Forge: A Rust Runtime for Ternary Model Evolution

Cross-Language AGI Infrastructure with GPU Training,

MCP Integration, and Gene Pool Management

ARKHEION AGI 2.0 — Paper 48

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

February 2026

Abstract

We present **Forge**, a high-performance Rust runtime for the ARKHEION AGI system, comprising 149 Rust source files and approximately 150,000 lines of code organized into 9 crates. Forge serves as the *performance-critical backbone* of the AGI infrastructure, handling ternary model storage (`.nucleus` format), GPU-accelerated training via AMD ROCm/HIP, gene pool evolution and management, cross-language Python bridging, and Model Context Protocol (MCP) server integration exposing 65+ tools for AI-assisted model engineering. The system achieves memory-safe, zero-cost abstraction performance through Rust's ownership model, while maintaining ergonomic interoperability with the 603K-line Python codebase through `forge-python` and `forge-bridge` crates. Key innovations include: (1) a ternary-native gene representation where each model parameter is a trit $\in \{-1, 0, +1\}$; (2) holographic compression encoding for gene pool storage; (3) GPU kernel dispatch for training with AMD RDNA2 hardware; and (4) a single source of truth (PHI SSOT) for the golden ratio constant across all crates.

Keywords: Rust, AGI runtime, ternary models, gene pools, GPU training, MCP, cross-language, ROCm, HIP, gene evolution

Epistemological Note

This paper documents an implemented and tested system. All metrics are measured from the actual codebase.

Heuristic:	**Empirical:**
"Gene pool" metaphor	149 files, ∼150K LOC
"Evolution" terminology	946 tests passing
"Brain" for inference	65+ MCP tools
"Forge" as crafting metaphor	GPU training verified

1 Introduction

The ARKHEION AGI system's Python codebase (603K LOC, 1,827 files) provides flexibility, rapid prototyping, and compatibility with the PyTorch/NumPy ecosystem. However, three requirements demand systems-level performance:

1. **Ternary model storage**: Packing trits ($\{-1, 0, +1\}$) efficiently requires bit-level manipulation inappropriate for Python
2. **GPU training**: Kernel dispatch, memory management, and training loops benefit from low-level hardware control
3. **Long-running services**: The MCP server must run continuously with minimal memory overhead

Forge addresses all three as a Rust workspace with 9 specialized crates, interoperating with Python through a bridge layer.

1.1 Contributions

1. 9-crate Rust workspace architecture
2. `.nucleus` format for ternary model storage
3. GPU training pipeline via AMD ROCm/HIP
4. 65+ MCP tools for AI-assisted model engineering

5. Cross-language Python bridge
6. Gene pool evolution with mutation and selection
7. Holographic compression for gene storage
8. ∼150K LOC with 946 tests

2 Crate Architecture

Table 1: Forge Workspace Crate Architecture

Crate	Responsibility	Files
forge-core	Data types, trit representation	18
forge-intel	Gene analysis, diagnostics	24
forge-brain	Inference, text generation	16
forge-bank	Persistent gene storage	14
forge-gpu	ROCm/HIP training kernels	12
forge-mcp	MCP server, 65+ tools	22
forge-bridge	Python↔Rust interop	11
forge-python	PyO3 bindings	8
forge-ui	Terminal UI, progress bars	10
Other (tests, examples)		14
Total		**149**

2.1 Dependency Graph

```
forge-mcp --> forge-intel --> forge-core
          |-> forge-brain --> forge-core
          |-> forge-bank  --> forge-core
          |-> forge-gpu   --> forge-core
forge-bridge --> forge-intel
             |-> forge-brain
forge-python --> forge-bridge
forge-ui --> forge-intel, forge-bank
```

All crates depend on `forge-core` for fundamental types.

3 forge-core: Ternary Fundamentals

3.1 Trit Representation

The fundamental unit is the *trit* (ternary digit):

Listing 1: Trit type

```
#[derive(Clone, Copy, PartialEq)]
pub enum Trit {
    Neg = -1,  // -1
    Zero = 0,  //  0
    Pos = 1,   // +1
}
```

A *gene* is a named array of trits representing a model parameter (weight matrix, bias vector, etc.):

Listing 2: Gene structure

```
pub struct Gene {
    pub id: String,
    pub trits: Vec<Trit>,
    pub shape: Vec<usize>,
    pub domain: String,
    pub quality: f64,  // phi score
}
```

3.2 .nucleus Format

The `.nucleus` file format stores gene pools:

1. **Header**: Magic bytes, version, gene count, metadata
2. **Gene table**: Sorted index of gene IDs, shapes, offsets
3. **Trit data**: Packed trits (5 trits per byte, practical; $3^5 = 243 \leq 256$), with a theoretical maximum of $\lfloor \log_3 256 \rfloor = 5$ complete trits
4. **Checksum**: SHA-256 integrity verification

3.3 PHI Single Source of Truth

The golden ratio constant is defined exactly once:

Listing 3: PHI SSOT

```
pub const PHI: f64 = 1.618033988749895;
pub const INV_PHI: f64 = 0.618033988749895;
pub const PHI_SQ: f64 = 2.618033988749895;
```

All crates reference `forge_core::PHI` ensuring consistency across the entire Rust codebase.

4 forge-intel: Gene Intelligence

The intelligence crate provides diagnostic and analytical tools for gene pools:

- **Diagnose**: Identify weak genes, dead genes (all zeros), and quality outliers
- **A/B Compare**: Compare two gene pools across quality metrics
- **Mutation History**: Track evolution lineage

- **Quality Metrics**: φ-score, entropy, trit distribution statistics
- **Auto-Clean**: Remove dead/duplicate genes

4.1 φ-Score

The quality of a gene is measured by its φ-score, computed from three sacred-geometry-inspired components:

$$\varphi\text{-score}(g) = 0.40 \cdot A_{\text{golden}} + 0.30 \cdot S_{\text{entropy}} + 0.30 \cdot F_{\text{fib}} \quad (1)$$

where:

- **Golden alignment** $A_{\text{golden}} = \frac{1}{1+\min(|r-\varphi|,\,|r-1/\varphi|)}$, with $r = t_+/t_-$, measures how close the positive/negative trit ratio is to φ or $1/\varphi$.
- **Entropy score** $S_{\text{entropy}} = -\sum_{v\in\{-1,0,+1\}} p_v \ln p_v \;/\; \ln 3$ is the normalized Shannon entropy over the trit distribution, maximized when trits are evenly balanced.
- **Fibonacci correlation** F_{fib} is the mean autocorrelation of the trit sequence at Fibonacci lag distances $(1, 2, 3, 5, 8, \ldots)$, normalized to $[0, 1]$, measuring self-similarity at Fibonacci-separated positions.

The score ranges roughly over $[0, \sim 1.6]$; higher values indicate better structural quality. This is a *heuristic* quality metric inspired by sacred geometry, **not** an IIT Φ calculation.

5 forge-gpu: GPU-Accelerated Training

5.1 AMD ROCm Integration

Forge dispatches GPU kernels via AMD ROCm/HIP, targeting the RDNA2 architecture (gfx1030, AMD Radeon RX 6600M):

Listing 4: GPU training dispatch

```
pub async fn train_epoch(
    pool: &mut GenePool,
    dataset: &Dataset,
    config: &TrainConfig,
) -> Result<EpochResult> {
    let device = hip::Device::default()?;
    let stream = device.create_stream()?;

    for batch in dataset.batches(config.batch_size) {
        let loss = forward_backward(
            &pool, &batch, &stream
        )?;
        apply_mutations(pool, &loss, config.lr)?;
    }

    Ok(EpochResult {
        loss: total_loss / n_batches,
        mutations: mutation_count,
    })
}
```

5.2 Training Pipeline

The `training_gpu.rs` module (982 LOC) implements:

1. Forward pass: ternary matrix multiplication
2. Loss computation: cross-entropy with label smoothing
3. Backward pass: gradient approximation for ternary weights
4. Mutation: probabilistic trit flips guided by gradients
5. Checkpoint: periodic `.nucleus` file saves

6 forge-mcp: Model Context Protocol Server

6.1 MCP Architecture

Forge implements a JSON-RPC 2.0 server following the Model Context Protocol (MCP) specification, enabling AI agents (GitHub Copilot, Claude, etc.) to interact with gene pools through natural language:

Listing 5: MCP tool example

```
#[mcp_tool(
    name = "forge_diagnose",
    description = "Diagnose gene pool health"
)]
pub async fn diagnose(
    file: String,
    domain: Option<String>,
) -> Result<DiagnosticReport> {
    let pool = load_pool(&file)?;
    intel::diagnose(&pool, domain.as_deref())
}
```

6.2 Tool Categories

65+ MCP tools organized by function:

Table 2: MCP Tool Categories

Category	Tools
Gene pool analysis	12
Gene bank management	8
Gene evolution	7
Brain (inference)	6
GPU training	5
Model conversion	4
Compression	4
Health monitoring	5
Benchmarking	3
Data export	4
Mutation/pruning	4
Multi-model ops	3
Total	**65+**

7 forge-brain: Inference Engine

The brain crate enables text generation from ternary models:

1. **Tokenization**: UTF-8 byte-pair encoding
2. **Forward pass**: Ternary matrix-vector products (only additions, no multiplications—since weights are $\{-1, 0, +1\}$)
3. **Sampling**: Temperature-scaled softmax with top-k/top-p filtering
4. **Deliberation**: Multi-revision reasoning (plan $\to$ reflect $\to$ refine)

7.1 Ternary Efficiency

With weights $w \in \{-1, 0, +1\}$, the matrix-vector product $y = Wx$ reduces to:

$$y_i = \sum_{j:w_{ij}=1} x_j - \sum_{j:w_{ij}=-1} x_j \tag{2}$$

This requires only additions and subtractions—no floating-point multiplications—enabling efficient inference on CPUs without dedicated tensor hardware.

8 Cross-Language Bridge

8.1 forge-bridge

The bridge crate provides Rust functions callable from both the MCP server and the Python codebase:

Listing 6: Bridge interface

```
pub trait ForgeBridge: Send + Sync {
    fn load_pool(&self, path: &str)
        -> Result<GenePool>;
    fn diagnose(&self, pool: &GenePool)
        -> Result<Report>;
    fn evolve(&self, pool: &mut GenePool,
        config: &EvolveConfig)
        -> Result<EvolveResult>;
}
```

8.2 forge-python (PyO3)

The Python crate wraps bridge functions as a native Python module using PyO3:

Listing 7: Python usage

```
import forge_python as forge

pool = forge.load_pool("model.nucleus")
report = forge.diagnose(pool)
print(f"Quality: {report.phi_score:.3f}")
```

9 Gene Pool Evolution

9.1 Mutation Operators

- **Random flip**: Change a random trit
- **Guided flip**: Flip trits where gradient magnitude is highest
- **Cross-over**: Combine trits from two parents
- **Amputate**: Zero out entire genes (destructive)
- **Prune**: Remove low-quality genes

9.2 Selection

Tournament selection with elitism: top 10% of genes survive unchanged, remaining 90% undergo mutation and selection.

9.3 Multi-Objective Pareto Evolution

The system supports Pareto-optimal evolution across objectives: quality (φ-score), size (parameter count), and inference speed (tokens/second).

Table 3: Forge Test Summary

Crate	Tests
forge-core	142
forge-intel	198
forge-brain	87
forge-bank	94
forge-gpu	56
forge-mcp	134
forge-bridge	78
forge-python	42
forge-ui	31
Integration	84
Total	**946**

10 Experiments

10.1 Test Suite

10.2 Performance

- Ternary matmul: 5.2× faster than equivalent float32 matmul on CPU (additions only).[1]
- .nucleus load time: < 100 ms for 268M parameter model
- MCP server memory: < 50 MB idle, < 500 MB during gene pool operations
- GPU training throughput: ∼2K tokens/s on AMD RX 6600M

11 Discussion

11.1 Why Rust?

1. **Memory safety**: No null pointers, data races, or buffer overflows—critical for long-running services
2. **Zero-cost abstractions**: Trait-based polymorphism with no runtime overhead
3. **Cross-compilation**: Single binary deployment without Python runtime
4. **Ecosystem**: cargo, crates.io, and excellent C/C++ interop for GPU libraries

[1] Measured on AMD Ryzen 5 with 1024×1024 matrices comparing optimized lookup-table ternary matmul against naive floating-point multiplication, single-threaded.

11.2 Limitations

- AMD ROCm support is less mature than CUDA
- PyO3 bridge adds complexity compared to pure Python
- 150K LOC Rust codebase maintained by 1 developer + AI
- Gene evolution is CPU-bound; GPU evolution planned

12 Conclusion

Forge provides the performance-critical runtime layer for the ARKHEION AGI system. Its 9-crate Rust workspace with ∼150K LOC and 946 tests delivers ternary model management, GPU training, gene evolution, and AI-accessible tooling through 65+ MCP tools. The cross-language bridge enables seamless interoperation with the 603K-line Python codebase. Forge demonstrates that Rust's ownership model, zero-cost abstractions, and memory safety guarantees are well-suited for AGI infrastructure where reliability and performance coexist.

References

[1] N. D. Matsakis and F. S. Klock II, “The Rust language,” in *Proc. ACM SIGAda*, pp. 103–104, 2014.

[2] S. Klabnik and C. Nichols, *The Rust Programming Language*. No Starch Press, 2019.

[3] F. Li et al., “Ternary weight networks,” in *NeurIPS* Workshop, 2016.

[4] I. Hubara et al., “Quantized neural networks: training neural networks with low precision weights and activations,” *JMLR*, vol. 18, pp. 1–30, 2018.

[5] Anthropic, “Model Context Protocol specification,” 2024. [Online]. Available: https://modelcontextprotocol.io

[6] AMD, “ROCm: Radeon Open Compute Platform,” 2024. [Online]. Available: https://rocm.docs.amd.com

[7] PyO3 Contributors, "PyO3: Rust bindings for the Python interpreter," 2024. [Online]. Available: https://pyo3.rs