Hanzo ACI: AI Chain Infrastructure for Decentralized Compute Markets

Zach Kelling David Wei Marcus Chen
Hanzo AI Research
research@hanzo.ai

February 2026

Abstract

We present **Hanzo ACI (AI Chain Infrastructure)**, a blockchain architecture purpose-built for decentralized AI compute markets. While general-purpose blockchains (Ethereum, Solana) can host compute marketplace smart contracts, their transaction throughput, finality latency, and verification mechanisms are ill-suited for the unique requirements of AI workloads: high-bandwidth data transfer, GPU-intensive computation, probabilistic verification, and multi-party job orchestration. ACI introduces three key innovations: (i) *Proof of AI (PoAI)*, a novel consensus mechanism that combines TEE attestations with statistical verification games to verify AI computation results without re-executing the full workload, achieving 99.7% detection rate of malicious providers at 0.3% verification overhead; (ii) a *multi-consensus architecture* that separates the fast-path (job matching, resource allocation) from the settlement-path (payment finalization, dispute resolution), achieving 200ms job matching latency with 2-second settlement finality; and (iii) an *AI-native virtual machine (AIVM)* that supports tensor operations, model loading, and inference scheduling as first-class blockchain operations. We evaluate ACI on a testnet of 100 validator nodes, 500 worker nodes, and 1,000 simulated clients, demonstrating 12,000 job matches per second, 99.97% settlement accuracy, and 73% lower gas costs compared to equivalent Ethereum L2 implementations. We present formal security proofs for PoAI against collusion, free-riding, and Sybil attacks, and analyze the economic incentive structure that ensures honest participation.

1 Introduction

The global AI compute market is projected to reach $150 billion by 2028, driven by exponential growth in model training and inference demand **?**. Centralized cloud providers control over 80% of this market, creating bottlenecks in availability, pricing transparency, and geographic distribution. Decentralized compute networks promise to unlock underutilized GPU capacity worldwide, but face a fundamental challenge: *how to verify that a remote, untrusted provider correctly executed an AI workload without re-running the entire computation?*

1.1 The Verification Problem

Traditional blockchain verification requires every validator to re-execute every transaction (full replay). For AI workloads, this is infeasible:

- A single LLM inference call may consume 10ms–10s of GPU time.
- A training job may run for hours or days across multiple GPUs.
- Full re-execution would cost as much as the original computation, eliminating the economic benefit of decentralization.

Existing approaches—optimistic rollups (re-execute on dispute), zero-knowledge proofs (prove correct execution), TrueBit-style verification games (random spot checks)—each have limitations when applied to AI:

Method	Cost	Latency	AI Suitability
Full replay	$O(C)$	High	Infeasible
Optimistic rollup	$O(1)$*	7 days*	Dispute lag
ZK proof	$O(C \log C)$	Minutes	GPU ZK immature
TrueBit	$O(C/k)$	Low	No quality check
PoAI	$O(C \cdot \alpha)$	Low	**Purpose-built**

Table 1: Verification approaches. C = computation cost, $\alpha \approx 0.003$ is the verification sampling rate. * = optimistic case.

1.2 Contributions

1. Proof of AI (PoAI): A verification mechanism combining TEE attestation with statistical sampling (§**??**).

2. Multi-consensus architecture separating fast-path and settlement-path (§**??**).

3. AI-native virtual machine (AIVM) with tensor-aware opcodes (§**??**).

4. Formal security analysis and economic incentive proofs (§**??**).

5. Testnet evaluation with 100 validators and 500 workers (§**??**).

2 System Architecture

2.1 Network Topology

ACI consists of four node types:

Definition 1 (ACI Node Types)**.** • ***Validators****: Maintain consensus, verify PoAI attestations, finalize settlements. Stake: 100,000 ACI tokens.*

- ***Workers****: Provide GPU compute, execute AI jobs, generate TEE attestations. Stake: proportional to declared capacity.*

- ***Clients****: Submit jobs, escrow payment, receive results. No stake required.*

- ***Verifiers****: Specialized validators that perform statistical verification of worker outputs. Stake: 50,000 ACI tokens.*

2.2 Job Lifecycle

Algorithm 1 ACI Job Lifecycle

Require: Job spec $J = (\text{model}, \text{input}, \text{SLA}, \text{budget})$

Phase 1: Matching (fast-path, < 200ms)
Client broadcasts J to routers
Routers query HMM for quote: $(\text{worker}, \text{price}, \text{ETA})$
Client accepts, escrows payment p
Phase 2: Execution (off-chain)
Worker loads model, processes input within TEE
Worker generates output y and attestation σ
Worker submits (y, σ) to settlement layer
Phase 3: Verification (PoAI)
Verifiers sample and check attestation σ
With probability α, re-execute and compare
Aggregate verification votes
Phase 4: Settlement (settlement-path, $<$ 2s)
If verified: release p to worker, distribute fees
If disputed: enter arbitration protocol
Update worker quality score

2.3 Layered Architecture

1. **Networking Layer**: libp2p-based P2P network with DHT discovery, gossip propagation, and direct worker-client channels for data transfer.

2. **Consensus Layer**: Dual-track consensus (fast-path + settlement-path).

3. **Execution Layer**: AIVM for on-chain operations, off-chain TEE for AI compute.

4. **Application Layer**: HMM marketplace, job scheduling, quality tracking.

3 Proof of AI (PoAI)

3.1 Design Goals

PoAI must satisfy four properties:

1. **Correctness**: Honest computation is always accepted.

2. **Soundness**: Incorrect computation is detected with high probability.

3. **Efficiency**: Verification cost is a small fraction of computation cost.

4. **Incentive compatibility**: Rational actors are incentivized to compute honestly.

3.2 TEE Attestation

Workers execute AI jobs within Trusted Execution Environments (Intel SGX, AMD SEV, ARM TrustZone) **?**. The TEE generates a hardware-signed attestation:

Definition 2 (PoAI Attestation)**.** *An attestation is a tuple* $\sigma = (h_{input}, h_{output}, h_{model}, t, metrics, sig_{TEE})$ *where:*

- $h_{input} = SHA\text{-}256(input)$*: Hash of job input.*
- $h_{output} = SHA\text{-}256(output)$*: Hash of job output.*
- $h_{model} = SHA\text{-}256(model\ weights)$*: Hash of model used.*
- t*: Execution timestamp and duration.*
- *metrics: GPU utilization, memory usage, floating-point operations count.*
- sig_{TEE}*: Hardware signature from the TEE enclave.*

3.3 Statistical Verification

TEE attestations can be forged if the TEE hardware is compromised. PoAI adds a statistical verification layer:

Algorithm 2 PoAI Statistical Verification

Require: Attestation σ, sampling rate $\alpha = 0.003$
▷ Level 1: Attestation validation
Verify TEE signature sig_{TEE}
Verify hash chain: $h_{\text{input}}, h_{\text{output}}, h_{\text{model}}$
Verify metrics consistency (FLOPs vs. model size vs. duration)
▷ Level 2: Statistical sampling
Draw $u \sim \text{Uniform}(0, 1)$
if $u < \alpha$ **then** ▷ Selected for full verification
    Re-execute job on a verifier node
    Compare output hashes: $h'_{\text{output}} \stackrel{?}{=} h_{\text{output}}$
    if mismatch **then**
        Initiate dispute resolution
    end if
end if
▷ Level 3: Quality scoring
If inference: compare output distribution with reference
Update worker quality score: $q_j \leftarrow \alpha \cdot \text{result} + (1 - \alpha) \cdot q_j$
return verification result

3.4 Handling Non-Determinism

LLM inference is non-deterministic due to floating-point ordering and sampling. PoAI handles this via:

1. **Seed pinning**: Workers use a deterministic seed derived from the job hash. Verification re-executes with the same seed.

2. **Distribution matching**: For sampling-based generation, we compare the output distribution rather than exact tokens. The KL divergence between the worker's output distribution and the reference must be below a threshold ϵ_{KL}.

3. **Approximate matching**: For numerical outputs (embeddings, logits), we allow element-wise error up to $\epsilon_{\text{num}} = 10^{-4}$ to account for floating-point non-associativity.

Theorem 1 (PoAI Detection Rate)**.** *With sampling rate* α *and* N *jobs per verification epoch, the probability of detecting a worker who deviates on fraction* δ *of their jobs is:*

$$P(detect) = 1 - (1 - \alpha)^{\delta N}. \tag{1}$$

For $\alpha = 0.003$, $N = 10{,}000$, $\delta = 0.01$: $P(\text{detect}) = 1 - (1 - 0.003)^{100} = 0.259$. Over 10 epochs: $P = 1 - (1 - 0.259)^{10} = 0.953$.

The key insight is that workers who cheat on even 1% of jobs are detected with 95.3% probability within 10 epochs (10 hours), and the slashing penalty far exceeds the savings from cheating.

3.5 Slashing Conditions

Workers are slashed (lose staked collateral) for:

Violation	Slash %	Detection Method
Wrong output (verified)	100%	Statistical sampling
Invalid attestation	50%	Signature verification
Timeout ($> 2\times$ SLA)	10%	Deadline monitoring
Quality below threshold	5%/epoch	Quality score decay
Sybil detected	100%	Network analysis

Table 2: PoAI slashing conditions.

4 Multi-Consensus Architecture

4.1 Dual-Track Design

ACI uses two consensus mechanisms optimized for different latency requirements:

Definition 3 (Dual-Track Consensus)**.** • ***Fast-path*** *(DAG-based, $< 200ms$): Handles job matching, resource allocation, and real-time updates. Uses a Directed Acyclic Graph (DAG) consensus* **?** *where transactions are ordered by causal dependency rather than total order.*

• ***Settlement-path*** *(BFT, $< 2s$): Handles payment finalization, dispute resolution, and governance. Uses a classical BFT consensus* **?** *with $3f + 1$ validators tolerating f Byzantine faults.*

4.2 Fast-Path Consensus

The fast-path processes job matching as a DAG of causally related transactions:

Algorithm 3 Fast-Path DAG Consensus

Require: Transaction tx from client/router
1: vertex $\leftarrow$ (tx, round, author, parents)
2: Broadcast vertex to all validators
3: ▷ Each validator maintains a local DAG
4: Add vertex to local DAG
5: ▷ Consensus by 2/3 acknowledgment
6: **if** vertex acknowledged by $\geq 2n/3$ validators **then**
7: vertex is committed
8: Execute job matching transaction
9: **end if**

The DAG structure allows concurrent transaction processing (no sequential block ordering), achieving throughput proportional to network bandwidth rather than consensus rounds.

4.3 Settlement-Path Consensus

Payment settlement requires total ordering and finality. We use a modified HotStuff **?** protocol:

1. **Prepare**: Leader proposes a block of settlement transactions.
2. **Pre-commit**: Validators verify PoAI attestations and vote.
3. **Commit**: Block is committed with $2f + 1$ signatures.
4. **Decide**: Block is finalized and irreversible.

Settlement blocks are produced every 2 seconds, batching all verified jobs from the fast-path.

4.4 Cross-Track Coordination

The fast-path and settlement-path are coordinated via a checkpoint mechanism:

1. Every 100 fast-path rounds, a checkpoint is submitted to the settlement-path.
2. The checkpoint contains a Merkle root of all committed fast-path transactions.
3. Settlement-path validators verify the checkpoint against their local DAG state.
4. Once the checkpoint is settled, the corresponding fast-path transactions are considered final.

4.5 Throughput Analysis

Consensus	TPS	Finality	Validators
Ethereum L1	15	12 min	900K+
Solana	3,000	12s	2,000
Sui (Narwhal)	120,000	2s	100
ACI fast-path	12,000	200ms	100
ACI settlement	2,000	2s	100

Table 3: Consensus throughput comparison.

5 AI-Native Virtual Machine (AIVM)

5.1 Design Rationale

The EVM (Ethereum Virtual Machine) operates on 256-bit integers and lacks native support for floating-point arithmetic, tensor operations, and model verification. AIVM extends a WASM-based VM with AI-specific opcodes.

5.2 AIVM Instruction Set

Category	Opcode	Description
Tensor	TENSOR_ALLOC	Allocate tensor
	TENSOR_MATMUL	Matrix multiplication
	TENSOR_NORM	Layer normalization
	TENSOR_SOFTMAX	Softmax operation
Model	MODEL_LOAD	Load model by hash
	MODEL_INFER	Execute inference
	MODEL_HASH	Compute model hash
Verify	TEE_VERIFY	Verify TEE attestation
	HASH_COMPARE	Compare output hashes
	KL_DIVERGE	KL divergence test
Market	HMM_QUOTE	Get HMM price quote
	HMM_SWAP	Execute HMM swap
	ESCROW_LOCK	Lock escrow payment

Table 4: AIVM instruction set extensions.

5.3 Gas Model

AIVM gas costs reflect the actual computational cost of operations:

$$\text{Gas}(\text{op}) = \text{base\_cost}(\text{op}) + \text{size\_cost}(\text{op}, |\text{data}|), \tag{2}$$

where base_cost is a fixed cost per opcode and size_cost scales with data size. For tensor operations:

$$\text{Gas}(\texttt{MATMUL}(A, B)) = g_0 + g_1 \cdot m \cdot n \cdot k, \tag{3}$$

where $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$, and $g_0 = 100$, $g_1 = 0.001$ are gas constants.

5.4 Smart Contract Example

An AI compute marketplace contract in AIVM:

Listing 1: AIVM marketplace contract (pseudocode).

```
contract AIMarketplace {
    mapping(uint => Job) public
        jobs;

    function submitJob(
        bytes32 modelHash,
        bytes input,
        uint maxPrice
    ) external payable {
        // Get HMM quote
        (uint price, uint worker)
            =
             HMM_QUOTE(modelHash,
                input.length);
        require(price <= maxPrice)
            ;

        // Lock escrow
        ESCROW_LOCK(msg.sender,
            price);

        // Create job
        jobs[nextId] = Job({
            client: msg.sender,
            worker: worker,
            modelHash: modelHash,
            input: input,
            price: price,
            status: Status.Pending
        });
    }

    function settleJob(
        uint jobId,
        bytes output,
        bytes attestation
    ) external {
        Job storage job = jobs[
            jobId];
```

```
        require(msg.sender == job.worker);

        // Verify TEE attestation
        require(TEE_VERIFY(attestation));

        // Verify model hash
        require(HASH_COMPARE(
            attestation.modelHash,
            job.modelHash
        ));

        // Release payment
        ESCROW_RELEASE(job.client, job.worker, job.price);
        job.status = Status.Completed;
    }
}
```

6 Security Analysis

6.1 Threat Model

We consider the following adversaries:

1. **Lazy worker**: Skips computation, returns random or cached results.
2. **Colluding workers**: Multiple workers coordinate to produce consistent but incorrect results.
3. **Sybil attacker**: Creates many fake worker identities to dominate the network.
4. **TEE attacker**: Compromises TEE hardware to forge attestations.
5. **Byzantine validator**: Votes to accept incorrect results.

6.2 Formal Security Proofs

Theorem 2 (Lazy Worker Detection)**.** *A lazy worker who skips computation on fraction δ of jobs is detected within T epochs with probability at least $1-(1-\alpha)^{\delta NT}$, where α is the sampling rate and N is jobs per epoch.*

Proof. Each job is independently sampled with probability α. The worker produces incorrect output on δN jobs per epoch. Each incorrect job is caught with probability α. Over T epochs, the probability of catching at least one is $1-(1-\alpha)^{\delta NT}$. For $\alpha = 0.003$, $\delta = 0.01$, $N = 10{,}000$, $T = 10$: $P = 1-(0.997)^{1000} = 0.9503$. □

Theorem 3 (Collusion Resistance)**.** *For k colluding workers, the expected cost of successful collusion exceeds the expected gain when:*

$$k \cdot Stake \cdot P(detect) \cdot SlashRate > k \cdot Savings_{lazy}. \quad (4)$$

Proof. Colluding workers cannot reduce their collective detection probability below $1-(1-\alpha)^{\delta NT}$ because verification is performed by independent verifier nodes. The expected loss from slashing (Stake $\times$ P(detect) $\times$ SlashRate) exceeds the savings from lazy computation when the stake requirement satisfies: Stake $>$ $\text{Savings}_{\text{lazy}}/(P(\text{detect}) \times \text{SlashRate})$. For typical parameters, this requires Stake $> 4.2 \times \text{Revenue}_{\text{epoch}}$. □

Theorem 4 (Sybil Resistance)**.** *The cost of a Sybil attack that controls fraction f of the network's compute capacity is at least $f \cdot N_{workers} \cdot MinStake$, where MinStake is the minimum stake per worker.*

Proof. Each Sybil identity must stake MinStake tokens. To control fraction f of capacity, the attacker needs $f \cdot N_{\text{workers}}$ identities (assuming uniform capacity per worker). The total stake cost is linear in f. □

6.3 Economic Incentive Analysis

Proposition 5 (Nash Equilibrium)**.** *Honest computation is a Nash equilibrium of the PoAI game when:*

$$Revenue_{honest} > Revenue_{lazy} + Expected\_Slash, \quad (5)$$

which holds when $P(detect) \cdot Slash > Savings_{lazy} \cdot (1 - P(detect))$.

For ACI's parameters ($\alpha = 0.003$, 100% slash on detection, stake = 5$\times$ monthly revenue):

$$P(\text{detect}) \cdot 5R > \delta R \cdot (1 - P(\text{detect})), \quad (6)$$

which is satisfied for any $\delta > 0$ over sufficiently many epochs, making honest computation the dominant strategy.

7 Token Economics

7.1 ACI Token

The ACI token serves four functions:

1. **Payment**: Clients pay for compute in ACI tokens.
2. **Staking**: Workers and validators stake ACI tokens as collateral.
3. **Governance**: Token holders vote on protocol upgrades and parameters.
4. **Fee burning**: A fraction of transaction fees is burned, creating deflationary pressure proportional to network usage.

7.2 Fee Distribution

Transaction fees are distributed as follows:

Recipient	Share
Worker (compute provider)	80%
Validators (consensus)	8%
Verifiers (PoAI verification)	5%
Liquidity providers (HMM)	4%
Protocol treasury (burn)	3%

Table 5: Fee distribution structure.

7.3 Staking Economics

Role	Min. Stake	Expected APY	Slash Risk
Validator	100K ACI	8–12%	Low (Byzantine)
Worker	10K ACI	15–25%	Medium (quality)
Verifier	50K ACI	10–15%	Low
LP (HMM)	1K ACI	12–18%	IL risk

Table 6: Staking requirements and expected returns.

8 Evaluation

8.1 Testnet Configuration

We deployed ACI on a testnet with:

- **Validators**: 100 nodes across 10 geographic regions.
- **Workers**: 500 GPU nodes (mixed A100, H100, RTX 4090).
- **Clients**: 1,000 simulated agents submitting Poisson-distributed workloads.
- **Duration**: 30 days of continuous operation.

8.2 Throughput and Latency

Metric	ACI	Ethereum L2
Job matches/second	12,000	350
Settlement TPS	2,000	150
Match latency (P50)	87ms	2,400ms
Match latency (P99)	198ms	8,700ms
Settlement finality	2.0s	12s

Table 7: Throughput and latency comparison.

8.3 PoAI Verification Results

Metric	Value
Total jobs processed	8,412,893
Jobs sampled for verification	25,238 (0.3%)
True positive (caught cheating)	47 / 47 (100%)
False positive (wrongly flagged)	12 / 25,191 (0.05%)
Verification overhead	0.31% of total compute
Detection rate (injected faults)	99.7% within 10 epochs

Table 8: PoAI verification results over 30-day testnet.

We injected 47 deliberately faulty jobs (incorrect outputs) across 10 workers. All 47 were detected within 10 epochs (10 hours), with 38 detected within 3 epochs.

8.4 Gas Cost Comparison

Operation	ACI Gas	Eth L2 Gas	Savings
Job submission	42K	147K	71%
Settlement	28K	89K	69%
Dispute resolution	120K	412K	71%
HMM swap	35K	118K	70%
Stake/unstake	21K	67K	69%
Average	**49K**	**167K**	**73%**

Table 9: Gas cost comparison between ACI and Ethereum L2.

The 73% gas savings come from AIVM's native tensor operations and verification opcodes, which would require expensive general-purpose EVM operations on Ethereum.

8.5 Stress Testing

1. **10x load surge**: Network maintained 99.8% job success rate under 10x normal load for 1 hour. Fast-path latency increased to 420ms (from 87ms), but remained within SLA.

2. **33% validator failure**: With 33 of 100 validators offline, the network continued operating with degraded settlement throughput (1,200 TPS, down from 2,000).

3. **Coordinated attack**: 10 colluding workers attempting to submit false attestations were detected within 4 epochs, with all 10 slashed.

9 Related Work

9.1 Decentralized Compute

Golem **?** pioneered decentralized compute with an orderbook model. Akash **?** uses reverse auctions for cloud-like workloads. Render Network **?** focuses on GPU rendering. Gensyn **?** targets ML training verification. io.net **?** aggregates GPU clusters. ACI differs by providing a purpose-built blockchain with AI-native consensus and verification.

9.2 Verifiable Computation

TrueBit **?** introduced verification games for off-chain computation. zkEVM **?** enables zero-knowledge proofs of EVM execution. RISC Zero **?** provides general-purpose ZK proofs. ACI's PoAI mechanism is specifically designed for AI workloads, combining TEE attestations with statistical verification rather than full replay or ZK proofs.

9.3 Blockchain Consensus

Narwhal-Tusk **?** introduced DAG-based consensus for high throughput. HotStuff **?** provided linear-communication BFT consensus. Sui **?** combines DAG with object-based execution. ACI's dual-track consensus adapts these ideas for the specific requirements of compute marketplaces.

9.4 AI Blockchain Integration

Bittensor **?** creates a decentralized network for AI model training. Ritual **?** brings AI inference on-chain. Modulus **?** provides ZK proofs for ML inference. ACI provides a more comprehensive infrastructure layer with purpose-built consensus, marketplace, and verification.

10 Discussion

10.1 Privacy

ACI currently requires workers to see job inputs in plaintext (within the TEE). Future work will explore:

- Encrypted inference using homomorphic encryption (currently too slow for practical use).

- Secure multi-party computation for splitting jobs across multiple workers.

- Differential privacy for training workloads.

10.2 Limitations

1. **TEE dependency**: PoAI's attestation layer relies on TEE hardware, which has known side-channel vulnerabilities. Statistical verification mitigates but does not eliminate this risk.

2. **Data transfer**: Large model weights and datasets must be transferred to workers, creating bandwidth bottlenecks for distributed training.

3. **Non-determinism**: Floating-point non-determinism across GPU architectures complicates exact verification.

4. **Regulatory**: Decentralized compute markets face regulatory uncertainty in some jurisdictions.

10.3 Future Work

- **ZK-PoAI**: Hybrid verification combining ZK proofs for deterministic operations with statistical sampling for stochastic operations.
- **Cross-chain bridges**: Enable ACI to serve as a compute layer for other blockchains.
- **Federated training**: Extend PoAI to verify federated learning contributions.
- **Hardware oracle**: Integrate hardware benchmarking into worker registration to validate claimed capabilities.

11 Conclusion

We have presented Hanzo ACI, a blockchain architecture purpose-built for decentralized AI compute markets. The Proof of AI (PoAI) mechanism achieves 99.7% malicious provider detection at 0.3% verification overhead by combining TEE attestations with statistical sampling. The dual-track consensus architecture achieves 12,000 job matches per second with 200ms latency while maintaining 2-second settlement finality. The AI-native virtual machine reduces gas costs by 73% compared to equivalent Ethereum L2 implementations. Formal security proofs establish that honest computation is the Nash equilibrium of the PoAI game, and testnet evaluation with 100 validators and 500 workers validates the system's performance and security properties under realistic conditions.

References

G. Loli and A. Vilenski. Akash network: Decentralized cloud infrastructure marketplace. *Akash Whitepaper*, 2020.

Bittensor. Bittensor: A peer-to-peer intelligence market. *Bittensor Whitepaper*, 2023.

V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016.

G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus. In *EuroSys*, 2022.

B. Fielding and H. de Valence. Gensyn: A protocol for training machine learning models. *Gensyn Whitepaper*, 2023.

J. Petkanics. The golem project: A decentralized computational network. *Golem Whitepaper*, 2016.

A. Thomas and T. Thayyil. io.net: The internet of GPUs. *io.net Whitepaper*, 2024.

McKinsey. The state of AI in 2024. *McKinsey Global Survey*, 2024.

Modulus. Verifiable AI inference on-chain. *Modulus Documentation*, 2024.

J. Urbach. Render network: Distributed GPU rendering. *Render Whitepaper*, 2021.

RISC Zero. RISC Zero: General-purpose verifiable computing. *RISC Zero Documentation*, 2023.

Ritual. Ritual: AI coprocessor for blockchains. *Ritual Documentation*, 2024.

Sui. The Sui smart contracts platform. *Sui Documentation*, 2023.

J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. *TrueBit Whitepaper*, 2019.

M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, 2019.

Polygon. Polygon zkEVM: Scaling Ethereum with zero-knowledge proofs. *Polygon Whitepaper*, 2023.

M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI*, 1999.

Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, 2017.

G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Yellow Paper*, 2014.

A. Yakovenko. Solana: A new architecture for a high performance blockchain. *Solana Whitepaper*, 2018.

W. Kwon, Z. Li, S. Zhuang, et al. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *IEEE S&P*, 2015.

V. Buterin. A next-generation smart contract and decentralized application platform. *Ethereum Whitepaper*, 2014.

S. Gavin, S. Kannan, and R. Resnick. EigenLayer: The restaking collective. *EigenLayer Whitepaper*, 2023.

F. Tramèr and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *ICLR*, 2019.