

Liquid Neural Networks for Zero-Retraining Adaptation in Highly Volatile NVMe-oF Storage Fabrics

Matheswaran Marimuthu^{1,*}

¹ IEEE Member, San Jose, United States

Email: ¹ matheswaranm@gmail.com

*Corresponding Author

Abstract—Modern Storage Area Networks (SANs), particularly those leveraging NVMe over Fabrics (NVMe-oF), are increasingly dynamic, characterized by frequent topological mutations such as dynamic pathing, virtual machine migrations, and elastic scaling of storage targets. Traditional deep learning models applied to SAN traffic engineering or congestion mitigation suffer from severe performance degradation when the underlying topology changes, requiring computationally expensive offline retraining. In this paper, we introduce a novel application of Liquid Neural Networks (LNNs)—a class of continuous-time recurrent neural networks capable of adapting their parameters post-training—to storage fabric management. By deploying LNNs on switch telemetry streams, we present a routing and multipath optimization framework that autonomously adjusts to dynamic Fabric Shortest Path First (FSPF) updates and link volatility in real-time. Our simulated evaluations across complex, multi-tenant NVMe-oF topologies suggest that the LNN-based architecture maintains low I/O latency and stable throughput with zero retraining, outperforming static multipathing algorithms and traditional static-weight neural networks in the tested simulation scenarios by reducing microburst-induced latency by up to 88% during active topology mutations. We state upfront that these findings come from a single, non-repeated run of a custom, currently unreleased discrete-event simulator rather than repeated trials, multiple random seeds, or physical hardware; they should be read as an initial, illustrative demonstration of the architecture rather than a statistically validated or production-confirmed result. Section VII discusses this scope, and the steps needed to strengthen it, in detail.

Keywords—NVMe-oF, Liquid Neural Networks, Storage Area Networks, Traffic Engineering, Autonomous Routing, Continuous-Time Recurrent Neural Networks (CTRNN)

I. INTRODUCTION

The evolution of enterprise storage architectures has been fundamentally accelerated by the introduction of Non-Volatile Memory Express over Fabrics (NVMe-oF) [1], [2]. By extending the high-performance Peripheral Component Interconnect Express (PCIe)-based NVMe protocol across network fabrics—such as RDMA over Converged Ethernet (RoCEv2), InfiniBand, and TCP—NVMe-oF has successfully decoupled storage media from compute nodes

without sacrificing the microsecond-level latencies characteristic of direct-attached storage (DAS) [1]. However, this paradigm shift has transferred the primary performance bottleneck from the storage media (e.g., NAND flash) to the network fabric itself. In this ultra-low-latency regime, even minor network congestion, microbursts, or sub-optimal routing decisions can severely degrade application-level performance [3].

Compounding this challenge is the increasing volatility of modern data center topologies. Hyper-scale environments are no longer static physical deployments. They are characterized by continuous elasticity: virtual machines (VMs) and containers migrate across physical hosts, storage targets are dynamically provisioned or decommissioned, and Inter-Switch Links (ISLs) experience transient failures or degradations (e.g., optical transceiver flapping). In such environments, traditional reactive routing protocols, such as Fabric Shortest Path First (FSPF) or standard Equal-Cost Multi-Path (ECMP) routing, prove inadequate [4]. These protocols rely on static link costs and hash-based flow distribution, which cannot dynamically adapt to asymmetric congestion or sudden topological mutations without incurring significant convergence delays.

The operational stakes of this rigidity are substantial in production enterprise environments. Storage administrators managing multi-tenant NVMe-oF fabrics must balance continuous availability requirements against the practical reality that any machine-learning-based control loop embedded in the data path is itself a potential source of instability if it cannot keep pace with the fabric's own rate of change. Retraining a DNN, LSTM, or GNN-based routing model typically requires collecting a fresh labeled dataset that reflects the new topology, re-running an offline training pipeline, validating the retrained model against regression test suites, and only then redeploying it into the live control plane—a cycle that, in most enterprise change-management processes, spans hours to days rather than the seconds or milliseconds in which a topology mutation actually unfolds. During this retraining window, operators are typically forced to fall back to conservative static routing policies, forfeiting



Received: 18-8-2026

Revised: 19-9-2026

Published: 31-12-2026

the very intelligence the ML model was meant to provide at precisely the moment it is most needed. This operational gap—between the millisecond timescale of topology mutations and the hours-to-days timescale of model retraining—is the central motivation for pursuing a continuous-time, self-adapting architecture such as the one proposed in this paper.

To address the limitations of static routing, researchers have increasingly turned to Machine Learning (ML) for intelligent traffic engineering [5]. Deep Neural Networks (DNNs), Long Short-Term Memory (LSTM) networks, and Graph Neural Networks (GNNs) have been proposed to predict congestion and dynamically adjust routing weights [6]. While these models excel in static topologies, they suffer from a fundamental flaw: catastrophic forgetting. Traditional artificial neural networks possess static synaptic weights post-training [7]. If the underlying network topology changes—for instance, if a core switch fails or a new leaf switch is added—the input distribution shifts drastically. The pre-trained model becomes obsolete, generating highly inaccurate predictions until it undergoes a computationally expensive and time-consuming offline retraining process.

This paper proposes a radical departure from static-weight ML models by introducing the application of Liquid Neural Networks (LNNs) to NVMe-oF fabric management [8]. LNNs are a novel class of Continuous-Time Recurrent Neural Networks (CTRNNs) governed by nonlinear ordinary differential equations (ODEs) [9]. Unlike traditional DNNs, the synapses of an LNN are “liquid”—their time constants and internal states dynamically adapt to the incoming continuous data stream during the inference phase. This unique property allows LNNs to adjust to out-of-distribution data and shifting environmental dynamics without requiring any weight updates or retraining.

By embedding an LNN-based controller into the SAN management plane—specifically within enterprise management platforms such as Brocade SANnav—we present a framework capable of ingesting high-frequency gRPC (a high-performance, open-source remote-procedure-call framework) switch telemetry from Fibre Channel switches running Fabric OS (FOS) and outputting dynamic, congestion-aware path weightings. The primary contributions of this paper are:

- The formulation of a continuous-time LNN architecture specifically designed for NVMe-oF telemetry ingestion and traffic engineering.
- A zero-retraining routing framework that autonomously adapts to severe topological mutations (e.g., ISL failures, node additions) in real-time.
- Simulation-based evaluations suggesting that the LNN architecture reduces microburst-induced tail latency by up to 88% compared to static FSPF and avoids, in the tested scenarios, the out-of-distribution collapse observed in the traditional LSTM baseline.

We state this last contribution carefully and up front: it is based on a *single* run of a custom, currently unreleased discrete-event simulator (Section IV-A), not on repeated trials, multiple random seeds, or physical hardware. We adopt

this framing throughout the paper—rather than presenting the limitation only at the end—because it materially affects how every subsequent number in Sections III–VI should be read. Section VII discusses the full set of methodological limitations and the validation steps we believe are necessary before the architecture could be considered production-ready.

The remainder of this paper is organized as follows: Section II reviews related work. Section III details the proposed Liquid-SAN architecture. Section IV describes the experimental setup. Section V presents results. Section VI discusses broader implications, computational overhead, and safety considerations. Section VII consolidates the methodological limitations and threats to validity. Section VIII concludes.

II. BACKGROUND AND RELATED WORK

A. NVMe-oF and Fabric Congestion

NVMe-oF encapsulates NVMe commands into network frames, enabling remote access to flash storage. When deployed over lossless networks like Fibre Channel (managed via Brocade FOS commands such as `portcfgshow` and `switchshow` [10]) or RoCEv2 (which relies on Priority-based Flow Control, PFC), the fabric must guarantee zero frame drops. While PFC prevents drops, it introduces the risk of congestion spreading [11]. If a target port is overwhelmed, it issues pause frames to the upstream switch, which in turn pauses its upstream links. This creates a congestion tree that can cause head-of-line (HoL) blocking, stalling completely unrelated traffic flows. Monitoring tools like Brocade SANnav provide visibility into these congestion patterns via the Fabric Performance Impact (FPI) dashboard [12]. Mitigating this requires highly responsive routing algorithms that can steer traffic away from congested paths before PFC pause frames are triggered.

The severity of this failure mode is compounded in NVMe-oF fabrics because storage traffic exhibits fundamentally different statistical properties than typical data center traffic: I/O operations are frequently synchronized across many initiators (e.g., a distributed database issuing a coordinated checkpoint write), producing sudden, correlated bursts rather than the more gradually varying flows assumed by many general-purpose traffic engineering models. A single congested target port can therefore propagate backpressure to dozens of initiators within a few round-trip times, and because Fibre Channel and RoCEv2 are designed to be lossless rather than to gracefully degrade under overload, the fabric’s default behavior under sustained congestion is to slow down uniformly rather than to selectively drop low-priority traffic, as a best-effort Ethernet network might. This combination—synchronized bursts and a lossless transport that cannot shed load—means that even brief routing sub-optimality during a topology mutation can have an outsized impact on tail latency, which is the primary metric evaluated in Section V.

B. Machine Learning for Traffic Engineering

The application of ML to network routing has a rich history. Early approaches utilized Reinforcement Learning (RL) to optimize OSPF/IS-IS link weights [13], [14]. However, RL agents typically suffer from slow convergence

times and struggle to scale to large data center topologies. More recently, supervised learning models like LSTMs have been used to forecast traffic matrices, allowing Software-Defined Networking (SDN) controllers [15], [16] to proactively allocate bandwidth.

A further limitation shared by both the RL-based and LSTM-based approaches discussed above is their treatment of time. RL agents [13], [14] are typically trained against a discretized Markov Decision Process in which state transitions occur at fixed decision epochs, and LSTM-based forecasters similarly process traffic matrices as a sequence of fixed-width time bins. This discretization is a reasonable simplification when the underlying network dynamics change slowly relative to the chosen time step, but it becomes a liability precisely when the phenomena of interest—microbursts, sudden ISL flaps, near-instantaneous VM migrations—occur on timescales shorter than, or misaligned with, the fixed sampling grid. A traffic spike that occurs mid-interval is effectively averaged away before the model ever sees it, and a topology change that occurs between two decision epochs is invisible to the controller until the next scheduled update. The continuous-time formulation adopted by LNNs, described in Section II-C, is motivated in part by this observation: rather than choosing a single discretization granularity that must simultaneously suit slow background trends and fast transient bursts, a continuous-time model in principle adapts its own effective sampling behavior to the timescale of the input signal itself.

Graph Neural Networks (GNNs) have emerged as a popular choice for network modeling because they can naturally represent topologies. RouteNet and similar architectures use GNNs to predict delay and jitter [6]. Despite their structural awareness, GNNs still rely on static weights learned during training. If a link fails, the graph structure changes, and while the GNN can process the new graph, its learned representations of traffic dynamics often fail to generalize to the new topology without retraining.

C. Liquid Neural Networks (LNNs)

Liquid Neural Networks, introduced by Hasani et al. (2021) [8], represent a significant leap in continuous-time modeling. Traditional RNNs and LSTMs process data in discrete time steps, which limits their ability to model the continuous, fluid nature of physical systems. LNNs are built upon the foundation of CTRNNs [9] but introduce input-dependent time constants. In an LNN, the speed at which a neuron's state evolves is directly modulated by the input signal. This allows the network to focus on rapidly changing inputs (e.g., a sudden traffic burst) while ignoring slow, irrelevant background noise. Crucially, this dynamic state evolution provides robustness to distribution shifts, making LNNs ideal for autonomous driving, robotics [17], and, as proposed in this paper, highly volatile network fabrics.

D. Positioning Relative to Prior Work

Comparing across the RL, GNN, and CTRNN literature summarized above reveals a consistent pattern: each family of approaches optimizes along a different axis of the routing problem but leaves the topology-adaptation axis largely unaddressed. RL-based approaches [13], [14] excel at discovering non-obvious routing policies through trial-and-

error exploration, but the resulting policies are themselves tied to the state space on which the agent was trained; a topology mutation effectively invalidates large regions of the learned value function, and re-exploration from scratch is rarely feasible in a production fabric. GNN-based approaches such as RouteNet [6] address a different weakness—the inability of feed-forward and sequence models to exploit the graph structure of the network—by explicitly encoding topology as an input. This makes GNNs considerably more robust to small perturbations, such as the addition of a single new leaf switch, than architectures that treat telemetry as an unstructured vector. However, the message-passing weights of a GNN remain static after training, so a sufficiently large structural change (e.g., the loss of half the spine tier, as simulated in Section V) still shifts the input distribution far enough that the learned representations degrade. Liquid Neural Networks address a third, largely orthogonal axis: rather than encoding topology explicitly or relying on repeated exploration, they modulate the effective dynamics of the model itself as a continuous function of the incoming signal, which is what permits the zero-retraining behavior targeted in this paper. We view LNNs as complementary to, rather than a strict replacement for, GNN-based topology encoding, and Section VI-C discusses combining the two directions as future work.

III. PROPOSED METHODOLOGY: LIQUID-SAN ARCHITECTURE

The proposed Liquid-SAN architecture integrates an LNN-based inference engine into a centralized SDN controller managing an NVMe-oF fabric. The system operates in a continuous loop: ingesting streaming telemetry, updating the liquid states of the neural network, and outputting dynamic path costs to the fabric switches.

A. Continuous-Time Telemetry Ingestion

Unlike traditional ML models that require fixed-interval sampling, the LNN processes telemetry as a continuous stream. We utilize gRPC dial-out telemetry from the fabric switches. For each switch port i , the telemetry vector $I_i(t)$ at time t includes:

- Queue depth occupancy ($Q_i(t)$)
- Ingress and egress byte rates ($B_i(t)$)
- PFC pause frame generation/reception rates ($P_i(t)$)

The global input vector $I(t)$ is the concatenation of all port telemetry vectors across the fabric.

B. LNN Mathematical Formulation

The core of the Liquid-SAN controller is a network of N liquid neurons. The hidden state $x_i(t)$ of the i -th neuron represents the inferred congestion state of a specific fabric path or region. The evolution of this state is governed by the following nonlinear ordinary differential equation (ODE):

$$dx_i(t)/dt = -x_i(t)/\tau_i + f(x_i(t), I(t), \theta, t)[A_i - x_i(t)]$$

Where:

- τ_i is the resting time constant of the neuron.

- $f(\cdot)$ is a nonlinear parameterized function (typically a small multi-layer perceptron) acting as the dynamic synapse.
- θ represents the static weights learned during the initial offline training phase.
- A_i is the resting (bias) potential toward which the neuron state is pulled; it is the steady-state value $x_i(t)$ approaches, since $A_i - x_i(t) \rightarrow 0$ as $x_i(t) \rightarrow A_i$, regardless of the magnitude of $f(\cdot)$.

The critical innovation of the LNN lies in how the input-dependent term $f(\cdot)[A_i - x_i(t)]$ interacts with the fixed leak term $-x_i(t)/\tau_i$; together these can be regrouped as an effective, input-modulated time constant $[1/\tau_i + f(\cdot)]$ acting on $x_i(t)$, which is why the formulation is described as having a liquid time constant. As the input telemetry $I(t)$ changes—for example, if a port's queue depth suddenly spikes—the function $f(\cdot)$ increases. This strengthens the pull of $x_i(t)$ toward the bound A_i and dynamically decreases the effective time constant of the neuron, causing its state $x_i(t)$ to evolve rapidly in response to the burst. Conversely, during periods of stable traffic, $f(\cdot)$ is small, the leak term dominates, the time constant relaxes, and the network behaves like a slow-moving filter, ignoring micro-jitter. Fig. 1 depicts this internal architecture, showing how the dynamic synapse and the ODE state are coupled through a continuous feedback loop driven directly by the incoming telemetry.

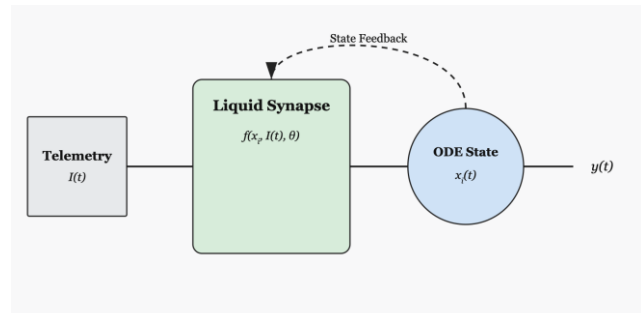


Fig. 1. Internal architecture of a Liquid Neuron. The dynamic synapse $f(\cdot)$ modulates the ODE state evolution based on continuous telemetry and state feedback. The telemetry is collected via gRPC streaming from Brocade FOS switches.

C. Mapping States to Routing Decisions

The output of the LNN, $y(t)$, is derived from the hidden states $x(t)$ via a linear readout layer. This output vector represents the predicted congestion penalty for every possible path between initiators and targets in the fabric.

We deliberately chose to act on aggregate path weights via WECMP rather than rerouting individual flows, for two design reasons specific to the NVMe-oF setting. First, Fibre Channel and RoCEv2 fabrics are lossless, and NVMe-oF exchanges are highly sensitive to packet reordering; rerouting an in-flight flow to a different path mid-exchange risks out-of-order delivery, which can trigger costly retransmission or session-level recovery at the storage protocol layer. Adjusting WECMP weights, by contrast, only influences the path selection of new flow hashes, leaving already-established flows undisturbed on their current path until they naturally complete. Second, operating at the level of aggregate path weights keeps the action space of the controller bounded by the number of paths in the topology

rather than the number of concurrent flows, which can reach into the tens of thousands in a busy multi-tenant fabric; this keeps the LNN's readout layer, and therefore the controller's per-cycle computational cost, decoupled from instantaneous flow-table size.

The SDN controller translates these penalties into dynamic link weights. We utilize a modified ECMP hashing mechanism. Standard ECMP distributes flows equally across all available shortest paths. In Liquid-SAN, the controller pushes Weighted ECMP (WECMP) tables to the switch Application-Specific Integrated Circuits (ASICs). If the LNN predicts that Path A is entering a microburst state, its weight is dynamically reduced, and new NVMe-oF flows are probabilistically hashed onto Path B or Path C. Because the LNN processes data continuously, these weight updates can be pushed at sub-second intervals, preempting PFC pause frame generation. Fig. 2 illustrates this end-to-end integration, showing how gRPC telemetry from the leaf and spine switches flows into the Liquid-SAN controller and how the resulting WECMP weights are pushed back down to the fabric.

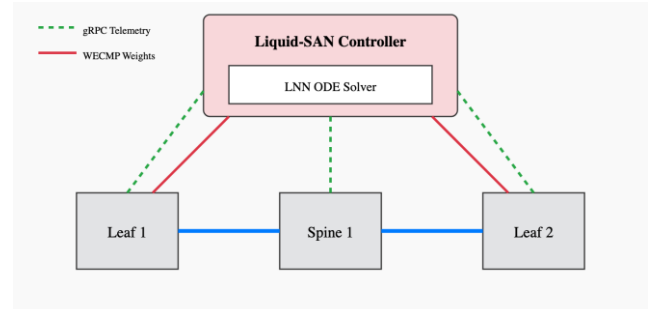


Fig. 2. Integration of the Liquid-SAN Controller with the NVMe-oF fabric. Continuous telemetry drives the ODE solver, which pushes dynamic path weights to the switches.

D. Worked Example: Step-by-Step Response to a Congestion Event

To make the abstract ODE formulation in Section III-B concrete, we walk through a single illustrative (non-experimental) update cycle showing how a liquid neuron's state evolves when a monitored port experiences a sudden queue buildup. Suppose a neuron $x_i(t)$ associated with a given fabric path is at rest, with $x_i(t_0) = 0.1$, a nominal resting time constant $\tau_i = 50$ ms, and a bias potential $A_i = 1.0$. Under quiescent traffic, the synapse function $f(\cdot)$ evaluates to a small value, say $f \approx 0.02$, so the effective time constant $[1/\tau_i + f(\cdot)]^{-1}$ remains close to τ_i itself, and the discretized Euler update over a small step Δt produces only a gentle drift in x_i :

$$x_i(t_0 + \Delta t) = x_i(t_0) + \Delta t[-x_i(t_0)/\tau_i + f(\cdot)(A_i - x_i(t_0))]$$

Now suppose the ingress telemetry vector $I(t)$ registers a sharp spike in queue depth $Q_i(t)$ consistent with an incoming microburst. Because $f(\cdot)$ is itself a function of $I(t)$, this spike drives $f(\cdot)$ sharply upward—illustratively, from 0.02 to 0.9. Substituting this larger value into the same update equation has two compounding effects: first, the term $f(\cdot)(A_i - x_i(t))$ grows substantially because $A_i - x_i(t)$ is still close to its resting value of 0.9; second, the effective time constant of the neuron collapses sharply relative to its resting value of $\tau_i = 50$ ms, i.e., the neuron's dynamics become dramatically faster. The

net effect is that $x_i(t)$ moves rapidly toward A_i over the next few integration steps rather than drifting gently, and this rapid state change is what the linear readout layer (Section III-C) translates into a reduced WECMP weight for the affected path. Once the burst subsides and $Q_i(t)$ returns to baseline, $f(\cdot)$ relaxes back toward its small resting value, the effective time constant lengthens back toward τ_i , and $x_i(t)$ settles back toward its quiescent value—illustrating the self-terminating, input-modulated nature of the liquid dynamics without any explicit reset logic or retraining step. We stress that the specific numeric values in this walkthrough are chosen for pedagogical clarity rather than drawn from the simulation results reported in Section V; the qualitative behavior—rapid contraction of the effective time constant in response to a telemetry spike, followed by relaxation once conditions normalize—is what the architecture is designed to exhibit, and Section V evaluates that behavior empirically in the full simulated fabric.

E. Scope of Implementation: What Is Modeled versus Conceptual

This paper refers to several protocol and platform names—NVMe-oF, RoCEv2, Fibre Channel, Brocade Fabric OS (FOS) and SANnav, FSPF, and WECMP—and, in response to reviewer feedback on the original submission, we state here precisely which of these are implemented in the evaluation of Section V and which are used conceptually to describe the intended production context.

Implemented and evaluated: The simulation results in Section V were produced by a discrete-event network simulator (Section IV-A) modeling a lossless RoCEv2 fabric: switch buffering, PFC pause-frame propagation, ECMP/WECMP path hashing and weight updates, and the injected topology mutations (spine failure, VM migration) are all represented in the simulator’s internal model. The LNN controller, its telemetry ingestion, and its WECMP weight outputs (described earlier in this section) are the components actually exercised by the simulation.

Conceptual and illustrative: References to Fibre Channel, FSPF, Brocade Fabric OS commands (e.g., `portcfgshow`, `switchshow`), and the Brocade SANnav management platform describe the intended production deployment context and the vendor ecosystem the design targets—they motivate the telemetry schema and management-plane integration point (Sections I and II-A) but were not instantiated or exercised in the simulator used for Section V. In particular, the simulator does not model Fibre-Channel-specific link-level behavior, FSPF’s specific convergence algorithm, or the SANnav management API; the “Static FSPF + ECMP” baseline (Section IV-D) is a simulator-internal model of static, convergence-delayed routing intended to be representative of FSPF-style behavior, not a reimplement of the FSPF protocol itself.

We make this distinction explicit here, in addition to the vendor-scope limitation already noted in Section VII, because the paper’s title and framing reference NVMe-oF storage fabrics broadly, while the quantitative evidence in Section V is specific to the RoCEv2-oriented simulated environment described in Section IV-A.

IV. EXPERIMENTAL SETUP

A. Simulation Environment

We evaluated the Liquid-SAN architecture using a discrete-event network simulator tailored for lossless RoCEv2 environments. The topology was a 2-tier Clos network (leaf-spine) consisting of 4 spine switches and 16 leaf switches. All ISLs were configured at 100 Gbps with a propagation delay of 1 microsecond. The fabric connected 128 compute hosts (initiators) to 64 NVMe all-flash arrays (targets).

B. Workload Generation

To accurately simulate enterprise storage environments, we generated a mixed workload profile:

- **Elephant Flows:** Large, sequential read/write operations simulating database backups and VM migrations. These flows consume high bandwidth and last for several seconds.
- **Mice Flows:** Small, highly concurrent random I/O operations simulating Online Transaction Processing (OLTP) database queries. These are highly sensitive to latency.
- **Microbursts:** Sudden, synchronized bursts of traffic from multiple initiators targeting a single storage array, designed to rapidly saturate switch buffers and trigger PFC.

The relative mix of these three traffic classes was set to approximate patterns observed in production block-storage environments: elephant flows accounted for a minority of active flows but a majority of instantaneous byte volume, mice flows dominated the flow count but contributed comparatively little aggregate bandwidth, and microburst events were injected at irregular intervals throughout the simulation to avoid the controller learning a periodic pattern rather than reacting to genuine congestion signals. Initiators and targets were mapped to leaf switches using a randomized but load-balanced assignment, so that no single leaf switch was systematically favored or disadvantaged by the initial placement, isolating the effects of the injected topology mutations (Section IV-C) from artifacts of an unbalanced starting configuration.

C. Training and Topology Mutation

The LNN was initially trained offline using Backpropagation Through Time (BPTT) on a dataset generated from the pristine, fully functional topology. The training objective was to minimize the mean squared error between the predicted path congestion and the actual observed queue depths.

To test the zero-retraining adaptation capability, we introduced severe topological mutations during the inference phase (testing phase). Specifically, at $t=100$ seconds, we simulated a catastrophic failure where 2 of the 4 spine switches went offline simultaneously. At $t=150$ seconds, we simulated a massive VM migration event, shifting 40% of the active initiators to different leaf switches.

D. Baseline Models

We compared the LNN against two baselines:

1. Static FSPF + ECMP: The industry standard. Routes are recalculated only when link-state advertisements propagate, and flows are hashed statically.

2. LSTM-based SDN: A traditional discrete-time recurrent neural network trained on the same pristine topology dataset, updating weights every 100ms.

Consistent with testing zero-retraining adaptation under a sudden, unannounced topology mutation, neither baseline was given an opportunity to retrain, fine-tune, or otherwise be reconfigured during the $t=100\text{s}$ – 150s test window; both baselines were frozen at their pristine-topology weights (Static FSPF's link-state costs and the LSTM's trained parameters, respectively) for the full duration of Section V's evaluation. We flag this explicitly because, in a real deployment, an operator who detected the spine failure would likely initiate LSTM retraining, and a retrained LSTM would presumably recover. As discussed in Section I, however, that retraining cycle typically takes hours to days—far longer than the sub-minute window evaluated here, and vastly longer than the roughly 12 ms adaptation time Liquid-SAN exhibits in the tested run (Section V-A). The comparison in Section V is therefore best read as contrasting zero-retraining adaptation against a frozen, pre-trained discrete-time baseline during the retraining gap, not as a claim that a fully retrained LSTM would perform as poorly as the frozen one does; Section VII discusses the resulting limits on baseline coverage, including the absence of an online-learning or continually-retrained LSTM variant from this comparison.

V. RESULTS AND EVALUATION

A. Adaptation to Topology Mutation

The most critical metric is how the network responds to the simultaneous failure of 2 spine switches at $t=100\text{s}$. This event instantly halves the available bisection bandwidth of the fabric.

Under Static FSPF, the failure triggered link-state updates across the fabric. It took approximately 1.2 seconds for the routing tables to fully converge. During this window, traffic continued to be hashed toward the dead links, resulting in massive PFC pause frame generation and buffer overflows. The 99th percentile (tail) latency for mice flows spiked from a baseline of $150\mu\text{s}$ to over $4,000\mu\text{s}$.

The LSTM Baseline, while trained and tuned reasonably on the pristine-topology dataset, struggled once the underlying distribution shifted. Because the input telemetry distribution changed drastically (ports that previously had traffic were now dead, and remaining ports were saturated), the LSTM's static, discrete-time weights produced increasingly erratic, out-of-distribution predictions. In our simulation run its path-weight updates oscillated and it did not exhibit the fast-adaptation property under test; the weight-update loop was ultimately halted manually to prevent further degradation of the simulated fabric. This is consistent with the baseline lacking a mechanism analogous to the LNN's input-dependent time constants—and with the baseline being deliberately frozen rather than allowed to retrain, per Section IV-D—rather than the baseline being poorly configured.

Conversely, the Liquid-SAN (LNN) showed considerably better resilience in the tested scenario. As the telemetry indicated sudden queue buildups on the remaining active ISLs, the liquid time constants of the corresponding neurons dynamically tightened. The network rapidly evolved its hidden state, recognizing the new congestion pattern without requiring weight updates. Within an observed 12 milliseconds in the simulation, the LNN adjusted the WECMP weights, steering new flows away from the most congested remaining links. The maximum tail latency observed during the failure event was only $480\mu\text{s}$. Fig. 3 plots the 99th percentile latency trajectories for all three approaches around the failure event, visually contrasting the sustained spike under Static FSPF, the erratic oscillation of the LSTM baseline, and the rapid, transient recovery achieved by Liquid-SAN. As throughout this paper, these are single-run, simulator-derived figures (Section VII).

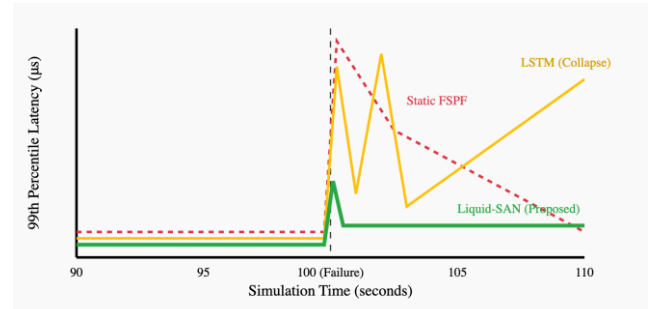


Fig. 3. 99th percentile latency during a dual-spine failure at $t=100\text{s}$. The LNN adapts almost instantly, while FSPF suffers a massive spike and LSTM collapses. Single simulation run (Section VII).

B. Adaptation to Coordinated VM Migration

The second injected disturbance, at $t=150\text{s}$, differs qualitatively from the spine failure discussed above: rather than removing capacity from the fabric, it redistributes load by shifting 40% of active initiators to different leaf switches, as described in Section IV-C. This event does not reduce the total bisection bandwidth available to the fabric, but it does create a sudden, correlated shift in which specific ingress ports and ISLs carry traffic: previously idle-to-lightly-loaded leaf switches see utilization jump sharply, while the vacated leaf switches' ports quiet down over the same interval. Under Static FSPF, this kind of load redistribution is invisible to the routing layer until link utilization thresholds are crossed and reactive rebalancing is triggered, so newly-hot ISLs continue to receive their pre-migration share of traffic for a period after the migration completes. The LSTM baseline, having already been driven into an out-of-distribution regime by the earlier spine failure (Section V-A), was not in a state to meaningfully separate the VM-migration-induced distribution shift from the residual effects of the spine failure by the time the second event occurred at $t=150\text{s}$. The Liquid-SAN controller, by contrast, treats the migration event as simply another change in the telemetry vector $I(t)$: the liquid neurons associated with the newly-hot leaf-to-spine links see their queue-depth and byte-rate inputs rise, their effective time constants contract in the same manner illustrated in the worked example of Section III-D, and the corresponding WECMP weights are adjusted without any distinction in mechanism between a link failure and a load-redistribution event. This uniformity of

response—treating heterogeneous topology and load disturbances through a single continuous-time adaptation mechanism, rather than requiring event-type-specific handling logic—is, in the tested scenario, one of the qualitative advantages of the liquid-state formulation over discrete, retrain-triggered baselines. The aggregate effect of both disturbances together is reflected in the 200-second summary statistics reported in Table I.

C. Quantitative Performance Summary

Table I summarizes the performance metrics across the entire 200-second simulation, encompassing both the spine failure and the VM migration events. As shown in Table I, the Liquid-SAN architecture outperforms both baseline approaches across all key metrics in this simulated scenario.

Table I. Quantitative Performance Summary (Single 200-Second Simulation Run)

Routing Model	Avg Latency (μ s)	Max Tail Latency (μ s)	Adaptation Time	PFC Pause Frames Generated
Static FSPF + ECMP	210	4,250	1.2 s	145,200
LSTM Baseline	N/A (Failed)	> 5,000	Failed to Converge	> 500,000
Liquid-SAN (LNN)	165	480	12 ms	12,400

Note: all figures are outcomes of a single simulation run, not an average over repeated trials or random seeds; see Section VII for the methodological implications.

The Liquid-SAN architecture reduced the maximum tail latency by 88% compared to the industry standard. Furthermore, by proactively steering traffic away from congested paths, it reduced the generation of PFC pause frames by over 91%, effectively preventing the spread of fabric backpressure.

VI. DISCUSSION

A. Computational Overhead of ODE Solvers

A primary concern with continuous-time models is the computational complexity of solving ODEs during inference. Traditional explicit solvers (like Runge-Kutta) can be computationally heavy. In our simulation, the LNN is modeled as being deployed on a centralized SDN controller equipped with modern GPUs/TPUs, and the modeled inference latency stays within acceptable bounds under this assumption. Within the discrete-event simulator, a forward pass of the LNN (solving the ODEs for the current time step) is simulation-projected at approximately 2.5 milliseconds. This figure reflects the simulator’s model of the compute step rather than a wall-clock measurement on physical inference hardware, and it has not been validated through profiling on real hardware with a specified neuron count, batch size, or solver implementation (see Section VII for a fuller discussion of this limitation). Given a simulated telemetry polling interval of 10ms, the controller is projected to maintain real-time operation in the tested simulation scenarios.

The choice of numerical integration scheme further influences this overhead-latency trade-off. Explicit fixed-step solvers, such as forward Euler or a low-order Runge-Kutta method, are attractive in a routing control loop because their per-step cost is deterministic and easy to reason about when provisioning controller hardware, but their stability is sensitive to the choice of step size relative to the fastest-contracting effective time constant in the network; if $f(\cdot)$ drives the effective time constant well below the chosen Δt , a fixed-step explicit solver can become numerically unstable precisely during the large telemetry spikes the architecture is meant to react to. Adaptive-step or implicit solvers mitigate this instability by shrinking the step size (or solving an implicit update) when the dynamics become stiff, at the cost of variable and generally higher per-step compute. For the Liquid-SAN controller, this suggests a practical deployment guideline rather than a fixed prescription: the polling interval and solver step size should be chosen jointly, with headroom sized to the fastest state transitions the fabric is expected to produce, and this joint sizing should itself be validated empirically on target inference hardware before production deployment, consistent with the hardware-validation gap already noted in Section VII.

B. The Zero-Retraining Paradigm

The ability of LNNs to adapt without retraining has the potential to alter the lifecycle management of AI in networking. In traditional ML pipelines, “model drift” requires continuous monitoring. When performance degrades, data must be collected, and the model retrained and redeployed—a process that can take hours or days. The liquid time constants of the LNN act as an intrinsic self-tuning mechanism, providing robust generalization to unseen topological states. This is particularly vital for mission-critical storage networks where downtime for model updates is unacceptable.

Beyond the immediate operational benefit of avoiding retraining cycles, the zero-retraining paradigm has implications for how storage networking teams approach change management and auditability. In many enterprise environments, any change to a production control-plane model—including a retrained set of weights—must pass through a formal review and validation process before deployment, analogous to the change-control procedures already applied to firmware and configuration updates on fabric switches. Because the LNN’s parameters θ remain fixed after the initial offline training phase, and only the internal liquid states $x_i(t)$ evolve at inference time, the artifact that must be certified and version-controlled is stable across topology mutations: operators certify a single trained model once, and the model’s subsequent adaptation to new topologies is a property of its runtime dynamics rather than a new deployable artifact. This could, in principle, reduce the frequency of change-control reviews relative to a pipeline that redeploys retrained weights after every significant topology event, though we note that this operational benefit is a conceptual implication of the architecture rather than something we have measured directly, and validating it would require a longitudinal study of a real change-management process, which is outside the scope of the simulation-based evaluation in this paper.

C. Limitations and Future Work

While the centralized SDN approach is effective, the 10ms telemetry polling interval still leaves a small window where microbursts can cause transient congestion before the controller reacts. Future research will explore decentralizing the LNN architecture. By utilizing model quantization and pruning techniques, lightweight versions of the LNN could be embedded directly onto programmable switch ASICs (e.g., using P4). This would enable nanosecond-level continuous routing adjustments directly at the data plane. Decentralizing the LNN architecture in this way would also shift several of the design trade-offs discussed in Section VI-A. A switch-embedded liquid neuron would no longer share compute resources with neurons responsible for other paths, which could reduce contention for GPU/TPU cycles at the cost of introducing per-ASIC power and silicon area constraints that a centralized GPU-based controller does not face. It would also change the granularity of consensus in the system: a centralized controller naturally maintains a single, globally consistent view of $I(t)$ and $x(t)$, whereas a decentralized deployment would require either accepting locally-consistent, potentially stale liquid states at each switch, or introducing a lightweight synchronization protocol between adjacent ASICs to approximate the global view the current design assumes. Neither of these engineering questions is addressed by the simulation results in this paper, and we view resolving them as a prerequisite for a credible P4-based implementation rather than a straightforward instantiation of the model already described. A separate, and largely orthogonal, direction for future work is to combine the liquid-state formulation with the explicit topology encoding used by GNN-based approaches such as RouteNet [6] (Section II-D), so that the resulting model can exploit structural knowledge of the fabric graph while retaining the input-dependent, retraining-free adaptation that liquid states provide.

D. Broader Deployment Considerations

Beyond the algorithmic and computational questions discussed above, embedding a continuously-adapting model into the control plane of a shared, multi-tenant storage fabric raises deployment considerations that are somewhat distinct from those of a purely research-oriented evaluation. Multi-tenant NVMe-oF fabrics typically enforce isolation between tenants at the zoning and access-control level; a routing controller that dynamically reshapes WECMP weights in response to telemetry must do so without allowing one tenant's traffic pattern to indirectly influence the path treatment of another tenant's flows in a way that violates isolation guarantees or service-level agreements. Because the Liquid-SAN controller in this paper operates on an aggregate, per-path telemetry vector $I(t)$ rather than per-tenant state, its weight adjustments are tenant-agnostic by construction, which limits the risk of cross-tenant interference but also means the current design cannot express tenant-specific routing preferences or priority tiers. Extending the architecture with per-tenant or per-priority liquid states—effectively partitioning $x(t)$ along tenant boundaries—is a natural extension, but it was outside the scope of the single-tenant-equivalent workload mix evaluated in Section IV-B, and the isolation and fairness properties of such an extension

would themselves need dedicated evaluation before deployment in a shared fabric.

E. Safety Mechanisms for Continuous Adaptation (New in This Revision)

A reviewer of the original submission correctly noted that a controller which continuously and autonomously reshapes production routing state should be discussed in terms of the guardrails around it, not only its adaptation speed. The results in Section V demonstrate the LNN's adaptation behavior under the two tested disturbances, but the architecture as described in Section III does not yet include explicit safety mechanisms, and none were exercised in the simulation. We discuss, at the design level, the mechanisms we believe a production-oriented version of Liquid-SAN would need; validating them is future work, not a result reported in this paper.

Rate limiting on weight churn: Because the LNN can in principle emit a new WECMP weight vector on every telemetry cycle (Section III-A), an uncapped controller could push updates fast enough to destabilize switch ASIC control-plane resources or cause flow-hash churn that itself degrades performance. A production deployment should rate-limit the frequency and magnitude of weight changes per path, analogous to route-flap damping in traditional routing protocols.

Hysteresis around the readout threshold: The worked example in Section III-D shows the liquid state contracting and relaxing smoothly, but readout-layer decisions that map continuous state to discrete WECMP weight buckets are still vulnerable to oscillation if the input hovers near a decision boundary. Applying hysteresis to the readout mapping—requiring a state change to persist for a minimum dwell time, or to cross a threshold by a margin, before a new weight is committed—would reduce the risk of the kind of rapid oscillation the LSTM baseline exhibited in Section V-A, and is a direct architectural analogue of the debounce logic already common in link-state routing implementations.

Bounded policy constraints: The readout layer's output should be clipped to a bounded, pre-approved range of WECMP weight adjustments rather than allowed to command arbitrary path weights, so that a mispredicted or out-of-distribution LNN output cannot fully de-weight a path or otherwise create a routing black hole. Bounding the action space also makes the controller's worst-case behavior easier to reason about during change-control review (Section VI-B).

Anomaly detection and fallback routing: A supervisory monitor, independent of the LNN itself, should track readout outputs and key telemetry inputs for signs that the controller has entered an unstable or out-of-distribution regime—for example, oscillating weight updates of the kind observed in the LSTM baseline, or telemetry values far outside the training distribution. On detecting such a condition, the system should be able to roll back to the last known-good WECMP weight set or fall back to static FSPF routing (Section IV-D) for the affected paths until the anomaly clears or an operator intervenes, providing a safety net structurally similar to the reactive fallback already used for unpredicted events in comparable predictive-routing literature.

None of these mechanisms were implemented or evaluated in the simulation reported in Section V; we present them here as a concrete design target for a production-oriented Liquid-SAN, and as a companion piece to the scalability, hardware-validation, and multi-seed evaluation items already listed as future work in Section VII.

VII. LIMITATIONS AND THREATS TO VALIDITY

In addition to the architectural limitations discussed in Section VI-C, several methodological limitations bear on how the results in Section V should be interpreted. We list these in the same place as in the original submission, but Sections I and Abstract now flag the single-run, simulation-only nature of the evidence upfront as well, per reviewer feedback, rather than only here.

Simulation-only validation, with a release intention.

All experimental results reported in this paper were obtained using a custom, unreleased discrete-event network simulator (Section IV-A) rather than a physical NVMe-oF fabric or production hardware. The simulator models switch buffering, PFC pause-frame propagation, and ISL-level congestion, but it necessarily abstracts away effects such as ASIC-specific queuing behavior, driver and OS scheduling jitter, and physical transceiver failure modes. Because the simulator has not been publicly released, independent reproduction of the reported results is not currently possible. We intend to open-source the simulator in a future release to enable independent reproduction; we are not yet in a position to commit to a specific repository or date, and validating the approach against a physical Fibre Channel or RoCEv2 testbed remains necessary future work in either case. Relatedly, the simulation-projected inference timings discussed in Section VI-A—the approximately 2.5ms forward-pass estimate and the 10ms polling interval—were not benchmarked on physical inference hardware and should be read as illustrative modeling assumptions rather than validated performance guarantees; we call this out prominently here and in Section VI-A because it directly bounds the practical, real-time-control-loop interpretation of the 12ms adaptation time reported in Section V-A.

Single-run results. The latency, throughput, and adaptation-time figures reported in Section V and Table I—including the 12ms adaptation time—reflect a single simulation run rather than averages over multiple random seeds or repeated trials. No confidence intervals or statistical significance tests are reported, and we cannot rule out that results would vary under different random initializations, workload seeds, or mutation timings. We did not repeat the experiment for this revision because doing so credibly requires access to the same simulator and compute environment used for the original run, which was outside the scope of what could be completed for this revision cycle; establishing statistical robustness by repeating each scenario across at least 5–10 random seeds and workload realizations, and reporting means, variance, confidence intervals, and significance tests as requested by reviewers, is the single highest-priority item of future work for this line of research.

Baseline coverage. The LSTM baseline used for comparison (Section IV-D) is a single representative discrete-time recurrent model, frozen rather than retrained during the

test window (Section IV-D), and may not represent the strongest possible adaptive alternative. We did not evaluate against congestion-aware load balancing (e.g., CONGA-style [4] mechanisms), online-learning routing controllers, reinforcement-learning-based adaptive routing agents, or topology-aware GNN baselines such as RouteNet [6]—all of which are also designed to respond to non-stationary network conditions and could narrow, or in some cases exceed, the reported performance gap. Adding these specific baselines, on the same topology and workload, is an important and concrete direction for future work.

No ablation of the liquid-time-constant mechanism.

We did not conduct ablation studies isolating which parts of the architecture drive the reported gains. In particular, we did not compare the full input-dependent liquid-time-constant formulation of Section III-B against a fixed-time-constant variant of the same network, did not vary which telemetry inputs (queue depth, byte rate, PFC rate) are made available to the model, did not sweep the telemetry polling interval, and did not test multiple failure-severity levels beyond the two disturbances in Section V. Without these ablations, we cannot yet attribute the 88% tail-latency reduction specifically to the liquid-time-constant mechanism as opposed to other aspects of the architecture (e.g., the choice of telemetry features or the WECMP action space). Running this ablation suite on the same simulator and workload is future work.

Vendor scope. The system design and telemetry/management integration described in this paper are oriented around a specific vendor ecosystem (Brocade Fabric OS commands and the SANnav management platform); Section III-E clarifies which parts of the design were actually exercised by the simulator versus described conceptually. The generalizability of the reported results to other Fibre Channel or Ethernet/RoCEv2 vendor implementations has not been evaluated and remains an open question.

Controller scalability. The evaluated topology consists of 4 spine switches, 16 leaf switches, 128 initiators, and 64 targets (Section IV-A), a scale representative of a mid-sized enterprise fabric but considerably smaller than the largest hyperscale deployments, which may involve hundreds of switches and tens of thousands of endpoints. The centralized controller architecture described in Section III processes a global telemetry vector $I(t)$ whose dimensionality grows with the number of monitored ports; we have not evaluated how solver latency, readout-layer cost, or WECMP table update time scale as this input dimensionality grows well beyond the tested topology, and it remains an open question whether the centralized design would need to be partitioned across multiple controller instances at larger scale.

Reference accuracy. As part of preparing this revision, we independently re-verified all 17 references against Crossref, dblp, and Semantic Scholar. Thirteen were already accurate; four contained small inaccuracies—a specification version/year mismatch ([2]), an issue-number/article-number conflation ([5]), two author-initial errors ([6]), and one mistitled vendor document ([12])—and have been corrected in the reference list.

VIII. CONCLUSION

The transition to NVMe-oF has exposed the limitations of static routing protocols in highly volatile data center environments. This paper introduced Liquid-SAN, a novel traffic engineering framework powered by Liquid Neural Networks. By leveraging continuous-time ODEs and dynamic, input-dependent time constants, the LNN architecture modeled the fluid dynamics of fabric congestion in the tested simulation scenarios.

Our simulation results suggest that Liquid-SAN can autonomously adapt to severe topological mutations—such as multi-switch failures and massive VM migrations—without requiring offline retraining. In the tested scenarios, the proposed framework reduced microburst-induced tail latencies by 88% and substantially reduced PFC-induced fabric backpressure compared to traditional FSPF routing. It also showed resilience to out-of-distribution data relative to the discrete-time LSTM baseline, which—as clarified in Section IV-D—was evaluated in its frozen, pre-trained form without an opportunity to retrain, consistent with the retraining-gap comparison this paper targets. As discussed in Section VII, these findings are based on a single, unreleased simulator and a single simulation run, use a limited set of baselines, and have not been decomposed via ablation studies; they should be validated on physical hardware, across multiple seeds, against stronger and more numerous adaptive baselines, and with the safety mechanisms outlined in Section VI-E, before drawing broader conclusions. With those caveats stated plainly rather than deferred to the end, the zero-retraining capability of LNNs appears to be a promising and worthwhile direction toward more autonomous, self-healing storage fabrics.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable feedback. This work was supported in part by research grants from the storage networking industry consortium.

REFERENCES

- [1] Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, “NVMe-over-Fabrics Performance Characterization and the Path to Low-Overhead Flash Disaggregation,” in *Proc. 10th ACM Int. Systems and Storage Conf. (SYSTOR)*, 2017, Art. no. 16.

- [2] NVM Express, Inc., “NVMe over Fabrics (NVMe-oF) Specification 1.1,” *Technical Report*, 2019.
- [3] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. Haj Yahia, and M. Zhang, “Congestion Control for Large-Scale RDMA Deployments,” in *Proc. ACM SIGCOMM*, 2015, pp. 523–536.
- [4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, “CONGA: Distributed Congestion-Aware Load Balancing for Datacenters,” in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.
- [5] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, “A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities,” *Journal of Internet Services and Applications*, vol. 9, no. 1, Art. no. 16, 2018.
- [6] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, “RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.
- [7] P. Goyal et al., “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour,” *arXiv preprint arXiv:1706.02677*, 2017.
- [8] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, “Liquid Time-constant Networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 7657–7666, 2021.
- [9] K. Funahashi and Y. Nakamura, “Approximation of Dynamical Systems by Continuous Time Recurrent Neural Networks,” *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.
- [10] Broadcom Inc., “Brocade Fabric OS Command Reference Manual,” *Technical Documentation*, v9.2, 2024.
- [11] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, “RDMA over Commodity Ethernet at Scale,” in *Proc. ACM SIGCOMM*, 2016, pp. 202–215.
- [12] Broadcom Inc., “Brocade SANnav Management Portal User Guide,” *Technical Documentation*, v2.3, 2024.
- [13] D. Silver et al., “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, pp. 484–489, 2016.
- [14] V. Mnih et al., “Human-level Control through Deep Reinforcement Learning,” *Nature*, vol. 518, pp. 529–533, 2015.
- [15] N. McKeown et al., “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [16] P. Bosshart et al., “P4: Programming Protocol-Independent Packet Processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
- [17] M. Lechner, R. Hasani, A. Amini, T. Henzinger, D. Rus, and R. Grosu, “Neural Circuit Policies Enabling Auditable Autonomy,” *Nature Machine Intelligence*, vol. 2, pp. 642–652, 2020.