Lite Paper v2.0

Cyborg Network

Verifiable AI inference at scale

Barath Kanna • Megha Varshini Tamilarasan

Abstract

We present a formal specification of the Cryptographically Yielded Blockchain-Orchestrated Resource Grid (CYBORG) chain, a decentralized AI inference protocol that integrates a Substrate-based runtime with multiple interoperable local peer-to-peer AI inference networks. Each network comprises heterogeneous AI-compatible hardware, optimized for domain-specific machine learning applications and orchestrated via a blockchain-backed consensus mechanism.

CYBORG, built on Polkadot, establishes a distributed AI execution environment where inference workloads are partitioned and executed across a network of embedded accelerators.

Each AI miner participates in the computational process while embedding a ZKML-enabled certification module, ensuring verifiable proof of execution without exposing model parameters or inference data. This cryptographic attestation mechanism mitigates adversarial modifications in the inference pipeline and guarantees deterministic correctness in model outputs.

The protocol enables low-latency, hyperlocal AI inference infrastructure at a global scale, optimized for real-time AI systems such as humanoids, autonomous robotics, and mission-critical cyber-physical systems. By leveraging provably secure computation and distributed orchestration, CYBORG aims to establish a

fault-tolerant, trustless AI execution framework that ensures scalability, privacy, and economic viability for next-generation machine intelligence

Cyborg is built with Vision 2030 in mind, fostering AI adoption through a decentralized, cost-efficient, and privacy-preserving inference network that supports the infrastructure demands of an AI-driven future.

Introduction

1.1 Nomenclature

In this paper, we introduce Cyborg Network, a decentralized, blockchain-governed AI inference protocol designed to orchestrate globally distributed AI processing nodes. Cyborg Network leverages a hybrid incentive model, integrating both cryptographic rewards and fiat-based incentives, to drive enterprise adoption while ensuring regulatory compliance.

The term CYBORG originates from its underlying architectural principle: Cryptographically Yielded Blockchain-Orchestrated Resource Grid. This nomenclature reflects its core design—a decentralized system where AI workloads are securely allocated, executed, and verified across a distributed network of inference nodes.

An early conceptual version of Cyborg Network was first outlined in research discussions on off-cloud AI inference scalability, addressing the challenges of cost-efficiency, data sovereignty, and fault tolerance. Unlike conventional cloud-based AI execution models, Cyborg Network proposes a globally scalable, trustless AI infrastructure that ensures deterministic AI processing with cryptographic attestation

1.2 Driving Factors

A decentralized AI inference network must be fault-tolerant, cryptographically verifiable, and economically sustainable. Traditional cloud-based inference pipelines suffer from latency constraints, cost inefficiencies, and opaque execution environments, making them unsuitable for AI systems requiring real-time, deterministic, and privacy-preserving computation. Cyborg Network introduces a blockchain-governed, hyperlocal AI execution layer, eliminating single points of failure and ensuring inference workloads remain provably correct, accessible, and censorship-resistant.

Bitcoin demonstrated the viability of decentralized economic coordination, ensuring immutable and censorship-resistant transactions. Ethereum expanded this model with Turing-complete smart contracts, unlocking programmable economic mechanisms but still constrained by execution costs and state coherency limitations. Polkadot further evolved the paradigm by enabling specialized, interoperable blockchains, optimizing both scalability and cross-network composability. Cyborg Network builds on this foundation, integrating zero-knowledge attestations, decentralized scheduling, and a hybrid incentive model to enable distributed, privacy-preserving AI inference at scale.

The architecture is governed by five core principles:

1. Resilience – Byzantine-resistant execution with no reliance on centralized entities.

2. Verifiability – ZKML-powered proof generation for deterministic and tamper-proof inference validation.

3. Scalability – Adaptive inference placement across geographically distributed AI accelerators, optimizing for latency and efficiency.

4. Economic Alignment – A hybrid incentive model, combining crypto-economic staking mechanisms with fiat-based enterprise adoption pathways.

5. State Coherency – Efficient cross-node synchronization, ensuring computational consistency across dynamic workloads.

Cyborg Network enforces deterministic execution guarantees while ensuring AI models operate with cryptographic integrity, low-latency response times, and regulatory-compliant data sovereignty. By extending Polkadot's principles of scalability, interoperability, and economic flexibility, it establishes a globally distributed, blockchain-secured AI processing framework, advancing decentralized inference beyond traditional paradigms.

1.3 Fundamentals

This section covers the core principles that underpin the design and functionality of our system, providing the necessary technical background for understanding its implementation.

1.3.1 Neural Networks

Neural Networks, structured as interconnected layers of artificial neurons, execute mathematical operations on input data through computational units known as operators. The connectivity of these layers is governed by weighted edges, which are iteratively adjusted during training, allowing the model to learn. These weights, or parameters, directly influence the computational and memory demands of inference.

As neural networks scale in complexity, so do the computational requirements and memory footprint necessary for both training and inference. Increased depth, neuron count, and architectural sophistication introduce an exponential rise in mathematical operations. Each operation demands storage and processing, creating bottlenecks on resource-constrained devices.

Inference in Deep Neural Networks (DNNs) involves propagating input data through successive layers to generate an output. Two primary constraints impact this process:

1. Memory footprint – The storage required for model parameters and intermediate activations.

2. Computational intensity – The Giga Operations Per Second (GOPS) needed for execution.

A device with inadequate memory cannot store the entire model, rendering inference infeasible. Similarly, limited computational throughput may introduce excessive latency, making real-time AI processing impractical on lower-powered hardware.

To address this, we frame the core problem as: How can large-scale AI inference be performed on hardware-limited devices? In practical terms, how can AI applications leverage high-capacity models without being constrained by local computational limits?

By abstracting DNNs as computational graphs, where data (tensors) flows between operators, we can decompose inference into distributed workloads. Tensors, which represent multi-dimensional data structures, are categorized into:

- Input tensors (X) – External inputs or static parameters.
- Activation tensors (Y) – Intermediate or final outputs generated by operators.

In the context of Cyborg Network, we delegate tasks to a localized cluster of custom AI accelerators that aggregate the processing as a unified network by combining individual differential neural layers processed at different machines within the same virtual network.

1.3.2 Edge Computing

Edge computing departs from traditional cloud-based architectures by executing computation closer to the data source, reducing latency and optimizing resource utilization. Unlike cloud models, edge computing distributes computational tasks across intermediate nodes such as cloudlets, micro data centers, or dedicated inference hardware, minimizing data transmission overhead.

Shi et al. [4]–[6] define edge computing as a network edge execution paradigm where downlink data corresponds to cloud services, and uplink data represents the Internet of Everything (IoE). Satyanarayanan [8] describes it as a model that deploys computing and storage resources closer to mobile devices and sensors to reduce latency. Zha et al. [9] extend this definition by emphasizing resource unification across geographically and network-proximate nodes, enabling distributed compute, storage, and networking for application services.

In an edge computing framework, workloads are offloaded from cloud infrastructure to edge nodes, leveraging localized compute resources to process data in real time. This approach enhances network efficiency, reduces reliance on cloud data centers, and supports low-latency, high-bandwidth applications. The architecture integrates compute, storage, and networking capabilities at the edge, addressing key industry requirements such as real-time processing, application intelligence, security, and privacy preservation [10].

Edge computing is widely adopted in scenarios demanding high-throughput, low-latency execution, particularly in AI-driven workloads, industrial automation, and IoT applications. Research continues to advance edge-native architectures, optimizing distributed execution models for fault tolerance, workload scheduling, and secure data processing [10]–[14].

1.3.3 Zero-Knowledge Machine Learning (ZKML) in Decentralized Inference

The increasing opacity of proprietary machine learning (ML) models has created fundamental challenges in model transparency, bias auditing, and result reproducibility. While open-source model architectures and weights have traditionally enabled scientific reproducibility and external validation, the commercialization

of foundation models—along with safety concerns over unrestricted access—has driven the industry towards closed-source deployments. This shift significantly impairs external verification, raising two major concerns:
1. Unverifiable Performance Claims – Model providers can assert benchmark superiority without independent validation, leading to skepticism about reported accuracy, robustness, and efficiency.

2. Unintended Bias and Opaque Decision-Making – Proprietary models prevent third-party auditing for algorithmic fairness, limiting the ability to detect and mitigate systemic biases in real-world deployments. Algorithmic audits and API-based evaluations offer partial solutions but remain impractical for high-risk ML applications without public interfaces—such as law enforcement predictive models, financial risk assessments, and internal enterprise AI systems. These limitations necessitate cryptographic mechanisms that allow external validation without exposing proprietary model weights or sensitive inference data.

1.3.4 Verifiable AI Inference via ZKML

Zero-Knowledge Machine Learning (ZKML) enables proof-based verification of model execution, ensuring that an AI system adheres to declared performance characteristics without revealing its internal workings. Using succinct Zero-Knowledge Proofs (ZKPs), an ML provider can generate cryptographic attestations that verify:

- The model executed correctly on a given input without tampering.
- The inference result adheres to documented performance standards (e.g., accuracy, fairness constraints).
- The same model architecture and parameters persist across different inference instances, preventing silent model degradation over time.

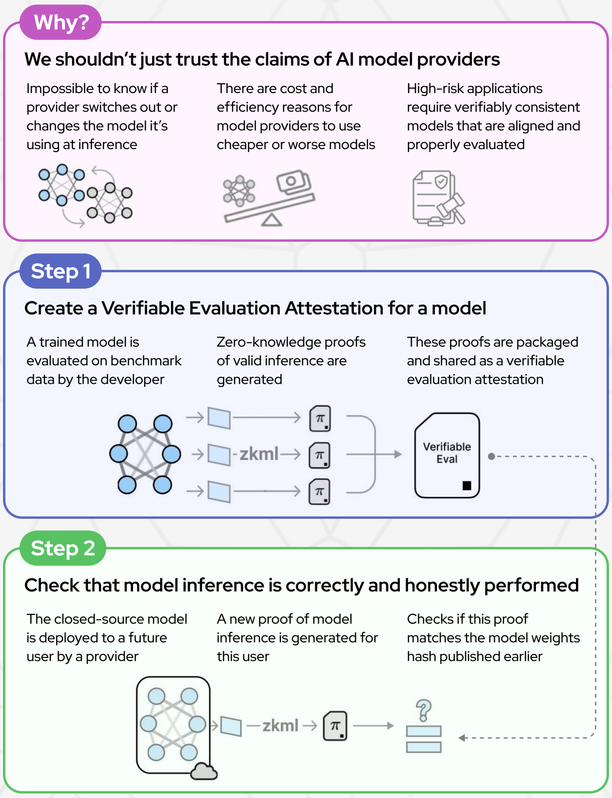
1.3.5 Integrating ZKML in Decentralized AI Architectures

To enforce verifiable AI execution across decentralized AI infrastructure, we implement a hybrid ZKML framework that:

1. Generates Zero-Knowledge Proofs for Model Execution – Ensuring integrity and correctness of inference results, even on untrusted nodes.

2. Enables Selective Disclosure for Model Auditing – Providing cryptographic evidence of fairness and safety without exposing proprietary model internals.

3. Ensures On-Chain Verifiability – Embedding ZKML proofs into blockchain-based decentralized inference systems for persistent, tamper-proof validation.


Why?

We shouldn't just trust the claims of AI model providers

Impossible to know if a provider switches out or changes the model it's using at inference

There are cost and efficiency reasons for model providers to use cheaper or worse models

High-risk applications require verifiably consistent models that are aligned and properly evaluated

Step 1

Create a Verifiable Evaluation Attestation for a model

A trained model is evaluated on benchmark data by the developer

Zero-knowledge proofs of valid inference are generated

These proofs are packaged and shared as a verifiable evaluation attestation

Step 2

Check that model inference is correctly and honestly performed

The closed-source model is deployed to a future user by a provider

A new proof of model inference is generated for this user

Checks if this proof matches the model weights hash published earlier

1.4 Trustless AI Execution in Distributed Inference Networks

As AI inference systems become increasingly integrated into critical domains such as healthcare, finance, and security, the need for verifiable execution mechanisms grows in importance. Traditional inference pipelines rely on centralized control and opaque verification processes, limiting trust in model outputs. In decentralized environments, where multiple nodes participate in inference computation, ensuring the integrity and privacy of AI execution becomes a complex challenge.

1.4.1 Security and Privacy in Distributed AI Inference

In high-stakes inference scenarios, cryptographic proof mechanisms are required to guarantee correctness while preserving confidentiality. A primary challenge arises when inference nodes operate outside a trusted execution environment, introducing risks such as:

- Result Tampering – Malicious or faulty nodes may return manipulated outputs.

- Unverifiable Execution – Model computations occur as a "black box," preventing external validation.

- Data Privacy Leakage – Sensitive input data may be exposed to untrusted compute nodes.

To mitigate these risks, Zero-Knowledge Proof (ZKP) systems provide a framework for verifiable yet privacy-preserving inference execution. By leveraging succinct proofs, inference nodes can cryptographically demonstrate that computations were executed correctly without revealing model internals or input data.

1.4.2 Zero-Knowledge Verification for Model Integrity

Decentralized inference introduces unique security threats, as individual nodes may behave dishonestly, affecting aggregated outputs. To address this, Zero-Knowledge Proof-based attestation mechanisms enable:

- Proof of Correct Execution – Each node generates a ZKP confirming that its assigned model computations were performed as expected.

- Selective Disclosure for Auditing – Cryptographic attestations allow external verification of inference correctness without exposing proprietary model architectures.

- Tamper-Proof Verification – On-chain integration of proofs ensures that any deviation from declared performance constraints is detectable.

These verification mechanisms provide a cryptographic assurance layer, ensuring that AI inference outputs remain trustworthy even in untrusted or adversarial environments.

1.5 Scalability Challenges in Secure Inference

Despite the advantages of cryptographic verification, Zero-Knowledge Machine Learning (ZKML) remains computationally expensive. Existing ZKP-based verification methods introduce substantial latency, with some proof-generation processes taking several minutes per token in large-scale language models. This computational overhead conflicts with the real-time performance demands of AI applications, necessitating further optimizations in:

• Proof Generation Speed – Reducing the computational cost of generating verifiable inferences.

• Efficient Proof Aggregation – Enabling multiple inference nodes to contribute proofs collectively, rather than verifying each result in isolation.

• Hybrid Off-Chain and On-Chain Validation – Balancing security and efficiency by performing proof generation off-chain while committing essential verification data on-chain.

Addressing these challenges is critical for the practical deployment of verifiable inference across distributed AI networks, ensuring that trustless execution does not compromise efficiency.

2. ZK Verification Process

This section explores the application of Zero-Knowledge (ZK) techniques in verifying AI inference integrity. It details how models are decomposed, computed across distributed nodes, and securely validated without exposing sensitive data

2.1.1 Model Decomposition and Verification.

A fundamental aspect of decentralized AI inference is the division of models into multiple computational components, each processed by separate computing nodes. These nodes execute their assigned operations and forward results to a central verifier. For example, a foundation model like LLaMA-3 [11] can be decomposed layer-wise, where each node sequentially processes specific layers, or width-wise, enabling parallel execution. The choice of decomposition strategy depends on application-specific requirements, and our approach supports both.

To ensure correctness and integrity in this setting, Zero-Knowledge Proofs (ZKPs) enable verification without exposing proprietary model weights. Each computing node (prover) generates a proof of correct inference execution, which the central server (verifier) validates without accessing the model's internal parameters. This prevents adversarial inference attacks while maintaining privacy.

2.1.2 Commitment and Proof Generation.

Each node commits to its assigned model fragment and computation results using cryptographic commitments. The proof generation process follows three steps:

1. Commitment: The prover commits to its model fragment, ensuring its integrity while keeping the weights hidden. This is achieved using generalized Pedersen commitments over an elliptic curve group G, formulated as [12]:

$$\text{Commit}(\mathbf{S}, r_S) = h^{r_S}\mathbf{g}^{\mathbf{S}} = h^{r_S}\prod_{i=1}^{d} g_i^{S_i}, \tag{1}$$

2. Proof Generation: The prover executes inference and produces a proof demonstrating correct computation under a finite field F, ensuring that operations—including matrix multiplications, activation functions, and token transformations—are faithfully executed

3. Verification: The central server verifies the proof, ensuring that the model decomposition is correctly followed and computations are consistent with expected outputs. This validation can be performed interactively or non-interactively using protocols such as Sum-Check for arithmetic operations and bit decomposition or lookup tables for non-arithmetic functions.

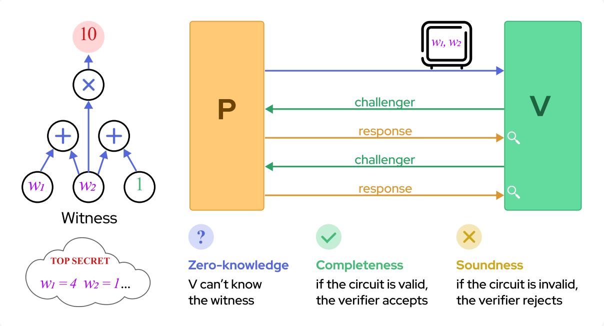


Figure 1: Zero-knowledge proof of circuit 10 = (w1 + w2)(w2 + 1) between a prover (P) and a verifier (V). Hereby, the goal of the prover is to prove to the verifier that P knows a w1 and w2 such that the claimed result "10" is indeed calculated by the equation (w1 +w2)(w2 + 1) (which is denoted by a circuit). The witness w1 = 4 and w2 = 1 are the secret of the prover. Zero-knowledge proof consists of a commitment process (denoted by the safe box) in the beginning, followed by several back-and-forth challenge and response processes between P and V in the interactive scenario. In the non-interactive scenario, the prover can challenge him or herself by the Fiat-Shamir heuristic [13] and the verifier only needs to verify the last response from the prover.

Mathematically, let f: $\mathbb{F}^d \rightarrow \mathbb{F}^n$ represent the transformation performed at a given layer of a foundation model, where the prover computes y = f(x, w) for some private model parameters w. The verifier must confirm that y is computed correctly without learning w. This is achieved through a ZKP of correct execution.

2.1.3 Verification of Model Execution

Each prover commits to the computation and provides a proof of correctness. The verification process ensures:

1. The claimed inference follows the expected model structure.

2. The computations remain within the expected finite field constraints.

3. The proof remains efficient, avoiding excessive computational overhead.

Since neural networks contain both arithmetic (e.g., matrix multiplications) and non-arithmetic (e.g., activation functions, token transformations) operations, verification requires encoding both types within the proof system.

2.1.4 Encoding Non-Arithmetic Operations in Zero-Knowledge Proofs

A fundamental challenge in verifying neural network execution lies in non-arithmetic operations such as activation functions. Two primary techniques are employed:
1. Bit Decomposition for Activation Functions
Consider the ReLU activation function [14]:

$$A = \mathrm{sign}(Z) \odot \mathrm{abs}(Z)\ , \tag{2}$$

where ⊙ is the Hadamard product, abs (Z) is the element-wise absolute value, and

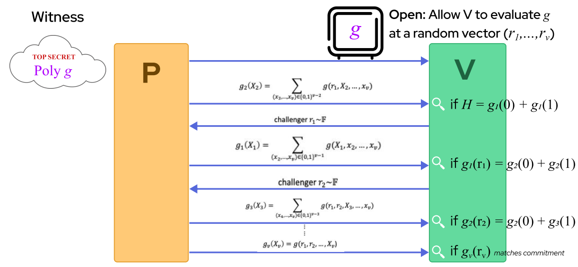

Witness
TOP SECRET
Poly g
g
Open: Allow V to evaluate g at a random vector $(r_1,\ldots,r_v)$
P
V
$g_2(X_2) = \sum_{(x_3,\ldots,x_v)\in\{0,1\}^{v-2}} g(r_1, X_2, \ldots, x_v)$
if $H = g_1(0) + g_1(1)$
challenger $r_1 \sim \mathbb{F}$
$g_1(X_1) = \sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v)$
if $g_1(r_1) = g_2(0) + g_2(1)$
challenger $r_2 \sim \mathbb{F}$
$g_3(X_3) = \sum_{(x_4,\ldots,x_v)\in\{0,1\}^{v-3}} g(r_1, r_2, X_3, \ldots, x_v)$
if $g_2(r_2) = g_2(0) + g_3(1)$
$g_v(X_v) = g(r_1, r_2, \ldots, X_v)$
if $g_v(r_v)$ matches commitment

Figure 2: Sum-Check protocol for $H \overset{?}{=} \sum_{(x_1,x_2,\ldots x_v)\in\{0,1\}^v} g(x_1, x_2, \ldots, x_v)$.

$$sign\ (z) = \begin{cases} 0\ , \quad z < 0, 1\ ,\ \text{otherwise} \end{cases} \tag{3}$$

We denote the binary representation of z as $(z_0, z_1, \ldots, z_Q)$ where $Q = \log_2 p$ (e.g., 128 or 256 in a finite field of order p). The bit decomposition must satisfy:

$$0 \overset{?}{=} z_i(z_i - 1), \quad \forall i \in \{0, \ldots, Q\}.$$

(4)

To verify ReLU execution in zero-knowledge, we enforce reconstruction constraints:

$$z \overset{?}{=} (2z_0 - 1) \sum_{i=1}^{Q} 2^{Q-i} z_i,$$

(5)

$$a \overset{?}{=} z_0 \sum_{i=1}^{Q} 2^{Q-i} z_i.$$

(6)

Using the Schwartz-Zippel Lemma, the correctness proof can be merged into a single Sum-Check equation:

$$2z_0 - 1)\left(\sum_{i=1}^{Q} 2^{Q-i} z_i\right) - z + (z_0 \sum_{i=1}^{Q} 2^{Q-i} z_i - a) r + \sum_{i=0}^{Q} z_i (z_i - 1) r^{i+2} \overset{?}{=} 0$$

(7)

where $r \sim \mathbb{F}$ is a randomly chosen field element to ensure probabilistic soundness.

2.1.5 Lookup Table Verification for Non-Arithmetic Computations

For operations such as Softmax, which are computationally expensive in a ZKP setting, we use a lookup table approach. Let T be a shared table containing valid input-output pairs for a non-arithmetic operation f, structured as:

$$T = \{(T_X, f(T_X))\} \subset \mathbb{F}^{n_2 \times (d_1 + d_2)}$$

(8)

To prove correct execution of f, the prover must show that its computed output Y matches a valid table entry. This is formulated as a subset argument:

$$\sum_{i=0}^{d_1 - 1} r^i X_i + \sum_{i=0}^{d_2 - 1} r^{i + d_1} Y_i \overset{?}{\in} \sum_i \mathrm{Col}_i(T) r^i.$$

(9)

Using the Sum-Check protocol, subset verification reduces to proving that the following polynomial identities hold:

$$\prod_{i \in [n_1]} (X + S_i) = \prod_{i \in [n_2]} (X + T_i)^{e_i}.$$

(10)

By taking the logarithmic derivative, we derive the final rational function identity:

$$\sum_{i \in [n_1]} \frac{1}{X + S_i} = \sum_{i \in [n_2]} \frac{e_i}{X + T_i}.$$

(11)

The prover commits to $e_i = |\{j : S_j = T_i\}|$, and this can be verified efficiently in zero-knowledge using the Sum-Check protocol.

The Pedersen commitment (1) satisfies the binding property: once sent to the verifier, the opening information *(r, S)* cannot be changed by the prover. At first glance, proving the ability to "open" the commitment may seem to require revealing the witness S to the verifier, potentially violating the zero-knowledge property. However, this concern is mitigated due to the homomorphic property of the Pedersen commitment. Specifically, given two commitments Commit *(S_1, r_1)* and *(S_2, r_2)* corresponding to tensors S_1 and S_2, we have:

$$\mathrm{Commit}(S_1, r_1) \cdot \mathrm{Commit}(S_2, r_2) = \mathrm{Commit}(S_1 + S_2, r_1 + r_2),$$

which results in the commitment of *(S_1 + S_2)*.

This property enables the prover to demonstrate knowledge of S without disclosing it by committing instead to a linear transformation:

$$S' \leftarrow S \cdot e + D,$$

(12)

where D is a d-dimensional hiding vector chosen by the prover, and e is a random scalar challenge sampled by the verifier. The vector D ensures $S.e+D$ that appears random to the verifier, preserving zero-knowledge. The existence of e guarantees special soundness: executing the protocol with two different challenges e_1 *and* e_2 leads to

$$S'_2 = S \cdot e_2 + D. \qquad S'_1 = S \cdot e_1 + D,$$

(13)

Since there are $2d$ unknowns *(* S *and* D *)* and $2d$ equations, two valid transcripts *(S, e₁, D)* and *(S, e₂, D)* uniquely determine S and D through Gaussian elimination, achieving special soundness.

Algorithm 1: Commitment Opening Procedure

This algorithm enables the prover to convince the verifier that they know a witness $s \in \mathbb{F}^d$ and a witness $t \in \mathbb{F}$ that are openings of commitments:

$$\text{Commit}(t, r_t) := h^{r_t} g^t,$$

such that $\langle s, y \rangle = t$, where h, g are randomly sampled generators from an elliptic curve group. The algorithm is provably complete, exhibits zero-knowledge, and ensures special soundness.

Steps:

1. Publicly known generators h, g, g_1, ……. , g_d are randomly sampled from the elliptic curve group G.

2. The prover sends commitments:

$$c_S = \text{Commit}(S, r_S) := h^{r_S} \prod_{i=1}^{d} g_i^{S_i},$$

$$c_t = \text{Commit}(t, r_t) := h^{r_t} g^t.$$

3. The prover picks $D \in \{0, 1, \ldots, p-1\}^d$ and hiding factors $r_1, r_2 \in \{0, 1, \ldots, p-1\}$, then sends commitments:

$$c_D = \text{Commit}(D, r_1) := h^{r_1} \prod_{i=1}^{d} g_i^{D_i},$$

$$c_{\langle D, y \rangle} = \text{Commit}(\langle D, y \rangle, r_2) := h^{r_2} g^{\langle D, y \rangle}.$$

4. The verifier selects a random challenge $e \in \{0, 1, \ldots, p-1\}$ and sends it to the prover.

5. The prover computes transformed values:

$$S' := S \cdot e + D, \quad r_{S'} := r_S \cdot e + r_1,$$

$$r_t' := r_t \cdot e + r_2.$$

6. The prover sends $(S', r_{S'})$ and r_t' to the verifier. These conceal S, r_S, t, r_t , ensuring zero-knowledge.

7. The verifier computes:

$$c_{S'} = \text{Commit}(S', r_{S'}) := h^{r_{S'}} \prod_{i=1}^{d} g_i^{S'_i},$$

$$c_{S'} \overset{?}{=} c_S^e \cdot c_D.$$

8. The verifier computes $t' := \langle S', y \rangle$ and its commitment:

$$c_t' = \text{Commit}(t', r_t') := h^{r_t'} g^{t'}.$$

It then checks:

$$c_t' \overset{?}{=} c_t^e \cdot c_{\langle D, y \rangle}.$$

9. If both checks pass, the verifier accepts; otherwise, they reject.

This algorithm ensures that the prover can demonstrate knowledge of S and t without revealing them, leveraging the homomorphic properties of Pedersen commitments. The zero-knowledge property remains intact, while the protocol guarantees soundness under multiple executions.

2.1.7 Proofs for Arithmetic Operations

Multilinear Extension

Zero-knowledge proofs primarily operate within a finite field F, rather than the real field R used in floating-point calculations. Arithmetic operations in this setting consist of addition and multiplication, which can be efficiently verified using the Sum-Check protocol, particularly the GKR protocol [15, 16]. The key idea in the Sum-Check protocol is to express a d-dimensional tensor $S \in F^d$ as a multi-variable polynomial $S_e : F^{\log_2 d} \to F$ via a transformation known as multilinear extension [18]:

$$S_e(u) = \sum_{b \in \{0,1\}^{\log_2 d}} S(b)\,\beta_e(u,b)\,,$$

(14)

where $b \in \{0,1\}^{\log_2 d}$ represents the binary index of tensor S, and $\beta_e(\cdot,\cdot) : F^{\log_2 d} \times F^{\log_2 d} \to F$ is the unique Lagrange interpolation polynomial:

$$\beta_e(u,b) = \prod_{i=1}^{\log_2 d} (u_i b_i + (1-u_i)(1-b_i))\,,$$

(15)

which satisfies the interpolation property:

$$\beta_e\left(b_1, b_2\right) = \begin{cases} 1, & if\ b_1 = b_2\,; \\ 0, & otherwise \end{cases}$$

(16)

This guarantees that S_e is the unique multilinear polynomial over F such that $S_e(u) = S(u)$ for all $u \in \{0,1\}^d$

Using this multilinear extension, we can rewrite arithmetic verification as the verification of the sum of a multi-variable, low-degree polynomial g:

$$H \overset{?}{=} \sum_{(x_1, x_2, \ldots, x_v) \in \{0,1\}^v} g(x_1, x_2, \ldots, x_v)\,,$$

(17)

which simplifies verification in zero-knowledge proof systems.

2.1.8 Sum-Check/GKR Protocol

Algorithm 2 describes the Sum-Check protocol (also known as the GKR protocol [15])for verifying Equation (17). The protocol proceeds in rounds [17]. In the first round, the prover sends a polynomial $g_1\ (X_1)$ and claims:

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

(18a)

If this claim holds, then $H = g_1(0) + g_1(1)$. To validate, the verifier randomly samples $r_1 \sim F$ and requests proof that:

$$g_1(r_1) \overset{?}{=} \sum_{(x_2, \ldots, x_v) \in \{0,1\}^{v-1}} g(r_1, x_2, \ldots, x_v)\,.$$

(18b)

In the second round, the prover sends $g_2(X_2)$ and claims:

$$g_2(X_2) \overset{?}{=} \sum_{(x_3, \ldots, x_v) \in \{0,1\}^{v-2}} g(r_1, X_2, x_3, \ldots, x_v)\,.$$

(18c)

By applying the Schwartz-Zippel Lemma recursively, the protocol continues until the final round, where all $\overset{?}{=}$ claims can be verified with high probability (failure probability at most $d/|F|$, where d is the polynomial degree). The final claim $g_v(r_v) = g(r_1, r_2, ..., r_v)$ can be verified via the commitment opening. The Sum-Check protocol ensures completeness through its structured proof construction. Soundness follows from recursive applications of the Schwartz-Zippel Lemma, while zero-knowledge properties are guaranteed by the Pedersen commitment scheme.

2.1.9 Reducing Linear Layers to Sum-Check Protocol

Linear layers are fundamental components of foundation models and can be mathematically represented as matrix multiplications. Given matrices $A, B, C \in F^{n \times n}$ such that $C = AB$, we define functions:

$$f_A(i_1, \ldots, i_{\log_2 n}, j_1, \ldots, j_{\log_2 n}) = A_{ij},$$

(19)

Where $(i_1, \ldots, i_{\log_2 n})$ and $(j_1, \ldots, j_{\log_2 n})$ represent binary indices of i and j, respectively. The multilinear extensions $\tilde{f}_A, \tilde{f}_B, \tilde{f}_C$ of these functions satisfy:

$$\tilde{f}_C(i_1, \ldots, i_{\log_2 n}, j_1, \ldots, j_{\log_2 n}) = \sum_{b \in \{0,1\}^{\log_2 n}} \tilde{f}_A(i_1, \ldots, i_{\log_2 n}, b) \cdot \tilde{f}_B(b, j_1, \ldots, j_{\log_2 n}) \cdot$$

(20)

Since Equation (20) conforms to the form of Equation (17), we can apply the Sum-Check protocol to verify the correctness of linear layers. Specifically, we define a degree-2 polynomial

$$g(b) = \tilde{f}_A(i_1, \ldots, i_{\log_2 n}, b) \cdot \tilde{f}_B(b, j_1, \ldots, j_{\log_2 n})$$

and verify $H = \tilde{f}_C(i_1, \ldots, i_{\log_2 n}, j_1, \ldots, j_{\log_2 n})$

at a random evaluation point .

$$(i_1, \ldots, i_{\log_2 n}, j_1, \ldots, j_{\log_2 n}) \in F^{\log_2 n \times \log_2 n}$$

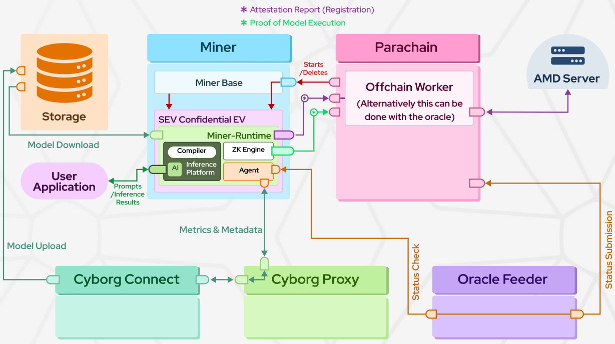
This enables the efficient verification of matrix multiplications within zero-knowledge proof systems.

3. System Design & Operation

This section provides an in-depth look into the structure and functionality of Cyborg Network. We detail how our network topology, architecture, and workflow enable AI inference at the edge. Additionally, we explore the role of miners in contributing computational power and the Cyborg Connect application that allows users to deploy, manage, and monitor AI workloads efficiently.

3.1 Network Topology

The Network consists of multiple interdependent components, each facilitating different aspects of AI inference execution and verification. These components include the blockchain, miner, Cyborg Connect (frontend), Oracle Feeder, Cyborg Proxy, and Storage Nodes, as illustrated in Diagram 2.



3.1.1 Worker Nodes and Execution Layer

Miners within the network function as worker nodes, a blockchain-native node definition specific to Cyborg. These nodes are non-consensus participants, operating independently of the blockchain's validation layer. Upon joining the network, a worker node remains in an idle state until assigned an AI model for inference by the runtime. Task assignment follows a deterministic scheduling mechanism that factors in node availability, compute capacity, and latency constraints.

Once inference execution begins, the miner generates periodic resource consumption reports, logging memory, compute cycles, and power usage. These reports are relayed to the blockchain to maintain an immutable record of execution costs. In parallel, the miner generates ZKML proofs, which serve as cryptographic attestations of the model's output, ensuring computational integrity. The blockchain verifies these proofs to validate inference correctness without exposing the underlying model weights or input data.

3.1.2 Storage and Model Management

AI models and associated weight files are stored in a distributed, cryptographically secured storage layer. This layer is implemented using CESS, a decentralized storage protocol optimized for high-performance data retrieval. Models are encrypted before storage, and only authorized worker nodes can decrypt them upon task allocation. The storage nodes operate independently from inference nodes, ensuring a separation between compute and data layers.

3.1.3 User Interaction and Authentication

Users interact with the system via Cyborg Connect, a web and mobile interface for managing inference requests. The interface does not directly communicate with the blockchain; instead, a proxy server handles user authentication and dashboard data retrieval. Private key authentication is required to decrypt user-specific inference logs, ensuring that only authorized users can access their execution data.

3.1.4 Oracle Integration and Data Flow

The Oracle Feeder serves as a bridge between the blockchain and the off-chain inference network. It transmits task parameters, model assignments, and inference results between the on-chain scheduling logic and off-chain worker nodes. The oracle operates asynchronously, ensuring low-latency data propagation while maintaining message integrity through cryptographic signatures.

The architecture follows a modular design, with independent execution, storage, and verification layers interacting through cryptographically secured communication channels. Each component operates in a permissionless environment, where task allocation, execution verification, and cost settlement occur without a centralized coordinator.

3.2 The Miner

The Cyborg miner is a purpose-built, high-performance edge server designed as an AI accelerator and blockchain node. Built on Nvidia's Jetson architecture, it integrates various hardware components to serve dual functions:

as an AI inference engine and as a fully operational blockchain node within the Cyborg Parachain ecosystem. The miner is equipped with 2TB of NVMe SSD storage, leveraging AES encryption for data-at-rest protection, ensuring high throughput and secure storage for large-scale AI models and datasets.

In terms of computational capability, the Cyborg miner is optimized for AI inference, capable of delivering up to 200 TOPS (Tera Operations Per Second). This performance enables the efficient execution of computationally intensive models such as deep learning inference tasks.

The miner operates strictly within the Cyborg blockchain network. It is pre-configured to function as a worker node in the blockchain, which ensures that all operations, including AI model assignments and task execution, are orchestrated through the Cyborg Connect platform. Communication with the miner is mediated via the blockchain's runtime environment, meaning the miner is not directly controlled by external commands but rather by on-chain instructions, enhancing security and consistency.

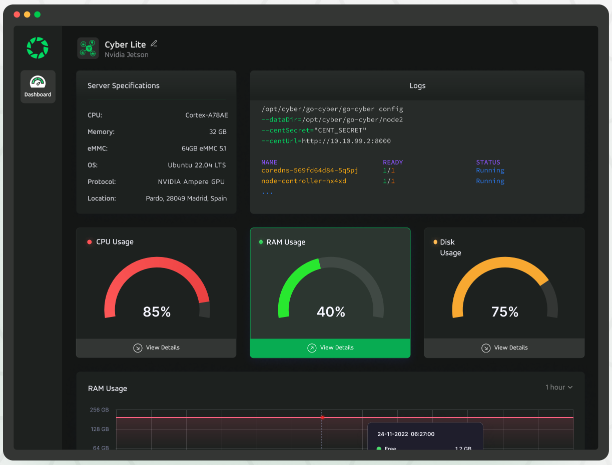
Geo-tagging of the miners ensures precise location tracking for optimal model deployment. This feature allows users to deploy models on miners located in specific geographic areas, improving inference performance by reducing latency and increasing computational efficiency.

The Cyborg miner is designed to be Zero-Knowledge (ZK) ready, supporting Zero-Knowledge Proof (ZKP) mechanisms. This feature facilitates privacy-preserving computations and secure validation of AI model outputs without disclosing sensitive data. The ZK architecture is fully integrated into the miner's execution pipeline, enabling the secure execution of models while maintaining confidentiality and integrity, particularly in scenarios requiring proof of correctness without exposing the underlying model or input data.

The Cyborg miners will be exclusively manufactured and distributed globally by our organization, ensuring consistent quality control and supply chain management. These miners will be made available for purchase directly to the public, with full technical support and integration capabilities. Additionally, we will establish a dedicated Customer Success team, composed of experts in both AI and blockchain technologies, to assist users with any inquiries or technical challenges they may encounter. The team will provide support for deployment, configuration, troubleshooting, and optimal usage of the Cyborg miner within the broader ecosystem, ensuring a seamless user experience.

3.3 Cyborg Connect

Cyborg Connect is a web and mobile-native software platform designed to deploy pre-trained AI models across globally distributed AI miners.



Server Specifications

CPU:	Cortex-A78AE
Memory:	32 GB
eMMC:	64GB eMMC 5.1
OS:	Ubuntu 22.04 LTS
Protocol:	NVIDIA Ampere GPU
Location:	Pardo, 28049 Madrid, Spain

Logs

```
/opt/cyber/go-cyber/go-cyber config
--dataDir=/opt/cyber/go-cyber/node2
--centSecret="CENT_SECRET"
--centUrl=http://10.10.99.2:8000

NAME                           READY          STATUS
coredns-569fd64d84-5q5pj       1/1            Running
node-controller-hx4xd          1/1            Running
...
```

Built on the Cyborg blockchain, the platform leverages its secure and transparent architecture while integrating key API-backed services:

- Payment Gateways – Supports seamless transactions in fiat and cryptocurrency, offering pay-per-use and subscription options.

- KYC & Compliance – Implements KYC/AML checks via Sumsub to ensure regulatory compliance while maintaining privacy.

- Resource Optimization – Dynamically allocates workloads based on performance, availability, and cost efficiency.

- Model & Data Security – Ensures secure model deployment with CESS-encrypted storage and privacy-preserving techniques like homomorphic encryption and split learning.

- Real-time Monitoring & Analytics – Provides insights into AI workload performance, miner uptime, and operational efficiency through an intuitive dashboard.

Deployment Process

1. The user submits a task via Cyborg Connect.

2. The app generates an ephemeral keypair and uses a Diffie-Hellman exchange with the worker's public key to establish an encryption key for CESS.

3. The ephemeral public key and model file ID are recorded on the Cyborg Parachain.

4. Miners monitor the parachain for task assignments.

5. Using the Diffie-Hellman secret, the miner downloads and decrypts the model (compilation may also be required).

6. The miner submits an attestation report confirming its execution state.

7. The Cyborg Parachain validates the attestation report.

8. The miner exposes endpoints for real-time monitoring via Cyborg Agent and inference result retrieval.

9. The user decrypts the results using their Diffie-Hellman key.

4. Scalability and Security in AI Workloads

Efficient AI deployment in a decentralized infrastructure requires advanced techniques to balance computational efficiency, security, and privacy. This section explores key optimizations such as model compression and sharding for resource-constrained environments, along with privacy-preserving mechanisms like secure multi-party computation (MPC) and homomorphic encryption. Additionally, we detail security protocols that ensure model integrity, prevent adversarial interference, and maintain confidentiality within AI execution environments.

4.1 Model Compression

Deploying machine learning models in production introduces constraints that are often overlooked during prototyping. In real-world applications, models must handle high request loads while maintaining low latency and high throughput.

- Latency: The time taken to generate a prediction after receiving an input.

- Throughput: The number of inference requests a system can process per unit time.

Optimizing for these factors requires accelerating model inference while minimizing resource consumption. Model compression techniques achieve this by reducing model size and computational complexity, often leading to significant speedups. While compression primarily targets memory efficiency, it also enhances inference performance, blurring the distinction between compression and optimization. The following sections explore key strategies for improving model efficiency.

4.1.1 Low Rank Factorization

Low-rank factorization is a structured model compression technique that decomposes weight matrices in neural networks into lower-rank approximations, reducing both computational complexity and memory footprint. Given a weight matrix , the goal is to approximate it as the product of two smaller matrices:

$$W \approx AB, \quad A \in \mathbb{R}^{m \times k}, \quad B \in \mathbb{R}^{k \times n}, \quad k \ll \min(m, n)$$

This decomposition constrains the network's representational capacity while preserving critical information, leading to reduced inference latency and lower memory usage.

In convolutional neural networks (CNNs), a practical case of low-rank decomposition involves factorizing 3×3 convolutions into 1×1 convolutions, as seen in SqueezeNet, effectively reducing parameter count and computational overhead.

In large language models (LLMs), low-rank adaptation (LoRA) [18] applies a similar principle to fine-tuning. Instead of updating the full parameter set of a pre-trained model, LoRA introduces low-rank matrices A and B into specific layers, allowing efficient adaptation to new tasks with minimal additional parameters. By freezing the original model weights and optimizing only the low-rank matrices, LoRA significantly reduces the computational burden of fine-tuning while maintaining expressivity.

4.1.2 Pruning

Pruning is a model compression technique that reduces the size and computational complexity of neural networks by eliminating redundant or low-importance parameters. Originally introduced in decision trees to mitigate overfitting by removing unnecessary branches, pruning has since been extended to neural networks, where it involves removing weights (edges) or entire neurons (nodes).

Pruning strategies fall into two main categories:

- Structured Pruning: Entire neurons, channels, or layers are removed, leading to a direct reduction in the model's size and computational cost. The resulting weight matrices shrink, improving both inference speed and memory efficiency.

- Unstructured Pruning: Individual connections (edges) are removed, creating sparse weight matrices. While this does not reduce the model's nominal size, it enables specialized sparse matrix optimizations for efficient storage and computation.

Mathematically, given a weight matrix W, pruning applies a mask M such that:

$$W' = M \odot W, \quad M \in \{0, 1\}^{m \times n}$$

Where $\odot$ represents the Hadamard product, and M determines which parameters are retained (1) or pruned (0).

The challenge lies in determining which weights to prune while minimizing accuracy degradation. Several approaches exist, including magnitude-based pruning (removing low-magnitude weights) and more advanced techniques such as Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS), which leverage second-order derivative information to assess weight importance.

We will employ low-rank factorization in Cyborg Connect to reduce computational overhead by decomposing large weight matrices into smaller, low-rank components. This enables efficient fine-tuning and adaptation of pre-trained models with minimal resource consumption, optimizing performance across decentralized AI miners.

Similarly, pruning eliminates redundant parameters, reducing model size and memory footprint without sacrificing accuracy. By leveraging structured pruning for direct computational savings and unstructured pruning for memory-efficient sparse representations, we ensure AI workloads run with maximum efficiency, maintaining high throughput with minimal resource expenditure.

4.2. Secure and Privacy-Preserving Inference

Ensuring security and privacy in decentralized AI inference requires robust verification mechanisms and data protection strategies. Our approach balances efficiency with rigorous integrity checks to mitigate risks such as tampering, unauthorized access, and data leakage. To achieve this, we integrate Finality-Based Verification (FBV) for model integrity and Split Learning (SL) [19] for privacy-preserving inference. These techniques enable secure execution across decentralized AI miners while maintaining high throughput and computational efficiency.

4.2.1 Finality Based Verification

Given the computational and scalability challenges associated with ZKML for verifying the integrity of LLMs in decentralized systems, we propose a Finality-Based Distribution Verification (CDV) strategy for general inference scenarios. This method leverages the collective agreement of multiple nodes to ensure correctness and integrity of model execution while preserving data privacy.

Finality-Based Verification Process

1. Redundant Execution: A subset of nodes {1, 2, ..., k} independently computes the output yi for the same input x using the model M with parameters θ:

$$y_i = M(x;\theta), \forall i \in \{1, 2, \ldots, k\}$$

2. Output Collection: The outputs {y1, y2, ..., yk} are securely collected for evaluation, requiring efficient communication protocols to protect data integrity.

3. Consensus Determination: A consensus algorithm C evaluates the collected outputs to determine an agreed-upon

result ycon:

$$y_{con} = C(\{y_1, y_2, \ldots, y_k\})$$

The consensus is valid if it meets a predefined criterion, such as majority agreement or a more sophisticated statistical validation.

4. Verification and Finalization: If the consensus result aligns with outputs from a sufficiently large subset of nodes, the model execution is verified. Otherwise, discrepancies indicate potential integrity issues, triggering further investigation or corrective measures.

This Finality-Based approach ensures robust verification of model integrity across decentralized nodes, mitigating the impact of faulty or malicious actors.

Verification in the Context of Model Sharding

In decentralized AI, ML models may be sharded across multiple nodes to enhance scalability. Each node i possesses a unique shard Mi of the complete model M, necessitating a specialized approach to CDV for fragmented model execution.

1. Shard Redundant Execution: Each shard Mi of the complete model M undergoes redundant execution by a designated subset of nodes. Each node within the subset computes:

$$y_{i,j} = M_i(x; \theta_{i,j}), \forall j \in \text{Subset of nodes for } M_i$$

This redundancy enables cross-verification, strengthening the validation process.

2. Redundant Output Collection and Verification: The outputs {yi,1, yi,2, ..., yi,m} for each shard i are collected and evaluated through a shard-specific consensus mechanism:

$$y_{con,i} = C_i(\{y_{i,1}, y_{i,2}, \ldots, y_{i,m}\})$$

The redundancy across nodes enhances the detection of discrepancies or faults in each shard

3. Shard Verification Completion: Upon reaching consensus for each shard i, the integrity of that shard's execution is confirmed before proceeding to the next stage.

4. Model Reconstruction: Once each shard has been independently verified, the shard-specific consensus results {ycon,1, ycon,2, ..., ycon,k} are combined to reconstruct the final model output:

$$Y_{final} = \sum_{i=1}^{k} y_{con,i}$$

This comprehensive verification framework ensures the correctness and security of AI inference within decentralized environments while maintaining scalability and efficiency

4.2.2 Data Privacy Protection via Split Learning

Recognizing the challenges posed by encrypting data for use in decentralized inference systems, we adopt Split Learning (SL) as a pragmatic solution to facilitate secure and efficient computation on encrypted data. Traditional encryption methods such as Homomorphic Encryption (HE), while securing data at rest and in transit, render direct computation costly by obscuring its format and structure.

This limitation is particularly problematic for processing with LLMs within a decentralized framework, where data privacy cannot be compromised.

Split Learning (SL) addresses these concerns by partitioning the computational model, allowing for data to be processed in segments without revealing sensitive information. In this approach, user data is protected by ensuring that it is never directly transmitted to any nodes—only the data embeddings from specific layers are exchanged, and each node accesses only the embeddings of certain layers.

Consider a neural network model N, such as Llama 2, composed of a sequence of 32 layers $\{L_1\ L_2,\ldots\ldots\ldots,L_{32}\}$, each with its own set of parameters Θ_i and activation function σ_i. The input to the network is X, and the output of the i-th layer, given input x_i, can be mathematically described as:

$$a_i = L_i(x_i;\Theta_i) = \sigma_i(W_i x_i + b_i)$$

where W_i and b_i are the weight matrix and bias vector of the i-th layer, respectively, and σ_i is a nonlinear activation function such as ReLU, sigmoid, or tanh.

Assuming the model is split at layer k, where the client handles layers $\{L_1,\ldots\ldots, L_k\}$ and the server handles layers $\{L_{k+1},\ldots\ldots, L_{32}\}$, the client computes the intermediate representation Z as follows:

$$Z = \sigma_k(W_k \cdot \sigma_{k-1}(\ldots\sigma_1(W_1 X + b_1)\ldots) + b_k)$$

This intermediate representation Z is then transmitted to the server, which continues the computation:

$$Y = \sigma_{32}(W_{32} \cdot \sigma_{31}(\ldots\sigma_{k+1}(W_{k+1} Z + b_{k+1})\ldots) + b_{32})$$

The loss function $L(Y, Y_{true})$ computes the error between the network output Y and the true labels Y_{true}, and the gradient of the loss with respect to the model's parameters is computed through backpropagation:

$$\frac{\partial L}{\partial \Theta_i} = \mathrm{ChainRule}\left(\frac{\partial L}{\partial Y}, \frac{\partial Y}{\partial a_{32}}, \ldots, \frac{\partial a_i}{\partial \Theta_i}\right)$$

For privacy protection during the transmission of Z from client to server, differential privacy techniques may be applied. Defining a privacy metric P that quantifies information leakage from Z, a proof of privacy preservation can be demonstrated such that for any ε-differential privacy guarantee, the information leakage remains below a predefined threshold:

$$P(Z) \le \varepsilon$$

It is noted that using differential privacy with SL enhances privacy at the cost of inference quality. Thus, within our framework, this is implemented as a tunable parameter, allowing users to balance privacy and model performance based on their requirements.

By leveraging Split Learning, we effectively navigate the complexities of data encryption within our decentralized inference system for LLMs. This approach preserves the confidentiality and integrity of user data while ensuring the operational feasibility of complex model computations, demonstrating a sophisticated balance between privacy preservation and computational efficiency.

5. Use Cases

This section provides a technical overview of key applications for Cyborg Network, emphasizing decentralized AI inference, model optimization, and privacy-preserving techniques. While the use cases listed here demonstrate broad industry impact, further expansions can be tailored based on specific enterprise needs.

5.1 AI Agents

Cyborg Network provides a robust execution layer for AI agents, enabling them to perform real-time decision-making, dynamic adaptation, and autonomous coordination across multiple domains. By leveraging distributed AI inference, agents can process large-scale data locally, avoiding centralized bottlenecks. Secure model verification ensures execution integrity, making it ideal for self-learning systems in smart cities, autonomous robotics, and industrial automation.

5.2 Smart Cities

From traffic optimization to predictive maintenance, smart city infrastructure benefits from real-time AI inference at the edge. Cyborg Network deploys AI models across public spaces and municipal IoT networks, reducing latency and cloud dependency. Privacy-preserving computation ensures that sensitive urban data remains secure while allowing federated intelligence across city nodes.

5.3 Autonomous Mobility

Autonomous vehicles (AVs) require low-latency perception models for navigation, object detection, and real-time decision-making. Cyborg Network enables decentralized inference, allowing AVs to process local sensor data while offloading complex computations to nearby nodes. Optimized model execution through pruning and quantization ensures faster response times and reduced power consumption.

5.4 Industrial IoT

AI-driven predictive maintenance, anomaly detection, and process automation demand efficient inference on-site, eliminating the need for constant cloud connectivity. Cyborg Network facilitates secure split learning, allowing manufacturers to process proprietary data locally while benefiting from distributed intelligence across multiple production lines.

5.5 Public Safety & Surveillance

AI-powered surveillance systems rely on real-time video analytics for anomaly detection, facial recognition, and threat identification. Cyborg Network enables distributed inference across edge nodes, reducing bandwidth consumption and improving response times. The Finality-Based Verification mechanism ensures that AI-generated alerts remain tamper-proof, while differential privacy safeguards sensitive biometric data.

5.6 Wearables & Personal AI

Wearable devices generate continuous biometric and behavioral data that require low-latency, on-device AI processing. Cyborg Network supports context-aware AI assistants, gesture recognition, and personalized health monitoring, ensuring that private user data remains encrypted while still benefiting from cloud-augmented intelligence.

5.7 Medical AI

Decentralized AI inference enhances medical imaging, disease prediction, and real-time patient monitoring while complying with healthcare privacy regulations. Hospitals can execute AI models on-premises without exposing patient data to external cloud servers. Fully Homomorphic Encryption (FHE) and ZK verification ensure both privacy and execution integrity, facilitating secure AI-driven diagnostics and research.

5.8 Agriculture & Precision Farming

Cyborg Network enables AI-driven automation in agriculture, optimizing crop yield prediction, pest detection, and autonomous farming machinery. Distributed inference ensures low-latency decision-making even in remote areas, while model compression techniques enable AI execution on resource-constrained IoT devices.

5.9 Decentralized Cloud Services

Beyond industry-specific applications, Cyborg Network serves as a foundation for decentralized cloud computing, allowing enterprises to deploy AI workloads dynamically without relying on centralized cloud providers. Developers can optimize for cost, performance, and security, while maintaining full ownership of their models and data.

Conclusion:

The Cyborg Network is poised to transform the landscape of edge computing by delivering a robust, decentralized, and transparent platform that caters to a wide range of industries and applications. Through real-time data processing and analysis, we're committed to providing unparalleled data privacy and security to meet the growing demand for efficient and low-latency solutions in an increasingly connected world. By leveraging the power of blockchain technology, the Cyborg Network goes beyond traditional cloud computing and data centers, nurturing an ecosystem of edge server providers, developers, and end-users. Our platform encourages participation and collaboration, making decentralized edge computing accessible and affordable for everyone.

In conclusion, the Cyborg Network is an ambitious vision, not just a project. Join us to transform data processing, transmission, and protection, and create a more connected, secure, and efficient world.