ATVENO White Paper

Atvenolabs

PoDW Consensus and the TensynHash Algorithm

Contents

Preface 2

Abstract 2

Atveno's Core Proposition 3

1. Intelligence Is Becoming a Network 3

2. Atveno as Open Infrastructure for Verifiable Intelligence 5

3. Atveno Network Architecture 8

4. PoDW Consensus and the TensynHash Algorithm 12

5. Deterministic Computation and Work Verification 18

6. An AI and Quantum-Friendly Computing Network 22

7. How the Atveno Network Operates 25

8. Network Participation and an Open Ecosystem 28

9. Network Value and Economic Model 31

10. Network Security and Technical Evolution 33

Appendix A Core Objects and Operating Relationships 37

Appendix B Key TensynHash Specification Summary 38

Preface

Intelligence is becoming a network-level capability. As AI moves beyond standalone software into a wider range of applications and computing infrastructure, compute resources, tasks, developers, and users are forming lasting connections around it. Open networks must therefore answer a broader set of questions: who can participate, what work the network recognizes, how nodes verify it independently, and how participants coordinate under shared rules.

Atveno explores open infrastructure for verifiable intelligence. It builds on PoW for security and incorporates AI-related computational structures and a quantum-friendly design into PoDW (Proof of Dyadic Work) consensus. The work algorithm is TensynHash, a synthesis of tensor-based computation and multiple computational mechanisms. Here, quantum-friendly means that a deterministic classical simulation of a five-qubit circuit participates in the work calculation. It does not imply quantum acceleration or post-quantum cryptography.

Convolution, gated memory updates, deterministic quantum-circuit simulation, and hash search form an interdependent computational path within the algorithm. Miners search for valid candidates; nodes recompute the work independently and assess block validity together with transaction execution and state checks.

Atveno aims to let more participants publish AI demand, contribute compute, build applications, and use the network. Over time, the technical role of consensus work can connect with the business value of application-level computing. Open coordination links work verification, service acceptance, data delivery, and value distribution into a network that can operate continuously.

Abstract

Atveno is open infrastructure for verifiable intelligence. PoW, AI-related computational structures, and a quantum-friendly mechanism together define the consensus work of PoDW, while a London-era EVM provides a programmable environment for execution and coordination. Miners contribute work, nodes verify independently, developers build on public interfaces, and users participate through wallet authorization.

TensynHash, the PoDW work algorithm, places T-bridge between A-hash and V-hash. It binds read-only memory prepared by convolution and LeakyReLU, deterministic feedback from the QM**5** five-qubit circuit, KDA-style gated updates, HardTanh clipping, and retrieval into one work path. The A-hash gate is fixed; mining difficulty is controlled by the V-hash target.

An algorithm proof consists of two eight-byte nonces. A node reconstructs the inputs and intermediate calculations from the block context, then verifies the block structure, transactions, and state. Valid work is necessary, but not sufficient, for block acceptance.

Consensus work and application services operate separately. Application-level AI, compute requests, and future hybrid-computing services require their own protocols for execution, acceptance, and settlement;

application demand does not directly change consensus work. This separation gives the base chain and application services distinct rules. The application services described here–including task registration, result acceptance, delivery, and settlement in Sections 6.2, 6.5, 7.4, 8.4, and 9.2–are planned and have not yet been implemented. The base chain provides PoW consensus, a London-era EVM, and standard interfaces.

Atveno's Core Proposition

Atveno centers participation and verification in an intelligent network. Participants can contribute work under chain rules, nodes can independently reach the same conclusion from the same data, and applications can use that foundation to coordinate and settle services.

Participation should be open to anyone who meets the applicable rules. Shared rules, independent verification, and clear roles underpin that principle; network configuration and participation rules define the specific entry points and resource requirements.

Moving from using intelligence to participating in it requires a connection among demand, work, verification, and coordination:

Intelligence → Work → Verification → Coordination → Open Network

1. Intelligence Is Becoming a Network

1.1 From Information Networks to Intelligent Networks

The internet connects information. Blockchains extend that model by allowing value, state, and rules to be agreed upon in an open environment. Participants can collaborate across organizations and geographies and form network relationships under a common protocol.

AI brings intelligence to search, decision-making, automation, and other applications, while creating sustained demand for computation. As tasks, resources, and users connect, intelligence begins to take on the characteristics of a network-level capability.

1.2 New Network Requirements in the Age of Intelligence

Training, inference, and more complex intelligent tasks require compute resources as well as ways to organize tasks and participants. Users, developers, resource providers, nodes, and machines may play different roles. An open network needs clear rules for their collaboration.

As these connections persist, value also depends on how the network organizes capabilities: how demand finds resources, how results are accepted, how contributions are recorded, and how parties deliver under predictable conditions.

1.3 Challenges for Open Intelligent Networks

1.3.1 Open Participation and Network Contribution

Open participation begins with recognizable contributions. In PoW, work is computational evidence bound to a candidate block, satisfying protocol conditions, and independently verifiable by a full node. Atveno embeds convolution, gated memory updates, and deterministic quantum-circuit feedback in TensynHash so that they participate in this work relationship.

Object	Role	Relationship to consensus
PoDW consensus work	Search for valid candidates with TensynHash	Passing the work test still requires full-block verification
AI-related computational structures	Convolution, activation, gated memory, and retrieval	Components of the fixed work function
Application-level compute tasks	Perform computation and delivery agreed with users	Accepted by the application; do not automatically add chain work

Application-level AI orders have their inputs, delivery, and acceptance defined by application protocols. Compute consumed, time online, or an executor's reputation alone cannot determine block validity. User orders do not directly increase the base chain's cumulative work.

1.3.2 Work and Reproducible Computation

Different implementations must reach the same conclusion for a given candidate. The TensynHash 2.9.2 Algorithm Specification therefore defines the program version, input encoding, integer widths, operation order, rounding, and error handling.

TensynHash uses deterministic integer arithmetic. Initial data and parameters are encoded as INT8; memory and principal intermediate results use INT32 fixed-point values; and the quantum circuit is simulated exactly with integer operations. Nodes reconstruct the calculation under these rules, while implementations remain free to optimize their engineering.

Parallel execution, caching, and hardware optimization must preserve the specified result. Reordering across rounding points, applying an activation twice, or repairing an invalid nonce could break that common basis.

1.3.3 Network Coordination and Value Distribution

PoDW verifies work conditions, the EVM verifies transactions and state transitions, and application acceptance checks delivery against an order. The object and scope of each conclusion must be clear. Valid work does not, by itself, establish the truth of external data or the quality of a service.

Consensus rewards, transaction gas, and application service fees correspond to different contributions and should be recorded and accounted for separately.

The chain stores the commonly agreed state; off-chain systems perform the corresponding computation, storage, and delivery. Timeouts, retries, disputes, and reorganizations need explicit rules so a failed task can be handled within the appropriate scope.

2. Atveno as Open Infrastructure for Verifiable Intelligence

2.1 What Is Atveno?

2.1.1 Open Infrastructure for Verifiable Intelligence

Atveno links intelligence-related computation, work conditions set by chain rules, and independent verification within a blockchain coordination system. Full nodes recompute consensus work from publicly available data and verify transactions and state transitions, giving participants a common protocol-level judgment.

At the application layer, this infrastructure offers a way to organize task registration, result records, and service settlement. Each service must still define its own acceptance mechanism before a computational result becomes deliverable application value.

2.1.2 Combining PoW, AI, and Quantum-Friendly Design

PoW supplies work-based rules for block production and fork choice. AI-related convolution, activation, and memory structures, together with deterministic circuit feedback from QM5, enter the work algorithm. The blockchain carries state, records, and network coordination.

PoDW is the consensus mechanism; TensynHash is its work algorithm. PoDW also encompasses block rules, difficulty policy, fork choice, and state validity. TensynHash specifies how a candidate is computed and assessed.

Their relationship lies in data dependencies: A-hash derives parameters; QM5 feedback affects memory updates; retrieval results and measured row indices bind the output; and that output forms the V-hash work context. In computational terms, mining cost comes mainly from the two hash-search stages, A-hash and V-hash. T-bridge, including QM5 and gated updates, runs once for each candidate that passes the A-hash gate and accounts for a small share of computation. The AI- and quantum-related structures matter for data dependency and result binding, rather than their share of compute cost.

2.1.3 An EVM-Compatible Open Blockchain Network

Execution compatibility is bounded by the London-era EVM, corresponding transaction types, and a controlled set of standard RPC methods. It covers the relevant bytecode, transaction encoding, gas, receipts, and logs. Each tool integration must be validated within that boundary.

PoDW work verification and EVM execution are separate layers; a failure in either layer invalidates the block. AI-related operations and quantum-circuit simulation perform work computation, while wallets authorize accounts and the EVM executes transactions.

2.2 Why Combine PoW, AI, and Quantum-Friendly Design?

2.2.1 AI and Intelligent Computational Structures

TensynHash turns AI computational structures such as convolution, activation, and gated memory into deterministic protocol work that nodes can recompute. AI-related operations therefore participate in the consensus decision while preserving the consistency required for independent node verification.

KDA's channel retention, prediction error, and directed updates provide a structural reference for memory transformation. Algorithm parameters are hash-derived; the algorithm does not train or learn from business feedback. Application-model capabilities remain properties of the relevant models and services.

2.2.2 PoW and the Foundations of Network Security

Blocks must satisfy the work conditions in the chain rules. Nodes compare competing forks by total difficulty, the sum of block-header difficulties. PoDW retains this security foundation while allowing independent nodes to verify changes to the work function. Probabilistic confirmation and the possibility of reorganizations still apply.

The actual cost of an attack depends on the algorithm, implementation, and available resources. It requires security analysis and engineering measurements; it cannot be inferred from the computational structure alone.

2.2.3 Quantum-Friendly Design and Deterministic Circuit Feedback

QM5 expresses amplitude transformations with a five-qubit circuit. Candidate-hash-derived bytes determine the sampling, and measured row indices feed back into memory gating and output binding. The circuit representation therefore affects the algorithm result, which nodes can reproduce in full with integer arithmetic on classical hardware.

Quantum-friendly refers to this design and its research direction. Miners do not need quantum computers, and random sampling from a physical device cannot directly replace the deterministic consensus path.

2.2.4 Blockchain and Open Coordination

The EVM expresses authorization, state records, and settlement conditions as programmable rules. P2P networking and storage allow independent nodes to obtain verification data. Keeping consensus work separate from application protocols lets applications expand services while the base chain continues to assess blocks under shared rules.



Figure 1 Atveno's Technical Composition

2.3 How the Atveno Network Comes Together

Intelligent demand starts the application path. PoDW defines protocol contributions to the base chain; independent verification establishes common judgments; and on-chain coordination connects participants. The two paths operate within one network, but an application order is not a prerequisite for consensus block production.

Miners search for TensynHash candidates from block context. Nodes recompute the work and check the complete block under chain rules. Input binding, targets, and algorithm versions come from those rules; identity, remote scores, or declarations of intermediate state cannot replace verification.

Applications organize compute requests, acceptance, and delivery on that foundation. The blockchain stores shared state so developers, resource providers, and users can coordinate under agreed terms. Consensus validity and application service quality are assessed separately.



Figure 2 How the Atveno Network Comes Together

2.4 Design Goals and Core Principles

Atveno maintains consistency through chain rules, independent verification, and deterministic execution, and manages protocol changes through explicit versions. Different engineering implementations must reach the same validity judgment for the same input.

2.4.1 Open Participation

Participation should be as open as the protocol permits, allowing miners, nodes, developers, and users to enter the network under their respective rules.

2.4.2 Network Security

Full nodes independently check work, chain policy, and EVM state transitions. Access services, mining pools, and gateways can facilitate participation but cannot replace consensus judgments.

Security analysis must cover work bypasses, extra degrees of search freedom, data availability, and worst-case verification resources. TensynHash makes no claim of memory hardness or resistance to specialized hardware. Algorithm versions adopted on chain remain frozen; replacing a version requires an upgrade activated at a specified block height under the chain specification.

2.4.3 Verifiability

The algorithm specification covers the complete computation path, supported by test vectors for normal, tampered, boundary, and malicious inputs. Consistency must also hold during synchronization and reorganizations; caches must not bypass contextual checks.

2.4.4 Coordination and Extensibility

Applications may extend off-chain computation and storage, but must specify task versions, acceptance, data availability, and failure handling. Changes to domain tags, gate order, quantization, or activation constants that affect work results must be governed as consensus upgrades.

2.5 Network Value and Long-Term Vision

Atveno seeks to let more participants contribute under chain rules: miners produce work, nodes preserve independent judgment, developers assemble coordination capabilities into applications, and users submit requests and receive results under clear conditions.

Long-term network value comes from reproducible work, reliable delivery, and sustained coordination. Consensus computation and application services follow distinct rules, providing connected foundations for an open infrastructure for an intelligent economy.

3. Atveno Network Architecture

3.1 Overall Architecture

3.1.1 Architecture Overview

Atveno's architecture comprises access, execution, consensus, work computation, P2P networking, and storage. PoDW uses TensynHash; the EVM handles transactions and application state. Nodes combine work verification and state checks to validate complete blocks.

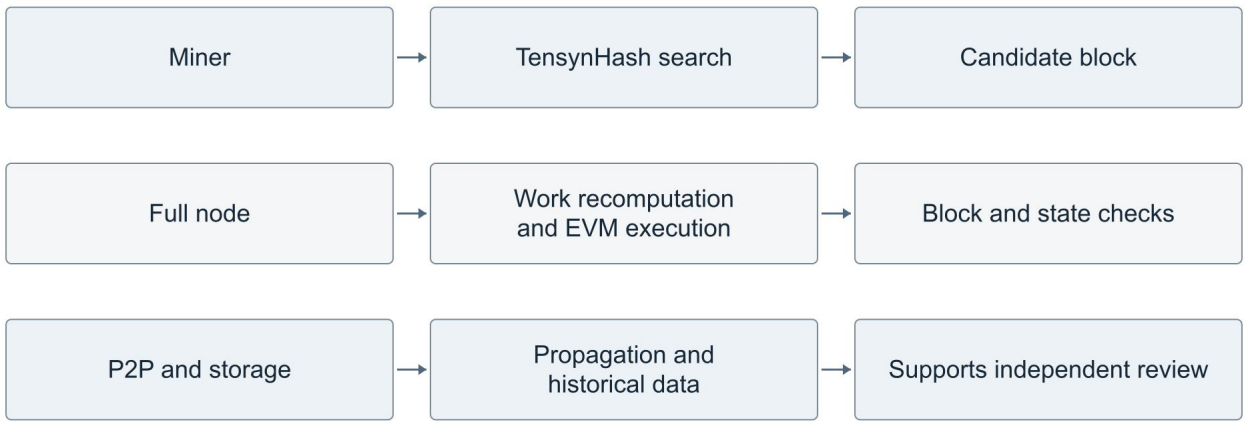


Figure 3 Atveno Network Architecture and Responsibilities

A consensus error invalidates a block; a failed application request is handled within the relevant application. Application schedulers, external data sources, and storage services should not become indispensable trusted parties for producing base-chain blocks.

3.1.2 Core System Components

The access layer receives signed transactions and queries; the execution layer produces state, receipts, and logs; and the consensus layer checks blocks, work, difficulty, and forks. Miners search for work, while P2P networking and storage propagate and retain the data needed for verification.

QM5 is part of T-bridge, and the algorithm verifier sits on a full node's work-validation path. Neither needs a remote quantum service or separate proof service to reach a conclusion.

Application-task layers use their own protocols to organize contracts, off-chain execution, storage, and acceptance components, extending services without changing base consensus rules.

3.1.3 Data, Computation, and State Flows

The data flow begins with a candidate block header determined by chain rules. The specified zeroing, serialization, domain hashing, and timestamp binding produce the 40-byte PoW header P. Chapter 4 defines the byte-level rules.

Miners prepare memory from P, perform the two-stage search, and submit the two nonces with block data. Nodes reconstruct the computation from the candidate and chain rules, check work, transactions, and state, and only then participate in chain selection.

When the block header or parent branch changes, the relevant work is judged in its new context. Applications should likewise restore request and settlement state according to canonical-chain events.

3.2 Blockchain and EVM Compatibility Layer

3.2.1 Base Blockchain Architecture

The blockchain layer handles propagation, state storage, consensus verification, fork processing, and RPC access. TensynHash defines the work function; transaction execution follows the London-era EVM.

A receiving node checks structure and the parent-block relationship, work, and chain policy before executing transactions and comparing the state and receipt roots. Only fully valid blocks enter chain selection; fast synchronization and checkpoints do not replace that judgment.

3.2.2 EVM Compatibility

The EVM executes transactions in block order against the parent state, handles signatures, nonces, gas, and failure rollback, and deterministically produces the new state, receipts, and logs. Compliant clients should obtain identical roots from identical inputs.

The block header uses standard fields, with the work proof stored in nonce and mixHash. Mining requires dedicated interfaces and version conventions. Standard Ethereum tools need adaptation within the stated London boundary; they do not automatically understand TensynHash work data or rules from later execution eras.

3.2.3 Smart Contracts and Application Integration

Developers build applications through transactions, RPC, events, and logs. Compute services also need task contracts, execution interfaces, acceptance policies, and settlement conditions. Public RPC should specify supported methods and resource limits; wallets handle account signatures.

Indexers and back-end services must process reorganizations by block hash. Contract entry points need replay protection and idempotency checks so one delivery cannot be consumed or paid for twice.

3.3 Core Network Components

3.3.1 Consensus and Network Nodes

Full nodes execute transactions, verify work, synchronize, store data, and select forks; they need not mine. Synchronization and reorganization replay must use the appropriate historical versions. Work digests and external caches cannot replace local recomputation.

3.3.2 Work Computation Components

Miners receive the PoW header and target, prepare a read-only cache, and search the nonces for both stages. They may schedule candidates in parallel, but each KDA update starts from C and never accumulates memory across candidates.

Optimization must preserve byte-level results and rounding order. After a task expires, a header changes, or a version switches, miners update inputs according to chain context. Nodes make the final validity judgment.

3.3.3 Algorithm Verification Components

The verifier recomputes the complete path from context, the two nonces, and the targets. Failures of the first-stage gate, nonce constraints, and final target must be distinguishable, and none may be repaired and accepted.

QM5 sampling is determined by hash-derived dice, and read-only memory is generated locally. External randomness or numerical range checks cannot replace the specified data provenance and computation.

3.3.4 On-Chain Coordination and Records

The base chain stores blocks, transactions, and state so nodes can reconstruct consensus judgments. Block rules define how the two work nonces are packaged.

Applications record task IDs, input and output commitments, execution versions, acceptance, and settlement events. Off-chain results must be checked against their commitments when retrieved; application acceptance remains responsible for service correctness.

3.4 On-Chain and Off-Chain Coordination

3.4.1 On-Chain Responsibilities

The chain handles commonly agreed state, authorization, and settlement conditions. Consensus inputs must be available with the block or deterministically reconstructible by nodes. An application state machine defines creation, submission, acceptance, timeout, and terminal states and supports recovery after reorganizations.

3.4.2 Off-Chain Computation and Execution

Miners search locally and nodes recompute locally; the work function need not be executed step by step inside the EVM. Application models and large inputs and outputs can also remain off-chain to limit propagation and storage costs.

Off-chain components must specify data commitments, access authorization, retrieval deadlines, runtime isolation, and appropriate encryption. Specific designs are responsible for availability and confidentiality; an on-chain record alone provides neither.

3.4.3 Work Verification and Application Coordination

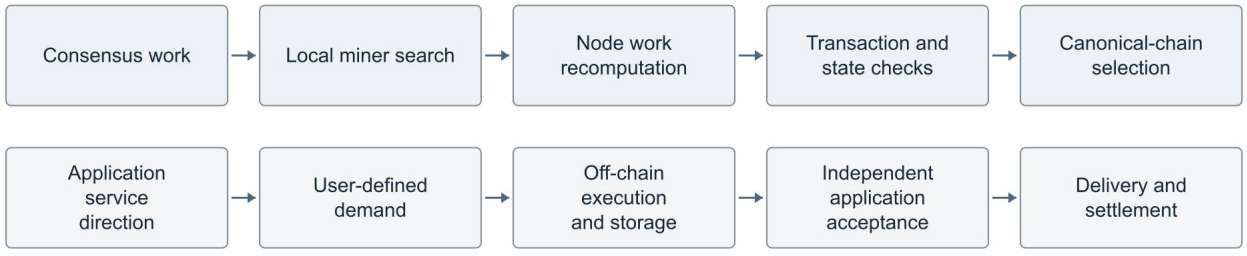


Figure 4 The Boundary Between Consensus and Applications

The consensus path checks work, transactions, and state. The application path advances acceptance and delivery under order terms. Both use the base chain's record-keeping capability, but each defines its own inputs, failure handling, and reward rules.

The coordination layer must define rejection, retry, expiry, dispute, and refund conditions. Application extensions cannot lower base-node work-validation standards or rewrite established consensus rewards.

4. PoDW Consensus and the TensynHash Algorithm

4.1 Atveno's PoDW Mechanism

4.1.1 Definition of Work

PoDW (Proof of Dyadic Work) is Atveno's consensus mechanism within the proof-of-work family. Dyadic refers to two levels of work: the A-hash gate is fixed, while the V-hash target adjusts with chain difficulty. TensynHash, its work algorithm, embeds deterministic AI-related memory computation and simulated quantum-circuit feedback in a two-stage hash path.

In this paper, AI Work means the protocol-specified convolution, activation, gated memory updates, retrieval, and related calculations in the algorithm. Together with QM5 and the two hash stages, it forms the complete consensus work. It is neither rewarded separately on the basis of self-reported compute power nor equivalent to running a user's model inference.

A valid candidate must pass the fixed A-hash gate, satisfy every T-bridge computation rule and nonce_v constraint, and meet the V-hash target. The algorithm proof contains nonce_a and nonce_v, each eight bytes. Nodes reconstruct the remaining data from context.

Work validity and block validity are separate layers. Chain rules also govern structure, parent block, timestamp, difficulty policy, transactions, and state. An algorithm proof does not establish that a user compute order was completed and does not directly determine token issuance.

4.1.2 Binding the PoW Header to the Block

TensynHash uses a 40-byte PoW header P, derived from the canonical block header and block timestamp. Any change to the block contents or timestamp changes P.

The TensynHash brand name is distinct from the consensus byte constants. The ASCII domain tags defined in the algorithm specification are consensus inputs. They have no trailing NUL or unspecified prefix. Changing these bytes changes the computation result.

4.1.3 Two-Stage Search and Block Acceptance

The same PoW header first prepares reusable read-only memory C. The miner searches nonce_a. After a candidate passes the fixed gate, T-bridge computes the second-stage header and prefix; the miner then searches the permitted nonce_v space for the final target. Nodes recompute validity at both stages.

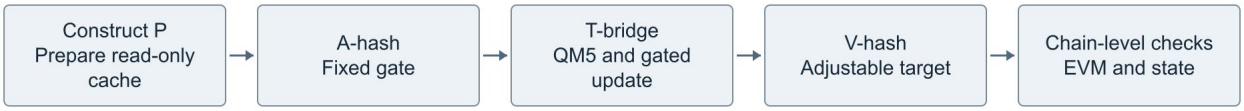


Figure 5 PoDW Work and Block Acceptance

Miners may use different search strategies, but cannot omit the algorithm's dependencies. T_a controls only first-stage admission; T_v varies with mining difficulty. Full nodes use the chain-prescribed target and cannot accept a looser parameter chosen by a miner.

4.2 PoDW and Network Security

4.2.1 Consensus Security

PoDW's basic security assumption remains the ability to sustain competition in valid work. Fork choice compares the total difficulty of competing forks: the sum of their block-header difficulties. This total excludes work spent on the A-hash gate and excludes uncle blocks. At low difficulty, total difficulty is not proportional to actual work expended, increasing reorganization risk early in the network's life. More confirmations reduce reorganization risk but do not provide deterministic finality.

Security also depends on the absence of cheap shortcuts through the work function, availability of verification data, and consistent rule execution across clients. QM5, convolution, and gated computation do not by themselves prove a higher attack cost or rule out algorithmic optimization or resource concentration.

4.2.2 Work and Attack Cost

Attack cost should be assessed by the ability to produce valid consensus work per unit time, accounting for algorithmic optimization, hardware, task switching, propagation, and access to resources. Nominal hash rate, matrix dimensions, or operation counts alone cannot establish a monetary security cost.

The A-hash gate is fixed, while the V-hash target is adjustable. Preparation, first-stage search, bridging, and second-stage search must be measured separately. The bridge result for one qualifying A-hash candidate can be reused across multiple nonce_v trials; it must not be charged again for every V-hash attempt. Theoretical admission probability is not measured throughput. Block cadence also depends on chain difficulty policy and the actual distribution of resources.

4.2.3 Network Threat Model

The threat model covers work bypasses, precomputation, replay, extra search freedom, numerical divergence, verification-resource exhaustion, and network isolation. The table identifies areas for analysis. Mitigations are security requirements, not evidence that the risks have been eliminated.

Threat	Checks and mitigations	Residual boundary
Precomputation and replay	Canonical header construction, candidate binding, historical-version rules	Reusable structures still need separate analysis
Computational shortcuts	Analyze equivalent simplifications, pipelining, and hardware advantages	Undiscovered optimizations cannot be ruled out
Numerical divergence	Consistent rounding points, widths, byte order, and activation positions	Common-mode client defects

Threat	Checks and mitigations	Residual boundary
Invalid candidates and verification DoS	Check encodings and gates first; bound concurrency and resources	Actual worst-case cost requires measurement
Majority work and network isolation	Compare total difficulty; use confirmation policies and diverse peers	Sustained majority attacks and long partitions

Risks can compound. Network isolation, for example, may extend the time spent searching on stale work while inconsistent rounding rules across clients may cause forks. Verification tests should cover combined conditions and require identical judgments for identical inputs across implementations.

4.3 How TensynHash Computes Work

4.3.1 Overall Structure and Numerical Rules

TensynHash comprises A-hash, T-bridge, and V-hash. T-bridge uses a 32×32 memory state, fixed convolution, QM5 feedback, and KDA-style updates to bind the first-stage result to the second-stage input.

Matrices use row-major layout, and concatenation uses fixed-length raw bytes. INT8 encodes initial data and parameters; memory state uses INT32 Q8; the retrieval output z uses INT32 Q18. T-bridge uses no floating point and does not compress updated state back into INT8.

R_s(x) denotes rounding x divided by 2 to the power s to the nearest integer, with ties to even. It differs from truncating division or directly right-shifting a negative number. The specified integer operation order, activation positions, and encodings are consensus semantics.

4.3.2 A-hash Admission Gate

A-hash is based on BLAKE3 and produces a 32-byte digest H_a for the first-stage admission check.

T_a is committed by chain rules and is 0x0000007fffffffff on mainnet. It does not change with mining difficulty or pool-share difficulty; changing it requires an upgrade. Under a uniform 64-bit output model, admission probability is 2^-25, or one successful nonce_a in 2^25 (33,554,432) trials on average. This estimates only the first-stage gate. It is neither the total cost of producing a valid block nor a measured throughput or security result.

4.3.3 Task Preparation and LeakyReLU

Preparation derives a 32×32 initial INT8 matrix S0 from the seed, with elements from -64 to 63. It applies convolution with a fixed 3×3 kernel, stride one, and one-element zero padding; multiplies by 256 to encode Q8; and then applies elementwise LeakyReLU with a negative slope of 1/8.

```
W = [[1,2,1], [2,4,2], [1,2,1]]
C_raw[r,c] = 256 * SUM(W[u,v] * S0[r+u-1,c+v-1])
C[r,c] = C_raw[r,c]              if C_raw[r,c] >= 0
C[r,c] = R_3(C_raw[r,c])         if C_raw[r,c] < 0
```

The preparation path yields C_raw in [-262144, 258048] and activated C in [-32768, 258048]. C is a read-only cache reusable for the same task. Candidate reads do not reactivate it, and candidate updates do not write back to C.

Convolution is prepared before the A-hash candidate loop. It should not be characterized as large-scale AI inference repeated for every first-stage trial. Cache reuse must be bound to the seed and the complete work context.

4.3.4 Candidate Parameters and QM5 Feedback

Once the first-stage gate passes, the algorithm derives query, key, gating, and sampling bytes in separate domains from ahead, nonce_a, and the original H_a. Query binds only ahead and nonce_a; key and gating also bind H_a. Gating derivation returns 49 bytes, of which the last 16 form dice.

QM5 initializes the amplitudes of a five-qubit circuit from the 32 integers in k_code and applies H, CNOT, CZ, and Toffoli gates in the specified order. Classical hardware simulates the circuit exactly with integer operations. The common normalization factor from four Hadamard gates cancels in probability ratios.

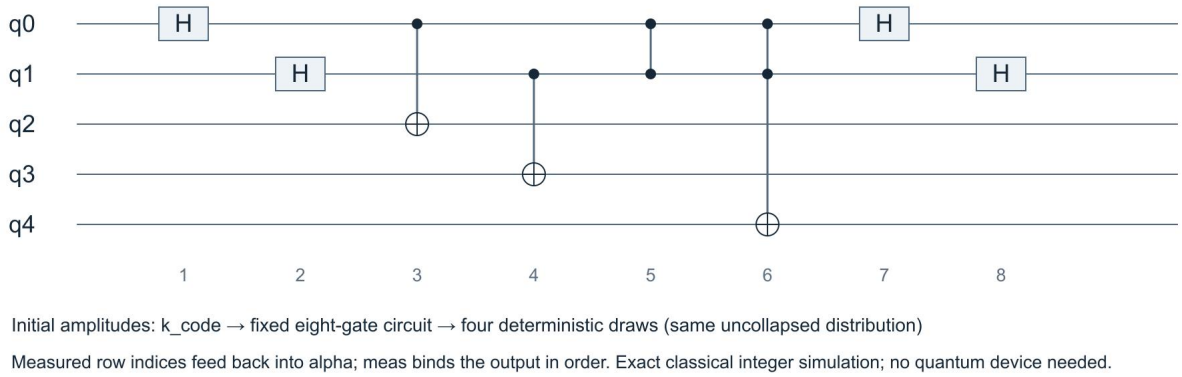


Figure 6 QM5 Five-Qubit Circuit and Deterministic Feedback

The squared output amplitudes are sampling weights. Dice is divided into four little-endian UINT32 values, each mapped to the same cumulative distribution. Sampling does not collapse the distribution, so row indices may repeat. If the total weight is zero, all four draws select row 0.

The alpha_code values for the four selected rows are set to 127, corresponding to a retention factor of 255/256. The row indices form the four-byte meas in order. They affect later gated updates and enter the final binding message directly. Sampling is fully determined by candidate inputs and uses no external quantum-random source.

4.3.5 Gated Memory Update and Retrieval

T-bridge uses KDA-style channel retention, prediction error, and a key-directed rank-one update. The logical values of query and key equal their signed encodings divided by 1,024; alpha lies in [1/2, 255/256] and beta in [1/256, 1]. These hash-derived values are neither trained projections nor the L2 normalization of the original model.

```
alpha_num[r] = INT32(alpha_code[r]) + 128
beta_num = INT32(beta_code) + 129
G[r,c] = R_8(alpha_num[r] * C[r,c])
prediction[c] = R_10(SUM(k_code[r] * G[r,c], r=0..31))
delta[c] = (INT32(v_code[c])+128)*256 - prediction[c]
u_raw[c] = R_8(beta_num * delta[c])
u[c] = min(32768, max(-32768, u_raw[c]))
S1[r,c] = G[r,c] + R_10(k_code[r] * u[c])
z[c] = SUM(q_code[r] * S1[r,c], r=0..31)
```

Here, v_code[c] = H_a[c] − 128 for c = 0..31.

QM5 feeds back into alpha before this update runs. HardTanh is applied after beta gating and Q8 rounding but before the key outer product. Its threshold of 32,768 corresponds to the Q8 logical value 128, and boundary values are preserved. Delta must not be clipped early, nor may activation be moved after the outer product.

S1 remains INT32 Q8; z is an INT32 Q18 integer numerator. No further activation or right shift is applied after retrieval. Implementations must preserve every rounding point. Real-number algebra can aid explanation but cannot replace the specified integer calculation.

4.3.6 Output Fusion and V-hash

The main-path retrieval result z is bound together with the original H_a retained on a side path, as well as ahead, nonce_a, and meas. The design draws on residual preservation of original information, but its actual operation is byte concatenation followed by BLAKE3 XOF, not elementwise tensor addition.

```
D_out = ASCII("ATVHASH-KDA-I8-Q8-QM-ACT2-OUT-V1")
M = D_out || ahead[40] || nonce_a[8] || H_a[32]
   || LE32_SIGNED(z[0..31])[128] || meas[4]
Buffer = B3(M, 42)
head_v = Buffer[0:40]
N_v = Buffer[40:42]
```

D_out is 32 bytes; the full message M is 244 bytes. z is serialized as little-endian two's-complement INT32 without first dividing by the Q18 scale factor. Nodes construct the message locally; no algorithm-proof fields beyond the two nonces are added.

Byte indices start at zero. The first two bytes of the V-hash nonce_v are fixed to N_v; byte 4 retains only the 0x80 bit, and byte 5 retains only the 0x07 bits. The other prescribed bytes retain candidate values, leaving 36 free search bits. Verifiers check the submitted value and must not mask it into validity.

V-hash is the second-stage hash. It takes head_v and nonce_v as inputs and produces a 32-byte digest H_v. Its target T_v adjusts with chain difficulty.

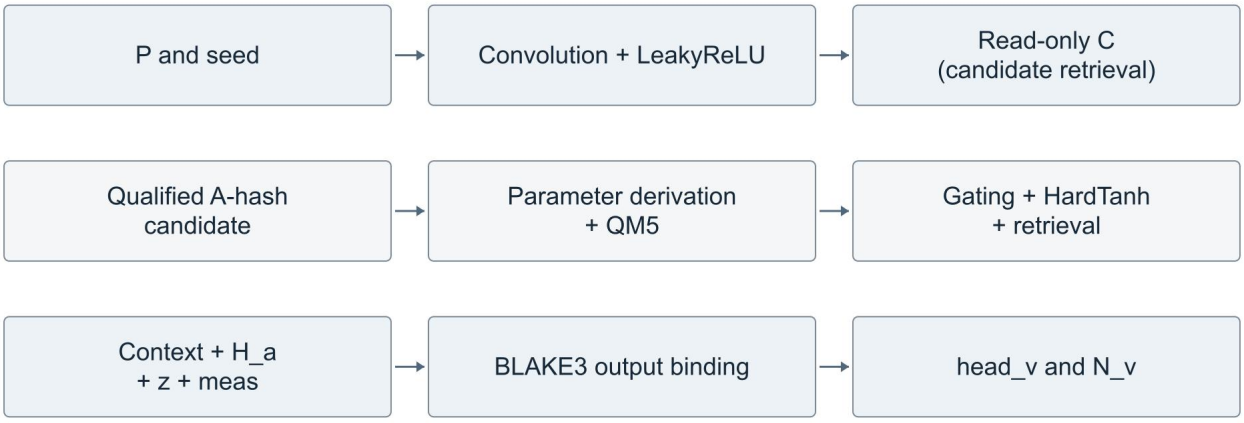


Figure 7 Stage Dependencies in TensynHash

4.3.7 Targets and Difficulty Policy

Valid work must pass both the first-stage gate and final target and satisfy all integer and nonce rules. T_a is fixed. T_v is a 32-byte big-endian target; H_v is interpreted as an unsigned 256-bit little-endian integer. Comparisons are inclusive.

```
a_pass = LE64(H_a[24:32]) <= T_a
v_pass = LE256(H_v[0:32]) <= BE256(T_v[0:32])
ValidWork = a_pass AND arithmetic_valid AND nonce_valid AND v_pass
```

A smaller T_v means higher mining difficulty. The two targets are verification inputs, not part of hash input or the T-bridge message; changing a target with computation inputs fixed does not change the digest. A zero target does not disable verification.

Target block time, difficulty-adjustment window, minimum difficulty, and total-difficulty accumulation are chain rules; none follows from the number of algorithm stages or the name Dyadic. Fork weights must follow the deterministic method prescribed by chain rules, not miner-reported AI scores or the size of measurement results.

4.4 Network Participation and Protocol Evolution

4.4.1 Network Participation

Miners produce work and full nodes verify it independently. One operator can run both without gaining extra authority over consensus. Pools may coordinate search, but nodes still check every candidate against the complete rules.

The TensynHash 2.9.2 Algorithm Specification defines header construction, parameter domains, computation rules, target interpretation, and test vectors used by nodes. Software composition and optimization strategies may vary, but specified results must be preserved. Input integrity and historical-version selection are checked under chain rules, not accepted on the assertion of a central service.

4.4.2 Network Incentives

Consensus rewards correspond to valid blocks on the canonical chain and are determined by monetary and fee rules. Work digests, gating parameters, and quantum-measurement row indices do not create balances or independent rewards.

Block rewards and application compute-service fees are accounted for separately. A failed application order cannot change a block reward confirmed under consensus rules. Blocks outside the canonical chain receive no block reward. A block included as an uncle by a canonical-chain block under the protocol rules may receive an uncle reward; the including block receives a small additional reward. Uncles do not contribute to total difficulty. Issuance and fee allocation follow the applicable rules.

4.4.3 Long-Term Security and Evolution

TensynHash domains, gate order, activation constants, rounding, and encoding need unified version management. An upgrade that changes the result must specify activation conditions and historical verification methods; it cannot take effect through an unagreed hot change of runtime parameters.

Algorithm evolution requires cross-checking specifications, reference implementations, independent vectors, malicious-input tests, and security analysis. Monitoring can reveal unusual work distribution, verification times, and reorganizations, but cannot bypass verification or change consensus rules automatically.

5. Deterministic Computation and Work Verification

5.1 From Search to Independent Verification

5.1.1 What a Work Proof Contains

A work proof contains nonce_a and nonce_v, totaling 16 bytes. A node recomputes the candidate using PoW header P and the targets. The block and transactions remain in chain data; intermediate states in debugging output receive no special trust.

Node recomputation grounds consensus judgments in inputs and rules rather than miner identity or equipment claims. The proof attests to a fixed work function; it is not a general execution credential for arbitrary AI computation.

5.1.2 Search Cost and Verification Cost

Work production repeatedly searches candidates. Verification recomputes the prescribed path for one submitted candidate and checks its targets. This cost difference reflects repeated search versus a single check; nodes still perform all computation required for that candidate.

Preparation, first-stage trials, T-bridge, second-stage trials, propagation, and verification should be measured separately. Actual cost depends on strategy, targets, implementation, and hardware. Data types or matrix dimensions cannot stand in for benchmarks.

5.1.3 The Network Value of Verifiability

The same inputs, version, and chain rules should yield the same digests and judgments, allowing every node to verify the same conclusion independently. Each application protocol defines acceptance criteria for its results and the treatment of external data truthfulness.

5.2 Work Objects and Context Binding

5.2.1 Input and Output Objects

Verification constructs P from chain context, receives two nonces, uses chain-prescribed T_v and fixed T_a, and locally reconstructs C, parameters, QM5 results, S1, z, Buffer, and both hash-stage digests.

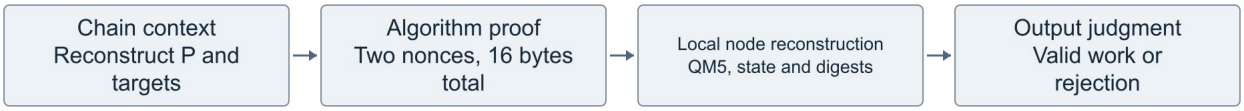


Figure 8 Work-Verification Inputs and Local Reconstruction

The protocol selects the algorithm version, while chain rules govern block packaging and timestamp validity. An interface may accept a byte sequence without making every such sequence a valid block header.

5.2.2 Domain Separation and Binding of Raw Digests

Initialization, query, key, gating, and final output use distinct domain tags. Fixed lengths, explicit encodings, and raw-byte concatenation define how objects are interpreted and prevent the same data from being reused in different positions.

The final message retains H_a, ahead, nonce_a, z, and meas in full. Even if quantization maps different intermediate results to the same z, the original H_a still enters the later binding. This design does not claim that the matrix path is injective or that the entire PoW has no collisions or shortcuts.

5.2.3 Context and Cache Lifecycle

Read-only C may be reused when its conditions hold, but every KDA candidate update starts from C. Query, key, gating, and dice depend on the current candidate; results from another nonce cannot be spliced in.

When the header, timestamp, nonce, or version changes, affected values must be recomputed. After a reorganization, the new parent-branch context must be checked. Reusing an array does not mean reusing a validity conclusion.

5.3 Fixed-Point Semantics and Cross-Implementation Consistency

5.3.1 Encoding and Numerical Representation

S0, convolution kernels, and parameter encodings use INT8. C, G, S1, and error-correction intermediates use INT32 Q8; z uses INT32 Q18. Hash and transport data use unsigned bytes. Array lengths exclude padding in language-level structs.

Q8 and Q18 denote implicit scaling, not floating-point storage formats. z_bytes encodes each of 32 signed integers as little-endian two's complement, totaling 128 bytes. The encoding of negative one is ff ff ff ff; a floating-point bit pattern is not permitted.

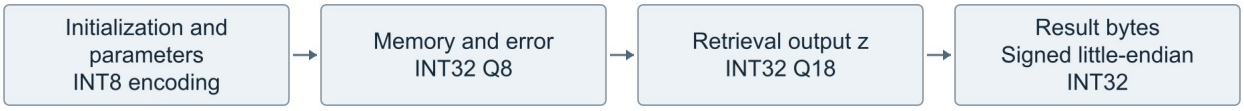


Figure 9 Numerical Representation in T-bridge

5.3.2 Rounding and Integer Bounds

R_s rounds to the nearest integer with ties to even. For example, R_1(1)=0, R_1(3)=2, R_1(-1)=0, and R_1(-3)=-2. Implementations must handle INT32_MIN safely and cannot substitute signed negation or left shift that may overflow.

The specification bounds products, accumulations, and partial sums within the appropriate integer ranges. Gating offsets are widened to INT32 before addition; beta_num can reach 256 and must not be narrowed back to INT8. T-bridge uses neither modular wraparound nor implicit saturation, only the explicitly specified HardTanh clipping. QM5 sampling-ratio multiplication uses UINT64.

5.3.3 Activation and Execution Order

LeakyReLU runs once during task preparation with a fixed negative slope of 1/8; reading a cached candidate does not reactivate it. HardTanh applies only to the error vector u_raw after beta gating and rounding, before the key outer product.

Optimizations must preserve these positions and elementwise rounding results. Algebraically simplifying real-number formulas, fusing operations, or moving clipping to the updated state may change consensus output and is not an equivalent implementation.

5.3.4 Known-Answer and Failure Tests

Known-answer tests record H_a, z, meas, Buffer, and H_v for given inputs so implementations can locate divergence at each layer. Normal vectors should be paired with negative and boundary tests; one matching final digest does not replace comprehensive validation coverage.

Specification example A passes the fixed A-hash gate and exercises the later path; its final validity still depends on comparison with the supplied T_v. Example B fails at the first stage. Example C uses a relaxed

test target to bypass the default gate solely for regression testing; it does not represent valid work submittable under default rules.

5.4 Node Verification and Network Confirmation

5.4.1 Verification Pipeline

A node first checks chain-data encoding, version, and context, then generates preparation state, computes A-hash, and checks the fixed gate. After admission, it derives parameters, runs QM5 feedback, updates and queries memory, constructs the second-stage header and nonce prefix, checks the submitted nonce_v, and then computes V-hash.

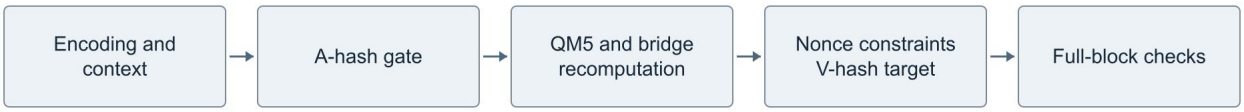


Figure 10 Node Work Verification and Block Confirmation

A failure of any arithmetic, encoding, context, nonce, or target condition causes rejection. Even after TensynHash succeeds, the block must pass chain-level difficulty policy, transaction execution, state-root checks, and other tests. An algorithm-level accept response does not accept the block as a whole.

5.4.2 Failure Classes and Resource Limits

A candidate that fails the A-hash gate never enters T-bridge; one that fails the nonce_v constraint never enters V-hash. If the final target fails, V-hash has been computed but the candidate is still invalid. Failure classes should be stable for cross-implementation and operational diagnosis.

Length, version, and other cheap checks should run first. Verification concurrency and message resources must be bounded. A verifier must not tolerate invalid input by replacing targets, correcting nonces, ignoring overflow, or trusting remote results.

5.4.3 Confirmation and Reorganization

Work verification produces a deterministic validity judgment, but confirmation after inclusion in the canonical chain remains probabilistic under PoW. Confirmation depth cannot eliminate sustained majority-work attacks; transactions and applications need appropriate risk policies.

Reorganizations roll back transactions, receipts, and application events with the canonical chain. Nodes replay under the appropriate historical algorithm rules; indexers and off-chain services restore state by block hash and keep task consumption, delivery, and payment idempotent.

5.5 The Extent of Verification

TensynHash's independent recomputation applies to fixed consensus work. Applications may register requests through the EVM, execute off-chain, and choose recomputation, redundant execution, statistical criteria, or authorized review. The trust assumptions and costs of each acceptance method should be disclosed.

An application task suitable for independent recomputation should fix its input, program version, precision, randomness, output encoding, and resource limits, and choose an acceptance policy before taking the order. Sampling-based or error-tolerant services also need statistical or numerical criteria, repeat trials, and dispute rules.

Open computing should begin with tasks whose boundaries are clear and costs measurable. Specifications and test vectors support reproducibility; data availability supports review; authorization and settlement rules handle failure. Bringing a new task into consensus requires renewed analysis of determinism, work distribution, cost, and availability.

Application extensions must preserve base-layer verification standards. External data, subjective quality, and real-world facts still require appropriate accountability; reproducible records and business correctness must be judged separately.

6. An AI and Quantum-Friendly Computing Network

6.1 An AI Computing Network

An AI computing network needs to connect sustained demand with resources at different scales and establish common interfaces for execution, acceptance, and settlement. Open participation allows more providers to contribute under clear conditions while helping demand-side users understand the service they will receive.

Atveno defines a deterministic work function at the consensus layer and leaves room at the application layer for services built around real demand. AI-related consensus operations use hash-derived parameters; memory updates do not accumulate knowledge across candidates. Individual applications define user training and inference separately.

6.2 AI Inference and Application-Level Services

The application-layer services in this section are planned and have not yet been implemented.

6.2.1 Bringing Inference Demand into the Network

An application-level inference request should specify the model or program version, input commitment, execution configuration, result requirements, acceptance method, budget, and validity period. Large inputs may use off-chain references, while contracts record authorization and status.

These are application-protocol objects. A missing, canceled, or failed order does not affect the base chain's continuing search for candidates under its own context.

6.2.2 Standardized Execution and Resource Matching

Providers may quote for scheduling by model, operator, precision, and capacity, with clear lease, cancellation, timeout, and retry terms. Deterministic tasks also need defined preprocessing, random seeds, numerical semantics, and result-equivalence rules.

Tasks that cannot promise bitwise consistency need error tolerances or statistical criteria, with acceptance costs included in service design. Provider identity and capability claims can guide resource selection but cannot by themselves prove a result correct.

6.2.3 Inference Results and Application Acceptance

Applications check results through pre-agreed recomputation, redundant execution, or authorized review, subject to retrieval, access, and payment conditions. A result hash binds content; acceptance policy judges model execution and output quality.

Failures, timeouts, and disputes follow their respective branches, and the settlement entry point prevents double consumption. The base chain records these states; the service protocol defines responsibility for delivery.

6.3 AI Computational Structures in Consensus Work

6.3.1 Convolution and Activation

Fixed 3×3 convolution and LeakyReLU transform a seed-derived matrix into read-only memory C. They use deterministic integer semantics and untrained weights. Preparation cost should be measured separately from candidate updates and second-stage search.

6.3.2 Gated Memory and Matrix Computation

The KDA-style update controls memory with retention factors, forms a prediction through key, derives an error from H_a, and writes a clipped correction in a directed manner before query retrieves the result. This ties memory transformation to the current candidate as recomputable protocol computation.

This structural borrowing does not run the full Kimi Linear model or inherit its training results or inference performance. Parameters, scaling, and execution semantics follow the TensynHash specification.

6.3.3 Parameter Derivation and the Meaning of Work

Hash-derived parameters place AI-related structures in each candidate's data dependencies. Fixed activation and rounding positions preserve consistency across implementations. Actual hardware efficiency, resistance to shortcuts, and business value still require separate performance, security, and application evidence.

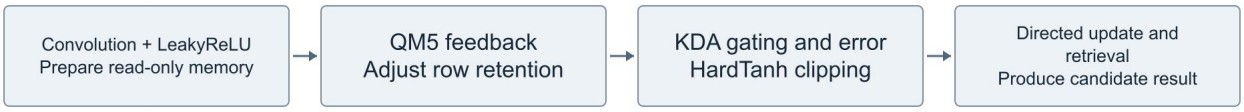


Figure 11 How AI Computational Structures Connect to QM5

6.4 Quantum-Friendly Design and the QM5 Module

6.4.1 Representation of the Five-Qubit Circuit

QM5 represents the five-qubit basis states with 32 amplitudes corresponding to KDA row indices, with q0 as the least significant bit. The fixed circuit is initialized from k_code and recomputed in full on classical devices using exact integer ratios.

6.4.2 Deterministic Sampling and Feedback

Four candidate-derived UINT32 values map to the same uncollapsed cumulative distribution, producing measured row indices whose order and repetitions are retained. Zero weight has a fixed output. The selected rows receive a retention factor of 255/256, and meas also enters the final binding.

This feedback links the circuit representation to memory updates and second-stage search, giving the quantum-friendly design a defined algorithmic role. Chapter 4 and Appendix B specify gate order and sampling rules.

6.4.3 Limits of the Quantum-Friendly Design

Quantum-friendly here means circuit representation, simulation, and deterministic feedback. It does not imply quantum acceleration, a mining advantage from quantum hardware, or general-purpose quantum services.

Post-quantum cryptography addresses separate mechanisms such as signatures and key establishment. QM5 does not automatically replace account signatures or guarantee that the whole chain withstands quantum attacks. Chapter 10 discusses the security boundary.

6.5 Application-Oriented Computing Extensions

The application-layer services in this section are planned and have not yet been implemented.

6.5.1 Distributed and Heterogeneous Computing

Applications may assign tasks to different resources, but must specify shard dependencies, versions, numerical semantics, and result merging. Reassignment needs to handle timeouts, missing shards, and duplicate delivery while recording input ranges and shard positions.

Distributed execution belongs to application protocols. It neither requires consensus miners to train models together nor changes independence between candidates. Task-specific acceptance rules determine whether different devices meet the required result consistency.

6.5.2 Constrained Hybrid-Computing Tasks

Potential directions include parameterized-circuit simulation, deterministic numerical computation, and batch experiment records. Each service needs a defined template, input range, execution interface, and acceptance criteria, with full-cost comparison against local or cloud alternatives.

If physical quantum devices are used in applications, device records and statistical acceptance may be necessary. Their random outputs cannot replace QM5's deterministic consensus path. Integration depends on the relevant specifications and verifiable capabilities.

6.5.3 AI Agents and Machine-to-Machine Demand

An agent may discover tasks, prepare requests, and orchestrate workflows. Authorization follows wallet signatures, budgets, and permission policies. Machine-to-machine settlement also needs replay protection, rate limits, revocation, and dispute handling.

External tools, real-world data, and subjective assessments retain their own trust boundaries. Automated workflows must respect account authorization and consensus rules.

6.5.4 AI-Native Applications

Developers can build around task registration, result commitments, reproducible acceptance, and constrained automated services. On-chain records connect off-chain computation; applications themselves provide scheduling, acceptance, and delivery.

Each service's specification and implementation define its capabilities. Clear interfaces, auditable fees, and manageable failures are common requirements for extending applications while preserving independent verification by base-layer nodes.

7. How the Atveno Network Operates

Network operation has two paths: consensus creates work from block context, while users or applications request computation. They share on-chain coordination but follow distinct verification and reward rules.

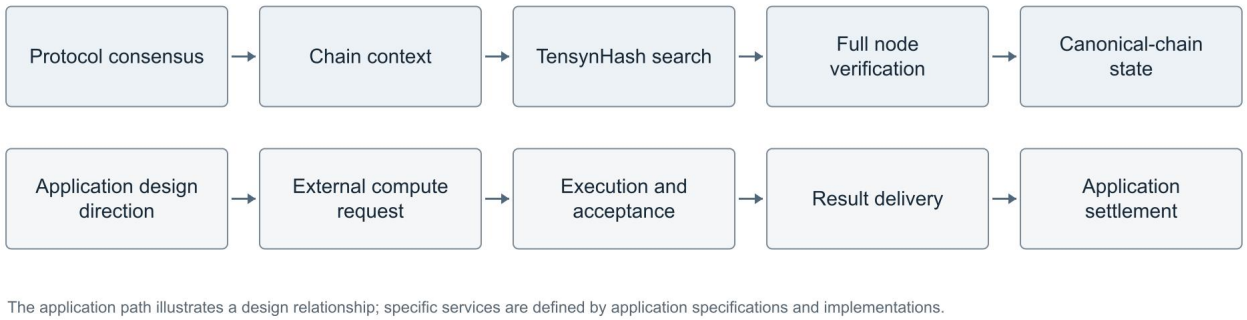


Figure 12 The Network's Two Operating Paths

7.1 From Candidate Block to Work Task

Nodes determine the parent block, transactions, and header under chain rules, zero specified fields during serialization, and construct the complete 40-byte P through TensynHash. The work object also contains fixed T_a, current T_v, and the active version.

Miners obtain work continuously from chain context; application orders are not an external prerequisite. Targets, timestamps, and task switches must match the current branch. Test parameters or work from another branch cannot be submitted as valid candidates.

Miners may search independently or receive work objects from pools. Access services pass tasks and collect candidates, but miners and verifiers still check inputs and configuration separately; server assertions cannot replace node judgment.

7.2 Computation and Candidate Production

7.2.1 Task Preparation

Miners generate read-only C through convolution and activation and reuse it when conditions permit. Each candidate update starts independently from C. Changes in work context require affected values to be recomputed so old conclusions do not carry into new tasks.

7.2.2 Two-Stage Search

The miner first searches nonce_a. After passing the fixed first-stage gate, it derives parameters, runs QM5 feedback, performs gated updates, HardTanh, and retrieval, and binds the original H_a to obtain the V-hash context and nonce prefix.

One head_v admits at most 2^36 compliant nonce_v candidates. Search must retain the prefix and bit constraints. When nonce_a or work context changes, affected data must be recomputed; invalid nonces cannot be repaired.

7.2.3 Candidate Submission

Miners submit both nonces and the block data required by chain rules. The verifier reconstructs intermediate states and matrices. After a task expires or the parent block or configuration changes, computation must use the new context.

7.3 Work Verification and Block Confirmation

7.3.1 Local Reconstruction

A full node reconstructs P and C from the candidate and chain rules, recomputes H_a, and checks the fixed gate. It rejects a failure immediately and enters T-bridge only after a pass.

7.3.2 Algorithm Verification

The node executes QM5, gated update, and retrieval as specified, produces S1, z, meas, and Buffer, checks nonce_v constraints, computes H_v, and compares it with T_v. Every stage uses the prescribed integer and rounding rules.

7.3.3 Chain-Level Checks and Confirmation

Once the algorithm is valid, the node checks block structure, chain-level difficulty policy, timestamp, transactions, and state root before total-difficulty accumulation and fork choice. A deterministic judgment on one work proof does not remove probabilistic confirmation or reorganization risk.

7.4 Application Requests and Result Delivery

7.4.1 Requests and On-Chain Records

An application first defines an authorized task with a fixed program or model, input commitment, execution version, acceptance policy, budget, and deadline. A contract retains necessary state while agreed storage holds large data. A unique task ID and nonce prevent duplicate requests.

Task state advances under call permissions and preconditions and is restored from canonical-chain events after a reorganization.

7.4.2 Execution, Acceptance, and Delivery

An executor validates inputs and computes, then submits a normalized result and commitment. An acceptor checks it against the policy agreed before the order was taken. The requester retrieves the result and checks its content. A matching hash binds content but cannot replace acceptance of the computation.

Failure, timeout, or unavailable data follows the relevant branch. Applications define recomputation, error-tolerance, and statistical acceptance; they cannot directly reuse a validity judgment for the fixed consensus work function.

7.4.3 Fees and Settlement

Fees are released according to acceptance and delivery conditions. Execution, review, storage, and failed retries are metered separately. Task rules set budgets, allocations, and refunds. Payments need protection against double consumption and recovery after reorganizations or revocation.

7.5 The Complete Network Loop

From chain context through candidate search and work verification to transaction execution and chain selection, consensus forms an independent loop. Every node should be able to verify work and state from the same data. Application errors must not block work verification, synchronization, or block production.

Applications proceed through request, execution, result submission, acceptance, delivery, and settlement. They define authorization, timeouts, and failure handling at each step and dispose of budgets and temporary data at terminal states. The diagram shows an application-protocol design; each application supplies its own components.

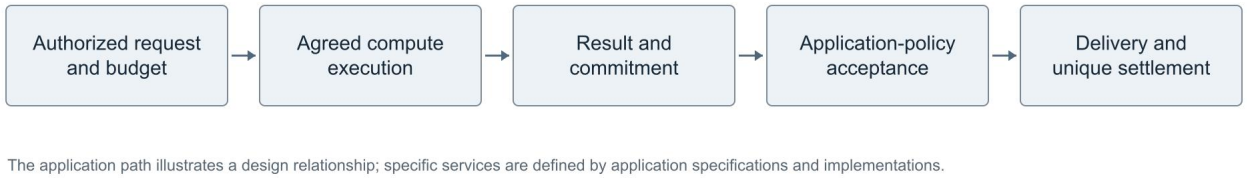


Figure 13 The Application Computing Service Loop

Consensus rewards support block production; gas pays for on-chain execution and records; application service fees pay for agreed computation and delivery. Each flow is accounted for under its own rules. Long-term value depends on reliable delivery, real demand, and sustained use.

8. Network Participation and an Open Ecosystem

8.1 Network Participants and Roles

8.1.1 Users and Requesters

Users submit transactions and call contracts through wallets. Compute requesters also specify inputs, result criteria, access terms, and budgets. Signatures and nonces constrain authorization; service specifications determine the acceptable task scope.

8.1.2 Miners and Compute Resource Participants

Miners participate in block production by searching with TensynHash. Application compute providers deliver services by completing orders. One operator can take both roles, but each has distinct work entry points, acceptance criteria, and revenue rules.

Device support depends on actual software and implementations. The algorithm's data types do not guarantee equal efficiency across devices. Participants should assess costs against compatibility, hardware, and energy use.

8.1.3 Nodes and Verification Participants

Full nodes can independently synchronize, store data, execute transactions, and verify work without mining. Verification is therefore a distinct form of participation, and its consensus judgment does not depend on remote proofs or application acceptors.

Application acceptors review services under order policies. That role carries no base-chain consensus privilege, and nodes do not audit every application's business outcome.

8.1.4 Developers and Application Participants

Within the London-era boundary, developers can use contracts, dApps, indexers, and off-chain services to build applications linking task registration, payment, and state coordination. Compute services must define their own execution, acceptance, and delivery interfaces.



Figure 14 Network Participants and Their Responsibilities

Tool integrations use actual network parameters and queryable versions, handling duplicate events, reorganizations, and unavailable data. Public interfaces allow parties to implement components separately; application protocols set the terms of collaboration.

8.2 Participation Relationships in the Network

Requesters set delivery goals and budgets, resource providers offer execution capacity, acceptors check results under agreed criteria, and developers express these relationships as application rules. Clear roles help each party identify contributions, costs, and responsibilities.

In consensus, miners produce work and nodes judge it independently. In applications, providers execute orders and the corresponding mechanisms accept services. Each relationship has its own correctness standard, with records and coordination supplied by the base chain.

8.3 EVM Compatibility and the Developer Environment

8.3.1 EVM Compatibility

Atveno targets London-era execution compatibility, including the corresponding EVM bytecode, protected legacy transactions, EIP-2930, EIP-1559, and controlled standard RPC methods. PoDW does not provide protocol-level deterministic finality semantics for safe or finalized.

Wallets and SDKs must use the actual chain ID, network parameters, and confirmation policy. Opcodes or semantics from later execution eras are not automatically within the compatibility scope.

8.3.2 Smart Contract Execution Environment

Contracts handle gas, reverts, logs, and state under London rules. The same parent state and block context should produce the same result. TensynHash is a protocol work path; the design does not assume that arbitrary contracts can call a dedicated precompile or general quantum service.

8.3.3 Developer and Tool Integration

Wallets, SDKs, contract tools, and explorers must check transaction encoding, signatures, nonces, gas, receipts, logs, queries, and errors individually. They must also adapt to Atveno's use of the nonce and mixHash fields. New interfaces need explicit resource limits and versions.

Public RPC methods and resources are bounded, and account signing remains in the wallet. Indexers track the canonical chain by block hash. Unknown extensions and data that affect verification semantics require explicit handling.

8.4 Developers and the Application Ecosystem

8.4.1 Building Applications

Consider reproducible numerical computation. An application can first record a program version, input commitment, result precision, and budget; an off-chain provider executes; an acceptance program recomputes or checks under agreed criteria; fees are released once the result can be retrieved. Requesters can view task state, and providers can assess delivery and payment terms beforehand.

This is an application design example. The relevant specification and implementation determine each service's capabilities and availability. Large objects can use off-chain references; applications need authorization, acceptance, dispute handling, and recovery after event rollback.

8.4.2 AI and Hybrid-Computing Applications

Another direction is an agent service constrained by policy. An agent makes compute requests within wallet authorization and budget; the application records tasks and results and settles under explicit conditions. Execution may be automated, but resource access, acceptance, and payment still follow their respective rules. Constrained inference and parameterized-circuit simulation can be assessed in the same way.

These directions require separate specifications, cost comparisons, and usable implementations. The fixed work function offers experience with deterministic computation; application teams must build scheduling, model services, and resource collaboration separately.

8.4.3 Expanding an Open Ecosystem

Ecosystem growth depends on public interfaces, identifiable versions, reviewable results, and manageable failures. New services should specify inputs and outputs, execution scope, acceptance, resource limits, and fees.

Application upgrades operate within their authorized scope. Changes affecting block validity must follow consensus-upgrade rules.

9. Network Value and Economic Model

9.1 Where Network Value Comes From

Network value comes from secure state coordination and usable services. Intelligent demand gives applications an entry point; nodes, miners, developers, and providers supply foundational capabilities. Whether wider participation creates value depends on actual use and reliable delivery.

9.2 Compute Demand and Network Value Flows

Applications turn compute demand into defined relationships for delivery, payment, and responsibility. The technical architecture explains how those relationships work; separate rules define specific prices and token issuance.

9.2.1 Bringing Demand into the Network

Before an order is accepted, a request specifies the computation, input commitment, version, acceptance criteria, budget, and validity period. Off-chain systems may provide quotes and scheduling. Requesters should understand what will be delivered, when payment is due, and how failure is handled.

9.2.2 Service Value in Computation and Acceptance

Execution, review, storage, and delivery have distinct costs. Each may be priced within a task budget and paid under specified conditions. Application revenue comes from agreed payments; consensus rewards follow block rules. They are accounted for separately.

9.2.3 Network Value Flows

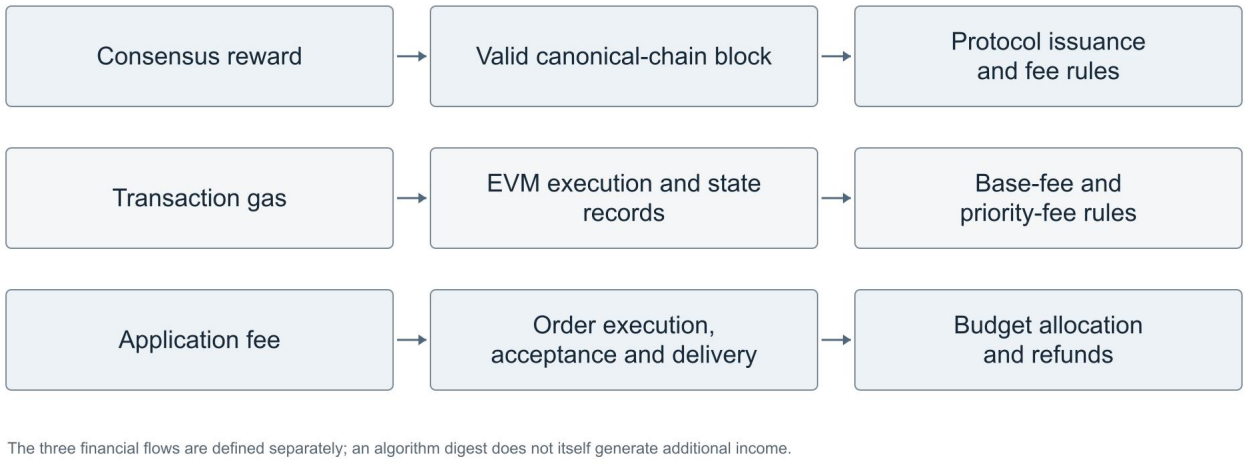


Figure 15 Three Boundaries of Network Value Flow

Gas pays for execution and recording; block rewards are protocol issuance; application fees pay for specific services. An application may reserve a budget in advance, distribute it after acceptance and delivery, and refund or retry under its rules after failure.

Unique task IDs and consumption state prevent duplicate claims. After a reorganization, delivery, payment, and revenue records should recover under the confirmation policy and canonical chain.

9.3 Network Participation and Incentives

9.3.1 PoDW and Network Incentives

PoDW incentives correspond to valid blocks and consensus contributions on the canonical chain. A separate set of Tokenomics *(add URL after publishing)* rules defines the issuance curve, block rewards, maturity, and fee allocation. Algorithm parameters and quantum feedback do not independently generate extra returns.

Incentives should be tied to publicly verifiable state so the same work is not rewarded twice. Participants must still account for equipment, energy, networking, and competition for work. The algorithm's structure cannot imply a fixed return.

9.3.2 Compute Services and Ecosystem Incentives

Real compute demand may generate service revenue. Developers and infrastructure contributors may also receive incentives under specific mechanisms. Fees, allocations, and participation terms should reflect protocol rules, costs, and actual contributions.

9.4 Network Fees and Their Relationship to Value

9.4.1 Network Fees and Revenue

London-era transaction fees follow gas and EIP-1559 rules: the protocol burns the base fee, while the priority fee goes to the recipient address specified in the block header. Failed transactions may still consume gas. Application accounting relies on confirmed canonical-chain receipts.

9.4.2 Application Services and Settlement Demand

Applications may charge for execution, acceptance, storage, and delivery and use protocols to manage escrow, allocations, refunds, and disputes. Service and economic specifications determine units, rates, and payment methods. More tasks may increase settlement use, but do not directly imply a token price or holding return.

9.4.3 The Long-Term Economic Loop

Reliable delivery can attract demand; paying demand supports resource supply; stable supply can improve the service experience. Evaluation of this loop must include execution, review, bandwidth, storage, retry, and coordination costs, with service revenue, protocol fees, and issuance subsidies accounted for separately.

This chapter sets out value and incentive principles. Separate documents define economic parameters; it provides no unverified fee rates, market sizes, or fixed returns.

10. Network Security and Technical Evolution

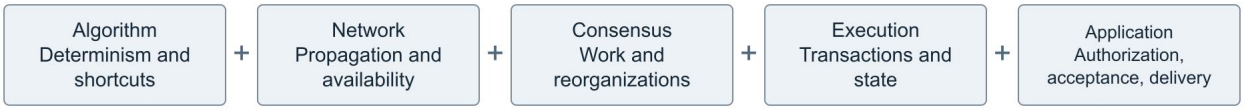


Figure 16 Security Boundaries and Scope of Analysis

10.1 Network Threat Model

10.1.1 Consensus Security

Full nodes independently verify block work and state transitions from public data. PoDW retains PoW's probabilistic confirmation and majority-work attack boundary. AI or quantum-circuit semantics in the work path do not change its form of finality.

Work verification, chain difficulty policy, timestamp checks, transaction execution, and state-root comparison together determine block validity. Historical synchronization and reorganizations must replay under the relevant versions; checkpoints and remote services cannot replace specified verification.

10.1.2 Network Attacks and Resource Consumption

Majority-work attacks, deep reorganizations, censorship, P2P isolation, and message suppression can coincide. Diverse peers, confirmation policies, and anomaly monitoring reduce some risk but cannot ensure normal service during partitions or sustained majority attacks.

Message length, parse depth, verification concurrency, and per-peer rates need limits. A fixed computational path helps estimate resource demands, but worst-case runtime must be measured on clients and devices rather than protected against using averages alone.

10.1.3 System Security Boundaries

Wallets authorize, miners search, full nodes judge consensus, and application execution and acceptance services deliver under order terms. Gateways and operators should not have authority to change validity.

Supply-chain compromise, key leaks, deployment errors, and common-mode client defects still require engineering controls. Reproducible builds, signed releases, least privilege, recovery exercises, and cross-implementation testing can reduce these risks.

10.2 Algorithm Execution and Result Security

10.2.1 Numerical Consistency and Implementation Risk

The main implementation risks are byte-order errors, inconsistent integer interpretation, divergent rounding, misplaced activations, cache misuse, and overflow. Implementations must ensure that the INT32 path in T-bridge does not wrap around.

The LeakyReLU negative slope, HardTanh threshold, gate order, and sampling rules are algorithm constants, not local tuning options. Optimization is permissible only when all specified results are preserved; operations cannot be fused across rounding points.

10.2.2 Work Bypasses and Precomputation

Security analysis must consider caching of fixed convolution, equivalent simplifications of the five-qubit circuit, precomputation, vectorization, pipelining, and reuse across candidates. The presence of matrices or memory state alone does not establish memory hardness; exact simulation of a fixed circuit alone does not establish resistance to specialized hardware.

Analysis should compare the ability to produce valid candidates per unit cost, including work distribution and extra search freedom. Binding the raw digest constrains replacement of intermediate values but cannot substitute for an analysis of shortcuts or security across the full PoW.

10.2.3 Verification Rejection and Version Handling

Nodes do not trust external H_a, z, gating, meas, or cached matrices. They reconstruct the current task and verify nonces and targets under the specification. A value within numerical range is not proof of correct provenance.

Unknown versions, malformed encodings, missing data, invalid nonces, and unmet targets must be rejected, not downgraded to warnings or accepted through remote adjudication. Historical and new rules need clear selection so old blocks are not recomputed incorrectly after an upgrade.

10.3 Quantum-Friendly Design and Cryptographic Boundaries

10.3.1 Security Meaning of Circuit Simulation

QM5 is a classical integer simulation of a fixed five-qubit circuit. Sampling is determined by hash-derived input and provides no new physical randomness. Quantum terminology describes its computational semantics; it does not mean the network must use quantum devices.

All four draws use the same uncollapsed distribution and may repeat row indices. Meas feeds back into gating and enters output binding. These rules must be reproduced exactly; random measurements from physical hardware cannot replace them directly without risking divergent node results.

10.3.2 Quantum Advantage and Post-Quantum Security

Quantum advantage requires comparing quantum and classical methods for a particular problem under specified correctness and cost conditions. Classical simulation of a quantum circuit does not establish such an advantage. Integrating physical quantum resources also requires reconciling noise and statistical acceptance with deterministic consensus.

Post-quantum cryptography concerns mechanisms against quantum attacks, including digital signatures and key establishment. This quantum-friendly design does not claim that account signatures have migrated or that the whole chain resists every quantum attack.

10.3.3 Hash Binding and Security Assumptions

Binding the raw digest in the final message is analyzed under BLAKE3's collision-resistance assumption. The algorithm specification reduces collisions in head_v to collisions in BLAKE3 output, using a 128-bit classical collision-security target for that layer.

This conditional analysis does not prove the security strength of PoDW as a whole or make a post-quantum claim. A 42-byte XOF output, INT8 naming, activations, and measurement feedback do not independently raise the claimed security level. Final-difficulty distribution and resistance to computational shortcuts need separate research.

10.4 Network Evolution and Long-Term Vision

10.4.1 Specifications and the Verification Path

The evolution of consensus capability requires mutually supporting specifications, reference implementations, independent test vectors, security analysis, and operational validation before entering the protocol through explicit versions and activation rules. This technical dependency order is neither a publication date nor a commercial commitment.

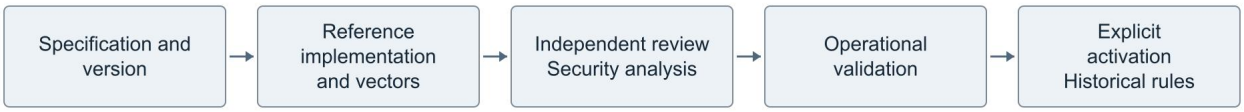


Figure 17 Verification Dependencies for Protocol Evolution

An upgrade should state which inputs, constants, or validity conditions it changes and validate historical replay, synchronization, and reorganizations. When defects arise, response rules must be consistent and auditable; one service cannot temporarily replace node judgment.

10.4.2 Network and Application Expansion

Chain capacity is constrained by gas, block cadence, propagation, and storage, while work verification adds computation for every full node. Scaling budgets must cover worst-case resource use and be assessed separately from application execution and data-delivery costs.

Applications may add templates and services without changing the work function. Changes involving new consensus computation need fresh analysis of work distribution, security assumptions, and verification cost; they cannot be activated implicitly by launching a service.

10.4.3 Long-Term Development of an Open Computing Network

Atveno aims to connect more developers, resource providers, and users under common rules so reproducible computation, reliable acceptance, and value coordination reinforce one another. Research in AI and quantum computing offers directions for application expansion. Long-term evolution keeps consensus stable,

applications extensible, and responsibilities clear; network value comes from sustained use and collaboration under shared rules.

Appendix A Core Objects and Operating Relationships

Object	Definition and role	Layer
PoDW	Proof of Dyadic Work, the consensus mechanism; Dyadic denotes the A-hash and V-hash levels	Consensus
TensynHash	Deterministic work algorithm used by PoDW	Work function
AI Work	Protocol-specified convolution, activation, gated updates, retrieval, and related computation	Consensus-work subprocess
PoW header P	40-byte input derived from the canonical block header and block timestamp	Work context
Algorithm proof	nonce_a and nonce_v, eight bytes each	Work submission
T-bridge	Deterministic computational layer linking A-hash and V-hash	Work function
QM5	Classical integer simulation and feedback from a fixed five-qubit circuit	Within T-bridge
C / S1 / z	Read-only memory, candidate-updated state, and retrieval result	Locally reconstructed data
Application task	User-agreed object of execution, acceptance, and delivery	Application
Verification	Work verification or application acceptance under the applicable rules	Depends on the object

Path	Source and process	Result and boundary
Consensus path	Block context → search → work recomputation → full-block verification	Block validity and chain selection
Application path	User request → execution → agreed acceptance → delivery and settlement	Application state; does not automatically add work
Quantum-friendly path	key and dice → fixed-circuit simulation → gated feedback and output binding	Deterministic classical execution

Appendix B Key TensynHash Specification Summary

B.1 Names and Protocol Constants

TensynHash is the algorithm name; PoDW is the consensus mechanism. The algorithm identifier is tensynhash-v2.9.2, and the profile identifier is TENSYNHASH-TBRIDGE-INT8-Q8-QM5-ACT2-V1. The domain tags in the table below are internal byte constants and remain unchanged.

Purpose	Specified ASCII domain tag	Bytes
PoW header	ATVHASH	7
Memory initialization	ATVHASH-MEM32-I8-INIT-V1	24
query	ATVHASH-KDA-I8-QUERY-V1	23
key	ATVHASH-KDA-I8-KEY-V1	21
Gating and dice	ATVHASH-KDA-I8-GATE-V1	22
Output binding	ATVHASH-KDA-I8-Q8-QM-ACT2-OUT-V1	32

All domain tags are literal ASCII without a trailing NUL. D_OUT includes the QM and ACT2 identifiers; it must not be replaced by an older output domain or omit measured row indices.

B.2 Exact Execution Rules for QM5

Basis states are numbered 0 through 31 and correspond to KDA row indices, with q0 as the least significant bit. Initial amplitudes are psi[i] = INT32(k_code[i]). The following eight gates are applied in order:

```
H(0); H(1); CNOT(0 -> 2); CNOT(1 -> 3);
CZ(0,1); TOFFOLI(0,1 -> 4); H(0); H(1)
```

For Hadamard H(t), each index i whose bit t is zero is paired with j = i OR (1 << t), transforming amplitudes (a,b) to (a+b,a-b), once per pair. CNOT swaps target-bit pairs when its control bit is one; CZ negates amplitudes when both bits are one; Toffoli swaps target-bit pairs when both control bits are one.

After omitting the common normalization of four Hadamard gates, the total squared amplitude is 16 times the initial sum of squares, at most 8,388,608. UINT64 holds the product of U_j and total during sampling; that multiplication is outside the claim that the entire path uses INT32.

```
cum[i] = SUM(psi[t]*psi[t], t=0..i)
total = cum[31]
for j = 0..3:
   U_j = LE32(dice[4*j:4*j+4])
   r_j = (UINT64(U_j) * total) >> 32
   o_j = first i with cum[i] > r_j     # total == 0: o_j = 0
   alpha_code[o_j] = 127
meas = BYTE(o_0) || BYTE(o_1) || BYTE(o_2) || BYTE(o_3)
```

B.3 Boundary and Regression Vectors

Case	Rule and purpose	Acceptance check
A	Default fixed A-hash gate; covers the full computation path	meas, Buffer, and H_v match the specification; compare the final target separately
B	Rejection case for the default fixed A-hash gate	Reject without entering the bridge path
C	Regression case with T_a at UINT64 maximum and T_v all ff	Runs under relaxed conditions; not a valid proof under default rules

Example A has meas `051f0519` and second-stage nonce prefix `532e`. Example C has meas `1816011a`, but its A-hash fails the default fixed gate. Tests should check the first-stage decision, bridge result, nonce constraints, and final target together, rather than relying only on a framework return status.

Under default rules, an algorithm proof is `16` bytes and the final binding message is `244` bytes. These must not be confused. The algorithm specification governs the complete byte vectors.