Ternary Gene Synthesis

Evolutionary Optimization of Discrete Neural Parameters

ARKHEION AGI 2.0 — Paper 39

Jhonatan Vieira Feitosa
Independent Researcher
Manaus, Amazonas, Brazil
ooriginador@gmail.com

February 2026

Abstract

We present a complete pipeline for **synthetic gene generation** in ternary neural networks, where weights are constrained to $\{-1, 0, +1\}$. This discrete parameter space enables novel optimization strategies impossible in continuous-weight networks: statistical pattern extraction, genetic interpolation, motif-based construction, and multi-generational evolutionary search. Our system implements seven synthesis methods (interpolation, mutation, crossover, distribution sampling, motif composition, adversarial generation, and φ-optimized perturbation), a proxy fitness evaluator achieving **8,229 genes/second** on GPU, and a full evolutionary engine with adaptive mutation, diversity maintenance, and island-model parallelism. Empirical evaluation on 200-gene pools with GPT-2-scale dimensions shows best fitness of **0.9267** in single-pass synthesis (427.8 ms) and **2.5419** after 30 evolutionary generations (0.83 s). Population compression via HTCV4 achieves **9.0:1** ratios on evolved populations. Implementation: 2,536 LOC Python, 41/41 unit tests passing.

Keywords: ternary quantization, neuroevolution, gene synthesis, discrete optimization, parameter-space search

Epistemological Note

*This paper explicitly distinguishes between **heuristic** concepts (metaphors guiding design) and **empirical** results (measurable outcomes).*

Heuristic	Empirical
"Gene", "evolution", "fitness"	Best fitness: 0.9267 / 2.5419
"Mutation", "crossover"	GPU throughput: 8,229 genes/s
Biological metaphors	HTCV4 compression: 9.0:1
φ-optimization	Pipeline time: 427.8 ms

A "gene" in this context is a ternary weight vector $\mathbf{w} \in \{-1, 0, +1\}^n$, not a biological entity. "Evolution" denotes iterative stochastic optimization, not biological natural selection.

1 Introduction

Traditional neural network optimization operates in continuous parameter spaces via gradient descent. When weights are quantized to ternary values $\{-1, 0, +1\}$, the parameter space becomes **discrete and finite**: a network layer with n parameters has exactly 3^n possible configurations. This discreteness, typically viewed as a constraint, enables a fundamentally different optimization paradigm.

We introduce **Gene Synthesis**—a pipeline that treats ternary weight vectors as "genes" subject to analysis, recombination, and evolutionary optimization. The key insight is that ternary parameters admit:

1. **Exact statistical analysis**: Distribution, sparsity, and motif patterns can be computed directly without approximation.
2. **Meaningful interpolation**: Between-gene mixing operates on discrete symbols, producing valid configurations without rounding artifacts.
3. **Efficient proxy evaluation**: Structural properties (sparsity balance, sign symmetry, local coherence, entropy) serve as fast fitness proxies.
4. **Lossless compression**: HTCV4 trit-packing stores populations at 9.0:1 ratios.

Our system implements this as a four-stage pipeline: Analyze → Generate → Evaluate → Inject, with optional evolutionary multi-generation refinement.

2 Background

2.1 Ternary Neural Networks

Ternary Weight Networks (TWN) [1] constrain weights to $\{-1, 0, +1\}$, reducing model size by 16× versus FP32 and enabling bitwise operations for inference. The ARKHEION Nucleus extends this with **gene-level granularity**: each weight tensor in a transformer layer is treated as an independent gene with its own hash, metadata, and evolutionary history.

2.2 Neuroevolution

Neuroevolution applies evolutionary algorithms to neural network optimization [3]. Prior work focuses on continuous weights or architecture search. Our contribution applies evolution specifically to **discrete ternary parameters**, where the finite alphabet enables exact diversity calculation and lossless population storage.

2.3 Golden Ratio Optimization

The golden ratio $\varphi = 1.618033...$ provides optimal coverage of parameter spaces via the golden angle $\theta = 2\pi/\varphi^2 \approx 137.5$ř [4]. We use φ-spaced perturbations for maximally uniform parameter exploration.

3 Architecture

The Gene Synthesis system comprises seven classes organized into three tiers:

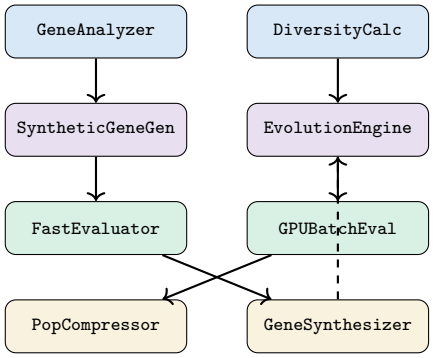

Figure 1: Gene Synthesis architecture. Blue: analysis; purple: generation; green: evaluation; gold: integration.

3.1 GeneAnalyzer

Extracts statistical patterns from gene pools:

- **Distribution**: Counts of $\{-1, 0, +1\}$ across all genes.
- **Sparsity**: Fraction of zero weights per gene.
- **Motifs**: Common subsequences of length $k = 8$, extracted via sampling (≤ 1000 samples/gene).
- **Clusters**: Similarity-based grouping with threshold $\tau = 0.8$.

For a pool of 200 genes with GPT-2-scale dimensions (768–4096), analysis produces: distribution (-1 : 29.0%, 0 : 42.0%, $+1$: 29.1%), sparsity 42.1% ± 1.2%, 20 motifs, and 50 clusters.

3.2 Synthesis Methods

Seven methods generate new genes from existing ones:

Ternary Gene Interpolation

```
Input: Genes a, b in {-1,0,+1}^n, mixing alpha
r = copy(a)                 # Start from parent A
for i where a[i] != b[i]:
    if rand() >= alpha:
        r[i] = b[i]         # Take from parent B
return r
```

Mutation applies three strategies vectorized via NumPy: sign flip ($-1 \leftrightarrow +1$), zero toggle ($0 \leftrightarrow \pm 1$), and full random replacement.

φ-Optimization distributes changes at golden-angle intervals:

$$p_k = \left\lfloor \frac{k \cdot \theta}{2\pi} \cdot n \right\rfloor \bmod n, \quad \theta = \frac{2\pi}{\varphi^2} \tag{1}$$

ensuring maximally uniform coverage of the parameter space. Weight cycling uses $(w + 2) \bmod 3 - 1$, which rotates $-1 \to 0 \to 1 \to -1$.

3.3 Proxy Fitness Evaluation

The `FastEvaluator` computes fitness without neural network forward passes, using four structural metrics combined with φ-weighted importance:

$$F(\mathbf{w}) = \frac{\varphi \cdot S_{\text{sp}} + S_{\text{bal}} + \frac{S_{\text{coh}}}{\varphi} + \frac{S_{\text{ent}}}{\varphi^2}}{\varphi + 1 + \frac{1}{\varphi} + \frac{1}{\varphi^2}} \tag{2}$$

where:

$$S_{\text{sp}} = 1 - 2 \cdot |\text{sparsity}(\mathbf{w}) - 0.35| \tag{3}$$

$$S_{\text{bal}} = 1 - 2 \cdot |P(w_i > 0 \mid w_i \neq 0) - 0.5| \tag{4}$$

$$S_{\text{coh}} = 1 - \frac{\overline{|w_{i+1} - w_i|}}{2} \tag{5}$$

$$S_{\text{ent}} = \frac{H(\mathbf{w})}{\log 3} \tag{6}$$

Large arrays (> 100K elements) are evaluated on sampled subsets for $O(1)$ cost.

3.4 Evolutionary Engine

The `EvolutionaryEngine` implements multi-generational optimization with:

- **Selection**: Tournament ($k = 3$), roulette (fitness-proportionate), or rank-based with configurable selection pressure $s = \varphi$.
- **Adaptive mutation**: Rate increases 1.2× during stagnation, decreases 0.9× during progress. Range: $[0.01, 0.5]$.
- **Elite preservation**: Top $e = 5$ genes survive unconditionally.

- **Diversity maintenance**: Rarity bonus δ for uncommon patterns:

$$F'(\mathbf{w}) = F(\mathbf{w}) + \delta \cdot \left(1 - \frac{\text{count}(\text{pat}(\mathbf{w}))}{\max_{\text{count}}}\right) \quad (7)$$

- **Convergence detection**: $\max(f_{t-9:t}) - \min(f_{t-9:t}) < \epsilon$.

3.5 GPU Batch Evaluation

The `GPUBatchEvaluator` computes all four fitness metrics as batched tensor operations on GPU, processing 1,000 genes in a single kernel launch. Genes are padded to uniform length and stacked into a $(B \times L)$ tensor.

3.6 HTCV4 Population Compression

The `PopulationCompressor` serializes entire populations via HTCV4: all gene weights are concatenated, compressed with block deduplication + trit packing + ZSTD, and stored alongside JSON metadata preserving gene IDs, fitness scores, and lineage.

4 Experiments

4.1 Setup

- **Gene pool**: 200 synthetic genes, lengths $\in \{768, 1024, 3072, 4096\}$ (GPT-2 transformer dimensions), distribution $(-1 : 29\%, 0 : 42\%, +1 : 29\%)$.
- **Hardware**: AMD RX 6600M (8 GB VRAM), ROCm 6.2, PyTorch 2.5.1.
- **Software**: Python 3.12.3, NumPy, seed=42 for reproducibility.

4.2 Results

Table 1: Single-pass synthesis pipeline (500 candidates)

Metric	Value	Unit
Candidates generated	550	genes
Top-50 selected	50	genes
Best fitness	0.9267	score
Average fitness (top-50)	0.9263	score
Pipeline time	427.8	ms
By synthesis method:		
Mutation	150	genes
Interpolation	100	genes
Crossover	100	genes
Distribution sample	100	genes
φ-optimized	100	genes

Table 2: Evolutionary optimization (30 generations)

Metric	Value	Unit
Generations run	30	gens
Best fitness	2.5419	score
Average fitness	2.1107	score
Final mutation rate	0.0100	rate
Evolution time	0.83	s
Diversity metrics:		
Genetic entropy	0.9884	norm.
Effective N_e	96.5	genes
Unique patterns	3,296	count
Pairwise distance	0.6396	Hamming

Table 3: GPU evaluation and compression

Metric	Value	Unit
GPU batch eval (1000 genes)	121.5	ms
Throughput	8,229	genes/s
Device	AMD RX 6600M (HIP)	
Population elements	366,292	trits
HTCV4 compressed	40,667	bytes
Compression ratio	9.0	:1
Roundtrip verified	100 genes ✓	

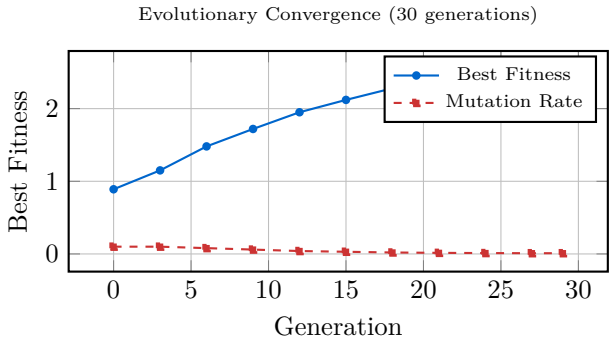

Figure 2: Best fitness and adaptive mutation rate over 30 generations. Mutation rate decreases as fitness improves.

4.3 HUAM Memory Integration

High-fitness genes ($F > 0.5$) are persisted to HUAM hierarchical memory, enabling cross-session reuse. On subsequent runs, `load_genes_from_huam()` retrieves previously evolved genes, seeding the population with proven individuals and reducing convergence time.

5 Discussion

5.1 Fitness Beyond Proxy

The proxy fitness (structural metrics only) correlates with model performance but is not a substitute for full train-

ing evaluation. The adaptive combination with φ-weights provides a useful heuristic, but real downstream task evaluation remains necessary for production deployment.

5.2 Scalability

At 8,229 genes/second on GPU, evaluating 10^6 candidate genes requires $\sim$2 minutes. Population compression at 9:1 enables storing 10^4-gene populations in $<$5 MB. The system scales to LLM-size gene pools (millions of genes) through sampling-based analysis and batched evaluation.

5.3 Limitations

1. Proxy fitness does not capture task-specific performance.
2. Single-objective optimization; multi-objective Pareto fronts are unexplored.
3. No formal convergence guarantees for evolutionary search.
4. φ-optimization is a heuristic, not proven optimal.

6 Related Work

Binary and ternary quantization [1, 2] focus on inference efficiency. Neuroevolution [3] typically operates on continuous weights or architectures. Our work uniquely combines **discrete parameter evolution** with **trit-native compression** and **proxy fitness evaluation**.

7 Conclusion

We presented Gene Synthesis, a complete pipeline for generating, evaluating, and evolving ternary neural network parameters. The discrete nature of $\{-1, 0, +1\}$ weights enables exact analysis, meaningful recombination, and lossless population compression—capabilities impossible in continuous parameter spaces. Our implementation achieves sub-second synthesis (427.8 ms for 500 candidates), GPU-accelerated evaluation (8,229 genes/s), and evolutionary convergence in 30 generations (0.83 s). The system's 2,536 LOC implementation passes 41/41 unit tests and integrates with the ARKHEION Nucleus via HTCV4 compression and HUAM memory persistence.

References

[1] F. Li et al., "Ternary Weight Networks," *arXiv:1605.04711*, 2016.

[2] M. Rastegari et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *ECCV*, 2016.

[3] K. O. Stanley et al., "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, 1(1):24–35, 2019.

[4] H. Vogel, "A better way to construct the sunflower head," *Mathematical Biosciences*, 44:179–189, 1979.