Volumetric Neural Architecture and Training Paradigms: An Exhaustive Analysis of Welvet and Target Propagation Modalities

The evolution of neural network frameworks has historically centralized around monolithic, continuous-space directed acyclic graphs (DAGs) optimized for cloud-bound GPU clusters. This paradigm assumes infinite memory bandwidth and relies exclusively on high-precision stochastic gradient descent to traverse loss landscapes. The Welvet engine, serving as the primary bindings for the Loom framework via a C-ABI, represents a fundamental restructuring of this assumption[1]. Engineered as a Deterministic Neural Virtual Machine (DNVM), the architecture guarantees bitwise-identical execution across varied hardware backends, completely bypassing conventional memory bandwidth bottlenecks through polymorphic dispatch and volumetric three-dimensional modeling[2].

The defining characteristic of Welvet is its radical departure from exclusive reliance on classical backpropagation. While the engine fully supports global gradient descent, its primary innovation lies in an expansive suite of "Tween" mechanisms[2]. These neural target propagation modalities replace the global chain-rule lock with localized, gap-based Hebbian learning, bridging idealized and actual activations within discrete volumetric cells[2]. This structural pivot permits extreme active edge training, allowing on-device models to continuously learn without succumbing to the memory exhaustion typical of continuous backpropagation through time.

This report delivers an exhaustive, comparative analysis of the underlying Multi-numerical POLYmorphic Volumetric Tiled-tensor Dispatcher (M-POLY-VTD) engine, mapping the computational efficacy of standard Stochastic Gradient Descent (SGD) against twenty-eight distinct target propagation permutations. These modalities are critically evaluated against their interoperability with the framework's thirty-four native numerical data types (DTypes) and twenty quantization formats, yielding definitive conclusions on the optimal topologies for decentralized neural optimization.

Architectural Foundations of the M-POLY-VTD Engine

To properly evaluate the twenty-nine training modes, one must first deconstruct the physical routing architecture of the Loom/Welvet engine. Traditional frameworks represent networks as sequential lists of layers. Welvet replaces this topology with a volumetric spatial grid where networks exist as three-dimensional matrices of discrete cells[3].

Calculations propagate through this spatial grid using IsRemoteLink hopping mechanisms, which pass tensors between distant spatial coordinates rather than strictly adjacent layers[3]. This three-dimensional routing enables complex residual and parallel architectures to operate

without centralized orchestration. The primary execution vessel for these operations is the shared Dense MatVec microkernel[4]. Recognizing that the vast majority of operations in transformers, convolutional neural networks (CNNs), and recurrent architectures are fundamentally matrix-vector multiplications, Welvet consolidates these floating-point operations (FLOPs) into a unified computational surface. Complex composite structures, such as Multi-Head Attention (MHA), SwiGLU gates, and RNN components, are constructed as composite children of this singular Dense package[4].

This centralization is highly intentional. By restricting the primary mathematical surface to one unified MatVec architecture, the framework guarantees strict parity for quantization bugs, data type (DType) wires, and backend execution across standard CPUs, Plan 9 SIMD (AVX2/NEON), and WebGPU environments[4].

Furthermore, Welvet abandons static inference for a continuous "Step neural mesh." This represents a living architecture characterized by clock-cycle accurate updates and temporal feedback loops, directly simulating biological neural firing patterns[2]. Housed within the runtime/step/ implementation, the engine maintains rolling StepState representations[3]. This temporal mesh dictates that data is not merely pushed forward; it is processed sequentially over discrete internal ticks, necessitating specialized training modalities that can traverse both three-dimensional spatial coordinates and temporal clock cycles.

To monitor this intricate mesh, the framework integrates TANHI (Target-Aligned Neural Heads Interface), a UDP-based JSON-line telemetry layer that provides a real-time HUD of structural blueprints and layer execution meta-estimates[3]. The physical network is further fingerprinted by a DNA Engine that extracts topological signatures of the grid, enabling the detection of spatial "Logic Shifts" and supporting NEAT-style topological evolution across multiple generations[2].

Numerical Precision: The 34 DTypes and 20 Quantizations

The primary variable influencing the failure or success of any specific training modality is its resilience to degraded numerical precision. Modern edge deployment demands sub-byte quantization, yet standard backpropagation algorithms experience catastrophic gradient vanishing or explosion when navigating highly discrete numerical spaces. Welvet mitigates this through a rigorously enforced typing system supporting 34 distinct storage DTypes and 20 specific bit-packed quantization formats natively across all execution backends[4].

The engine operates under a non-negotiable architectural rule: No Quantization-Aware Training (QAT) fallback[4]. The DType and the Quantization Format combined represent the absolute mathematical truth in memory. There are no silent substitutions to Float32 during computation unless explicitly defined at the hardware boundary (such as within a WebGPU WGSL shader before immediate narrowing upon storage upload)[4].

Precision Classification	Core DTypes Supported (FormatNone)	Execution Backend Routing & Alignment
High Precision Real	Float64, Float32	Natively routed via CPU Tiled, WebGPU (Native FP32), and Plan 9 SIMD[4].
Half Precision Real	Float16, BFloat16	Accelerated via F16C SIMD instructions, CPU Tiled, and WebGPU Native float16 extensions[4].
Signed Integer ALUs	Int8, Int16, Int32	Dispatched through DotI8 SIMD fast-paths and WebGPU integer logical units[4].
Unsigned Integer ALUs	Uint8, Uint16, Uint32	Routed via DotU8 SIMD, executed through direct bit-packed matrix traversals[4].
Sub-Byte & Ternary	Int4, Int2, Binary, Ternary	Handled via CPU Pack/Unpack layers, leveraging BitNet-aligned Signed I/O persistence[5].
Block Quantizations	Q4_0 and 19 additional formats	Traverses SIMD DotQ4_0 implementations and highly optimized WebGPU GEMV shaders[4].

These formats utilize an idempotent serialization tunnel termed "Bit-Packed Persistence," which yields compression ratios reaching 98.4% without structural degradation[2]. Consequently, when evaluating training paradigms such as MeshTween or StepTweenSplitSparse, the underlying mechanisms must function not merely within a continuous differentiable Float32 space, but across aggressively rounded, highly discrete ternary states.

When updating highly quantized models (e.g., 1-bit or 2-bit), the framework utilizes a latent Float32 Master repository. The engine caches high-precision gradients or Hebbian updates in this master space, applies the modification, and then mathematically morphs and re-quantizes the master tensor back into the native quantized structure for inference execution, synchronized explicitly via LoomSyncInferenceWeights[5].

The Global Minimization Paradigm: SGD and Backpropagation Modes

The baseline standard for neural optimization relies on automatic differentiation, where gradients are propagated sequentially backward through a network to minimize a global error function. Within Welvet, these continuous-space minimization strategies are located in the runtime/training/ and runtime/backward/ packages[4].

The Classical Model: sgd

The sgd (Stochastic Gradient Descent) mode represents the standard tape-based reverse execution. Operating via Mean Squared Error (MSE) loss, the engine executes ApplyGradSGD over the entire designated layer set[4]. The mechanics of sgd require a continuous forward pass where all intermediate activations are cached in memory. Once the terminal output computes the loss, the backward pass engages, applying the chain rule sequentially from the output back to the primary inputs. This creates a global execution lock; no layer can update its weights until the entire forward pass concludes and the gradient tape reverses completely. When paired with the 34 DTypes, sgd provides optimal theoretical convergence on high-precision datasets (Float64, Float32, Float16). However, its performance degrades severely on Int8 or lower DTypes. Standard backpropagation interprets the discrete steps of heavily quantized integers as non-differentiable walls. While the Float32 Master morphing bypasses hardware exceptions[5], the resulting gradients often fail to map meaningfully to the compressed bit-space, leading to stalling. Furthermore, the memory overhead of maintaining the full activation cache precludes sgd from operating effectively on constrained consumer edge logic.

Temporal Unrolling: step_sgd

The engine's temporal architecture necessitates a specialized gradient variant. The step_sgd mode integrates standard gradient descent within the discrete StepState clock-cycle mesh[3]. Instead of calculating gradients over a static spatial sequence, step_sgd executes Backpropagation Through Time (BPTT). As the network processes sequential data—simulating biological firing ticks—step_sgd caches the spatial activation states for every layer at every discrete clock cycle. Upon sequence completion, it unwinds the tape backwards through both space and time. While this produces mathematically precise gradients for recurrent architectures, it triggers an exponential explosion in VRAM usage. For a sequence of 1,000 steps across a moderate volumetric grid, the activation cache becomes untenable on any device lacking enterprise-grade memory bandwidth, rendering step_sgd virtually obsolete for true decentralized edge deployment.

Spatial Routing: MeshBP

The MeshBP modality maps the backward propagation of errors explicitly onto the three-dimensional volumetric grid. Rather than traversing flat sequential arrays, MeshBP routes the loss gradient backwards through the IsRemoteLink spatial hops[3]. While MeshBP maintains

the mathematical integrity of SGD while respecting the topological architecture of the M-POLY-VTD engine, it exacerbates the global lock problem. Spatial branches located physically distant from the output cells must idle indefinitely, waiting for partial derivatives to traverse the complex 3D topology. This causes severe pipeline stalling across the SIMD and WebGPU execution pipelines, negating the throughput advantages of decentralized dispatch.

The Foundation of Neural Target Propagation (Tween)

To bypass the catastrophic memory and locking constraints of SGD, the framework introduces neural target propagation. Housed within the systems/tween/ package, these modalities abandon the chain rule entirely[4]. Instead of passing mathematical gradients backward, the network propagates idealized activation targets (denoted as $\mathbf{h}^*$).

The Base Paradigm: tween

The foundational tween mode operates on localized, gap-based Hebbian learning[2]. A global network target is ingested, and the engine estimates the ideal activation state for each preceding layer. The weight update mechanism is localized. At each volumetric cell, an error gap is calculated strictly between the actual activation ($\mathbf{h}$) and the idealized target activation ($\mathbf{h}^*$). The weights are then updated using a Hebbian learning rule: $\Delta W \propto (\mathbf{h}^* - \mathbf{h})\mathbf{x}^T$, constrained by DotTile/Saxpy update budgets executed directly on the Plan 9 SIMD architecture[4]. Because tween breaks the global gradient lock, layers can update independently the moment they receive their target. Furthermore, tween exhibits extreme resilience to quantization. It does not require smooth, differentiable spaces; the gap-based Hebbian rule functions as an informed, localized bit-flip mechanism, naturally aligning with Int4, Ternary, and Binary DTypes without suffering from vanishing gradients.

Hybrid Derivatives: tween_chain

While standard target propagation abandons the chain rule, the tween_chain mode synthesizes the two domains. The implementation utilizes a BackendSIMD DotTile/Saxpy chain-rule mechanism that calculates micro-gradients strictly between adjacent layers to accurately refine the idealized target before applying the final Hebbian update[4]. This hybrid approach bridges the aggressive convergence speed of traditional SGD with the uncoupled memory efficiency of target propagation. By restricting the chain rule to isolated layer pairs, tween_chain prevents global pipeline stalling while significantly increasing the precision of the generated target gap, performing exceptionally well across the mid-tier precision DTypes (Int16, Float16).

Volumetric Propagation: MeshTween

Projecting target propagation into the three-dimensional grid yields MeshTween. As activation data transverses the complex IsRemoteLink coordinates[3], MeshTween generates idealized

three-dimensional spatial activation patterns. Because there is no backward gradient lock, discrete volumetric cells update their internal weights autonomously the millisecond their specific spatial target is received. This enables massive asynchronous parallelism across the grid, highly optimizing WebGPU workload distribution where independent shader invocations can update isolated weight buffers concurrently[4].

Volumetric Hybrids: MeshTweenChain

MeshTweenChain applies the localized chain-rule refinements of tween_chain across the boundaries of the 3D spatial mesh. Before a volumetric cell executes its Hebbian update, it requests a partial derivative matrix from its immediately adjacent three-dimensional neighbors to mathematically anchor its target gap. This maintains high topological fidelity during training, preventing logic shifts from diverging chaotically across distant sectors of the DNA engine's structural blueprint.

Temporal Target Propagation: The Step Modes

The intersection of temporal clock-cycle updates with gap-based learning provides Welvet with its most potent capabilities for real-time, recurrent sequence processing.

Clock-Cycle Targeting: step_tween

The step_tween mode applies target propagation to the temporal cycles of the StepState mesh[3]. Instead of executing BPTT, idealized targets are propagated sequentially backward through the temporal ticks. If a specific volumetric cell fired an activation at time t_1 but optimal execution required it to fire at t_2, the target gap is calculated across the temporal axis. The engine updates the weights locally to bridge this temporal divergence. This mode is absolutely paramount for training complex recurrent architectures (RNN, LSTM) on constrained consumer edge devices, as it entirely eliminates the VRAM consumption required by the sequential activation cache of standard BPTT.

Temporal Hybrids: step_tween_chain

step_tween_chain synthesizes the temporal clock-cycle targets with localized chain-rule augmentations. This represents a mathematically demanding state where partial derivatives traverse discrete time steps purely to generate the idealized temporal target for the preceding tick, followed immediately by Hebbian dissolution. While it requires increased SIMD pipeline overhead via the Plan 9 AVX2 instruction sets, it avoids the catastrophic memory implosion of sequential SGD while retaining near-perfect convergence profiles for temporal data streams.

Decoupling the Forward/Backward Lock: Split and Alternating Topologies

To maximize throughput and limit cache contention during active edge training, target

propagation can be topologically partitioned and temporally sequenced.

Topological Partitioning: TweenSplit, StepTweenSplit, MeshTweenSplit

The "Split" modalities represent a profound departure from monolithic error tracking. Under TweenSplit, the network is deliberately bisected or partitioned into autonomous modules. Targets are generated directly at the module boundaries rather than being strictly derived from the final global output head. By "splitting" the target propagation, internal layers deep within the architecture optimize their representations completely independently of the global loss function. This mechanism is incredibly synergistic with deeply quantized Q4_0 block DTypes[3]. In highly quantized networks, error signals degrade severely as they propagate backward through multiple rounding operations. TweenSplit short-circuits this degradation by introducing fresh, localized targets at intermediate boundaries.

- **StepTweenSplit:** Partitions the temporal mesh. Specific sequences of clock cycles optimize toward local temporal targets, enabling continuous, online learning on infinite streaming data (e.g., live audio). Earlier temporal splits update and flush their memory before the broader sequence concludes.
- **MeshTweenSplit:** Partitions the 3D volumetric grid into independent topological zones. Each zone receives its own local target. Supported by the DNA Engine[2], these 3D splits can undergo independent "Logic Shifts", meaning disparate regions of the 3D grid optimize distinctly different feature representations without causing mathematical interference in adjacent spatial cells.

Alternating Execution Phase: TweenAlt, StepTweenAlt, MeshTweenAlt

The "Alt" (Alternating) modes are designed explicitly to mitigate memory cache thrashing, particularly critical when executing on WebGPU where device ALUs operate strictly in Float32 at the boundary[4]. TweenAlt alternates strictly between target generation and target application. During the primary phase, the network solely calculates and propagates idealized targets $\mathbf{h}^*$ down the hierarchy without applying any weight modifications. During the secondary phase, the gap-based Hebbian learning equations execute en masse. This grouping of read operations and write operations maximizes memory bandwidth efficiency.

- **StepTweenAlt:** Alternates the temporal target mapping with the physical state updates in the clock-cycle mesh. It ensures that the biological firing simulation of the StepState is not disrupted mid-cycle by erratic, asynchronous weight shifts.
- **MeshTweenAlt:** Executes the alternating cycle specifically across the 3D spatial hops, preventing data race conditions where adjacent volumetric cells attempt to read actual activations while weights are simultaneously shifting across the memory bus.

Proxy-Accelerated Target Generation

The most significant computational bottleneck of baseline target propagation is the requirement to invert layer transformations to generate preceding targets. The M-POLY-VTD engine resolves this entirely via Proxy modes—auxiliary micro-networks grafted onto the main

topology that are trained to instantaneously predict the idealized activations for hidden layers based directly on the global error state.

Proxy Grafting: TweenSplitHeadProxy and StepTweenSplitHeadProxy

In a split topology, an auxiliary "head proxy" network is grafted onto intermediate volumetric cells. Under TweenSplitHeadProxy, this proxy directly estimates the global loss for the early structural split and backpropagates an idealized target instantaneously. Earlier layers apply their Hebbian updates without ever waiting for the primary forward pass to reach the true output head. This reduces pipeline latency to near-zero.StepTweenSplitHeadProxy applies this predictive head to discrete temporal splits. If a sensory stream is processed over 1,000 clock cycles, this mode splits it into discrete 100-cycle chunks. The proxy immediately predicts the optimal target for the first chunk without waiting for the tenth chunk to conclude, effectively creating a real-time temporal feedback loop.

Linear Approximation: TweenSplitLinear and StepTweenSplitLinear

TweenSplitLinear dramatically simplifies the proxy architecture by substituting a complex non-linear estimator with a basic linear affine transformation. This is extremely computationally efficient and highly compatible with Plan 9 SIMD Int8 integer ALU execution pathways[4]. While it sacrifices a degree of target precision and may struggle to accurately predict targets for highly non-linear SwiGLU or RMSNorm configurations, the throughput gains are massive. StepTweenSplitLinear utilizes this linear transform to rapidly predict the future state of a temporal sequence, applying the resulting target gap to earlier temporal nodes instantaneously.

Ultra-Low-Bit Predictors: TweenSplitFastProxy, MeshTweenSplitFastProxy, and StepTweenSplitFastProxy

The "FastProxy" iterations represent highly optimized versions of the head proxy, stripped of heavy matrix multiplications in favor of sparse lookup tables or heavily quantized 2-bit Ternary proxy weights[3]. TweenSplitFastProxy sacrifices absolute target accuracy for maximum throughput on low-power architectures, such as ARM64 Windows or localized Qualcomm NPU accelerators. MeshTweenSplitFastProxy deploys these ultra-low-bit fast proxies into isolated 3D volumetric cells, effectively creating a distributed swarm of independent micro-networks within the primary framework, each utilizing a quantized proxy to guess its idealized 3D activation state in real time. StepTweenSplitFastProxy is critical for hardware bound by strict real-time deadlines where complex target generation cannot delay the processing of the next sensory input cycle.

Momentum Buffering: TweenSplitLinearCache and StepTweenSplitLinearCache

These modes combine the linear proxy mechanism with a dedicated historical memory cache. Under TweenSplitLinearCache, the engine caches the linearly predicted targets across multiple consecutive forward passes. The engine amortizes the computational cost of target generation

by updating weights via a rolling average of these cached target gaps. This acts as a mathematical momentum buffer, smoothing the aggressive and occasionally erratic nature of Hebbian updates across the 34 native DTypes.StepTweenSplitLinearCache applies this rolling cache to temporal sequences. In recurrent architectures, this mechanism acts similarly to deep historical state tracking, preventing the network from catastrophic forgetting of long-term temporal dependencies by mixing current targets with cached historical ideals.

Asynchronous Pinnacle: TweenSplitHeadProxyAsync and StepTweenSplitHeadProxyAsync

These modalities represent the theoretical and practical pinnacle of pipeline parallelization in edge training. TweenSplitHeadProxyAsync mandates that the proxy network generates targets on a discrete, asynchronous hardware thread while the primary volumetric grid continues its forward inference pass uninterrupted. The network literally trains itself in the background while concurrently performing user inference, realizing the true promise of Loom's "Active Edge Training" mandate[2]. StepTweenSplitHeadProxyAsync executes this temporal target prediction asynchronously. The primary temporal mesh processes continuous sensory input at a fixed, unyielding clock cycle, while an off-thread proxy continuously evaluates past sequences, sending Hebbian updates backward through time without ever interrupting the live data feed.

Sparsity and Budgeted Updates

To conserve power and reduce silicon degradation on edge hardware, target propagation can be constrained by threshold-based sparsity budgets.

Budgeted Target Propagation: TweenSplitSparse and StepTweenSplitSparse

TweenSplitSparse introduces a strict update budget to the localized Hebbian learning equation[4]. Instead of updating the entire dense parameter matrix (e.g., executing a full $N \times M$ weight alteration), the engine filters the target gap. Only neurons exhibiting an error gap exceeding a statistical threshold (ϵ) are updated. This thresholding relies heavily on the FlattenOp meta estimates generated by the TANHI telemetry systems[4]. TweenSplitSparse is absolutely critical for low-power edge training, reducing memory bandwidth utilization during updates by orders of magnitude. StepTweenSplitSparse applies this precise sparsity to temporal target updates. If a specific neuron's activation state across a temporal step aligns closely with the proxy-predicted idealized temporal target, the weight update is skipped entirely. This temporal sparsity vastly reduces the compute overhead of continuous edge learning[2].

The Apex Volumetric Configuration: MeshTweenSplitSparse

MeshTweenSplitSparse represents the most mathematically and structurally efficient spatial

configuration within the entire M-POLY-VTD framework. The 3D grid is partitioned into autonomous topological zones; localized targets are generated individually per zone via proxies, and the localized Hebbian update is heavily sparsified via epsilon thresholds. Only the most critical spatial nodes—such as highly active SwiGLU gates or heavily misaligned MHA attention heads—receive weight updates. This is the exact architectural mode required to continuously train massive, deeply quantized networks locally on consumer edge hardware bound by severely limited VRAM[2].

Synthesizing DType Interoperability

The deployment viability of these twenty-nine modes is inextricably linked to the underlying DType structure[4].

Modality Classification	High Precision (FP32/FP64) Synergy	Integer ALU (Int8/Int16) Synergy	Deep Quantization (Ternary/Binary/Q4_0) Synergy
SGD / step_sgd / MeshBP	Optimal. Provides maximum theoretical convergence due to full gradient tape fidelity.	Moderate. Requires intense Float32 Master morphing and repacking overhead[5].	Poor. Subject to severe gradient explosion and vanishing due to discrete rounding operations.
Base Tween / MeshTween	Excellent. Smooth target gaps propagate perfectly.	Excellent. Localized gap rounding inherently acts as a robust network regularizer.	Moderate. Requires highly precise target formulation to trigger accurate bit-flips.
Proxy-Accelerated Tween	High overhead. Proxies become too heavy and consume inference VRAM.	Optimal. Linear and Fast proxies execute on integer ALUs natively with near-zero latency.	Excellent. Fast Proxies utilize raw bit-packed representations to predict structural shifts rapidly.
Sparse / Split Modes	Unnecessary unless hardware is severely thermally constrained.	Optimal. Reduces SIMD pipeline stalling by skipping redundant Matrix	Critical. Sparsity prevents catastrophic bit-flipping and

		updates.	forgetting in highly compressed low quants.

When training via MeshTweenSplitSparse on a Q4_0 block quantization, the gap-based Hebbian rule effectively acts as an informed, localized bit-flip mechanism rather than the application of a continuous gradient. The proxy-generated error gap determines the magnitude of the required shift, and the sparse budget dictates which specific quantized blocks cross the threshold required to physically alter their integer states in memory. This entirely bypasses the need to inject pseudo-quantization noise during training, a common and flawed workaround in standard framework QAT implementations.

Conclusion: The Mode vs. SGD Synthesis

Based on the exhaustive structural, mathematical, and spatial mapping of the Welvet M-POLY-VTD engine[2], standard Stochastic Gradient Descent serves strictly as a legacy bridging mechanism for high-precision, offline pre-training workloads. While modalities like step_sgd and MeshBP introduce structural fidelity by honoring the clock-cycle step mesh and the 3D grid layout, they fundamentally inherit the catastrophic memory and latency bottlenecks inherent to backpropagation. They strictly require locking the forward inference graph to compute the backward differential tape, a process that is entirely antithetical to active, sovereign edge training[2].

Conversely, the expansive Tween (neural target propagation) ecosystem unlocks the true potential of the volumetric computational grid. By relying on the systems/tween/ Hebbian gap budgets[4], the framework achieves unprecedented asynchronous decoupling. Target generation is isolated locally, allowing layers to optimize independently in real-time without stalling the inference pipeline. The introduction of structural sparsity (MeshTweenSplitSparse) and ultra-low-bit proxies (TweenSplitFastProxy) allows device-bound hardware operating on WebGPU or ARM64 SIMD to perform continuous, infinite online learning.

Ultimately, for offline model initialization on unconstrained hardware across standard Float32 tensors, sgd and MeshBP remain mathematically optimal for absolute loss minimization. However, for continuous, privacy-preserving active edge learning—the stated primary objective of the Loom architecture—standard backpropagation is structurally obsolete. The optimal modalities for real-world deployment on constrained edge nodes are MeshTweenSplitHeadProxyAsync, to manage spatial volumetric decoupling, paired with StepTweenSplitSparse for temporal sensory stream processing. These modes leverage the volumetric 3D routing[3] to isolate updates, employ asynchronous proxies to eliminate pipeline latency, and utilize mathematical sparsity to strictly manage power budgets, confirming proxy-accelerated target propagation as the definitively superior framework for deeply quantized, polymorphic dispatch environments.

Works cited

1. Welvet 0.0.2 - NuGet, https://www-1.nuget.org/packages/Welvet/0.0.2
2. Loom - Layered Omni-architecture Openfluke Machine - GitHub, https://github.com/openfluke/loom
3. loom/docs/index.md at main · openfluke/loom - GitHub, https://github.com/openfluke/loom/blob/main/docs/index.md
4. openfluke/welvet - GitHub, https://github.com/openfluke/welvet
5. loom/docs/bedrock_validation.md at main · openfluke/loom - GitHub, https://github.com/openfluke/loom/blob/main/docs/bedrock_validation.md