, cost-effective reasoning, coding, and multilingual tasks.

Table 1: Original (Meta's) llama-3.3-70B-Instruct vs. our "two-for-one" PEARL-certified variant. Both executions were done with 4×H200 GPUs. We explore several parallelism techniques. TMADs, i.e., Tera MADs, is a metric counting number of Multiply-Add (MAD) operations. *Useful* MADs is the total number of MAD operations *done anyway* that are used for mining.

Model	**Parallelism**	**Score** (MMLU)	**Throughput** (tok/sec)	**Time** (sec)	**Useful MADs** (TMADs/sec)
Meta's LLaMA 70B	PP=4	0.8198	15,269.81	441.100	–
Meta's LLaMA 70B	TP=4	0.8193	13,218	510	–
Meta's LLaMA 70B	DP=2, TP=2	0.8197	13,162	512	–
Meta's LLaMA 70B	DP=4	*OOM – bf16 model (∼140 GB) exceeds single GPU VRAM*			
PEARL-certified	PP=4	0.8190	17,206.26	391.457	806
PEARL-certified	TP=4	0.8180	13,264.38	507.789	620
PEARL-certified	DP=4	0.8198	18,291.66	368.229	981