TenQ: GPU-Oriented Proof of Work for TRAIDAG

Deterministic memory-bound work with dependent execution phases

Tenrai Technologies
tenrai-technologies.org

Technical Whitepaper v1.0 • September 2026

ABSTRACT

TenQ is the proof-of-work function used by the TRAIDAG work layer. It maps the canonical, anchor-bound work preimage to a 256-bit value and retains the nonce, extranonce and target interface used by mining templates, pool accounting and reward binding. The production profile combines a 32 MiB epoch cache, a 2 GiB read-only dataset, an 8 KiB attempt-local scratchpad, four dependent 256-instruction programs and 1,024 execution rounds.

The construction separates reusable epoch state from nonce-specific execution. Full evaluation keeps the dataset resident, while Light evaluation reconstructs requested dataset nodes from the cache; both paths implement the same consensus function. The instruction schedule evolves between phases from prior execution state, and the round function combines irregular dataset access, private writable state, mixed integer arithmetic and explicit cross-lane dependencies. These properties target efficient execution on commodity GPUs while increasing the breadth of resources that a specialized implementation must reproduce.

Revision 5 is evaluated on both general-purpose CPUs and an NVIDIA GeForce RTX 2070 under Windows. With the 2 GiB production profile and a batch of 4,096 attempts, CUDA reached 28,964.1 evaluations/s and OpenCL 28,832.3 evaluations/s. A constrained AMD EPYC virtual machine reached 1,866.6 Full evaluations/s with five workers. The paper specifies the construction, establishes deterministic and Full/Light equivalence, derives bounded logical work, analyzes hardware specialization and quantum target search, and relates the resulting verifier budget to high-rate TRAIDAG admission.

Document scope

This paper describes the consensus-relevant TenQ construction and its production profile. The byte-exact packing rules, constants and conformance vectors are normative in the accompanying work specification and source release. Measurements reported here characterize the stated implementations and machines; they are not protocol parameters. Ledger ordering, stake-certified finality and reward settlement are defined by the TRAIDAG Technical Whitepaper [1].

Authors

Qiro, Voss and Neryn — Tenrai Technologies
Cryptis and ZeroBytes — Cryptix Network
Shard — Rift Project

Author names are pseudonyms.

Contents

1 Introduction

TRAIDAG separates permissionless block production from ledger ordering. Miners publish work blocks concurrently, while a stake-certified protocol determines which proposals enter the committed registration queue and which queue prefix is executed. TenQ provides the work predicate for that production layer. A valid block binds its header to a finalized work anchor and satisfies the target committed by that anchor [1].

The design is shaped by two competing requirements. Mining should map efficiently to commodity GPUs, because widely available hardware lowers the entry cost for independent miners. Verification, however, must remain fast enough for nodes operating at high work-block rates. TenQ addresses this tension by separating shared epoch memory from attempt-local execution state. The epoch cache and dataset are reused across many nonce attempts; registers, scratchpad contents, program keys and branch state evolve independently for each attempt.

Deterministic cross-platform execution is a consensus requirement. The work function therefore uses fixed-width integer arithmetic, explicit byte order and explicit read-before-write round semantics. Floating-point arithmetic, device timing and implementation-defined overflow are absent from the validity path. The same abstract machine can be mapped to scalar CPUs, multithreaded CPU verification, CUDA and OpenCL without changing the resulting work value.

Specialization resistance is pursued by composition rather than by a single bottleneck. Dataset access stresses memory capacity, bandwidth and latency; the scratchpad adds private writable state; the instruction mix exercises a broader integer datapath; and phase-dependent program generation limits complete-attempt precomputation. The objective is a small system-level advantage for specialized hardware after memory, arithmetic, control, power and development cost are considered together. This is the same economic question addressed by programmable work designs such as RandomX and ProgPoW, although TenQ uses a different state machine and integration model [4, 5].

The principal results of this paper are: an exact production specification for a GPU-oriented work function; a proof of deterministic evaluation and exact Full/Light equivalence; closed-form bounds on the logical instruction and memory workload; a resource-level analysis of ASIC/FPGA specialization; an oracle-cost treatment of quantum target search; and CPU/GPU measurements tied to the TRAIDAG verification budget.

2 Work interface and production profile

2.1 Canonical input and target

Let H denote the 312-byte canonical TRAIDAG work header and P the work preimage,

$$P = \texttt{TRAI\_POW\_V1} \parallel H, \qquad |P| = 323 \text{ bytes}. \tag{1}$$

The header binds chain identity, protocol version, algorithm identifier, immutable work reference, parent and payload commitments, payload length and mass, reward lock, nonce and extranonce. Scalar fields use the canonical TRAIDAG encoding; nonce and extranonce are opaque 32-byte strings [1, 2].

An authenticated work reference resolves to the anchor context

$$A = (\mathit{chain}, \mathit{revision}, \mathit{profile}, h_A, T_A),$$

which fixes the target and the epoch material applicable to the block. The work predicate is

$$V = \mathrm{TenQ}_A(P), \qquad \mathrm{validWork}(P, A) \iff \mathrm{BE256}(V) \le T_A. \tag{2}$$

The comparison is inclusive. Difficulty changes apply to newly anchored work; they do not reinterpret a previously produced block.

2.2 Production parameters

Table 1 gives the revision-5 production profile. Epoch memory is read-only during attempt execution and can be shared across concurrent workers. The register file and scratchpad belong to one attempt.

The epoch number is derived from finalized anchor height,

$$e(A) = \left\lfloor \frac{h_A}{34560} \right\rfloor. \tag{3}$$

This is a consensus counter. Its wall-clock duration follows the finalized-height cadence of the network rather than a local clock.

Table 1. TenQ production profile, consensus revision 5.

Parameter	Production value
Algorithm identifier	`0x02`
Profile identifier	`0x54454E10`
Epoch cache	32 MiB
Materialized dataset	2 GiB
Dataset node	128 bytes
Cache mixing passes	3
Cache accesses per reconstructed node	160
Logical lanes	32
Registers per lane	16×32 bit
Scratchpad	2,048 × 32-bit words (8 KiB)
Programs per attempt	4
Instructions per program	256
Rounds per attempt	1,024

3 Construction

3.1 Arithmetic and cryptographic primitives

Words are unsigned 32-bit or 64-bit integers. Addition, subtraction and multiplication wrap modulo the relevant power of two; shifts are logical; rotation counts are reduced modulo word width. Multiplication-high returns the upper half of the full unsigned product. Division and remainder use unsigned arithmetic with an odd, nonzero denominator.

SHAKE256 is instantiated according to FIPS 202 with the 24-round Keccak-$f[1600]$ permutation [3]. Internal words are serialized little-endian before absorption. The final 32-byte output is interpreted as a big-endian integer only for the target comparison in Equation (2).

A fixed-width internal primitive $F(X, d_0, d_1)$ accepts 32 input words and two 64-bit domain words, packs them into one Keccak state according to the work specification, applies Keccak-$f[1600]$, and returns the first 32 output words. F is used as an internal compression transform; the outer input and finalization interfaces remain domain-separated SHAKE256 calls.

3.2 Epoch cache and dataset

The epoch seed is

$$E = \mathrm{SHAKE256}(\texttt{TENRAI/TENQ/EPOCH} \parallel \mathit{chain} \parallel \mathrm{LE32}(p) \parallel \mathrm{LE32}(5) \parallel \mathrm{LE32}(e), 64), \tag{4}$$

where p is the profile identifier. Cache nodes are initialized from E and their indices. Three ordered passes then update the cache in ascending node order. Every update combines the current node, the preceding updated node and two data-dependent references, creating a sequential recurrence within each pass.

A dataset node is a deterministic function of the completed cache, the epoch seed and its node index. A 32-word accumulator is initialized from the seed and index, updated through 160 cache references, and finalized with F. Once cache construction is complete, distinct dataset nodes are independent and can be materialized in parallel.

Full evaluation stores all dataset nodes. Light evaluation stores only the cache and reconstructs a requested node when it is referenced. Both paths call the same dataset-node function; they differ only in whether its output was materialized earlier or recomputed on demand.

3.3 Attempt state and dependent programs

For canonical preimage P,

$$S = \mathrm{SHAKE256}(\texttt{TENRAI/TENQ/ATTEMPT} \parallel P, 64). \tag{5}$$

The seed initializes 32 logical lanes, each with 16 registers, the 8 KiB scratchpad and the first program key. Four phases execute consecutively. Registers, scratchpad contents, the next-node index and branch state persist across phase boundaries. The key for phase $q + 1$ is derived from the terminal state of phase q.

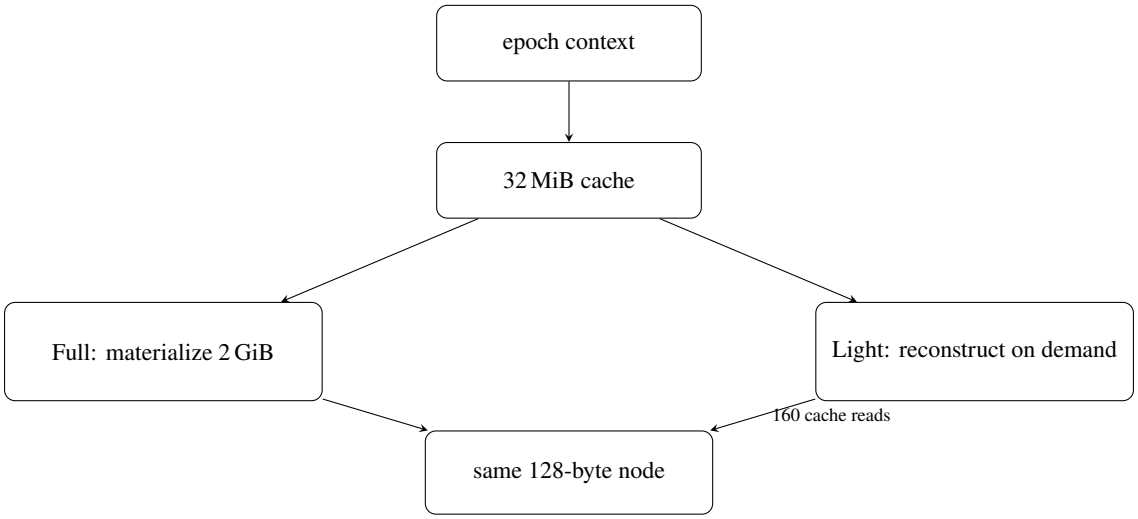

Figure 1. Full and Light evaluation are two implementations of the same dataset-node function. Dataset residency changes memory and recomputation cost, not consensus output.

Program generation uses epoch opcode weights and the phase key. The 256 instruction slots are assigned to 16 opcodes, permuted by a four-round Feistel permutation, and populated with register selectors, immediates and rotation counts. The instruction set combines addition, subtraction, XOR, rotations, multiplication, multiplication-high, division, remainder, byte swap, comparison and logical shifts. Destination and selected source register indices are distinct by construction.

The phase dependency is central to the schedule. The first program can be generated from the attempt seed; later programs become available only after the preceding phase has executed. Complete-attempt instruction order is therefore nonce-dependent rather than an epoch-wide static sequence.

3.4 Round transition

Each round reads one dataset node and one scratchpad word per lane. All scratchpad reads observe the state at the beginning of the logical round. Writes occur only after those reads complete. Lane ℓ writes inside its own 64-word region, eliminating write/write races, while reads may address the complete scratchpad.

Every lane executes eight base instructions and, when the previous branch flag is set, four additional instructions. Lane contributions are reduced in a fixed order, mixed across lanes, and used to derive the next dataset index and branch flag. The first round starts with the extra branch disabled. CPU and GPU implementations may schedule work differently, but they must preserve this logical barrier and reduction order.

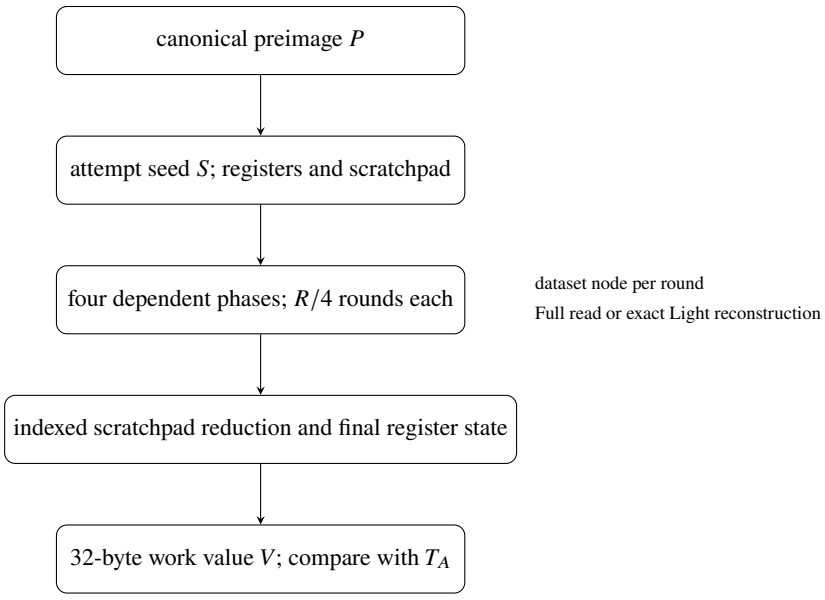

Figure 2. Execution path for one work attempt. The target comparison follows the complete state transition.

3.5 Final state reduction

Partition the scratchpad into 64 ordered 128-byte chunks $X_0, \ldots, X_{63}$, and let a, b be the first two 64-bit words of S. Revision 5 defines

$$L_c = F(X_c, a \oplus c, b + c), \tag{6}$$

$$U = \bigoplus_{c=0}^{63} L_c, \tag{7}$$

$$Z = F(U, a \oplus \texttt{0xf01d}, b \oplus \texttt{0xf01d}). \tag{8}$$

The chunk index enters the compression context before the XOR aggregation. The final work value is

$$V = \mathrm{SHAKE256}(\texttt{TENRAI/TENQ/FINAL} \parallel S \parallel \mathrm{LE32}(p) \parallel \mathrm{LE}(r_0) \parallel \cdots \parallel \mathrm{LE}(r_{31}) \parallel \mathrm{LE}(Z), 32). \tag{9}$$

The function has no secret parameters or device-specific inputs.

4 Formal properties

4.1 Deterministic evaluation

Theorem 4.1 (Deterministic evaluation). For a fixed canonical preimage, authenticated anchor context and prescribed epoch material, all conforming evaluators return the same work value, independently of physical thread scheduling and lane placement.

Proof. The anchor context fixes the profile and epoch seed. Ordered cache recurrences uniquely determine the completed cache; each dataset node is then a pure function of that cache, seed and index. The attempt seed uniquely determines registers, scratchpad and the first program key. Suppose two evaluators agree at the start of a round. The dataset index fixes the node; read-before-write semantics fix scratchpad inputs; the program and previous branch flag fix every instruction. Fixed-width unsigned arithmetic fixes the register updates, disjoint write regions fix the next scratchpad, and the specified cross-lane reduction fixes the next dataset index and branch state. Induction over all rounds and phase transitions gives identical final state, so Equation (9) receives the same byte string and returns the same value. □

The result applies to the abstract machine defined by the specification. A conforming backend is one that reproduces the published consensus vectors for that machine.

4.2 Full and Light equivalence

Proposition 4.2 (Exact reconstruction). If Full and Light evaluation use the same completed cache and epoch seed, they return identical TenQ work values for every valid preimage.

Proof. At dataset index i, the Full path reads the stored output of the dataset-node function. The Light path evaluates that same function at i. Every round therefore receives identical dataset words; Theorem 4.1 gives identical terminal state and final output. □

This equivalence exposes the memory–computation trade-off directly: dataset residency is an implementation choice, while the work value remains a single consensus object.

4.3 Program composition

Let $w_k > 0$ be the transformed epoch weight of opcode k, $W = \sum_{k=0}^{15} w_k$, and $C_k = \sum_{j<k} w_j$. The number of instruction slots allocated to opcode k is

$$n_k = \left\lfloor \frac{256(C_k + w_k)}{W} \right\rfloor - \left\lfloor \frac{256 C_k}{W} \right\rfloor. \tag{10}$$

The allocation intervals partition $[0, 256)$ and the Feistel stage is a permutation, hence $\sum_k n_k = 256$. Base opcode counts are fixed for one epoch even though instruction order, operands, immediates and later phase keys vary by attempt.

In the production profile, one phase executes 256 rounds and eight base instructions per lane per round. The 256-entry base program is traversed eight times per lane. Across four phases and 32 lanes, opcode k therefore executes $1024 n_k$ times in the base portion of an attempt. Branch-dependent instructions are additional.

4.4 Bounded logical work

Let E_b be the number of rounds in which the extra branch is enabled. The first round starts with the branch clear, so

$$0 \le E_b \le R - 1, \qquad N_{\text{ins}} = 32(8R + 4E_b). \tag{11}$$

For $R = 1024$, the attempt executes between 262,144 and 393,088 lane instructions. Full evaluation requests 128 KiB of logical dataset bytes per attempt. Light evaluation requests the same 1,024 logical nodes and reconstructs each through 160 cache references, corresponding to 20 MiB of logical cache reads. Hardware transaction size, cache locality and memory-controller behavior determine the physical traffic seen by a device.

5 Specialization analysis

5.1 Resource composition

A public deterministic work function can be implemented in an ASIC or FPGA. The relevant security objective is therefore economic specialization resistance: the ratio between accepted-proof throughput and the complete cost of the mining system. For implementation j, let r_j be attempts per second, P_j electrical power, K_j an amortized hardware-and-operation cost rate, and p the common target success probability. Two useful system-level measures are

$$\eta_j^{\text{energy}} = \frac{r_j p}{P_j}, \qquad \eta_j^{\text{cost}} = \frac{r_j p}{K_j}. \tag{12}$$

TenQ distributes work across memory, arithmetic, private state and control. The 2 GiB dataset is read-only and epoch-scoped, so a Full miner shares one resident working set across many concurrent attempts. The scratchpad and register state, by contrast, scale with mining concurrency. This asymmetry is deliberate: large reusable memory raises the cost of the memory hierarchy, while attempt-local writable state consumes on-chip capacity and constrains occupancy.

Full and Light evaluation define the two endpoints of a broader time–memory continuum. A specialized implementation may retain a subset of dataset nodes, cache hot regions, reconstruct selected nodes, or build a custom multi-level memory hierarchy. Missing nodes ultimately resolve to the same deterministic reconstruction from the 32 MiB cache. The relevant specialization trade-off is the shared memory system that supports concurrent engines: throughput depends on how effectively a design exchanges dataset residency for the 160-reference reconstruction path.

5.2 Arithmetic and program dependence

The instruction set deliberately mixes operations with different implementation profiles: additions and XORs, rotations and shifts, full and high multiplication, comparisons, byte swaps, division and remainder. A narrow accelerator can optimize individual operations, but the complete design must sustain the weighted mix while preserving register dependencies and memory access order.

Integer division illustrates why opcode count alone is an incomplete hardware metric. Quotient-0 and quotient-1 cases can sometimes reduce to comparisons and subtraction, while a general divider remains comparatively expensive. GPUs also pay nontrivial cost for integer division, so simply increasing divider frequency would not automatically improve relative GPU efficiency [6].

Fixed base opcode counts remove one obvious source of program-selection bias: every complete program in an epoch contains the same number of each base opcode. Phase-key chaining addresses a different degree of freedom. The program for phase $q + 1$ depends on the terminal state of phase q, so a miner learns a later program only after paying for the preceding phases. Rejection of an unfavorable later program therefore carries an explicit sunk cost. Any residual benefit from dependency-aware, operand-aware or latency-aware selection is tied to conditional program distributions and real hardware costs rather than to a free epoch-wide filter.

5.3 State binding and shortcut surface

The final reduction binds chunk position before the XOR aggregation. This removes the permutation symmetry of a position-free XOR fold and prevents direct cancellation by merely swapping chunk positions. More general generalized-birthday or algebraic strategies depend on the set of internal states reachable from valid headers, not on freely chosen 128-byte strings [11].

Revision 5 therefore makes a concrete empirical proposition: a competitive miner must provision a broad mix of memory capacity and access bandwidth, private writable state, integer arithmetic and control. The magnitude of any ASIC or

FPGA advantage is measured at the complete-system level against optimized GPU implementations. The architecture is designed to make that comparison expensive in several dimensions at once rather than to rely on a single nominal memory size or one unusually costly opcode.

6 Quantum target search

6.1 Search complexity

Under an ideal uniform-output model, a complete attempt is accepted with probability

$$p = \frac{T_A + 1}{2^{256}}. \tag{13}$$

Independent classical search requires $1/p$ attempts on average. Grover search and amplitude amplification reduce the ideal oracle-query complexity to order $p^{-1/2}$ [7, 8].

For TenQ, the oracle is the complete work predicate rather than a bare 256-bit comparator. A reversible implementation must account for the dependent round function, dataset or cache access, scratchpad updates, branch state, final reduction, target comparison and the uncomputation needed by the chosen circuit. TenQ therefore changes the cost of one quantum oracle evaluation; it does not change Grover's asymptotic square-root query law.

6.2 Resource model

Let c_C denote the cost of an optimized classical attempt and c_Q the cost of one implemented quantum amplification step. Suppressing constant factors and parallel scheduling gives the comparison scale

$$\frac{\text{classical cost}}{\text{quantum cost}} \asymp \frac{c_C}{c_Q\sqrt{p}}. \tag{14}$$

A concrete value for c_Q requires a reversible realization of the arithmetic and memory strategy, including coherent lookup, ancilla management, circuit depth, uncomputation and fault-tolerance overhead. The classical 2 GiB dataset is therefore an input to a quantum resource analysis, not a direct statement about an equal amount of mandatory coherent quantum memory. Mining-specific studies reach the same distinction between query advantage and practical cost [9, 10].

6.3 Anchor lifetime

TRAIDAG decouples work-block production from work-derived ledger selection. Producing a sibling block does not invalidate another miner's finalized work anchor. The relevant search horizon is the remaining registration lifetime of that anchor, together with relay and admission latency, rather than the mean interval between work blocks. This matters when comparing classical and quantum mining strategies at high BPS: the target opportunity is defined by the anchor window, not by a selected-chain race [1].

7 Performance evaluation

7.1 Measurement environments

CPU measurements characterize verifier capacity. They were collected on Linux x86-64 under KVM with an AMD EPYC 9V74 guest, five visible vCPUs and a cgroup allowance of four CPU-seconds per second. GCC 14.2 compiled the C++ implementation in release mode with OpenMP; the 2 GiB production dataset was fully materialized. Five repetitions were recorded for each Full worker count.

GPU measurements characterize production mining on an NVIDIA GeForce RTX 2070 under Windows. CUDA and OpenCL executed consensus revision 5. The production `prod2g` profile used the 2 GiB dataset with batches of 4,096 and 16,384 attempts. A reduced `bench` profile was additionally measured at 65,536 attempts. For each recorded configuration, Table 2 reports the median throughput and observed minimum–maximum range.

7.2 GPU production results

At batch 4,096, the production-profile medians differ by 0.46%: 28,964.1 eval/s for CUDA and 28,832.3 eval/s for OpenCL. At batch 16,384, the difference is 0.87%. Throughput declines modestly as the production batch grows, from

Table 2. RTX 2070 benchmark results under Windows. The `prod2g` rows use the 2 GiB production profile.

Backend	Profile	Batch	Median eval/s	Observed range	μs/eval
CUDA	bench	4,096	61,954.3	52,494.1–63,748.1	16.14
CUDA	bench	16,384	61,747.0	60,253.9–61,973.7	16.20
CUDA	bench	65,536	61,257.6	60,386.8–61,310.3	16.32
OpenCL	bench	4,096	63,529.0	61,831.9–64,573.6	15.74
OpenCL	bench	16,384	62,572.1	62,422.1–62,679.5	15.98
OpenCL	bench	65,536	61,446.7	61,361.0–61,641.4	16.27
CUDA	prod2g	4,096	28,964.1	28,775.4–29,025.9	34.53
CUDA	prod2g	16,384	28,260.9	28,215.4–28,282.8	35.38
OpenCL	prod2g	4,096	28,832.3	28,384.8–29,190.6	34.68
OpenCL	prod2g	16,384	28,017.1	27,991.7–28,115.7	35.69

28,964.1 to 28,260.9 eval/s for CUDA and from 28,832.3 to 28,017.1 eval/s for OpenCL. On this device the two backends therefore converge on essentially the same production throughput envelope.

The reduced `bench` profile remains above 61,000 eval/s across all tested batches. Its higher throughput reflects the smaller memory and round parameters and is included to show batch-size scaling outside the production profile.

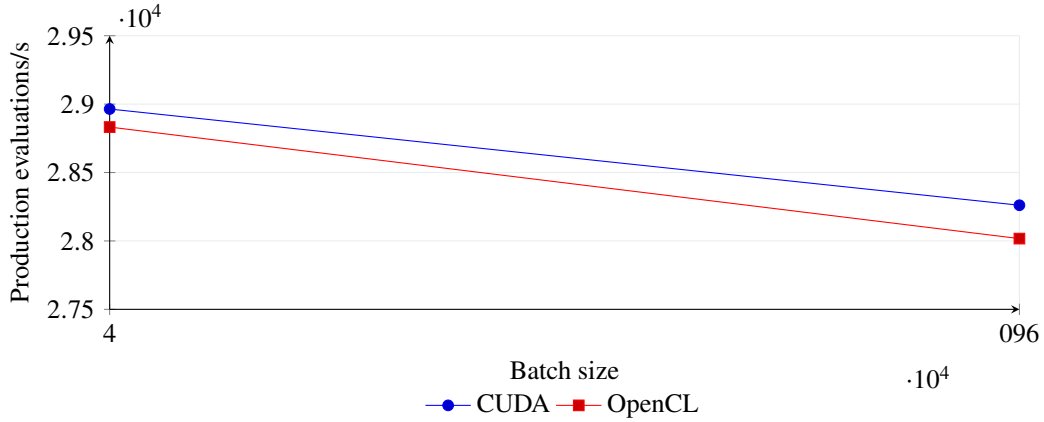

Figure 3. Production-profile throughput on the RTX 2070. CUDA and OpenCL remain within 1% at matched batch sizes.

7.3 CPU verification results

Table 3. Production-profile CPU measurements. Five repetitions per row.

Path	Workers	Median eval/s	Observed range
Full	1	570.5	566.2–576.4
Full	2	1,151.0	1,131.6–1,164.1
Full	4	1,893.7	1,887.7–2,287.2
Full	5	1,866.6	1,739.1–2,006.9
Light	1	32.33	29.63–34.56

Cache construction took 2.221 s and full dataset construction 111.981 s in the same environment. The one-worker Full/Light throughput ratio was 17.65. The four-CPU quota explains the flattening between four and five workers; five visible vCPUs do not represent five dedicated physical cores.

For a work-block rate B, let u be the mean number of direct parent headers that have not already been verified when a block is processed. Ignoring invalid traffic, the proof-evaluation demand is

$$\lambda_{\text{proof}} = B(1+u), \qquad 0 \le u \le 16. \tag{15}$$

At 100 BPS, cached parents require 100 evaluations/s, or 5.36% of the measured five-worker capacity. The cold upper case with 16 unchecked parents requires 1,700 evaluations/s, or 91.07%. Parent-proof reuse is therefore a first-order part of the verifier budget at high BPS.

8 Integration with TRAIDAG

8.1 Anchors, nonce space and difficulty

TenQ retains nonce, extranonce and target difficulty because they form the established boundary between TRAIDAG mining templates and work registration. Difficulty controls the probability that a completed attempt is accepted; the amount of computation in one attempt is fixed by the active profile. Pools can partition nonce or extranonce space without changing the payload, reward destination or anchor.

Before verification, a node resolves the block's work reference through finalized TRAIDAG history. That anchor fixes the target, the TenQ revision, the profile and the epoch context. A later target change cannot alter the validity of an already anchored block.

8.2 Epoch rollover and retention

Epoch material remains necessary while an admissible block or directly referenced parent can still depend on it. A node may keep the current Full dataset resident and use Light reconstruction for older retained contexts. This preserves exact validity while reducing resident memory at the cost of additional reconstruction work.

Profile changes are consensus changes. Dataset size, cache size, round count, reconstruction count, instruction semantics and domain constants are fixed by the activated work profile and cannot be treated as local tuning parameters.

8.3 Useful-compute boundary

The nonce-and-target interface also gives TRAIDAG a stable boundary for a later useful-compute extension. Such an extension would require a separate job format, result-verification rule and settlement protocol. The v1 compute lane remains disabled; revision 5 of TenQ is the proof-of-work function described in this paper [1].

9 Discussion

TenQ deliberately separates three questions that are often conflated in proof-of-work design. Consensus correctness asks whether heterogeneous implementations compute the same value; Theorem 4.1 and Proposition 4.2 address that question. Mining economics asks how much a specialized implementation can improve cost per accepted proof; Section 5 reduces that question to a system-level balance across memory, private state, arithmetic and control. Quantum search asks how the public target predicate behaves under amplitude amplification; Section 6 preserves Grover's generic query model while making the oracle the complete dependent work function.

The performance measurements provide an initial calibration of those design goals. On the RTX 2070, two independent GPU backends deliver almost identical production throughput. The CPU verifier remains fast enough for high-rate admission when previously verified parent headers are reused, while the Light path supplies an exact lower-memory fallback for retained historical contexts. Broader hardware generations will change absolute rates, but they do not change the consensus function or the comparison framework.

10 Conclusion

TenQ defines a deterministic GPU-oriented proof-of-work function for TRAIDAG. Its production profile combines reusable epoch memory with nonce-specific writable state, four phase-dependent integer programs and explicit round semantics. Full and Light evaluation are mathematically equivalent, the logical work per attempt is bounded, and the function remains compatible with conventional nonce search, target difficulty and pool template allocation.

The specialization strategy is architectural. A Full miner shares a large epoch working set, concurrent attempts carry private state, and later instruction schedules depend on earlier execution. Competitive hardware must therefore balance memory capacity and latency, on-chip storage, arithmetic throughput and control rather than optimize a single narrow hash pipeline. Generic quantum search retains its square-root query advantage, but each oracle call must implement the complete reversible TenQ transition.

On an NVIDIA GeForce RTX 2070 under Windows, revision 5 sustains approximately 29,000 production evaluations/s through both CUDA and OpenCL. The measured CPU verifier reaches 1,866.6 Full evaluations/s on a four-CPU-quota virtual machine. Together with exact Full/Light equivalence and anchor-bound work semantics, these results give TenQ a concrete execution model for high-rate, permissionless block production without coupling proof of work to TRAIDAG

ledger ordering.

A Canonical work fields

Table 4. Zero-based field offsets in the 323-byte work preimage.

Offset	Bytes	Field
0	11	literal `TRAI_POW_V1`
11	48	chain identifier
59	2	protocol version
61	1	algorithm identifier (`0x02`)
62	48	work reference
110	48	parent commitment
158	48	payload commitment
206	4	payload length
210	8	payload mass
218	41	reward lock
259	32	nonce
291	32	extranonce

Scalar header fields follow the canonical TRAIDAG big-endian encoding; byte strings are copied unchanged. TenQ binds these bytes into the work value. Payload validity, reward-lock validity and transaction semantics remain separate node checks.

B Revision-5 integer instruction set

Let $x = r[\text{dst}]$, $y = r[a]$, $z = r[b]$, and $m = \text{data} \oplus \text{imm} \oplus \text{salt}$. Each instruction assigns the destination register. hi denotes unsigned multiplication-high and `bswap` reverses four bytes. Addition, subtraction and multiplication wrap at 32 bits.

Table 5. Revision-5 program operations.

Opcode	Assigned expression
0	$x + (y \oplus m)$
1	$x - (z + m)$
2	$x \oplus \text{rotl}_{32}(y, r) \oplus m$
3	$x \cdot (y \mid 1)$
4	$\text{hi}(x \oplus m, y \mid 1) \oplus z \oplus x$
5	$\text{rotl}_{32}(x + y, (z + r)\ \&\ 31) \oplus m$
6	$\text{mix}_{32}(x \oplus y \oplus m) + z$
7	$x + \text{hi}(y \mid 1, z \oplus m)$
8	$\lfloor (x \oplus m)/(y \mid 1) \rfloor \oplus \text{rotl}_{32}(x, r)$
9	$((x + m) \bmod (y \mid 1)) \oplus \text{rotl}_{32}(x, r)$
10	$\text{bswap}(x) + (y \oplus m)$
11	$x \oplus (y < z\ ?\ m : \neg m)$
12	$x + (y \gg (z\ \&\ 31)) + m$
13	$(\text{hi}(x, y) \cdot (z \mid 1)) \oplus m \oplus x$
14	$\text{rotl}_{32}(x \oplus z, r) + \text{mix}_{32}(y + m)$
15	$x + y + z + m$

The symbol r in rotation arguments is the decoded rotation count, not the register array. Exact program packing, salts, phase reductions and domain constants are defined by the work specification [2].

References

[1] Tenrai Technologies and contributors. *TRAIDAG: Certified Ordering over a Proof-of-Work Block Graph*. Technical Whitepaper v1.2.3, September 2026.

[2] Tenrai Technologies. *TenQ v1 Work Profile*. Source release 1.0.0, consensus revision 5, September 2026. Accompanying specification, frozen vectors and consensus profile.

[3] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS 202, 2015. DOI: 10.6028/NIST.FIPS.202.

[4] tevador and RandomX contributors. *RandomX design*. Project design document and reference implementation.

[5] IfDefElse and contributors. *ProgPoW: Programmatic Proof-of-Work*. Design and reference implementation.

[6] NVIDIA. *CUDA C++ Best Practices Guide*. Integer arithmetic and instruction optimization.

[7] L. K. Grover. "A fast quantum mechanical algorithm for database search." Proceedings of STOC, 1996, pp. 212–219.

[8] G. Brassard, P. Høyer, M. Mosca and A. Tapp. "Quantum Amplitude Amplification and Estimation." *Contemporary Mathematics* 305, 2002, pp. 53–74.

[9] R. R. Nerem and D. R. Gaur. "Conditions for Advantageous Quantum Bitcoin Mining." 2021. arXiv:2110.00878.

[10] O. Sattath. "On the insecurity of quantum Bitcoin mining." 2019. arXiv:1804.08118v4.

[11] D. Wagner. "A Generalized Birthday Problem." CRYPTO 2002, LNCS 2442, pp. 288–304.

[12] Tenrai Technologies. *TenQ v1.0.0 Performance Measurements*. September 2026. CPU: Linux x86-64/KVM, AMD EPYC 9V74 guest. GPU: NVIDIA GeForce RTX 2070 under Windows, CUDA and OpenCL.