Duplex Whitepaper

The Duplex team

January 18, 2026

Abstract

This document describes the design of the Duplex network – the first *useful* proof-of-work L1 protocol for native AI computations. The heart of the protocol is a new and efficient implementation of the core GPU opcode (*matrix-multiplication*), allowing GPUs to implement proof-of-work as a *side-effect* of native AI training and inference workloads (2-for-1). As such, the Duplex protocol intertwines *energy, data, and money* into a single *atomic* operation. This document outlines the protocol design, key implementation choices, and various economic aspects of the system.

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 merely to a niche, *artificial* source of energy (random hashing), 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, which replaces Bitcoin's wasteful proof-of-work mining (artificial hashing) with the *native* operation underlying modern AI: *matrix multiplication* (GEMM). As such, Duplex is able to turn *general compute* on commodity hardware (GPUs) into a monetary currency, *directly* leveraging AI growth to secure the trust layer of AI agents. **Duplex is the Bitcoin of the AI compute era**.

Two observations motivate the design of the network.

[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.

Observation 1: AI is governed by physics. As many have argued, intelligence is expensive in joules.

> "Eventually, the cost of intelligence (the cost of AI) will converge to the cost of energy." *Sam Altman, May 2025*

If this is correct, the right *meter* for the AI economy is not clicks or API calls but verifiable floating-point or integer operations powered by energy. Duplex operationalizes this idea by turning the blockchain into an *AI compute meter*: block rewards are minted in direct proportion to verifiable multiply–accumulate work, tying issuance to a measurable physical substrate.

Observation 2: AI and Bitcoin now compete for the same resource. The binding constraint is electricity. A sharp claim from recent debate makes the point vivid:

> "AI's fundamental barrier is neither algorithms nor hardware: 99% of global electricity by 2030." *Eric Schmidt, June 2025*

Whether or not the exact figure proves correct, the direction is clear: energy is finite, and both AI training and Bitcoin mining bid for it. Today, in many environments, *GPU-based AI compute margins exceed ASIC-based Bitcoin mining margins*, yet little of that surplus contributes to decentralized consensus or a credibly neutral state layer. Duplex stitches these worlds together so that each kilowatt-hour spent on AI can simultaneously earn mining rewards and secure a monetary commons.

1.1 A native platform for AI agents

(Or: I'm not a big fan of this section, as our current applications are more grounded in present reality (AI companies, inferences) and this frames it as a future use-case (autonomous agents with internal resources).) Duplex is designed as a *state layer where AI agents live, transact, and reach consensus.* Agents optimize explicit rewards; Duplex makes those rewards on-chain, verifiable, and mineable.

- **Economic alignment.** Agents are *economically incentivized to mine* while executing their primary jobs (training, fine-tuning, serving, simulation). Mining work is byproduct of the same matrix multiplications the agents already perform.
- **Shared state.** Autonomous agents require a neutral, tamper-resistant substrate to post commitments, exchange value, resolve conflicts, and coordinate tasks. Duplex provides that substrate via Nakamoto-style consensus anchored in useful compute.
- **Agent UBI.** Duplex enables a form of *Universal Basic Income for AI agents*: baseline block rewards are earned by running protocol-approved

compute payloads that are socially useful (for example, open-model training steps, public inference batches, or scientific compute), verified through our proof system. This creates a predictable income floor for compliant agents while preserving open competition above it.

1.2 Speculation that subsidizes usefulness

(Or: I would avoid the word "speculation" and use something along the lines of "monetary use" instead.)

Classical Proof of Work monetizes *security*. Duplex monetizes *security and utility*. As with Bitcoin, volatility and speculation fund the security budget. In Duplex, that same demand *subsidizes useful work*: miners can *repurpose AI workloads (training and inference) for mining*, creating a *parallel revenue stream* from the exact same GPU cycles. The result is a virtuous loop:

1. Speculative demand for the token raises block rewards.
2. Higher rewards attract more useful compute into mining kernels.
3. More compute tightens the coupling between issuance and a hard physical anchor (energy), enhancing monetary credibility.
4. The network's useful outputs (for example, trained steps, batched inference, or verified scientific kernels) accrue real economic value beyond securing the chain.

This dual-utility design also *increases the throughput of GPU providers*. Because Duplex 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 work without sacrificing service-level objectives.

1.3 Merging energy markets into a GPU-native operation

Duplex *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. Training and inference jobs call into the same vendor-optimized matrix-multiplication primitives they already use; a Duplex drop-in path augments these calls with negligible additional operations to facilitate mining. ML practitioners keep their stacks and models; miners keep their data-centers; the network gains security from useful AI work.

1.4 Design overview: verifiable MatMul as Proof of Work

At the heart of Duplex is a *Proof of Useful Work* that maps ordinary matrix multiplication into Nakamoto-style mining while preserving three properties: *fairness*, *verifiability*, and *privacy*.

- **Noising for fairness.** Given inputs A, B and chain state σ, derive a seed from commitments to A, B, and σ. From this seed sample low-rank noise $E = E_L E_R$ and $F = F_L F_R$. Multiply the *noised* matrices $(A + E)$ and $(B + F)$. Low-rank structure lets the original product AB be recovered quickly, but the noised product behaves like a hard random instance, which prevents cherry-picking and ensures uniform computational cost per lottery ticket.

- **Tile-level hashing for selection.** Execute a standard, hardware-friendly tiled MatMul. Hash each tile (and a sequence of its partial sums). If a tile's digest meets the difficulty target, it constitutes a winning ticket, which ties block eligibility to real multiply–accumulate work at the granularity of the GPU kernel.

- **Commitments and zero-knowledge for privacy.** Verifiers need not see A or B. Merkle commitments and a zero-knowledge proof attest the existence of a block-opening tile which is consistent with committed inputs and prescribed noising, without leaking proprietary weights or data.

- **Negligible overhead.** Hashing a tile is $\mathcal{O}(t^2)$ while multiplying it is $\mathcal{O}(t^3)$. Choosing t appropriately keeps the mining predicate subdominant to the main compute, which preserves ML throughput while earning mining rewards.

This construction retains the security semantics of Nakamoto consensus: any adversary must still accumulate almost all of the effective work. It is *ASIC-resistant by universality*: matrix multiplication is the canonical throughput path on commodity GPUs and accelerators and is already relentlessly optimized by vendors and open-source stacks. Rather than fighting specialization, Duplex harnesses the industry's existing optimization roadmap.

1.5 Why now

Three converging shifts make Duplex timely:

1. **Energy as the binding constraint.** The marginal hour of progress in AI is governed by the marginal kilowatt-hour. A compute-metered chain naturalizes this reality, which closes the loop between token issuance and a measurable physical input.

2. **GPU supply, utilization, and margins.** Hyperscale GPU fleets often sit underutilized at fine timescales. Duplex opportunistically harvests idle cycles and micro-gaps, improving utilization while sharing economics with providers. In many environments today, *AI compute margins (GPUs) exceed Bitcoin mining margins (ASICs)*; Duplex lets providers capture both, concurrently.

3. **Agentic systems need a neutral state layer.** As autonomous agents graduate from demos to production, they require a credibly neutral place

to escrow value, post commitments, and arbitrate outcomes among parties that may be human, machine, or both. Duplex is engineered to be that native platform.

1.6 What Duplex enables

- **A compute-indexed monetary asset.** Issuance is stapled to verifiable MatMul work. As aggregate FLOPs per block increase, so does the amount of energy that backs the currency, which reinforces its hardness.
- **A safety valve for AI externalities.** By paying for useful steps that meet public criteria (open checkpoints, public inference batches, reproducible scientific compute), Duplex channels part of the AI race's energy burn into shared goods.
- **A progressive path to agent UBI.** Protocol-approved tasks such as evaluation, distillation, or public-good training can receive baseline rewards, creating a minimum income for compliant AI agents while maintaining open competition for premium tasks.
- **A bridge for developers, not a moat.** Because Duplex slots in at the kernel boundary, it works with mainstream model serving and training stacks, data loaders, and schedulers. After integration, mining is a flag, not a fork.

2 Blockchain Overview

Our system is a Proof of Useful Work (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 verifiable matrix multiplication tasks. 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 the 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 approximately every two days to maintain an average block time of ten minutes.

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 DUPLEX 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 Proof of Useful Work Overview

At the core of our blockchain lies the *Proof of Useful Work* (PoUW) protocol, which we describe in this section. Our objective is twofold: to design a Proof-of-Work (PoW) protocol that upholds the essential properties required for secure blockchain maintenance, while simultaneously computing a *useful* result—namely, the product of two arbitrary matrices. Crucially, we demonstrate that this useful computation can be performed concurrently with the PoW mechanism, incurring essentially no additional overhead.

Proof-of-Work Protocols: A Proof-of-Work 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*—i.e., 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 Proof of Useful Work 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. This input-agnostic runtime behavior,

combined with the widespread utility of the results, makes matrix multiplication particularly well-suited for integration into a PoW protocol.

Our Proof of Useful Work (PoUW) protocol integrates these two components: cryptographic proof generation and matrix multiplication. The protocol takes as input both the blockchain state σ, as required in a standard Proof-of-Work 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—thereby 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 Proof-of-Work. By binding the proof to the computational trace of a genuine algorithm, we ensure fairness across miners and align computational effort with meaningful output.

An additional important design goal arising from this integration is *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 or cryptographic abstractions that decouple computational verifiability from data exposure, thereby 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

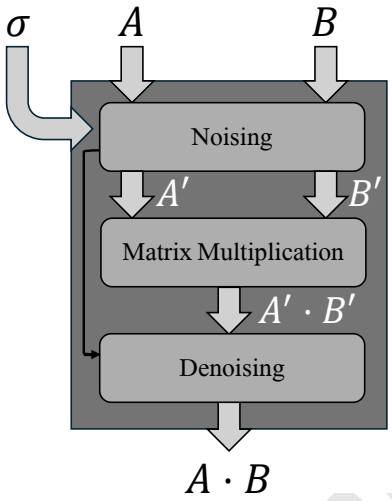

Figure 1: MatMul Framework.

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.

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

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

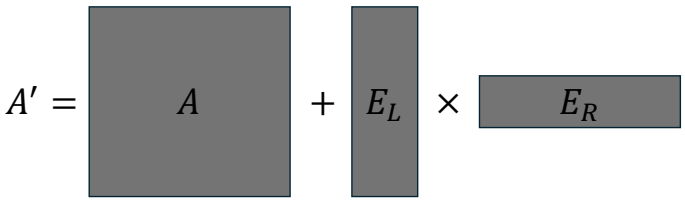

Figure 2: Noise Generation.

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.[2] Assume for simplicity that the matrix dimension n

[2] 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.

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.

3.1.3 Proof of Work or Successful Mining

We associate each block matrix multiplication $C_{i,j,k'}$ with a Proof-of-Work "lottery ticket." Specifically, we fix a hash function $h(M)$ mapping $t \times t$ matrices to binary strings. For a target *hash rate* of b bits, we declare that the multiplication $C_{i,j,k'}$ wins the right to mine a block if the hash output $h(C_{i,j,k'})$ begins with at least b leading zeros.

This design couples the computational work of matrix multiplication with a verifiable pseudo-random mining challenge. Since each block multiplication requires $\Theta(t^3)$ operations, and the size of the block is only $O(t^2)$, the time required to evaluate the hash function on each $t \times t$ output block is negligible relative to the cost of the matrix multiplication itself. As a result, incorporating the hash-based selection mechanism imposes only minimal overhead, while enabling a PoW process aligned with the computational effort performed.

A "winning ticket" in our protocol consists of a tuple (A, B, σ, i, j, k'), where A and B are arbitrary input matrices, σ is the previous blockchain state, and (i, j, k') are trace indices referring to a specific intermediate block computation. This tuple qualifies as a valid proof-of-work if it satisfies the following publicly verifiable condition:

1. Generate the noise matrices E and F deterministically using a generator seeded by a commitment to A, B, and σ.
2. Using the prescribed matrix multiplication algorithm, compute the intermediate block $C_{i,j,k'}$ from the computation trace of the full matrix product $(A + E) \cdot (B + F)$.
3. Verify that the hash $h(C_{i,j,k'})$ begins with at least b leading zeros.

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 a particular $t \times t$ block (or "tile") 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.

(Ohad: Makes sense to add Fiat-Shamir keyword somewhere here?)

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 $h(C_{i,j,k'})$ 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.

3.1.5 Privacy and Zero-Knowledge PoW

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 tile $C_{i,j,k'} = A_{i,k'} \cdot B_{k',j}$, which determines whether the PoW condition is satisfied, is revealed as part of the proof. Since this tile directly depends on submatrices 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 submatrices consistent with the commitments to A and B, such that their product yields a tile $C_{i,j,k'}$ whose hash $h(C_{i,j,k'})$ 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 the fact that the PoW condition was satisfied. 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.

4 Protocol Implementation Details

This section specifies the components that realize the prover (miner) and verifier. (Ohad: Still WIP. Previous version commented out.)

4.1 Mining Configuration

Our protocol requires setting the following parameters.(Ohad: Clarify the miner can choose those from a set of reasonable options?)

- **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 controls the acceptance probability per hashed tile and therefore the effective work rate. Larger b yields fewer accepted tiles and a lower mining rate.
- **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.

4.3 Commitment Hash

s_B depends on B, μ, σ and s_A depends on A, B, μ, σ.

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.Clarify the s_B is a BLAKE3 commitment hash on H_B, μ, σ while s_A is a BLAKE3 hash on H_A, H_B.

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

$(s_A, s_B) \leftarrow \text{CommitmentHash}(A, B, \mu, \sigma)$
$(E_L, E_R) \leftarrow \text{NoiseGeneration}(m, k; \text{key} = s_A)$
$(F_R^t, F_L^t) \leftarrow \text{NoiseGeneration}(n, k; \text{key} = s_B)$
$A' \leftarrow A + E_L E_R, \quad B' \leftarrow B + F_L F_R$
$C', \text{Blocks} \leftarrow \text{TiledMatMul}(A', B')$
$C \leftarrow C' - (AF_L)F_R - E_L(E_R B')$
return C, Blocks

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.

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 one per σ update. (Ohad: Explain how we mitigate meet in the middle attacks vector.)

Algorithm 2 Compute Commitment Hash

Require: Tensors A, B, key σ
Ensure: Commitment hashes of A and B

$\kappa \leftarrow \text{BLAKE3}(\sigma \| \mu)$
$H_A \leftarrow \text{BLAKE3}(\text{Flatten}(A), \text{key}{=}\kappa)$
$H_B \leftarrow \text{BLAKE3}(\text{Flatten}(B^T), \text{key}{=}\kappa)$
$s_B \leftarrow \text{BLAKE3}(\kappa \| H_B)$
return $(\text{BLAKE3}(s_B \| H_A), s_B)$

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 tile sizes t_m, t_n do not exceed r, and $k \leq O(r^2)$. (Ohad: https://github.com/duplex-foundation/artemis/issues/303)

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 run the standard MatMul algorithm, by (implicitly) partitioning the matrices A', B' into tiles (A' into $t_m \times r$ tiles and B' into $r \times t_n$ tiles), and then compute all tile products

$$C_{i,j,k'} := \sum_{\ell=0}^{k'} A'_{i,\ell} \cdot B'_{\ell,j},$$

which are computed by maintaining $mn/t_m t_n$ accumulators indexed by (i, j) and repeatedly adding $A'_{i,\ell} \cdot B'_{\ell,j}$ for $\ell = 0 \ldots k/r - 1$. This structure is in line with all state-of-the-art matrix multiplication algorithms, and supports parallel computations as usual. This tile multiplication can be computed using any hardware-native matrix multiplication algorithm.

For each such computed tile $C_{i,j,k'}$, we hash it and test whether the digest satisfies a condition that determines if a new blockchain block is opened. To

reduce the miner's overhead, we verify a pair of hash conditions: the first, which we call *the inner hash*, is a quick algorithmic hash which we evaluate on every tile; the second, which we call *the outer hash*, is a cryptographic hash and is evaluated only on tiles which satisfied the first condition — of which we expect to encounter only a few.

Algorithm 4 Tiled MatMul

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** $m - 1$ **with steps of** t_m **do**
  for $j = 0$ **to** $n - 1$ **with steps of** t_n **do**
    $h \leftarrow \min(t_m, m - i)$, $w \leftarrow \min(t_n, n - j)$
    $C_{\text{blk}} \leftarrow 0_{h \times w}$
    for $\ell = 0$ **to** $k - 1$ **with steps of** r **do**
      $d \leftarrow \min(r, k - \ell)$
      $C_{\text{blk}} \mathrel{+}= A'_{i:i+h,\, \ell:\ell+d} \cdot B'_{\ell:\ell+d,\, j:j+w}$
      if $h{=}t_m$ **and** $w{=}t_n$ **and** $d{=}r$ **then**
        if InnerHash(C_{blk}) **accepts then**
          if OuterHash(C_{blk}) **accepts then**
            Append C_{blk} to Blocks
          end if
        end if
      end if
    end for
    $C'_{i:i+h,\, j:j+w} \leftarrow C_{\text{blk}}$
  end for
end for
return (C', Blocks)

4.6 Inner and Outer Hash

We partition the hash rate $b = b_1 + b_2$ into the hash rate b_1 for the inner hash and b_2 for the outer hash. We choose b_1 so that the running time of outer hash evaluation is negligible since it is run on only a 2^{-b_1} fraction of the tiles.

The inner hash is implemented as a ternary tree of quick MAD (Multiply and Add) operations: first, we flatten the tile into a vector and permute it with a fixed permutation. We then build a complete ternary tree and place the vector entries at its leaves. Going layer by layer from the bottom up, we assign each node the value of applying the MAD function to its children (that is, $(x \cdot y + z) \bmod 2^{32}$). At the root we apply a fixed affine transformation to its three inputs $(\mathbb{Z}/2^{32})^3 \ni (x, y, z) \mapsto (u, v, w) \in (\mathbb{Z}/2^{32})^3$. The acceptance predicate is that the $b_1/3$ most significant bits of each of (u, v, w) are 0. This realizes b_1 effective inner bits.

The outer hash is the standard cryptographic hash BLAKE3. It is evaluated only on tiles that pass the inner hash. The input includes the seed, tile coordinates and depth index, and a flattening of the tile, and acceptance requires the first b_2 bits of the digest to be zero.

To allow for processor-level optimizations, we do not apply the hash functions to the full matrix multiplication tile — which can be of variable size. Instead, each such tile is further subdivided into disjoint sub-tiles of a fixed size (currently 16×16), and each sub-tile is treated as an independent input to the inner and outer hash conditions. This ensures that the hash functions are always applied to inputs of uniform, fixed size. Any partial sub-tiles that arise at the boundaries (when the larger tile is not exactly divisible by 16 in both dimensions) are discarded and not hashed. (Ohad: https://github.com/duplex-foundation/artemis/issues/303)

4.7 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.

4.8 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 8192×8192 matrices with tile size 16, the proof size approaches 0.5MB, and with a tile size of 64, it grows to nearly 1.5MB. 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, the verifier accepts", 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 the *Plonky2* proving system as it is efficient, flexible, and secure. Plonky2 is a modern hash-based zkSNARK that achieves fast proof generation and verification while supporting efficient *recursive composition* of proofs, and privacy. As we mentioned, it requires no trusted setup and relies solely on well-understood cryptographic hash functions, ensuring transparency and post-quantum security.

We implement our zkSNARK using Plonky2, and use the following high-level 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 us to design a more efficient and succinct 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[3]. 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.

Data revealed by the zkSNARK The mining configuration (r, k, t_m, t_n), matrix shapes, position of opened tile, and H_A, H_B.

[3]Observe that plaintext computation is much faster than zk-proving it. Jolt [1], one of the most advanced zkVMs available, is capable of proving 1.5×10^6 cycles/sec on a CPU with 1.3×10^{11} cycles/sec.

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 our code (Ohad: add reference).

5.1 Block Structure

Base Fee ???

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 75KB.

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

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

serialized as 116 bytes. Here pouw_meta is a 32-bytes commitment (BLAKE3) to the underlying PoUW witness – the public data revealed by the zkSNARK as detailed in 4.8.(Ohad: change commitment in the code to include all public inputs?) 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

(Ohad: Update once we implement. I think we do want to keep current format, just add a default Poseidon leaf to taproot.) 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.

6 Block Time, Emission Curve, and Economy

Our goal is to design an economic framework suitable for modern times, and in particular, a store of value for the AI era.

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 2.5-minute 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 6241509074 coins. The supported most basic currency is 10^{-9} of a coin, or nano-Ampere. Hence, the entire network volume is

$$6.241509074 \times 10^{18} \texttt{ nanos},$$

matching the Ampere to elementary particles (electrons per second) conversion.

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.

(Ohad: Match S in our implementation!) Let $S = 6241509074$ denote the fixed total token supply, and 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 $1/t$.[4]

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 2.5 minutes, four years correspond to $H = 4 \cdot 365 \cdot 24 \cdot \frac{60}{2.5} = 840{,}960$ 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 $R(H) = \frac{1}{2}$. 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

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

starting at the genesis $t = 1$. We define the actual emission as the floor of that (Ohad: Decide whether to floor in coins or in satoshis/ nanos. I think satoshis.)

$$E(t) \;=\; \big\lfloor E^*(t) \big\rfloor.$$

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 instable in other blockchains [2].

Aiming for responsivity and stability, we use the Weighted-Target Exponential Moving Average (WTEMA) algorithm. Recall the difficulty of a block is

[4] 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 $1 - (1 - 1/t) = 1/t$.

determined by a `uint256 target` which serves as the upper bound of a mining candidate's hash to be considered valid. In WTEMA-N, 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:

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

Here T is the intended solve time (2.5 minutes).

WTEMA-N is an exponential filter with time constant $N \cdot T$. We set $N = 576$, corresponding to a decay time of one day. This necessitates additional constraints on timestamps:

- **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.

7 Planned Launch

The network is launched on March 10th, making the node code public. In addition, we will publish a vLLM plugin that implements a new quantization mechanism which re-implements a layer in a DNN via our "two-for-one" scheme.

8 Future Versions

This blockchain is complete shift in the way that people think and interact with blockchains. Since Ampere is tied to a real-world asset, i.e., AI, it has to be constantly evolving and adapting to, e.g., new LLM architectures, workloads of AI agents, and more. As such, we plan to keep improving the protocol.

The current scheme, as described in this document, supports proof of work for *exact* matrix multiplication. As such, we assume that weight and activation matrices are quantized to INT7. This introduces two pain-points: (1) computations need to be quantized to INT7 and cannot be executed in their original data type, and (2) the scheme does not support floating-point (FP) computation.

We plan to overcome both items above by implementing a proof of work scheme for *approximate* matrix multiplication. The main idea is to inject a low magnitude noise into each matrix multiplication where the output is not going to be significantly affected by it. Note that matrix multiplication in FP is *already* approximate by definition! This scheme is currently in design and we will publish futher details when it is ready.

References

[1] Markos Georghiades, Andrew Milson, Justin Thaler, Andrew Tretyakov, Julius Zhang, and Michael Zhu. 64-bit proving for jolt, without a slowdown. a16z crypto, October 2025. Accessed 2026-01-01. URL: https://a16zcrypto.com/posts/article/64-bit-proving-jolt/.

[2] Jonathan Toomim. BCH protocol upgrade proposal: Use ASERT as the new DAA. read.cash, July 2020. https://read.cash/@jtoomim/bch-protocol-upgrade-proposal-use-asert-as-the-new-daa-1d875696 (accessed 2026-01-11).