Shape-Preserving Networks: Encoding Structural Knowledge through Derivative Constraints and Nested Integration

Miguel Suau*, Alec Edwards, Alex Durkin, Ian Howell, Giuseppe Pinto, Marc Artola, Daniel Jarne, Sayam Kumar
Phaidra
*miguel.suau@phaidra.ai

ABSTRACT

We present *Shape-Preserving Networks* (SPNs), a neural network architecture that guarantees monotonicity, curvature, and interaction structure by construction through nested integration. Unlike Physics-Informed Neural Networks (PINNs), where physics is enforced by embedding the governing physical laws as penalties directly into the loss function during training and hence provide no guarantees, or through explicit inequality constraints, SPNs encode derivative constraints directly into the forward pass, ensuring global satisfaction across the entire domain. The architecture decomposes functions into univariate and pairwise terms, constructing each from its derivatives with prescribed signs. We evaluate SPNs on synthetic benchmarks, a real-world chiller power prediction task, and a pendulum world model for model-based control. Results demonstrate physically plausible out-of-distribution extrapolation, improved robustness to outliers, and greater stability across training runs compared to standard networks, while the additive decomposition provides interpretable components that align with known physics.

ACM Reference Format:
Miguel Suau*, Alec Edwards, Alex Durkin, Ian Howell, Giuseppe Pinto, Marc Artola, Daniel Jarne, Sayam Kumar. 2026. Shape-Preserving Networks: Encoding Structural Knowledge through Derivative Constraints and Nested Integration. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 12 pages.

1 INTRODUCTION

Data-driven models are widely used for modelling complex systems, yet they offer no guarantees on how they behave outside of their training distributions. For purely predictive tasks, where new inputs are drawn from the same distribution as the training data, this lack of guarantees is largely acceptable since prediction accuracy is the primary objective, and how the model achieves it is secondary. However, many real-world applications are interventional in nature; the model is used to evaluate actions or operating conditions that have not been observed before. In such settings (for example, building climate control, power grid optimisation, ecosystem management) predictions in unseen operating regimes must remain physically plausible [5]. A model that predicts decreasing cooling power at higher temperatures, or one that predicts a decrease in speed when the applied force is increased, can lead to dangerous decisions. Standard neural networks offer no built-in mechanism to prevent such behaviour.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

Physics-based models, grounded in first-principles formulations such as governing differential equations and conservation laws, are renowned for their high fidelity and strong extrapolative capabilities. However, their practical deployment is often hindered by the need for extensive system-specific knowledge, significant modelling effort, and a substantial computational burden. Moreover, for many systems, tractable governing equations simply do not exist.

In practice, however, even when a complete physics model is unavailable, we often possess qualitative knowledge about how the inputs affect the outputs. We know that certain quantities increase together, that certain effects saturate, or that two factors contribute independently. This kind of structural knowledge (monotonicity, curvature, and interaction structure) can guide generalisation into unseen regimes if it can be reliably encoded into the model.

Consider a pendulum. Without writing down equations, we know that gravity and torque contribute independently; that the gravitational restoring force grows as the pendulum tilts from vertical but the rate of increase diminishes (the force saturates near the horizontal), giving a concave shape; that more torque always produces more acceleration; and that at the vertical positions the gravitational torque is exactly zero. This structural information (monotonicity, curvature, interaction structure, and known anchor values) should constrain a learned model, yet a standard neural network cannot guarantee any of these properties.

Physics-informed machine learning (PIML) has made significant progress in encoding physical knowledge into models [6, 11]. Most existing approaches, however, assume access to governing equations that can be expressed in closed form (e.g., partial differential equations or conservation laws) or enforce constraints through penalised loss terms or constrained optimisation. These strategies complicate training and typically only guarantee constraint satisfaction at a finite set of collocation points rather than globally. An alternative is to encode physical knowledge directly into the network architecture, as in Hamiltonian neural networks [3] or specialised monotonic networks [15, 16], but each is tailored to a particular invariant and none scales naturally to the broad range of qualitative structural knowledge described above.

Contributions. We introduce *Shape-Preserving Networks* (SPNs), an architecture that encodes derivative constraints directly into the forward pass via nested integration. The key idea is that a smooth function can be systematically constructed from its derivatives; by designing those derivatives to have the correct sign, the resulting function inherits the desired shape by construction. Building on unconstrained monotonic networks [15], which integrate over strictly positive functions to guarantee monotonicity, we extend the approach to curvature and to pairwise interactions through nested integration. Additionally, anchor points allow the output

to pass through known (input, output) pairs exactly while preserving all shape properties. Monotonicity, curvature, interaction signs, and anchor constraints are guaranteed globally within the prespecified domain, without constrained optimisation or penalty terms, thereby removing the burden from the training process entirely. We demonstrate SPNs on synthetic benchmarks, a real-world chiller power prediction task, and a pendulum world model for model-based control.

Our experiments show that encoding shape knowledge provides several practical advantages: physically plausible out-of-distribution extrapolation where standard networks diverge, improved robustness to outliers, greater stability across training runs, and enhanced interpretability through the additive decomposition, where individual components can be inspected and compared with known physics.

2 RELATED WORK

The drive to harness the complementary strengths of physics-based and data-driven modelling has led to the rapid expansion of PIML [6, 11]. This integration is typically achieved through several synergistic strategies. A common approach, pioneered by the development of Physics-Informed Neural Networks (PINNs), involves incorporating governing equations, such as differential equations or conservation laws, into the model's loss function, penalising deviations from known physical behaviour during training [5, 6, 12]. Alternatively, models can be designed with architectures or prior distributions that inherently respect physical structures, such as specific symmetries or invariants [9]. Physical constraints can thus be enforced either softly, through regularisation, or strictly, via constrained optimisation and specialized architectural design [1]. By embedding these laws, PIML models generally maintain physical consistency and generalise better than purely data-driven approaches.

Despite the successes of PIML, enforcing qualitative structural knowledge, such as strict monotonicity, concavity, or specific nonlinear equality and inequality constraints, remains a persistent challenge. Historically, efforts to embed such knowledge have often relied on formulating the training process as a constrained optimisation problem or utilizing specialised search algorithms [7, 14]. More recently, an array of specialised neural network architectures, lattice networks, and projection methods have been proposed to enforce strict structural conditions, particularly monotonicity, to ensure model reliability [8, 13, 16]. Advanced projection methods, such as those solving Karush-Kuhn-Tucker (KKT) conditions, have also been introduced to enforce linear and nonlinear constraints up to machine precision [4].

Whilst mathematically sound, enforcing hard constraints through optimisation or complex projection significantly complicates the training landscape. Sequential projection approaches often incur high computational costs due to repetitive projection operations, while simultaneous projection approaches are limited in scalability because the neural network components induce dense linear algebra blocks [10]. These methods increase computational overhead and frequently require careful, often manual, balancing of data and physics residuals during the training process [4]. Furthermore, hard-constrained PCML can be highly sensitive to noisy or biased measurements, and many of these optimisation approaches only guarantee constraint satisfaction at specific collocation points, rather than globally across the entire continuous domain [10].

To circumvent these optimisation bottlenecks, some recent frameworks have begun embedding physics directly into the network's forward pass via its derivatives. A notable example is Hamiltonian Neural Networks (HNNs), which enforce strict energy conservation by predicting a scalar energy function and deriving the system's dynamics through its exact gradients [3]. Unconstrained monotonic networks [15] introduced the idea of integrating over strictly positive functions (via softplus) to guarantee monotonicity by construction. The SPN architecture proposed in this work builds on the monotonic networks approach, extending it to curvature and to pairwise interactions through 2D nested integration.

3 SHAPE-PRESERVING NEURAL NETWORK

We present the SPN formulation in the continuous domain, which provides a clear and intuitive understanding of how shape properties are preserved by construction. In Section 3.4, we then describe a practical implementation that discretizes this formulation on a finite grid. Figure 1 provides a visual overview of the architecture.

The architecture builds on monotonic networks [15], which integrate over strictly positive functions. Case 1 (Section 3.2) follows this directly; Cases 2–3 extend it to curvature via nested integration, and pairwise terms (Section 3.3) handle interactions.

3.1 Overview

Definition 3.1. The Shape-Preserving Neural Network (SPN) encodes sign constraints on the Jacobian and Hessian directly into the network structure so that they hold by construction, regardless of the training data. It decomposes the output function into a sum of univariate and pairwise terms:

$$f(\mathbf{x}) = \sum_i U_i(x_i) + \sum_{i<j} P_{ij}(x_i, x_j) \tag{1}$$

where $f\colon \mathcal{X} \to \mathbb{R}$ with $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X} \subset \mathbb{R}^n$. Each $U_i(x_i)$ is a univariate term controlling per-variable monotonicity and curvature, and each $P_{ij}(x_i, x_j)$ is a pairwise term controlling cross-variable interactions.

This additive structure allows for control over the signs of the first and second derivatives of the output function f. In particular,

- **Jacobian signs** (first derivatives, monotonicity): $\frac{\partial f}{\partial x_i} > 0$ or $\frac{\partial f}{\partial x_i} < 0$
- **Hessian diagonal** (curvature): $\partial^2 f / \partial x_i^2 > 0$ (convex) or < 0 (concave)
- **Hessian off-diagonal** (interaction): $\partial^2 f / \partial x_i \partial x_j > 0$ or < 0

In addition to derivative constraints, the SPN supports *anchor points* (Section 3.6): known (input, output) pairs that the function must pass through exactly, such as boundary conditions or known equilibria.

Next, we describe the construction of the univariate terms, followed by the pairwise terms.

3.2 Univariate Terms

Each univariate component $U_i(x_i)$ controls the first and second derivatives of f with respect to x_i.

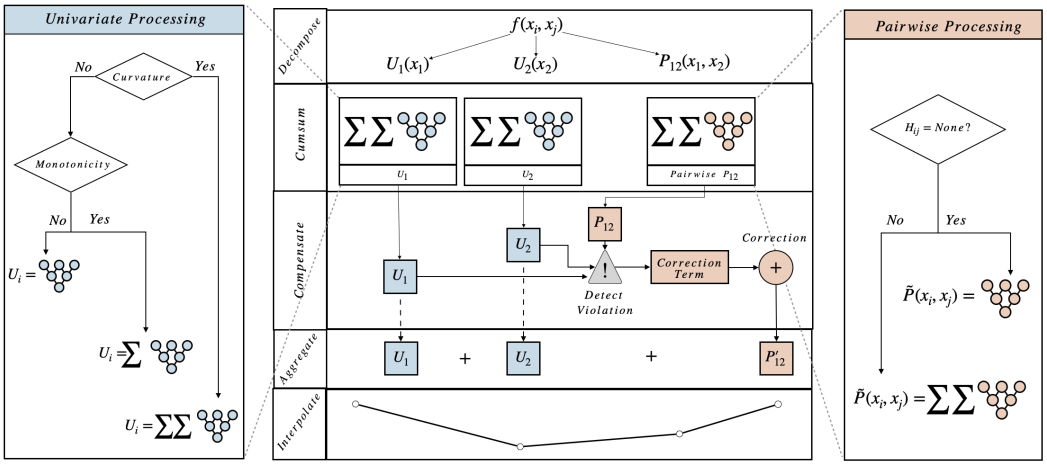


Figure 1: Visual representation of the SPN architecture, showing the univariate and pairwise processing pipelines.

Let $J_i \in \{+1, -1, \text{None}\}$ and $H_{ii} \in \{+1, -1, \text{None}\}$ denote the desired signs of the first and pure second derivatives. Let $g_i : \mathbb{R} \to \mathbb{R}$ be a neural network, σ a strictly positive activation (e.g. softplus), and b_0, b_1 learnable biases. When both J_i and H_{ii} are None, no structure is imposed. The remaining cases are:

Case 1: Monotonicity only ($J_i \in \{+1, -1\}$, H_{ii} = None). When only a monotonicity constraint is imposed, the univariate term is constructed as a single integral:

$$U_i(x_i) = b_0 + \int_0^{x_i} J_i \cdot \sigma(b_1 + g_i(t))\, dt. \tag{2}$$

The integrand $J_i \cdot \sigma(\cdot)$ has a definite sign, so the resulting function is monotonically increasing ($J_i = +1$) or decreasing ($J_i = -1$).

Case 2: Curvature only (J_i = None, $H_{ii} \in \{+1, -1\}$). When only convexity or concavity is enforced, a nested integral controls the monotonicity of the first derivative:

$$U_i(x_i) = b_0 + \int_0^{x_i} \left(b_1 + \int_0^{t} H_{ii} \cdot \sigma(g_i(s))\, ds \right) dt. \tag{3}$$

The inner integral accumulates terms of sign H_{ii}, making the first derivative monotonically increasing ($H_{ii} = +1$, convex) or decreasing ($H_{ii} = -1$, concave), while the function itself is free to be non-monotonic.

Case 3: Monotonicity and curvature ($J_i \in \{+1, -1\}, H_{ii} \in \{+1, -1\}$). When both monotonicity and curvature constraints are imposed, the construction nests both mechanisms:

$$U_i(x_i) = b_0 + \int_0^{x_i} J_i \sigma\left(b_1 + J_i \int_0^{t} H_{ii} \sigma(g_i(s))\, ds \right) dt. \tag{4}$$

The inner integral encodes the curvature constraint as in Case 2. The outer σ wrapped with J_i then ensures that the integrand has a definite sign, encoding the monotonicity constraint as in Case 1. Together, both constraints hold by construction.

3.3 Pairwise Interaction Terms

The pairwise component $P_{ij}(x_i, x_j)$ controls the mixed partial derivative $\frac{\partial^2 f}{\partial x_i \partial x_j}$ (Hessian off-diagonal). Let $H_{ij} \in \{+1, -1, \text{None}\}$ denote the desired sign of this mixed derivative, and let $g_{ij} : \mathbb{R}^2 \to \mathbb{R}$ be a continuous function of both variables (in practice, a neural network). As before, $b_1 \in \mathbb{R}$ is a learnable bias.

Case 1: Constrained interaction ($H_{ij} \in \{+1, -1\}$). The pairwise term uses nested integration to encode the sign constraint on the mixed partial:

$$P_{ij}(x_i, x_j) = \int_0^{x_i} \sigma\left(b_1 + H_{ij} \int_0^{x_j} \sigma(g_{ij}(s, t))\, ds \right) dt. \tag{5}$$

The inner integral accumulates strictly positive values scaled by H_{ij}, and the outer σ ensures positivity of the outer integrand. As a result, the mixed partial derivative $\frac{\partial^2 P_{ij}}{\partial x_i \partial x_j}$ inherits the sign of H_{ij}.

Case 2: Free interaction (H_{ij} = None). The pairwise term is evaluated directly:

$$P_{ij}(x_i, x_j) = g_{ij}(x_i, x_j). \tag{6}$$

No integration or positive-definite wrapping is applied; g_{ij} directly outputs the interaction surface and the mixed partial is completely free.

No interaction ($H_{ij} = 0$). When $H_{ij} = 0$, no pairwise component is created for the (x_i, x_j) pair and the function is purely additive in those variables: $f(x_i, x_j) = U_i(x_i) + U_j(x_j)$.

3.4 Discrete Implementation

In practice, the integrals over g cannot be computed in closed form when g is a neural network. We approximate them via left Riemann sums on a discrete grid.

3.4.1 Domain Discretization. We discretize each input into a grid $\{x_1, \ldots, x_N\}$ with spacings $\delta_k = x_k - x_{k-1}$ ($\delta_1 = 0$). Each integrand value is multiplied by δ_k, so the discrete sum converges to the integral as the grid is refined, making the model *resolution-invariant*.

3.4.2 Univariate Discretization. When curvature is constrained ($H_{ii} \neq$ None), the discretization proceeds as follows:
(1) **Compute base values:** $g_k = \sigma(g(x_k)) \geq 0$
(2) **Apply Hessian sign:** $h_k = H_{ii} \cdot g_k$
(3) **Inner cumulative sum** (first derivative):

$$S_k = \sum_{j=1}^{k} h_j \cdot \delta_j \tag{7}$$

When curvature is free ($H_{ii} =$ None), steps 1–3 are replaced by $S_k = g(x_k)$ directly. The remaining steps apply in both cases:
(4) **Apply Jacobian sign** (monotonicity):

$$\phi_k = \begin{cases} J_i\, \sigma(b_1 + J_i S_k) & J_i \neq \text{None} \\ b_1 + S_k & \text{otherwise} \end{cases} \tag{8}$$

(5) **Outer cumulative sum** (function values):

$$U_{i,k} = b_0 + \sum_{j=1}^{k} \phi_j \cdot \delta_j \tag{9}$$

3.4.3 Pairwise Discretization. For constrained interactions ($H_{ij} \in \{+1, -1\}$), the two domains are discretized as $\{x_u^{(i)}\}$ and $\{x_v^{(j)}\}$ with step sizes $\delta_u^{(i)}$, $\delta_v^{(j)}$:
(1) **Evaluate 2D network:** $G_{u,v} = g(x_u^{(i)}, x_v^{(j)})$
(2) **Apply softplus:** $G^+_{u,v} = \sigma(G_{u,v}) \geq 0$
(3) **Inner cumulative sum** (over x_j):

$$S_{u,v} = H_{ij} \sum_{k=1}^{v} G^+_{u,k} \cdot \delta_k^{(j)} \tag{10}$$

(4) **Apply outer softplus:** $T_{u,v} = \sigma(b_1 + S_{u,v})$
(5) **Outer cumulative sum** (over x_i):

$$P_{u,v} = \sum_{k=1}^{u} T_{k,v} \cdot \delta_k^{(i)} \tag{11}$$

For free interactions ($H_{ij} =$ None), no integration is needed: $P_{u,v} = g(x_u^{(i)}, x_v^{(j)})$ directly.

3.4.4 Interpolation. Since query points may not coincide with grid nodes, we interpolate between neighboring grid values. For a univariate term U_i, given a query point x with $x_k \leq x < x_{k+1}$, we define $\alpha = (x - x_k)/(x_{k+1} - x_k)$ and compute:

$$\hat{U}_i(x) = (1 - \alpha)\, U_{i,k} + \alpha\, U_{i,k+1} \tag{12}$$

For pairwise terms, bilinear interpolation is used analogously. Because the SPN decomposes into univariate and pairwise terms, only 1D and 2D interpolation is ever needed, regardless of the input dimension, avoiding the exponential cost of multilinear interpolation in high dimensions.

Shape preservation. Linear interpolation preserves monotonicity: if $U_{i,k+1} \geq U_{i,k}$ for all k, the interpolated values are also non-decreasing. Discrete convexity is preserved at grid nodes (second differences remain non-negative), but the interpolated function is piecewise-linear between nodes and therefore has zero curvature almost everywhere in the continuous sense. Bilinear interpolation preserves monotonicity along each axis and the sign of the mixed partial within each grid cell. In practice, a sufficiently fine grid (50–100 points per dimension) makes this distinction negligible.

3.5 Compensation Mechanism

When pairwise terms are added to univariate terms, they can violate the univariate shape properties. The compensation mechanism adds correction terms to restore them.

3.5.1 Problem Statement. Consider $f(x_i, x_j) = U_i(x_i) + U_j(x_j) + P_{ij}(x_i, x_j)$. Even if U_i is decreasing in x_i, the pairwise term P_{ij} might be increasing in x_i, causing the total to violate the monotonicity constraint.

3.5.2 Exact Compensation (Iterative). This is the minimal correction strategy. It updates the compensation step-by-step, recomputing the violations after each update.

For each column v (fixed x_j), define the discrete slope along x_i:

$$d_{u,v}^{(i)} = P_{u,v} - P_{u-1,v} \tag{13}$$

At each step u, compute the violation after all previous corrections.

For increasing ($J_i = +1$):

$$\text{violation}_u = \max\Big(0, -\min_v d_{u,v}^{(i)}\Big) \tag{14}$$

For decreasing ($J_i = -1$):

$$\text{violation}_u = \max\Big(0, \max_v d_{u,v}^{(i)}\Big) \tag{15}$$

Then apply a cumulative correction:

$$C_u = C_{u-1} + \text{violation}_u \tag{16}$$

and shift all subsequent points:

$$P_{k,v} \leftarrow P_{k,v} + J_i \cdot C_u \quad \text{for all } k \geq u \tag{17}$$

This guarantees minimal correction but requires iterating over u because each update changes the slopes used to compute later violations.

The same procedure is applied symmetrically along x_j.

3.5.3 Vectorized Approximation. To avoid iterative updates, we compute a single-pass, worst-case correction:

$$\text{violation}_u = \begin{cases} \max\Big(0, -\min_v d_{u,v}^{(i)}\Big), & J_i = +1 \\ \max\Big(0, \max_v d_{u,v}^{(i)}\Big), & J_i = -1 \end{cases} \tag{18}$$

Then integrate once:

$$C_u = \sum_{k=1}^{u} \text{violation}_k \tag{19}$$

and apply:

$$P'_{u,v} = P_{u,v} + J_i \cdot C_u \tag{20}$$

This is guaranteed to satisfy the monotonicity constraint but can be conservative (the correction may be larger than necessary) because it uses the worst-case violation at each step without re-evaluating after corrections.

3.5.4 Curvature Compensation. Curvature compensation follows the same pattern but uses second differences:

$$d_{u,v}^{(2)} = d_{u,v}^{(1)} - d_{u-1,v}^{(1)} \tag{21}$$

The exact version integrates the violation twice with iterative updates. The vectorized version computes a single-pass worst-case violation and applies two cumulative sums.

3.6 Anchor Points

Anchor points allow the SPN output to pass through specific (input, output) pairs exactly while preserving shape properties by construction. The SPN output is a cumulative sum of increments $f(x) = f_0 + \sum_k \phi_k \delta_k$ with ϕ_k of definite sign. A single anchor is satisfied by a shift: $y(x) = f(x) + (y_a - f(x_a))$. Two anchors at domain boundaries use shift and uniform scaling:

$$y(x) = \gamma \cdot (f(x) - f(x_{\min})) + y_{\min}, \quad \gamma = \frac{y_{\max} - y_{\min}}{f(x_{\max}) - f(x_{\min})} \tag{22}$$

Since $\gamma > 0$ (assuming monotonically consistent anchors), scaling preserves both monotonicity and curvature. For 2D SPNs, anchors must be axis-aligned and at domain boundaries; the fixed terms contribute a constant and the varying contribution is scaled uniformly.

When both anchors share the same output value ($y_{\min} = y_{\max}$), a shift combined with a linear correction is used:

$$f(x) = g(x) - g(x_{\min}) + y_{\text{anchor}} + s \cdot (x - x_{\min}) \tag{23}$$

where $s = -(g(x_{\max}) - g(x_{\min}))/(x_{\max} - x_{\min})$. This preserves curvature (the linear term has zero second derivative) but the tilt can violate monotonicity. Accordingly, same-value anchors should only be used when no monotonicity constraint is imposed; when monotonicity is required, the two anchor values must be distinct and monotonically consistent.

3.7 Properties and Guarantees

Table 1 summarizes the full set of shape parameters. All guarantees follow from a single observation: $\sigma(z) > 0$ and $\sigma'(z) > 0$ for all z.

Table 1: Summary of shape parameters and their effects.

Parameter	Effect	Guarantee
$J_i = +1$	$\frac{\partial f}{\partial x_i} \geq 0$	Non-decreasing in x_i
$J_i = -1$	$\frac{\partial f}{\partial x_i} \leq 0$	Non-increasing in x_i
$J_i =$ None	Free	No monotonicity requirement
$H_{ii} = +1$	$\frac{\partial^2 f}{\partial x_i^2} \geq 0$	Convex in x_i
$H_{ii} = -1$	$\frac{\partial^2 f}{\partial x_i^2} \leq 0$	Concave in x_i
$H_{ii} =$ None	Free	No curvature requirement
$H_{ij} = +1$	$\frac{\partial^2 f}{\partial x_i \partial x_j} \geq 0$	Positive interaction
$H_{ij} = -1$	$\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$	Negative interaction
$H_{ij} =$ None	Free	No interaction sign requirement
$H_{ij} = 0$	No interaction	Purely additive

Monotonicity. In the univariate construction, the integrand of the outer integral is $J_i \cdot \sigma(\cdot)$. Since $\sigma(\cdot) > 0$, the integrand has the same sign as J_i, so the integral is monotonically increasing ($J_i = +1$) or decreasing ($J_i = -1$). The compensation mechanism (Section 3.5) restores this when pairwise terms are present.

Curvature. The inner integral accumulates terms $H_{ii} \cdot \sigma(g_i(s))$. Since $\sigma(g_i(s)) > 0$, the first derivative is monotonically increasing ($H_{ii} = +1$, convex) or decreasing ($H_{ii} = -1$, concave). Compensation restores this property when pairwise terms are added.

Interaction sign. For a constrained pairwise term, the mixed partial derivative is:

$$\frac{\partial^2 P_{ij}}{\partial x_i \partial x_j} = \sigma'\Big(b_1 + H_{ij} \int_0^{x_j} \sigma(g_{ij})\, ds\Big) \cdot H_{ij} \cdot \sigma(g_{ij}(x_j, x_i)). \tag{24}$$

Since $\sigma' > 0$ and $\sigma > 0$, the sign is determined entirely by H_{ij}. Compensation corrections are applied uniformly across x_j (effectively univariate in x_i), so their mixed partial is zero and the interaction sign is unchanged.

4 EXPERIMENTS

We evaluate the SPN on three tasks: (i) synthetic function approximation, (ii) real-world chiller power prediction with anomalous data, and (iii) a pendulum world model for model-based control. In all settings we compare against a standard multi-layer perceptron (MLP) of comparable size.

4.1 Function Approximation

4.1.1 Setup. We consider synthetic test functions with known analytical derivatives, allowing us to specify the correct Jacobian and Hessian sign constraints.

1D functions.

- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$, $x \in [-5, 5]$. Monotonically increasing ($J = +1$), curvature unconstrained (changes sign at $x = 0$).
- **Log-utility**: $f(x) = \log(x + 1)$, $x \in [0, 10]$. Monotonically increasing ($J = +1$) and concave ($H_{11} = -1$).

2D functions.

- **Exponential**: $f(x, y) = -e^{-(x+y)}$, $(x, y) \in [0, 5]^2$. Increasing ($J = [+1, +1]$), concave ($H_{\text{diag}} = [-1, -1]$), negative interaction ($H_{12} = -1$). Anchor at the origin.
- **Quadratic**: $f(x, y) = 0.1(x^2 + y^2)$, $(x, y) \in [-3, 3]^2$. No monotonicity ($J = [\text{None}, \text{None}]$), convex ($H_{\text{diag}} = [+1, +1]$), no interaction ($H_{12} = 0$).
- **Cobb–Douglas**: $f(x, y) = x^\alpha y^{1-\alpha}$, $\alpha = 0.3$, $(x, y) \in [0.5, 5]^2$. Increasing ($J = [+1, +1]$), concave ($H_{\text{diag}} = [-1, -1]$), free interaction.

4.1.2 Training and Evaluation. For each function, $N = 1000$ noisy samples ($\sigma = 0.05$) are drawn from the central 50% of the domain (training); the remaining region serves as the OOD test set. Both models are trained for 4000 epochs with Adam (lr $= 10^{-3}$, MSE loss). The SPN uses 50 grid points per dimension and two-hidden-layer g-networks of width 64; the MLP uses matched two-hidden-layer width-64 ReLU networks. Results are mean ± std over 5 seeds.

4.1.3 Results. The SPN uses linear interpolation between grid nodes, which allows it to represent smooth functions in a discrete domain. The MLP generally achieves a lower training MSE on functions with complex curvature (e.g., quadratic), reflecting its greater flexibility, while the SPN matches or even outperforms the MLP on training loss for simpler monotonic functions (log-utility, Cobb–Douglas). Both models show comparable in-distribution test performance, with the SPN achieving lower ID MSE on four of five functions. The key advantage of the SPN emerges in the out-of-distribution region, where the structural constraints (monotonicity,

Table 2: Train, ID and OOD MSE ($\times 10^{-3}$, mean ± std, 5 seeds) for SPN and MLP. Bold = better model.

	Train MSE		ID MSE		OOD MSE	
Function	SPN	MLP	SPN	MLP	SPN	MLP
Sigmoid	2.38 ± 0.10	**2.30 ± 0.09**	**2.45 ± 0.19**	2.55 ± 0.22	**3.01 ± 0.99**	44.60 ± 51.68
Log-utility	**2.39 ± 0.10**	2.41 ± 0.11	**2.45 ± 0.21**	2.46 ± 0.22	**3.34 ± 0.88**	40.16 ± 15.98
Exponential	2.46 ± 0.10	**2.25 ± 0.10**	**2.25 ± 0.45**	2.43 ± 0.52	**12.70 ± 2.04**	23.68 ± 17.78
Quadratic	2.50 ± 0.11	**1.71 ± 0.13**	**2.32 ± 0.48**	3.22 ± 0.45	**53.41 ± 9.57**	78.72 ± 30.35
Cobb–Douglas	**2.38 ± 0.10**	2.38 ± 0.10	2.38 ± 0.49	**2.37 ± 0.45**	**16.20 ± 5.66**	26.00 ± 9.25

curvature, and interaction signs) guide the extrapolation along physically plausible directions. The SPN achieves a lower OOD MSE on all five functions, and the MLP's OOD performance is both worse on average and far more variable across seeds (e.g., sigmoid OOD std of 51.68 vs. 0.99), reflecting the instability of unconstrained extrapolation. The effect is most striking on the 1D functions, where the MLP's OOD MSE is 12–15× worse than the SPN's. Figure 2 illustrates the 1D cases: the SPN extrapolates smoothly beyond the training domain, while the MLP diverges. 3D surface visualizations of the learned functions for all 2D tasks are provided in Appendix A (Figures 5–7).

4.2 Application: Chiller Power Prediction

Modern high-performance computing (HPC) facilities operate under a fixed total power budget that is shared between IT equipment and cooling infrastructure. Chiller plants reject the heat generated by the computing nodes to the outside air, and their power consumption increases with both ambient temperature and IT load. When the outside temperature is low, less cooling power is needed, freeing capacity for additional computation. Accurately predicting chiller power consumption as a function of ambient temperature and IT load therefore enables dynamic power reallocation between computing and cooling, maximizing useful compute while maintaining thermal safety margins.

We apply the SPN and MLP to this real-world function approximation task using facility monitoring data from CINECA's Marconi100 supercomputer [2]. The dataset contains minute-level measurements of ambient temperature, total IT power consumption, and total chiller power consumption over a multi-month period (~690k samples after preprocessing). Crucially, the dataset contains an anomalous period on 2021-08-14 where a chiller plant shut down, causing chiller power to fall to roughly half its expected value (see Figure 8 in the appendix for a time-series view of this event). Such outliers are dangerous for safety-critical prediction: a model that fits them would underestimate chiller power needs during normal operation, potentially leading to thermal failures.

4.2.1 Setup. The SPN encodes two physical relationships: chiller power increases with ambient temperature ($J_1 = +1$) and with IT load ($J_2 = +1$). These monotonicity constraints make the SPN *architecturally* unable to predict lower chiller power at higher temperature or load, regardless of the training data.

4.2.2 Training and Evaluation. Both models are trained for 50 epochs (80/20 split, Adam, lr = 10^{-3}, batch 1024). The SPN uses 100 grid points and g-networks of width 256; the MLP uses two hidden layers of width 256. We report the *Monotonicity Violation Rate* (MVR): the fraction of adjacent predictions violating monotonicity (mean ± std, 5 seeds).

Table 3: Chiller power prediction (mean ± std, 5 seeds). MVR = fraction of slices where monotonicity is violated. The SPN guarantees MVR = 0% by construction.

	Test MSE	R^2	MVR (Temp)	MVR (IT)
SPN	1246.8 ± 9.8	0.875 ± 0.001	**0.00 ± 0.00**%	**0.00 ± 0.00**%
MLP	**1208.9 ± 8.3**	**0.879 ± 0.001**	14.25 ± 2.20%	21.41 ± 1.39%

4.2.3 Results. Table 3 and Figure 3 summarize the results. The MLP achieves marginally lower test MSE (1208.9 vs. 1246.8) precisely *because* it fits the outlier shutdown period, but it violates monotonicity in 14.25% of temperature slices and 21.41% of IT load slices. The SPN achieves 0% MVR by construction. Figure 3 shows that the MLP predicts decreasing chiller power at higher IT load in some regions, which is a dangerous underestimate in safety-critical applications.

4.3 World Model

We apply the SPN to learn transition dynamics $s_{t+1} = f(s_t, a_t)$ for the Pendulum environment. The SPN world model embeds physics-informed structure directly into the network architecture, whereas the MLP baseline learns the same mapping without structural constraints. Both models share the same semi-implicit Euler integration scheme; only the function that predicts velocity increments differs.

4.3.1 Setup. The state is $s = (\theta, \dot{\theta})$ (angle and angular velocity), and the action $a = u$ is a scalar torque.

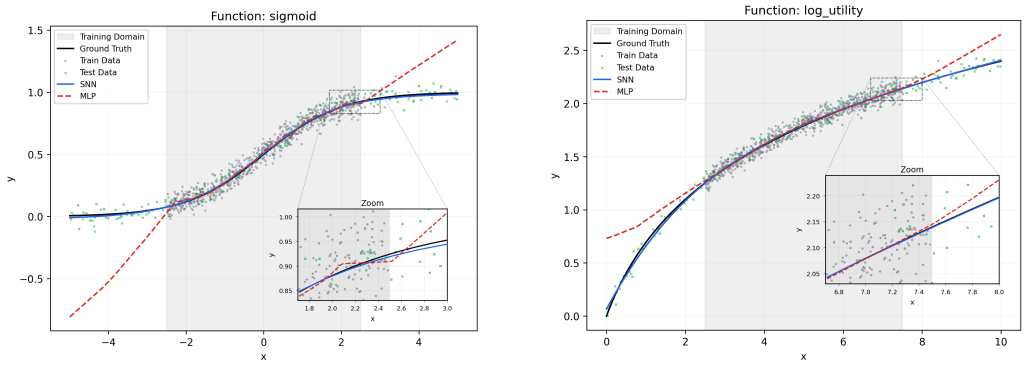
SPN architecture. The SPN world model decomposes the angular acceleration into two additive univariate terms:

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \underbrace{\mathrm{SPN}_{\mathrm{gravity}}(|\theta_t|) \cdot \mathrm{sign}(\theta_t)}_{\text{gravity}} + \underbrace{\mathrm{SPN}_{\mathrm{control}}(u_t)}_{\text{control}} \tag{25}$$

$$\theta_{t+1} = \theta_t + \dot{\theta}_{t+1} \cdot \Delta t \tag{26}$$

where $\Delta t = 0.05$ s.

The gravity component exploits the antisymmetry of the gravitational torque: $\sin(-\theta) = -\sin(\theta)$. By having the SPN operate on $|\theta|$ and multiplying the output by $\mathrm{sign}(\theta)$, we enforce this symmetry by construction. The structural constraints are:



(a) Sigmoid **(b) Log-utility**

Figure 2: SPN vs. MLP predictions on 1D functions. The gray-shaded region indicates the training domain. The SPN extrapolates smoothly outside the training domain, while the MLP diverges.

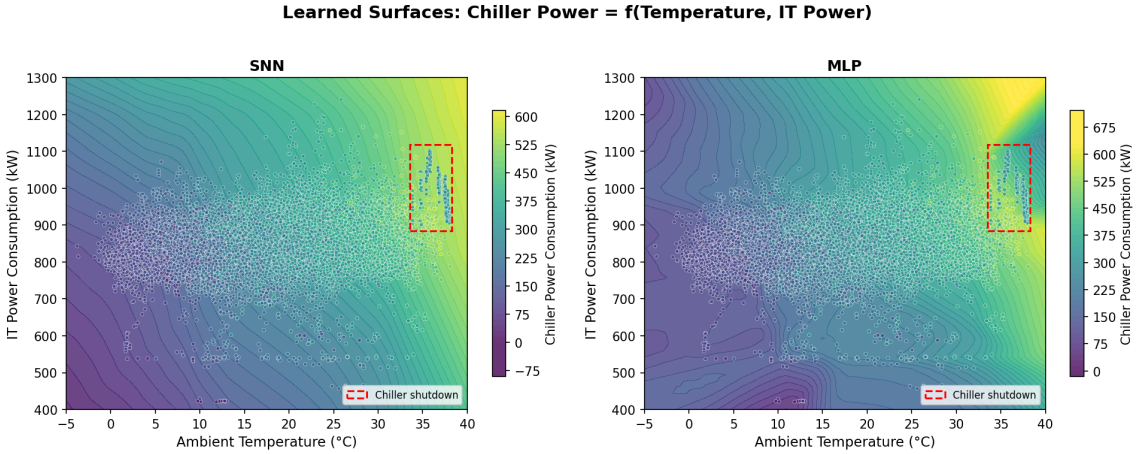


Figure 3: Learned chiller power surfaces. The dashed red box marks the 2021-08-14 shutdown (see Figure 8). The SPN maintains a monotonically increasing surface, while the MLP fits the outlier, producing non-monotonic predictions.

- $\text{SPN}_{\text{gravity}}$: concavity ($H_{11} = -1$), reflecting the fact that $\sin(\theta)$ is concave on $[0, \pi]$. Anchor points at $\theta = 0$ and $\theta = \pi$ with value 0 (no gravitational torque at the vertical positions).
- $\text{SPN}_{\text{control}}$: monotonically increasing ($J = +1$), since more torque produces more angular acceleration. Anchor at $u = 0$ with value 0 (no torque, no acceleration).

MLP baseline. The MLP takes $(\theta, \dot{\theta}, u)$ as input and outputs the velocity increment $\Delta\dot{\theta}$. It uses the same Euler integration scheme as the SPN. Two hidden layers of width 256 with ReLU activations.

4.3.2 Training and Evaluation. We collect 100 episodes of trajectory data using a random policy starting from the bottom position ($\theta = \pi$). Models are trained for 200 epochs with Adam (lr $= 10^{-3}$, batch size 128) using MSE loss on $\dot{\theta}$ only (the position update is deterministic given the velocity). All results are averaged over 5 random seeds.

We evaluate world models along two axes: (i) *single-step prediction accuracy*, measured by MSE on $\dot{\theta}$ for both the training set and a held-out validation set (same distribution), and (ii) *multi-step trajectory prediction*, where the model is rolled out autoregressively for T steps given an initial state s_0 and an action sequence $(a_0, \ldots, a_{T-1})$. We report the mean L2 error between predicted and real state trajectories, averaged over multiple test episodes. Multi-step evaluation is significantly harder than single-step, as errors compound over time.

We consider two evaluation regimes for multi-step prediction:

- **In-distribution (ID):** trajectories collected with the same random policy used for training, starting from $\theta = \pi$ (pendulum hanging down).
- **Out-of-distribution (OOD):** trajectories generated by a CEM planner using the *true environment* as an oracle. These trajectories visit near-upright states ($\theta \approx 0$) that the random training policy rarely reaches, testing whether the learned model generalizes to the states that matter most for control.

Additionally, we evaluate downstream control performance by using each learned model as the world model inside a CEM planner running for 200 closed-loop control steps. At each step, the CEM planner uses the learned model to simulate action sequences, selects the best action, executes it in the true environment, and re-plans from the resulting state. The CEM reward measures the cumulative

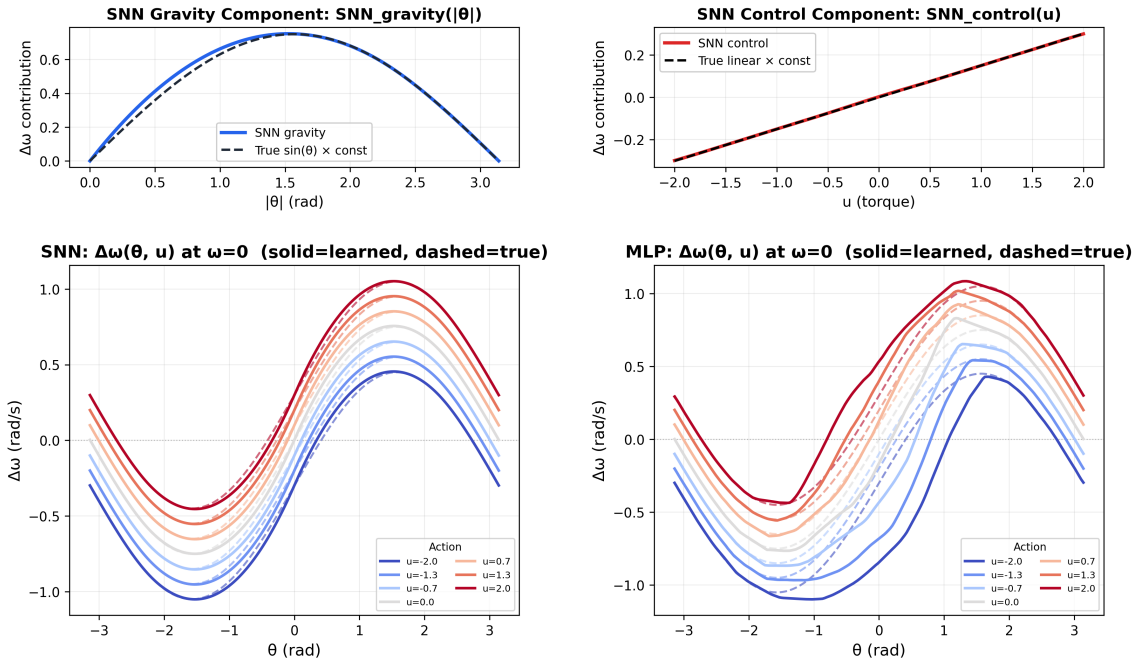


Figure 4: Learned pendulum dynamics. Top left: SPN gravity component recovers the true $\sin(\theta)$ torque with concavity and anchor constraints. Top right: SPN control component learns a monotonically increasing torque-to-acceleration mapping. Bottom: total predicted $\Delta\omega(\theta, u)$ at $\dot{\theta} = 0$ (solid = learned, dashed = true). The SPN yields interpretable, physically consistent components.

reward obtained when the planned actions are executed in the true environment; higher (less negative) reward indicates better swing-up and balancing performance. We also report the Oracle CEM reward, where the planner has access to the true environment dynamics, serving as an upper bound on achievable performance. All results are averaged over 5 random seeds.

Table 4: World model evaluation (mean ± std, 5 seeds). MSE on $\dot{\theta}$ ($\times 10^6$); L2 = mean state distance over rollouts; CEM reward over 200 steps. Oracle uses true dynamics.

Metric	SPN	MLP	Oracle
Train MSE ($\times 10^6$)	**1.31 ± 0.92**	10.75 ± 6.92	—
Test MSE ($\times 10^6$)	**1.32 ± 0.96**	11.66 ± 6.83	—
ID rollout L2	**0.03 ± 0.02**	0.16 ± 0.13	—
OOD rollout L2	**2.78 ± 0.06**	4.11 ± 1.75	—
CEM reward	**−403.1 ± 1.9**	−682.5 ± 127.8	−402.8 ± 2.2

4.3.3 Results. Table 4 summarizes the results. The SPN outperforms the MLP on every metric, with 8× lower single-step MSE and 5× lower ID rollout L2, showing that the physics-informed decomposition reduces error compounding during autoregressive prediction.

The advantage is most pronounced on OOD trajectories: the SPN's rollout L2 is 1.5× lower (2.78 vs. 4.11) with far less variance across seeds (std 0.06 vs. 1.75). On downstream control, the SPN achieves a CEM reward of −403.1, nearly matching the Oracle (−402.8), while the MLP degrades to −682.5 with high variance. The 200-step horizon amplifies model errors, causing the MLP planner to consistently fail at swing-up while the SPN matches Oracle performance. Figure 4 confirms that the learned gravity and control components closely match the true physics.

5 CONCLUSION

We have introduced *Shape-Preserving Networks* (SPNs), a neural network architecture that encodes derivative constraints directly into the forward pass through nested integration, guaranteeing monotonicity, curvature, and interaction structure by construction without constrained optimisation or penalty terms.

Experiments on synthetic benchmarks, chiller power prediction, and a pendulum world model demonstrate that structural knowledge yields physically plausible OOD extrapolation, robustness to outliers, training stability, and interpretability through additive univariate and pairwise components.

At the same time, the shape-preserving construction comes with trade-offs. The reduced flexibility relative to unconstrained networks can lead to a modest increase in in-distribution error on some tasks. Training incurs additional computational overhead, though forward passes can be sped up at inference time by caching. Finally, the current architecture is limited to univariate and pairwise terms, meaning higher-order interactions are not directly captured.

Future work could explore extensions to higher-order interactions, more efficient discretization strategies, and applications to additional domains where qualitative structural knowledge is available but full physics models are not.

REFERENCES

[1] Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine. 2021. Enforcing analytic constraints in neural networks emulating physical systems. *Physical Review Letters* 126, 9 (2021), 098302.

[2] Andrea Borghesi, Carmine Di Santi, Martin Molan, Mohsen Seyedkazemi Ardebili, Alessio Mauri, Massimiliano Guarrasi, Daniela Galetti, Mirko Cestari, Francesco Barchi, Luca Benini, Francesco Beneventi, and Andrea Bartolini. 2023. M100 ExaData: A Data Collection Campaign on the CINECA's Marconi100 Tier-0 Supercomputer. *Scientific Data* 10 (2023), 288. https://doi.org/10.1038/s41597-023-02174-3

[3] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. 2019. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 32.

[4] Ashfaq Iftakher, Rahul Golder, Bimol Nath Roy, and M.M. Faruque Hasan. 2026. Physics-informed neural networks with hard nonlinear equality and inequality constraints. *Computers & Chemical Engineering* 204 (2026), 109418. https://doi.org/10.1016/j.compchemeng.2025.109418

[5] Zixin Jiang, Xuezheng Wang, Han Li, Tianzhen Hong, Fengqi You, Ján Drgoňa, Draguna Vrabie, and Bing Dong. 2025. Physics-informed machine learning for building performance simulation-A review of a nascent field. *Advances in Applied Energy* 18 (2025), 100223. https://doi.org/10.1016/j.adapen.2025.100223

[6] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. 2021. Physics-informed machine learning. *Nature Reviews Physics* 3, 6 (2021), 422–440. https://doi.org/10.1038/s42254-021-00314-5

[7] Herbert Kay and Lyle H. Ungar. 1993. Deriving Monotonic Function Envelopes from Observations. In *Working Papers of the Seventh International Workshop on Qualitative Reasoning about Physical Systems (QR'93)*. Orcas Island, Washington, USA. https://www.qrg.northwestern.edu/papers/Files/qr-workshops/QR93/Kay_1993_Deriving_Monotonic_Function_Envelopes.pdf

[8] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. 2022. Certified Monotonic Neural Networks. arXiv:2011.10219 [cs.LG] https://arxiv.org/abs/2011.10219

[9] M. Mattheakis, P. Protopapas, D. Sondak, M. Di Giovanni, and E. Kaxiras. 2020. Physical Symmetries Embedded in Neural Networks. arXiv:1904.08991 [physics.comp-ph] https://arxiv.org/abs/1904.08991

[10] Angan Mukherjee and Victor M Zavala. 2026. Physics-constrained machine learning for chemical engineering. *Current Opinion in Chemical Engineering* 51 (2026), 101228. https://doi.org/10.1016/j.coche.2026.101228

[11] Truong X. Nghiem, Ján Drgoňa, Colin Jones, Zoltan Nagy, Roland Schwan, Biswadip Dey, Ankush Chakrabarty, Stefano Di Cairano, Joel A. Paulson, Andrea Carron, Melanie N. Zeilinger, Wenceslao Shaw Cortez, and Draguna L. Vrabie. 2023. Physics-Informed Machine Learning for Modeling and Control of Dynamical Systems. In *2023 American Control Conference (ACC)*. 3735–3750. https://doi.org/10.23919/ACC55779.2023.10155901

[12] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707.

[13] Davide Sartor, Alberto Sinigaglia, and Gian Antonio Susto. 2025. Advancing Constrained Monotonic Neural Networks: Achieving Universal Approximation Beyond Bounded Activations. arXiv:2505.02537 [cs.LG] https://arxiv.org/abs/2505.02537

[14] Laurentiu A. Tarca, Bernard P.A. Grandjean, and Faïçal Larachi. 2004. Embedding monotonicity and concavity in the training of neural networks by means of genetic algorithms: Application to multiphase flow. *Computers & Chemical Engineering* 28, 9 (2004), 1701–1713. https://doi.org/10.1016/j.compchemeng.2004.01.003

[15] Antoine Wehenkel and Gilles Louppe. 2021. Unconstrained Monotonic Neural Networks. arXiv:1908.05164 [cs.LG] https://arxiv.org/abs/1908.05164

[16] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. 2017. Deep Lattice Networks and Partial Monotonic Functions. arXiv:1709.06680 [stat.ML] https://arxiv.org/abs/1709.06680

A ADDITIONAL FIGURES

Figures 5–7 show 3D surface plots and corresponding contour plots of the learned functions for the 2D prediction tasks. Each figure displays three panels: the ground truth function, the SPN prediction, and the MLP prediction. Training data (black circles) and test data (red squares) are overlaid. The 3D surfaces illustrate the overall shape of each model's output, while the contour plots reveal finer details such as monotonicity violations and irregular level sets in the MLP predictions. Together, they show that the SPN's structural constraints produce smooth, physically plausible surfaces that closely match the ground truth, even in the out-of-distribution region, whereas the MLP diverges outside the training domain.

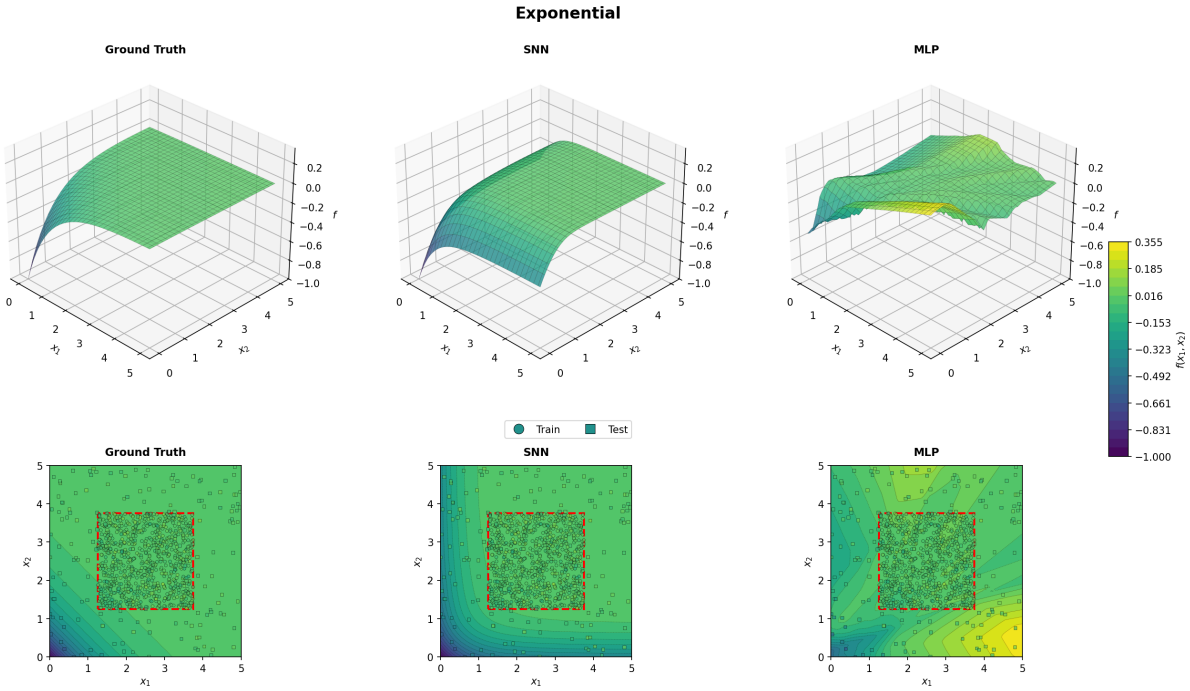


Figure 5: 3D surface plots and contour plots for the Exponential function. The SPN captures the convex structure and extrapolates smoothly, while the MLP produces irregular surfaces outside the training domain.

Figure 8 shows a time-series view of the anomalous chiller shutdown event on 2021-08-14, referenced in Section 4.2. During a roughly four-hour window (~14:14–17:55 UTC), chiller power dropped from approximately 560 kW to 305 kW despite ambient temperatures exceeding 35°C and IT loads remaining above 800 kW. This anomalous period, highlighted by the red-shaded region, corresponds to the dashed red box in Figure 3. A model that fits this outlier (as the MLP does) would dangerously underestimate cooling requirements under normal operation.

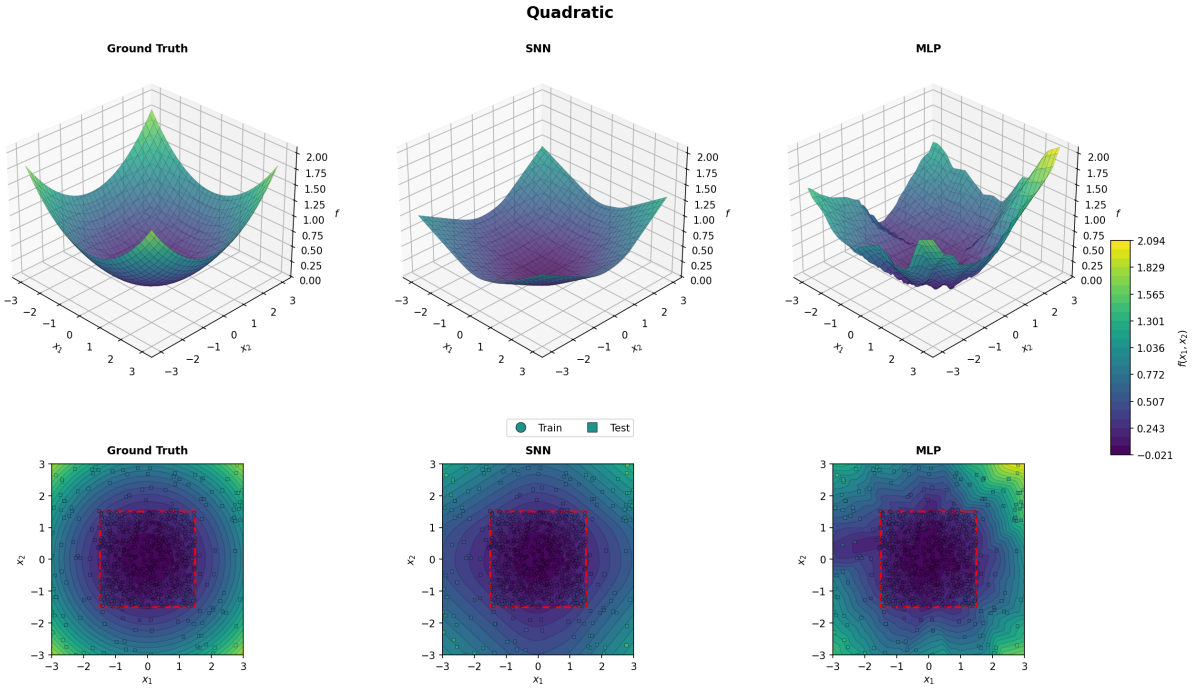


Figure 6: 3D surface plots and contour plots for the Quadratic function. The SPN enforces the correct convexity structure, yielding a smooth bowl shape that closely matches the ground truth even in the OOD region.

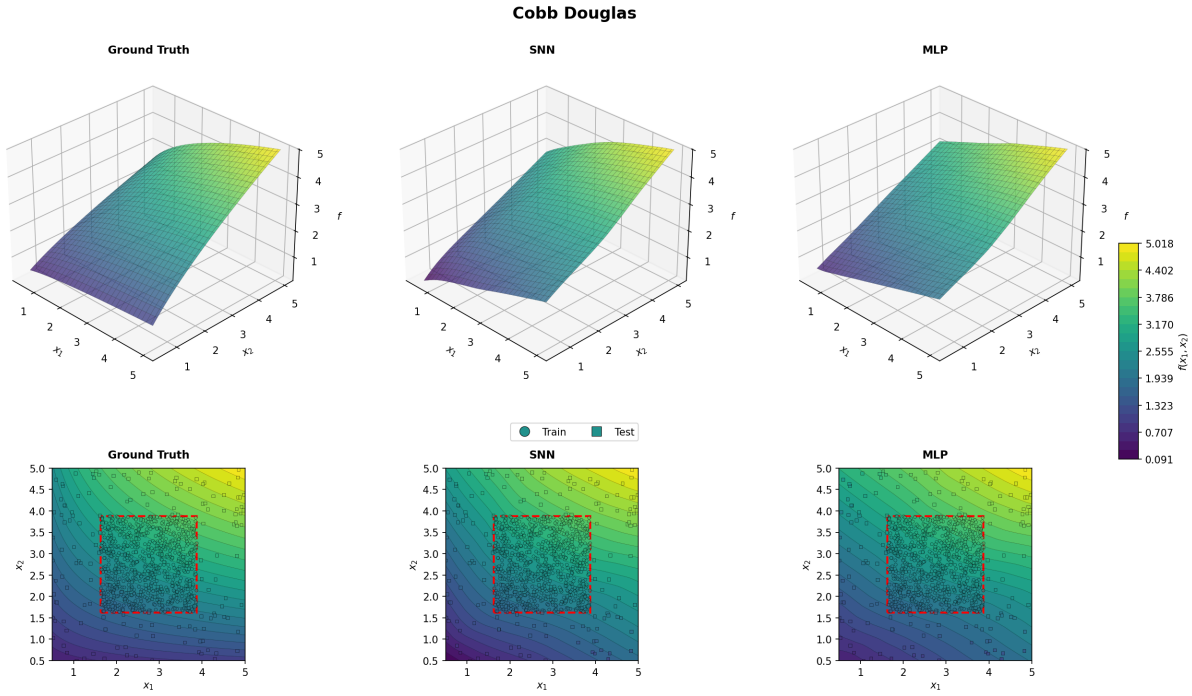


Figure 7: 3D surface plots and contour plots for the Cobb–Douglas function. The SPN's monotonicity and concavity constraints produce a physically plausible surface, whereas the MLP diverges outside the training domain.

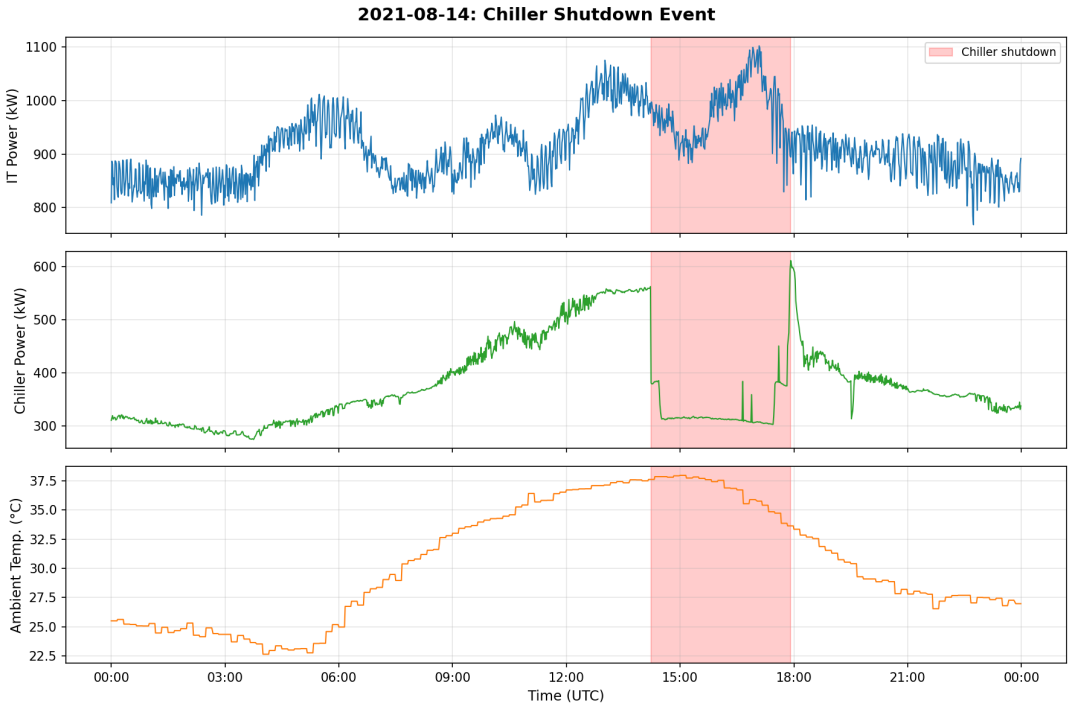


Figure 8: Time-series view of the chiller shutdown event on 2021-08-14. The red-shaded region marks the period (~14:14–17:55 UTC) during which chiller power dropped from ~560 kW to ~305 kW despite ambient temperatures exceeding 35°C and IT loads above 800 kW. This anomalous period corresponds to the dashed red box in Figure 3.