CONSILIUM EXECUTION BLUEPRINT v9.1 (MVP)

Target: Bittensor SN4 (Optimization / Dense QUBO)
MVP Deadline: 28 Feb 2026
Status: Engineering Specification (for implementation by 2-person team + AI)
Confidentiality: Internal / do not distribute

---

0. Executive Summary

This document is an implementation-focused translation of the "E8 + Topology + Quantum Scout" approach for Bittensor SN4.

Core idea (what to build): - SN4 provides **Dense QUBO** instances `Q` under a strict time window. - We extract fast **topological signals** from the task's interaction structure ("holes/loops" → invariants like β_1) using an IBM/Qiskit **Quantum Scout** with strict timeouts and hard fallbacks. - We construct a deterministic **8D topological latent space** and a projection matrix `W ∈ R^(8×N)` ("topological principal components"). - We **quantize** the latent space against the **E8 root system** (240 roots = vertices of the Gosset polytope `4_21`) by snapping to the nearest E8 root vector. - We **lift** the chosen E8 root direction back to an `N`-bit seed via W^{+}, then run a short **Lift & Repair** local search on the original `Q` to produce a valid submission.

MVP goal: - Implement the full SN4 miner loop on **testnet** and demonstrate non-random positive incentive/score under strict latency constraints.

What is intentionally *not* explained here: - Philosophical / "why E8" motivation. The spec focuses on I/O and determinism so an AI can implement it.

---

1. MVP Definition (Scope)

1.1 Goal

Achieve stable positive incentive/score on SN4 **testnet** by running a single miner instance continuously and submitting valid solutions.

1.2 Success Metrics (KPIs)

Define these numbers before Week 2 and keep them stable for the month: - **Median incentive/score ≥ S_target** over a continuous run of T_eval hours. - **p95 end-to-end latency ≤ T_p95** seconds (job received → tx submitted). - **Acceptance rate ≥ R_accept** (valid solutions / total attempts). - **Stability:** ≥ 24h continuous operation without crash.

Recommended initial targets (edit after first baseline run): - S_target = 0.001 - T_p95 = 45s - R_accept = 80% - T_eval = 24h

1.3 Non-Goals (MVP)

- No GUI/front-end (CLI only).
- No autoscaling / multi-node cluster management.
- No mainnet risk in February (testnet only).
- No “enterprise” observability; keep only minimal structured logs.

1.4 Strategic Context (Sniper Doctrine)

SN4 is treated as a **time-windowed optimization market**. The intended niche is: - Dense / high-connectivity QUBOs where naive GPU heuristics struggle with state update costs. - We ignore low-complexity instances and focus on tasks where the E8+topology pipeline gives asymmetric advantage.

---

2. The Interface: Bittensor SN4 Contract (Must-Be-True)

> **Important:** v9.0 contained placeholder method names. For v9.1, treat SN4 I/O as **contract-first**: verify the real Synapse schema and submission call in the actual SN4 codebase before coding.

2.1 Task Source

Miner receives tasks from SN4 validators via a Bittensor Synapse.

MVP deliverable: - A single function that converts "Raw Synapse" → `(Q, meta)` where: - `Q`: dense matrix (`N×N`, `float32` preferred) - `meta`: includes at minimum `job_id`, `deadline_ts` or `timeout_s`, and any validator-provided scoring hints.

2.2 Task Data Model (Schema to confirm)

Minimum fields we assume exist (or can be derived): - `N` (int): number of variables (expected 2000–5000). - `Q` (matrix): QUBO weights, typically in `[-1, 1]` (confirm). - `domain` (enum): variables are either `{0,1}` or `{-1,1}` (confirm; implement conversion). - `time_window_s` (float): hard deadline for response (confirm).

2.3 Solution Output Model

Miner must output: - `solution`: list/array of length `N` (binary/spins per subnet contract) - optional `energy` (float) if accepted by protocol (confirm) - signature / wallet metadata as required by Bittensor

2.4 Scoring Objective (MVP assumption)

Assume validators reward: - Lower energy `H(s)` is better - Latency is a multiplier/penalty - Invalid format or deadline miss → zero or negative

Define energy in a single canonical form internally, and convert at boundaries: - Spin form: $s \in \{-1, +1\}^N$, $H(s) = s^T Q s$ - Binary form: $x \in \{0, 1\}^N$, $H(x) = x^T Q x$

---

3. System Architecture (Single-Node MVP)

3.1 Components

All components run on one host (laptop or server), with external calls to IBM/Qiskit and an optional solver service.

- **Ingest**: Synapse → dense `Q`, `meta`
- **Quantum Scout**: `(Q, meta)` → topological features `F` (with strict timeout + fallback)
- **W Builder**: `(Q, F)` → `W ∈ R^(8×N)` + cache key
- **E8 Quantizer**: `W`, `Q` → candidate roots `r_i` and/or best `r*`
- **Lift & Repair**: `(W, r*, Q)` → valid `N`-bit solution `s`
- **Submit**: `s` → on-chain/testnet submission
- **Logger**: writes one JSON line per attempt (minimal observability)

3.2 Degraded Modes (Required for MVP)

The pipeline must always produce *some* output before deadline: - **Degraded-Q**: Quantum Scout timed out → use cached `F_cache` - **Degraded-W**: W build failed → use cached `W_cache` or deterministic default `W_default` - **Degraded-S**: Repair cannot finish → return best-so-far seed solution

3.3 Hardware Roles (Conceptual)

- **Orchestrator (Local host):** parsing, caching, building `W`, lift & repair, submission.
- **Scout (IBM Quantum / Qiskit Runtime):** topological feature estimation under strict timeout.
- **Hammer (optional external solver):** if configured, solves/warms-starts candidate solutions within a time slice.

---

4. The “Sniper Loop” Pipeline (Deterministic, Deadline-Aware)

4.1 Hard Time Budget (initial)

This is a starting point; tune after first testnet run.

- Step 1 Ingest: `0.5s`
- Step 2 Quantum Scout: `3.0s` (hard timeout)

- Step 3 Build W: 5.0s
- Step 4 E8 Quantize: 0.2s
- Step 5 Lift & Repair: ≤ (deadline - elapsed - submit_budget) (target 2–20s)
- Step 6 Submit: 1.0s

4.2 Filtering (Sniper Scope)

We intentionally ignore tasks that do not match the dense/high-connectivity niche.

Define **effective density**:

```
density_eps(Q, eps) = count(|Q_ij| > eps) / (N*N)
```

Initial policy: - If N < 2000: DROP (optional; tune) - If density_eps(Q, eps=1e-3) < 0.3: DROP (tune)

4.3 End-to-End Steps

1) **Ingest** - Parse synapse, validate shapes/types, convert to internal canonical domain (spin or binary). - Compute cheap stats: N, density_eps, min/max weight, symmetric check.

2) **Quantum Scout (timeout hard)** - Compute topological features F: - must include at least beta1_estimate (even if approximate) - may include additional scalars: components_estimate, spectral_near_zero, etc. - If timeout/error: load F_cache keyed by (subnet_version, N_bucket, density_bucket) or last-good.

3) **Build W** - Build W ∈ R^(8×N) deterministically from (Q, F) (Section 6). - Cache W for reuse across tasks (keyed by N_bucket + beta1_bucket). - Precompute W⁺ (pseudo-inverse) when W changes.

4) **E8 Quantize** - Generate E8 roots (240 vectors in R^8) deterministically (Appendix A). - Compute reduced matrix Q8 = W Q Wᵀ (8×8). - Select root candidates r by minimizing reduced energy E8(r) = rᵀ Q8 r. - Keep top B candidates (beam width; MVP

start B=8).

5) **Lift & Repair (anytime)** For each candidate r in order: - Lift: $s\_seed = W^{+} r$ (vector in R^N) - Binarize/spinize: s0 = binarize(s_seed) (Section 6) - Repair: run greedy local search on original Q under a strict time slice. - Track best (energy, solution) and stop when remaining time is low.

6) **Submit** - Submit best solution found. - No artificial delays: submit as soon as ready.

---

5. Mathematical & Data Definitions

5.1 QUBO Canonicalization

Pick one internal representation for MVP (recommended: **spin** {-1,+1}): - If subnet uses {0,1}, convert to spin or implement both energy functions and repair deltas.

5.2 E8 Root System (Gosset Polytope 4_21)

We use the E8 root system as a fixed codebook of 240 vectors in R^8: - All roots have equal norm ($||r||^2 = 2$) - Two families: - Type A: permutations of (±1, ±1, 0, 0, 0, 0, 0, 0) → 112 roots - Type B: (±1/2, ..., ±1/2) with an even number of minus signs → 128 roots

Engineering note: - In MVP we do **not** need to implement Weyl-group actions explicitly; we only need the deterministic root codebook and snapping.

5.3 Projection Matrix W

Corrected definition (v9.1): - W ∈ R^(8×N) maps N-dimensional states into an 8D latent space. - v = W s, where v ∈ R^8, s ∈ R^N (a candidate state/seed). - W is derived from topological structure (Section 6), not hand-tuned.

5.4 Snapping / Quantization

Given v ∈ R^8, snap to nearest E8 root:

```
r* = argmin_{r in Roots_E8} ||v - r||_2
```

Since all roots share a constant norm, this is equivalent to maximizing dot product:

```
r* = argmax_{r in Roots_E8} <v, r>
```

5.5 Pseudo-Inverse

Given `W` has full row rank, compute:

```
W⁺ = Wᵀ (W Wᵀ)^(-1)     # dimensions: (N×8)
```

Compute and cache `W⁺` whenever `W` changes.

---

6. E8 Regularizer (Engineering Specification)

6.1 Inputs / Outputs

Inputs: - `Q`: dense `N×N` float32 - `F`: topological feature vector (at minimum `beta1_estimate`)

Outputs: - `W`: `8×N` float32 - `W_plus`: `N×8` float32 - `Roots_E8`: list of 240 vectors in `R^8` (cached global constant)

6.2 Building `W` (MVP-Implementable)

We need 8 stable “topological principal components” for the task graph.

MVP method (deterministic, fast, reproducible): 1) Build a **skeleton graph** `G_skel` from `Q`: - For each node `i`, keep the `k` strongest interactions by `|Q_ij|`. - Symmetrize the edge set. - Set `k = clamp(k_min, k_max, f(beta1_estimate, N))`. 2) Build sparse adjacency A and graph Laplacian $L0 = D - A$. 3) Compute 8 eigenvectors associated with the smallest non-trivial eigenvalues of L0: - Stack them as rows of W (after normalization / orthonormalization).

Recommended MVP defaults: - k_min = 32, k_max = 256 - f(beta1, N) = round(16 + 2*min(beta1, 64)) (tune)

Notes: - This matches the "topological components" intent and keeps runtime feasible at N≈5000. - The Quantum Scout controls beta1_estimate, which controls skeletonization and stability of W.

6.3 Reduced Energy Selection in E8 Space

Compute reduced matrix:

```
Q8 = W Q Wᵀ     # 8×8
```

For each root r:

```
E8(r) = rᵀ Q8 r
```

Select beam B roots with the lowest E8(r).

6.4 Lift & Repair

For each candidate root r: 1) Lift:

```
s_seed = W⁺ r      # N-dimensional real vector
```

2) Binarize/spinize: - If internal domain is spins: s0_i = +1 if s_seed_i >= 0 else -1 - If internal domain is binary: x0_i = 1 if s_seed_i

>= 0 else 0 3) Repair (greedy local search on original Q): - Run until time slice exhausted or no improving flips. - Always maintain best-so-far solution.

Repair must be **anytime**: - If time remains: continue improving - If close to deadline: stop and return current best

Implementation detail for speed: - Maintain g = Q s (or Q x) and update g after each flip so each flip is $O(N)$, not $O(N^2)$. (See Appendix B.)

6.5 Why this can work (engineering framing)

The E8 root codebook provides a compact set of highly symmetric directions in 8D. The hypothesis is: - The 8D embedding concentrates “energy landscape shape” into a small space. - Snapping to E8 roots yields seeds that are closer to good minima than random initialization.

This is an empirical claim for MVP: - We validate only by testnet score and golden-set energy distributions.

---

7. Quantum Scout (Engineering Specification)

7.1 Purpose

Return a **topological feature vector F fast**, under strict timeouts, robust to IBM API issues.

Minimum required output for MVP: - beta1_estimate (float or int) - confidence (0..1) or source (quantum | cache | fallback)

7.2 I/O Contract

Input: - Q (dense N×N) Output: - F = { beta1_estimate, ... }

7.3 Algorithm Sketch (LGZ-like Rank/Laplacian Estimation)

We estimate β_1 via the kernel dimension of the combinatorial 1-Laplacian L1: - Construct a simplicial representation from Q (MVP: use skeleton graph and optional 2-simplices by threshold). - Build boundary operators and L1. - Encode L1 as a Hermitian operator / Hamiltonian. - Use Qiskit Runtime primitives to estimate spectral weight near zero.

Heat-trace estimator (conceptual):

```
beta1 ≈ Tr(exp(-τ L1))    for large τ
```

MVP reality requirement: - The runtime program must return within `T_quantum=3s` or we fall back. - If quantum execution cannot meet this, run the same estimator on a smaller sampled subcomplex (document the sampling).

MVP fallback (must be implemented regardless): - If quantum is unavailable, estimate a graph-level cycle count on the skeleton: - For a graph (1-complex), `beta1_graph = m - n + c` (edges - nodes + connected components). - Use this only as a degraded signal for choosing `k` and stabilizing `W`.

7.4 Timeouts, Budgets, Fail-Safes

- Hard timeout: `T_quantum = 3.0s` (configurable).
- Budget: `≤ Q_calls_per_hour` (initial 10/h).
- Fail-safe: on timeout or non-2xx response:
- Use cached `F_cache`
- Mark attempt as `degraded_quantum=true`

7.5 Cache Keys

Cache `F` by coarse buckets: - `N_bucket = round(N / 250) * 250` - `density_bucket = round(density_eps / 0.05) * 0.05` - `weight_stats_bucket` (optional)

---

8. External Solver (Optional Accelerator)

MVP allows one of these: - **A)** No external solver: use only Lift & Repair (Section 6.4). - **B)** External solver service (DA / cloud GPU) that accepts a QUBO and returns a candidate solution.

If using an external solver, define a stable interface:

8.1 API Contract (example)

Request:

```
{
  "job_id": "string",
  "domain": "spin|binary",
  "Q": "packed matrix or URL",
  "time_limit_s": 20.0,
  "warm_start": "optional seed solution"
}
```

Response:

```
{
```

```
  "solution": [0,1,0,...],

  "energy": -123.45,

  "solver_time_s": 18.7,

  "status": "ok|timeout|error"

}
```

8.2 Failure Policy

- If external solver fails/slow: immediately continue with local Lift & Repair using `s_seed`.

---

9. Testing & Validation (MVP, Minimal)

9.1 Golden Dataset (local)

Save ~20 real tasks from testnet to `tests/data/` as JSON (synapse snapshots).

Required local command: - `run_pipeline --input tests/data/golden_01.json --deadline 45s`

Acceptance (local): - Produces a correctly shaped solution. - Computes energy without NaN/overflow. - Runs under the deadline budget.

9.2 Testnet Run (the real proof)

Run miner on SN4 testnet for `≥ 2h` (then `24h`). Log and track: - latency per stage - final energy - accepted vs rejected submissions - score/incentive if available

---

10. Implementation Roadmap (Feb 2026)

Week 1 — Skeleton

- Implement Ingest + Submit to testnet.
- Implement logging.
- Dummy solver produces valid-shaped random solutions (for protocol verification).

Week 2 — Brain (Quantum + E8 core)

- Implement E8 roots generator + snapping.
- Implement W builder (skeleton graph + spectral embedding).
- Implement Quantum Scout call + strict timeout + cache.

Week 3 — Lift & Repair

- Implement W^+ + lifting + binarize/spinize.
- Implement greedy repair with `O(N)` flip deltas.
- Golden dataset pipeline.

Week 4 — Tuning & Proving

- Tune time budgets, beam width `B`, skeleton `k`, thresholds.
- 24h testnet run and KPI evaluation.

---

11. Risks & Mitigations (MVP)

1) **IBM latency / quota** → strict timeout + cached features + degraded mode. 2) **SN4 protocol changes** → isolate synapse parsing and submission behind interfaces. 3) **Dense matrix performance** → enforce float32 + BLAS + memory budget checks. 4) **Projection error (W bad)** → beam width >1 + repair randomized restarts. 5) **Correlation gap (E8 reduced energy ≠ real energy)** → evaluate top-B roots + pick best after repair.

---

12. Reference Implementation Outline (Minimal, AI-Friendly)

Recommended Python module boundaries (names are suggestions; keep interfaces stable): - miner/ingest.py: synapse parsing → (Q, meta) - miner/quantum_scout.py: Q → F with timeout + cache + fallback - miner/w_builder.py: (Q, F) → W, W_plus - miner/e8.py: root generator + snapping + reduced energy ranking - miner/lift_repair.py: lift, binarize/spinize, greedy repair, energy - miner/submit.py: submit to subnet - miner/main.py: sniper loop + budgets + logging

Minimal CLI commands for MVP: - miner run --subnet 4 --network testnet --wallet <name> - miner replay --input tests/data/golden_01.json --deadline 45s

---

13. Configuration (MVP)

13.1 Environment Variables

- IBM_TOKEN (if using Qiskit Runtime)
- SOLVER_API_KEY / SOLVER_URL (if using an external solver)
- BT_WALLET_NAME / BT_WALLET_HOTKEY (per Bittensor conventions; confirm)

13.2 Runtime Parameters (config file or CLI)

- Timeouts: T_ingest, T_quantum, T_W, T_repair, T_submit
- Beam width: B
- Skeletonization: k_min, k_max, eps_density
- Determinism: RNG_SEED (for repair restarts; default fixed)

Appendix A — Deterministic Generator for 240 E8 Roots

Type A (112 roots): - Choose 2 positions out of 8 for non-zero entries. - Assign each of the two entries a sign ±1. - All permutations of positions and sign choices.

Type B (128 roots): - All 8-tuples of ±1/2 with an even number of negative signs.

Implementation note: - Return a stable ordering (lexicographic) for reproducibility.

Appendix B — Greedy Repair: Fast ΔE Updates (Spin Form)

For spins `s ∈ {-1,+1}^N`, energy:

```
H = sᵀ Q s
```

Maintain `g = Q s` (vector length `N`).

Flipping spin `i` (`s_i := -s_i`) changes energy by:

```
ΔH_i = -4 * s_i * g_i + 4 * Q_ii
```

(Derive/confirm based on the exact diagonal convention used by SN4 tasks; adjust once the dataset is verified.)

After flipping `i`, update:

```
g := g + (-2*s_i_old) * Q[:, i]
```

This makes each flip `O(N)`.

Appendix C — Minimal Structured Log Schema (JSONL)

One line per task attempt:

```
{
  "ts": "iso8601",
  "job_id": "string",
  "N": 5000,
  "density_eps": 0.92,
  "degraded_quantum": false,
  "beta1": 37.0,
  "beam_B": 8,
  "energy": -123.45,
  "latency_total_s": 31.2,
```

```
  "latency": { "ingest": 0.2, "quantum": 2.9, "W": 3.8, "repair": 23.1,
"submit": 1.0 },

  "submit_status": "ok|rejected|timeout|error"

}
```

Appendix D — MUST-CHECK Items (Do Before Coding)

- Confirm SN4 synapse schema and actual submission API in the current subnet repository.
- Confirm variable domain and diagonal conventions in energy computation.
- Confirm whether the validator expects energy or only a solution vector.
- Confirm IBM/Qiskit runtime latency and select an executable strategy that fits `T_quantum`.