Sprecher Networks: A Parameter-Efficient Kolmogorov-Arnold Architecture

Christian Hägg* Kathlén Kohn† Giovanni Luca Marchetti‡ Boris Shapiro§

December 21, 2025

Abstract

We present *Sprecher Networks* (SNs), a family of trainable neural architectures inspired by the classical Kolmogorov–Arnold–Sprecher (KAS) construction for approximating multivariate continuous functions. Distinct from Multi-Layer Perceptrons (MLPs) with fixed node activations and Kolmogorov-Arnold Networks (KANs) featuring learnable edge activations, SNs utilize shared, learnable splines (*monotonic* and *general*) within structured blocks incorporating explicit shift parameters and mixing weights. Our approach directly realizes Sprecher's specific 1965 "sum of shifted splines" formula in its single-layer variant and extends it to deeper, multi-layer compositions. We further enhance the architecture with optional lateral mixing connections that enable intra-block communication between output dimensions, providing a parameter-efficient alternative to full attention mechanisms. Beyond parameter efficiency with $O(LN + LG)$ scaling (where G is the knot count of the shared splines) versus MLPs' $O(LN^2)$, SNs admit a sequential evaluation strategy that reduces peak forward-intermediate memory from $O(N^2)$ to $O(N)$ (treating batch size as constant), making much wider architectures feasible under memory constraints. We demonstrate empirically that composing these blocks into deep networks leads to highly parameter and memory-efficient models, discuss theoretical motivations, and compare SNs with related architectures (MLPs, KANs, and networks with learnable node activations).

1 Introduction and historical background

Approximation of continuous functions by sums of univariate functions has been a recurring theme in mathematical analysis and neural networks. The Kolmogorov–Arnold Representation Theorem [7, 1] established that any multivariate continuous function $f : [0, 1]^d \to \mathbb{R}$ can be represented as a finite composition of continuous functions of a single variable and the addition operation. Specifically, Kolmogorov (1957) showed that such functions can be represented as a finite sum involving univariate functions applied to sums of other univariate functions of the inputs.

David Sprecher's 1965 construction. In his 1965 landmark paper [10], David Sprecher provided a constructive proof and a specific formula realizing the Kolmogorov–Arnold representation. He showed that any continuous function $f : [0, 1]^n \to \mathbb{R}$ could be represented as:

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \Phi\left(\sum_{p=1}^{n} \lambda_p \, \phi\big(x_p + \eta \, q\big) + q \right) \tag{1}$$

for a single *monotonic* inner function ϕ, a continuous outer function Φ, a constant shift parameter $\eta > 0$, and constants λ_p. This construction simplified the representation by using only one inner function ϕ, relying on shifts of the input coordinates ($x_p + \eta q$) and an outer summation index shift ($+q$) to achieve universality. The key insight of *shifting input coordinates* and summing evaluations under inner and outer univariate maps is central to Sprecher's specific result.

*Department of Mathematics, Stockholm University, Stockholm, Sweden. Email: hagg@math.su.se
†Department of Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden. Email: kathlen@kth.se
‡Department of Mathematics, KTH Royal Institute of Technology, Stockholm, Sweden. Email: glma@kth.se
§Department of Mathematics, Stockholm University, Stockholm, Sweden. Email: shapiro@math.su.se

From a shallow theorem to a deep architecture. Sprecher's 1965 result is remarkable because it guarantees universal approximation with a single hidden layer (i.e., a shallow network). This mirrors the history of Multi-Layer Perceptrons, where the Universal Approximation Theorem also guaranteed the sufficiency of a single hidden layer. However, the entire deep learning revolution was built on the empirical discovery that composing multiple layers, while not theoretically necessary for universality, provides vast practical benefits in terms of efficiency and learnability.

This history motivates our central research question: can the components of Sprecher's shallow, highly-structured formula be used as a new kind of building block in a *deep*, compositional architecture? We propose to investigate this by composing what we term *Sprecher blocks*, where the vector output of one block becomes the input to the next. It is crucial to emphasize that this deep, compositional structure is our own architectural proposal, inspired by the paradigm of deep learning, and is not part of Sprecher's original construction or proof of universality. The goal of this paper is not to generalize Sprecher's theorem, but to empirically evaluate whether this theorem-inspired design is a viable and efficient alternative to existing deep learning models when extended into a deep framework.

Modern context. Recent work has revitalized interest in leveraging Kolmogorov-Arnold representations for modern deep learning. Notably, Kolmogorov-Arnold Networks (KANs) [9] were introduced, proposing an architecture with learnable activation functions (splines) placed on the *edges* of the network graph, replacing traditional linear weights and fixed node activations.

Architectural landscape. Understanding how novel architectures relate to established ones is crucial. Standard Multi-Layer Perceptrons (MLPs) [4] employ fixed nonlinear activation functions at nodes and learnable linear weights on edges, justified by the Universal Approximation Theorem [2, 5]. Extensions include networks with *learnable activations on nodes*, sometimes called Adaptive-MLPs or Learnable Activation Networks (LANs) [3, 11, 9], which retain linear edge weights but make the node non-linearity trainable. KANs [9] represent a more significant departure, moving learnable splines to edges and eliminating linear weights entirely, using simple summation at nodes. Sprecher Networks (SNs), as we detail below, propose a distinct approach derived directly from Sprecher's 1965 formula. SNs employ function blocks containing shared learnable splines (ϕ, Φ), learnable mixing weights (λ), explicit structural shifts (η, q), and optionally, lateral mixing connections for intra-block communication. This structure offers a different alternative within the landscape of function approximation networks.

2 Motivation and overview of Sprecher Networks

While MLPs are the workhorse of deep learning, architectures inspired by KAS representations offer potential benefits, particularly in interpretability and potentially parameter efficiency for certain function classes. KANs explore one direction by placing learnable functions on edges. Our *Sprecher Networks* (SNs) explore a different direction, aiming to directly implement Sprecher's constructive formula within a trainable framework and extend it to deeper architectures.

SNs are built upon the following principles, directly reflecting Sprecher's formula:

- Each functional block (mapping between layers) is organized around a shared *monotonic* spline $\phi(\cdot)$ and a shared *general* spline $\Phi(\cdot)$, both learnable.
- Each block incorporates a learnable scalar shift η applied to inputs based on the output index q.
- Each block includes learnable mixing weights λ_i (a vector, not a matrix) that combine contributions from different input dimensions, with the same weights shared across all output dimensions.
- The structure explicitly includes the additive shift αq inside the outer spline Φ, where α is a scaling factor (typically $\alpha = 1$) that maintains consistency with Sprecher's formulation.
- Optionally, blocks can include lateral mixing connections that allow output dimensions to exchange information before the outer spline transformation, enhancing expressivity with minimal parameter overhead.

Our architecture generalizes this classical single-layer shift-and-sum construction to a multi-layer network by composing these functional units, which we term *Sprecher blocks*. The mapping from one hidden layer representation to the next is realized by such a block. Unlike MLPs with fixed node activations, LANs with learnable node activations, or KANs with learnable edge activations, SNs concentrate their learnable nonlinearity into the two shared splines per block, applied in a specific structure involving shifts and learnable linear weights. This imposes a strong inductive bias, trading the flexibility of independent weights/splines for extreme parameter sharing. Diversity in the transformation arises from the mixing weights (λ), the index-dependent shifts (q), and when enabled, the lateral mixing connections.

Concretely, each Sprecher block applies the transformation:

$$(x_i)_{i=1}^{d_{\text{in}}} \;\mapsto\; \Big[\Phi^{(\ell)}\Big(s_q^{(\ell)} \;+\; \tau^{(\ell)} \sum_{j\in\mathcal{N}(q)} \omega_{q,j}^{(\ell)}\, s_j^{(\ell)}\Big)\Big]_{q=0}^{d_{\text{out}}-1},$$

where $s_q^{(\ell)} = \sum_{i=1}^{d_{\text{in}}} \lambda_i^{(\ell)}\, \phi^{(\ell)}(x_i + \eta^{(\ell)} q) + \alpha q$. Lateral mixing (if enabled) modifies the pre-activation $s_q^{(\ell)}$ *before* $\Phi^{(\ell)}$, whereas residual connections (if enabled) are applied *after* $\Phi^{(\ell)}$ and do not introduce lateral coupling across q. For scalar outputs, the outputs of the final Sprecher block are aggregated (via summation); for vector outputs, no final summation is applied.

In Sprecher's original work, one layer (block) with $d_{\text{out}} = 2n + 1$ outputs (where $n = d_{\text{in}}$) was sufficient for universality. Our approach stacks L Sprecher blocks to create a deep network progression:

$$d_0 \to d_1 \to \cdots \to d_{L-1} \to d_L,$$

where $d_0 = d_{\text{in}}$ is the input dimension, and d_L is the dimension of the final hidden representation before potential aggregation or final mapping. This multi-block composition provides a deeper analog of the KAS construction, aiming for potentially enhanced expressive power or efficiency for complex compositional functions. Although the overall SN family is universal already for $L = 1$ (Theorem 1), we do not yet have a universality characterization under fixed depth/width constraints for $L > 1$ blocks (nor for our vector-output extension); we explore this empirically (see Section 6).

Definition 1 (Network notation)**.** *Throughout this paper, we denote Sprecher Network architectures using arrow notation of the form $d_{\text{in}} \to [d_1, d_2, \ldots, d_L] \to d_{\text{out}}$, where d_{in} is the input dimension, $[d_1, d_2, \ldots, d_L]$ represents the hidden layer dimensions (widths), and d_{out} is the final output dimension of the network. For scalar output ($d_{\text{out}} = 1$), the final block's outputs are summed. For vector output ($d_{\text{out}} > 1$), an additional non-summed block maps from d_L to d_{out}. For example, $2 \to [5, 3, 8] \to 1$ describes a network with 2-dimensional input, three hidden layers of widths 5, 3, and 8 respectively, and a scalar output (implying the final block's outputs of dimension 8 are summed). $2 \to [5, 3] \to 4$ describes a network with 2-dimensional input, two hidden layers of widths 5 and 3, and a 4-dimensional vector output (implying an additional output block maps from dimension 3 to 4 without summation). When input or output dimensions are clear from context, we may use the abbreviated notation $[d_1, d_2, \ldots, d_L]$ to focus on the hidden layer structure.*

3 Core architectural details

In our architecture, the fundamental building unit is the *Sprecher block*. The network is composed of a sequence of Sprecher blocks, each performing a shift-and-sum transformation inspired by Sprecher's original construction.

3.1 Sprecher block structure

A Sprecher block transforms an input vector $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ to an output vector $\mathbf{h} \in \mathbb{R}^{d_{\text{out}}}$. The data flow through a single block is illustrated in Figure 1. This transformation is implemented using the following shared, learnable components specific to that block:

- **Monotonic spline $\phi(\cdot)$:** A non-decreasing spline mapping $\mathbb{R} \to [0, 1]$ (piecewise-linear in our implementation; see Section 7) that is applied coordinate-wise. While we denote the domain as $\mathbb{R}$ for

generality, in practice each $\phi^{(\ell)}$ operates on a bounded domain determined dynamically by the network's structure and current parameters, as detailed in Section 7.4. This function is shared across all input-output connections within the block and its coefficients are learnable. Monotonicity is enforced during training through appropriate parameterization (see Section 7).

- **General spline $\Phi(\cdot)$:** A spline mapping $\mathbb{R} \to \mathbb{R}$ (piecewise-linear or C^1 piecewise-cubic Hermite/PCHIP; see Section 7, without monotonicity constraints) whose domain and codomain are determined by the network structure and can be either fixed or made trainable. If trainable codomains are used, they can be parameterized in various ways, such as by center and scale (half-width) parameters, allowing the spline to adapt its output range during training. This function is also shared across the block and its coefficients are learnable.

- **Mixing weights vector λ:** A vector $\{\lambda_i\}$ of size d_{in}, whose entries are learnable. These weights linearly combine the contributions from different input dimensions after transformation by ϕ. Crucially, these weights are shared across all output dimensions within the block, maintaining fidelity to Sprecher's original formulation.

- **Shift parameter η:** A learnable scalar η. This parameter controls the magnitude of the input shift $x_i + \eta q$, which depends on the output index q. While Sprecher's original construction requires $\eta > 0$, practical implementations may allow η to take any real value during training.

- **Lateral mixing parameters (optional):** When enabled, learnable parameters that allow intra-block communication between output dimensions. This includes a scalar $\tau^{(\ell)}$ (lateral scale) and weight vector(s) $\omega^{(\ell)}$ that enable outputs to exchange information before passing through $\Phi^{(\ell)}$. Two variants are supported: *cyclic* (each output receives from its cyclic neighbor) and *bidirectional* (each output mixes with both neighbors).

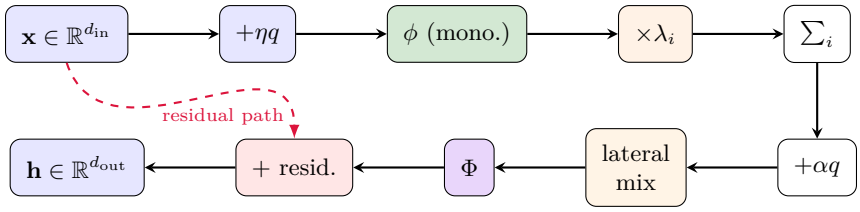


Figure 1: Data flow through a single Sprecher block. Each input x_i is shifted by ηq (where q indexes outputs), passed through the shared monotonic spline ϕ, weighted by λ_i, and summed. The pre-activation $s_q = \sum_i \lambda_i \phi(x_i + \eta q) + \alpha q$ undergoes optional lateral mixing before being transformed by the shared general spline Φ. Residual connections (dashed) provide direct gradient paths.

Concretely, we first define a single Sprecher block as an operator. Let $B^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ denote the ℓ-th Sprecher block with parameters $\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)}$, a fixed spacing constant α (set to 1 in all experiments unless noted), and optionally $\tau^{(\ell)}, \omega^{(\ell)}$. Given an input vector $\mathbf{x} = (x_1, \ldots, x_{d_{\text{in}}}) \in \mathbb{R}^{d_{\text{in}}}$, the block computes its output vector $\mathbf{h} \in \mathbb{R}^{d_{\text{out}}}$ component-wise as:

$$[B^{(\ell)}(\mathbf{x})]_q = \Phi^{(\ell)}\left(s_q^{(\ell)} + \tau^{(\ell)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(\ell)} s_j^{(\ell)} \right),$$

where $s_q^{(\ell)} = \sum_{i=1}^{d_{\text{in}}} \lambda_i^{(\ell)} \phi^{(\ell)}(x_i + \eta^{(\ell)} q) + \alpha q$ represents the pre-mixing activation, α is a fixed channel-spacing constant (we set $\alpha = 1$ in all experiments unless explicitly stated otherwise), and $\mathcal{N}(q)$ denotes the neighborhood structure:

- **No mixing:** $\mathcal{N}(q) = \emptyset$ (reduces to original formulation)

- **Cyclic:** $\mathcal{N}(q) = \{(q+1) \bmod d_{\text{out}}\}$ with scalar weights $\omega_q^{(\ell)}$

- **Bidirectional:** $\mathcal{N}(q) = \{(q-1) \bmod d_{\text{out}}, (q+1) \bmod d_{\text{out}}\}$ with weights $\omega^{(\ell)}_{q,\text{bwd}}$ for the $(q-1) \bmod d_{\text{out}}$ neighbor and $\omega^{(\ell)}_{q,\text{fwd}}$ for the $(q+1) \bmod d_{\text{out}}$ neighbor

While $\alpha = 1$ maintains theoretical fidelity, alternative values may be explored to improve optimization dynamics in deeper networks. Note that q serves dual roles here: as an output index ($q = 0, \ldots, d_{\text{out}} - 1$) and as an additive shift parameter within the formula.

In a network with multiple layers, each Sprecher block (indexed by $\ell = 1, \ldots, L$ or $L + 1$) uses its own independent set of shared parameters $(\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)})$ and optionally $(\tau^{(\ell)}, \omega^{(\ell)})$. The block operation implements a specific form of transformation: each input coordinate x_i is first shifted by an amount depending on the output index q and the shared shift parameter $\eta^{(\ell)}$, then passed through the shared monotonic spline $\phi^{(\ell)}$. The results are linearly combined using the learnable mixing weights $\lambda_i^{(\ell)}$, shifted again by the output index scaled by α. When lateral mixing is enabled, these pre-activation values undergo weighted mixing with neighboring outputs. Finally, the result is passed through the shared general spline $\Phi^{(\ell)}$. Figure 2 illustrates this spline-sharing structure within a single block, highlighting the key architectural difference from KANs. Stacking these blocks creates a deep, compositional representation.

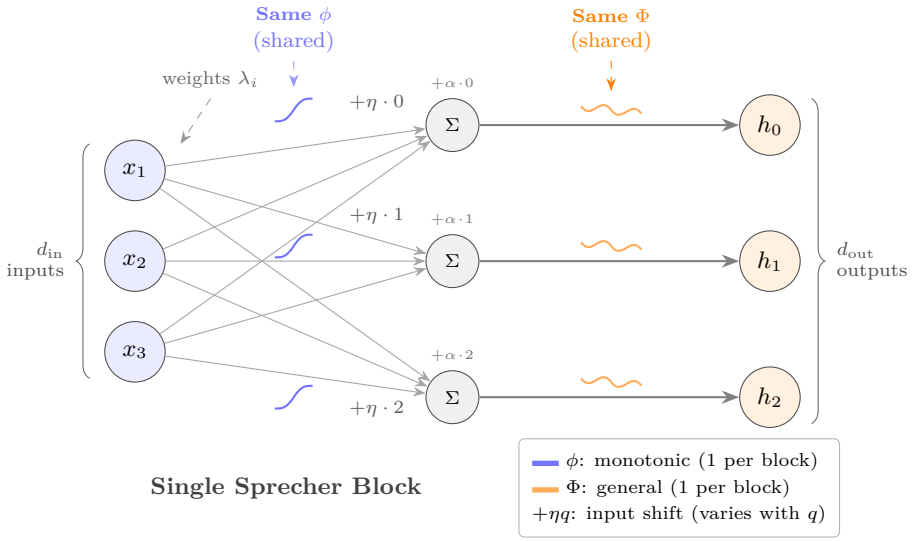


Figure 2: Internal structure of a Sprecher block showing spline sharing. Unlike KANs where each edge has a unique learnable spline, a Sprecher block uses only **two shared splines**: one monotonic ϕ applied to all shifted inputs, and one general Φ applied to all weighted sums. The mixing weights λ_i are shared across all output dimensions. Diversity across outputs arises from the index-dependent shifts $+\eta q$ (applied before ϕ) and $+\alpha q$ (added to each sum). This extreme parameter sharing yields $O(d_{\text{in}} + G)$ parameters per block versus $O(d_{\text{in}} \cdot d_{\text{out}} \cdot G)$ for KANs.

Remark 1 (Computational considerations)**.** *While the block operation as written suggests computing all d_{out} outputs simultaneously, implementations may compute them sequentially to reduce memory usage from $O(B \cdot d_{\text{in}} \cdot d_{\text{out}})$ to $O(B \cdot \max(d_{\text{in}}, d_{\text{out}}))$ where B is the batch size. This sequential computation produces mathematically identical results to the parallel formulation. This is particularly valuable when exploring architectures with wide layers, where memory constraints often limit feasible network configurations before parameter count becomes prohibitive. Section 7.3 details this sequential computation strategy.*

3.2 Optional enhancements

Several optional components can enhance the basic Sprecher block:

- **Residual connections (Cyclic):** When enabled, residual connections are implemented using a dimension-adaptive modulo-based cyclic assignment that maintains architectural coherence. Like convolutional layers that achieve parameter efficiency through local connectivity patterns, cyclic residuals

use periodic patterns to reduce parameters from $O(d_{\text{in}} \times d_{\text{out}})$ to $O(\max(d_{\text{in}}, d_{\text{out}}))$. This method adapts based on dimensional relationships between adjacent layers:

- **Identity** ($d_{\text{in}} = d_{\text{out}}$): Uses a single learnable weight w_{res}
- **Broadcasting** ($d_{\text{in}} < d_{\text{out}}$): Each output dimension cyclically selects an input via $1+(q \bmod d_{\text{in}})$ and applies a learnable scale
- **Pooling** ($d_{\text{in}} > d_{\text{out}}$): Input dimensions are cyclically assigned to outputs via $(i-1) \bmod d_{\text{out}}$ with learnable weights

Specifically, the block output with residual connection becomes:

$$[B^{(\ell)}_{\text{res}}(\mathbf{x})]_q = [B^{(\ell)}(\mathbf{x})]_q + \begin{cases} w_{\text{res}} \cdot x_{q+1} & \text{if } d_{\text{in}} = d_{\text{out}} \\ w^{\text{bcast}}_q \cdot x_{1+(q \bmod d_{\text{in}})} & \text{if } d_{\text{in}} < d_{\text{out}} \\ \sum_{i:(i-1) \bmod d_{\text{out}} = q} w^{\text{pool}}_i \cdot x_i & \text{if } d_{\text{in}} > d_{\text{out}} \end{cases}$$

where $w^{\text{bcast}} \in \mathbb{R}^{d_{\text{out}}}$ and $w^{\text{pool}} \in \mathbb{R}^{d_{\text{in}}}$ are learnable weight vectors. Empirically, these constrained connections often perform comparably to full projection residuals while using substantially fewer residual parameters.

- **Lateral mixing connections (distinct from cyclic residuals):** Inspired by the success of attention mechanisms and lateral connections in vision models, we introduce an optional intra-block communication mechanism. Before applying the outer spline $\Phi^{(\ell)}$, each output dimension can incorporate weighted contributions from neighboring outputs:

$$\tilde{s}^{(\ell)}_q = s^{(\ell)}_q + \tau^{(\ell)} \cdot \begin{cases} \omega^{(\ell)}_q \cdot s^{(\ell)}_{(q+1) \bmod d_{\text{out}}} & \text{(cyclic)} \\ \omega^{(\ell)}_{q,\text{fwd}} \cdot s^{(\ell)}_{(q+1) \bmod d_{\text{out}}} + \omega^{(\ell)}_{q,\text{bwd}} \cdot s^{(\ell)}_{(q-1) \bmod d_{\text{out}}} & \text{(bidirectional)} \end{cases}$$

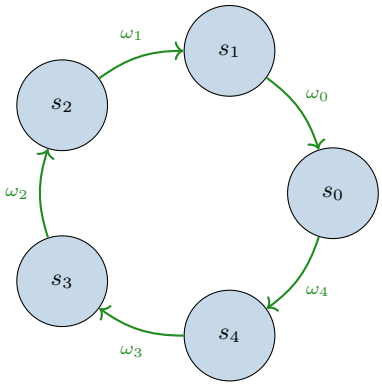
  This mechanism allows the network to learn correlations between output dimensions while maintaining the parameter efficiency of the architecture, adding only $O(d_{\text{out}})$ parameters per block. Empirically, we find this particularly beneficial for vector-valued outputs and deeper networks.

- **Normalization:** Batch normalization can be applied to block outputs to improve training stability, particularly in deeper networks. See Section 3.3 for details.

- **Output affine head:** The network may include a learnable affine head applied to the final network output: $f_{\text{final}}(\mathbf{x}) = s_{\text{out}}\, f(\mathbf{x}) + b_{\text{out}}$, where $s_{\text{out}}, b_{\text{out}} \in \mathbb{R}$ (2 parameters) are shared across all output dimensions. In our experiments we initialize b_{out} to the mean of the training targets (averaged over all outputs when $m > 1$) and set $s_{\text{out}} = 0.1$.

Remark 2 (Lateral mixing as structured attention)**.** *The lateral mixing mechanism can be viewed as a highly constrained form of self-attention where each output dimension "attends" only to its immediate neighbors in a cyclic topology (see Figure 3). Unlike full attention which requires $O(d^2_{\text{out}})$ parameters, our approach maintains linear scaling while still enabling cross-dimensional information flow. This design choice reflects our philosophy of extreme parameter sharing: just as the Sprecher structure shares splines across all connections within a block, lateral mixing shares the communication pattern across all samples while learning only the mixing weights.*

Remark 3 (Lateral mixing resolves the shared-weight optimization plateau)**.** *Wide single-layer Sprecher Networks exhibit a particularly severe optimization challenge beyond the typical difficulties of shallow networks. Due to the shared weight vector λ (rather than a weight matrix), all output dimensions process identical linear combinations of inputs, differing only through the deterministic index-dependent shifts $x_i \mapsto x_i + \eta q$ (inside ϕ) and the additive term $+\alpha q$ (in the pre-activation), leading to nearly identical outputs early in training. This symmetry can cause outputs to collapse into similar representations, reducing effective capacity and slowing convergence.*



Cyclic lateral mixing

$\tilde{s}_q = s_q + \tau\omega_q s_{(q+1) \bmod d_{\text{out}}}$

Pre-activation:

$s_q = \sum_{i=1}^{d_{\text{in}}} \lambda_i \phi(x_i + \eta q) + \alpha q$

After lateral mixing:

$\tilde{s}_q = s_q + \tau \sum_{j \in \mathcal{N}(q)} \omega_{q,j} s_j$

Output:

$h_q = \Phi(\tilde{s}_q)$

Figure 3: Lateral mixing enables cross-dimensional communication before the outer spline Φ. In the cyclic variant (shown), each pre-activation s_q receives a scaled contribution from its neighbor $s_{(q+1) \bmod d_{\text{out}}}$, parameterized by scale τ and per-output weights ω_q. This adds only $O(d_{\text{out}})$ parameters while breaking the symmetries inherent in the shared-weight structure. The bidirectional variant additionally includes contributions from $s_{(q-1) \bmod d_{\text{out}}}$.

Cyclic residual connections help by injecting input information directly into each output, but in very wide layers this may still be insufficient to break output symmetry and avoid plateaus. As an illustrative example, a $2 \to [120] \to 1$ *network trained on the 2D regression benchmark (*Toy-2D-Complex*)* $f(x_1, x_2) = \exp(\sin(11x_1)) + 3x_2 + 4\sin(8x_2)$ *over* $[0,1]^2$ *(with a fixed* 32×32 *training grid) reaches an MSE of* 8.55 *after* 40,000 *epochs with cyclic residuals (broadcast weights;* 120 *residual parameters). Replacing the cyclic residual with a dense projection residual (*$2 \times 120 = 240$ *residual parameters) provides no improvement (MSE:* 8.55*), while adding cyclic lateral mixing (a scalar* τ *plus* 120 *mixing weights) adds only* 121 *parameters and drops the MSE to* 2.12*. This demonstrates that the problem is not simply the general difficulty of optimizing wide shallow networks (which affects MLPs too), but specifically the need to break symmetries created by the shared-weight constraint. The lateral mixing mechanism provides the minimal cross-dimensional communication needed to differentiate the outputs beyond mere shifting, enabling successful optimization where traditional approaches fail.*

Remark 4 (Cyclic residuals as dimensional folding)**.** *The cyclic assignment pattern can be understood geometrically as a form of dimensional folding. When* $d_{\text{in}} > d_{\text{out}}$*, the modulo operation* $(i-1) \bmod d_{\text{out}}$ *effectively "wraps" the higher-dimensional input space around the lower-dimensional output space, analogous to wrapping a string around a cylinder. This creates a regular pattern where input dimensions are distributed uniformly across outputs, ensuring complete coverage and direct gradient paths. The cyclic pattern maximally separates consecutive inputs; for instance, with* $d_{\text{in}} = 7, d_{\text{out}} = 3$*, consecutive inputs* $\{1,2,3,4,5,6,7\}$ *map to outputs* $\{0,1,2,0,1,2,0\}$ *respectively, preventing the local clustering that might occur with contiguous block assignments.*

3.2.1 Toy-$4 \to 5$ synthetic regression task

We additionally consider a higher-dimensional vector-valued regression benchmark, TOY-$4 \to 5$, with inputs $\mathbf{x} = (x_1, x_2, x_3, x_4) \in [0,1]^4$ and outputs $f(\mathbf{x}) \in \mathbb{R}^5$ defined by

$$
\begin{aligned}
f_1(\mathbf{x}) &= \sin(2\pi x_1)\cos(\pi x_2),\\
f_2(\mathbf{x}) &= \exp\big(-2(x_1^2 + x_2^2)\big),\\
f_3(\mathbf{x}) &= x_3^3 - x_4^2 + \tfrac{1}{2}\sin(5x_3),\\
f_4(\mathbf{x}) &= \mathrm{sigmoid}\big(3(x_1 + x_2 - x_3 + x_4)\big),\\
f_5(\mathbf{x}) &= \tfrac{1}{2}\sin(4\pi x_1 x_4) + \tfrac{1}{2}\cos(3\pi x_2 x_3),
\end{aligned}
$$

where $\mathrm{sigmoid}(t) = \frac{1}{1+e^{-t}}$. Unless otherwise stated, training inputs are sampled i.i.d. uniformly from $[0,1]^4$.

Table 1: Cyclic residuals vs. linear residual projections on synthetic regression tasks. We report the best MSE on a fixed 1,024-point training set (measured in evaluation mode) achieved within the stated number of epochs. Toy-2D-Complex refers to the scalar 2D regression benchmark defined above; Toy-2D (vector) refers to the vector-valued 2D benchmark in Fig. 10; Toy-$4 \to 5$ refers to the synthetic regression task defined in Section 3.2.1. "Linear" uses a learned scalar when $d_{\text{in}} = d_{\text{out}}$ and a learned matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ otherwise; "cyclic" uses the pooling/broadcast construction when $d_{\text{in}} \neq d_{\text{out}}$ (and a learned scalar when $d_{\text{in}} = d_{\text{out}}$). The last columns count residual-path parameters only. For tasks with $d_{\text{out}} > 1$, MSE is computed as the mean of $(\hat{y} - y)^2$ over all output dimensions and samples.

Task	Architecture	Epochs	MSE (linear)	MSE (cyclic)	#resid params (lin./cyc.)	Ratio
Toy-2D-Complex	$2 \to [10, 10, 10] \to 1$	10,000	4.33×10^{-2}	$\mathbf{1.79 \times 10^{-3}}$	22/12	1.8×
Toy-2D-Complex	$2 \to [10, 11, 12, 13, 14, 15, 16, 17] \to 1$	50,000	$\mathbf{9.77 \times 10^{-6}}$	3.88×10^{-5}	1322/108	12.2×
Toy-2D-Complex	$2 \to [10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11] \to 1$	50,000	$\mathbf{1.52 \times 10^{-5}}$	3.89×10^{-5}	1670/175	9.5×
Toy-2D (vector)	$2 \to [50, 50, 50] \to 2$	10,000	7.50×10^{-6}	$\mathbf{2.46 \times 10^{-6}}$	202/102	2.0×
Toy-2D (vector)	$2 \to [10, 11, 12, 13, 14, 15, 16, 17] \to 2$	5,000	$\mathbf{5.47 \times 10^{-3}}$	9.09×10^{-2}	1356/125	10.8×
Toy-$4 \to 5$	$4 \to [30, 40] \to 5$	10,000	$\mathbf{8.01 \times 10^{-3}}$	1.60×10^{-2}	1520/110	13.8×
Toy-$4 \to 5$	$4 \to [10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11, 10, 11] \to 5$	10,000	$\mathbf{3.24 \times 10^{-4}}$	5.51×10^{-3}	1745/186	9.4×

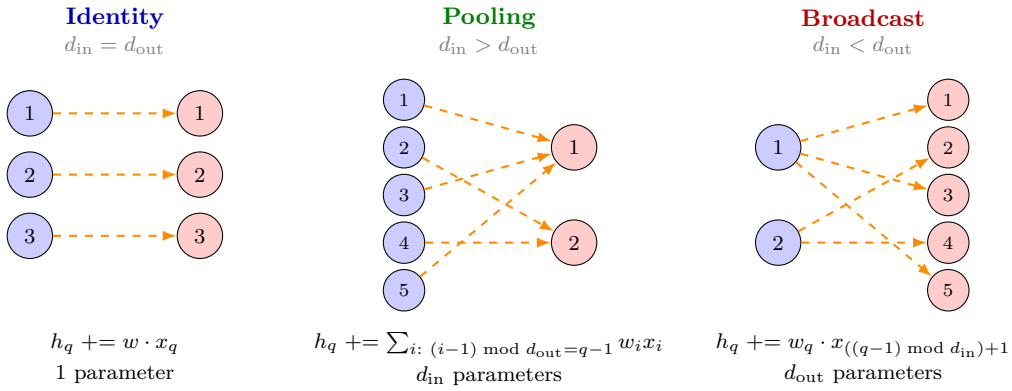


Figure 4: Dimension-adaptive cyclic residual connections (indices shown 1-based for readability). **Left:** When dimensions match, a single scalar weight applies element-wise. **Center:** When $d_{\text{in}} > d_{\text{out}}$ (pooling), multiple inputs are cyclically assigned to each output via modular indexing. **Right:** When $d_{\text{in}} < d_{\text{out}}$ (broadcast), inputs are cyclically reused across outputs. All cases maintain $O(\max(d_{\text{in}}, d_{\text{out}}))$ parameters, preserving the linear scaling of the architecture.

Remark 5 (Structured sparsity and architectural coherence)**.** *The cyclic residual design can be viewed as implementing a structured sparse projection where each output connects to approximately $d_{\text{in}}/d_{\text{out}}$ inputs (pooling case) or each input connects to approximately $d_{\text{out}}/d_{\text{in}}$ outputs (broadcasting case). This represents another form of weight sharing that complements the main Sprecher architecture: while Sprecher blocks*

Table 2: Ablation on Toy-2D-Vector. Mean test RMSE ± std over 10 seeds (5,000 epochs per run). Lower is better.

Setting	Test RMSE
Base SN (no residuals/mixing/domain tracking)	0.325 ± 0.015
+ cyclic residuals	0.226 ± 0.065
+ bidirectional mixing	0.211 ± 0.055
+ domain tracking	0.046 ± 0.050
+ resampling (full SN)	$\mathbf{0.029 \pm 0.020}$

share splines across output dimensions with diversity through shifts, residual connections share structure across connections with diversity through cyclic assignment. The unexpected effectiveness of this severe constraint suggests that for gradient flow and information propagation, the specific connection pattern may matter less than ensuring uniform coverage and balanced paths, a hypothesis that warrants further theoretical investigation.

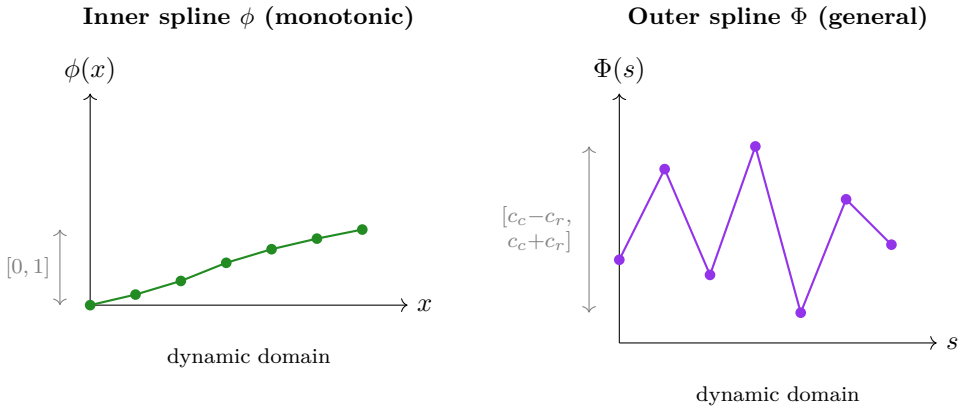


Figure 5: The dual spline system in Sprecher Networks. Left: The inner spline ϕ is monotonic (non-decreasing) with fixed codomain $[0, 1]$, parameterized via cumulative sums of softplus-transformed increments; it is strictly increasing on its spline domain and uses constant extension outside that domain (0 for inputs below the leftmost knot, 1 above the rightmost knot). Right: The outer spline Φ is a general (non-monotonic) univariate spline (piecewise-linear or cubic PCHIP) with optional learnable codomain parameters (c_c, c_r) defining center and radius. It uses linear extrapolation outside its domain. Both domains can be updated during training as λ and η evolve; in some experiments we freeze domain updates after a warm-up period (and in our PINN experiments we disable domain updates altogether).

3.3 Normalization considerations

The unique structure of Sprecher Networks requires careful consideration when incorporating normalization techniques. While standard neural networks typically normalize individual neuron activations, the shared-spline architecture of SNs suggests a different approach: normalizing the entire vector output $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$ of each Sprecher block as a cohesive unit.

Batch normalization can be placed either *before* or *after* a Sprecher block. In both placements it acts on the vector features $\mathbf{h}$, not on the internal scalars $s_q^{(\ell)}$, so the additive $+\alpha q$ and any lateral mixing remain entirely inside $\Phi^{(\ell)}$.

- **After block (default).** Apply BN to the block output $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$.
- **Before block.** Apply BN to the input $\mathbf{h}^{(\ell-1)} \in \mathbb{R}^{d_{\ell-1}}$ before it enters the block.

The transformation takes the familiar form:

$$\tilde{\mathbf{h}} = \gamma \odot \frac{\mathbf{h} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

where $\gamma, \beta \in \mathbb{R}^d$ are learnable affine parameters, and the statistics μ and σ^2 are computed per-dimension across the batch.

This approach maintains the parameter efficiency central to SNs: each normalization layer adds only $2d$ parameters, preserving the $O(LN)$ scaling. Moreover, by treating the block output as a unified representation rather than normalizing individual components, the method respects the architectural philosophy that all output dimensions arise from shared transformations through $\phi^{(\ell)}$ and $\Phi^{(\ell)}$.

We often find it beneficial to skip normalization for the first block (default setting), allowing the network to directly process the input features while still stabilizing deeper layers. This selective application is particularly important when theoretical domain computation assumes inputs in $[0, 1]^n$, as normalization could shift the data outside expected ranges.

When computing theoretical domains (Section 7.4), the bounds must account for normalization:

- **Training mode**: Conservative bounds assume standardization to approximately $[-4, 4]$ (covering 99.99% of a standard normal distribution), with affine transformation applied afterward.
- **Evaluation mode**: Uses the actual per-channel running statistics (and affine parameters, if enabled) to compute tighter bounds.

Our empirical findings suggest that the combination of normalization with residual connections and lateral mixing proves particularly effective for deeper SNs, enabling successful training of networks with many blocks while maintaining the characteristic parameter efficiency of the architecture.

3.4 Layer composition and final mapping

Let L be the number of hidden layers specified by the architecture $[d_1, \ldots, d_L]$. In our framework, a "hidden layer" corresponds to the vector output of a Sprecher block. The mapping from the representation at layer $\ell - 1$ to layer ℓ is implemented by the ℓ-th Sprecher block.

Let the input to the network be $\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_0}$ (where $d_0 = d_{\text{in}}$). The spline core of the ℓ-th Sprecher block ($\ell = 1, \ldots, L$) produces a vector $\tilde{\mathbf{h}}^{(\ell)} \in \mathbb{R}^{d_\ell}$, computed component-wise as:

$$\tilde{\mathbf{h}}_q^{(\ell)} = \Phi^{(\ell)} \left(\sum_{i=1}^{d_{\ell-1}} \lambda_i^{(\ell)} \, \phi^{(\ell)} \left(\mathbf{h}_i^{(\ell-1)} + \eta^{(\ell)} \, q \right) + \alpha q + \tau^{(\ell)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(\ell)} s_j^{(\ell)} \right) + R^{(\ell)}\big(\mathbf{h}^{(\ell-1)}\big)_q, \quad q = 0, \ldots, d_\ell - 1. \tag{2}$$

where the lateral mixing term is included when enabled, and $s_j^{(\ell)}$ denotes the pre-mixing activation for output j. The term $R^{(\ell)}(\mathbf{h}^{(\ell-1)})_q$ denotes an (optional) residual connection (e.g., the cyclic residual of Fig. 4 or a standard learned linear projection); set $R^{(\ell)} \equiv 0$ when residuals are disabled. When BatchNorm is used, it is applied before or after each block as described in Section 3.3. Throughout, $\tilde{\mathbf{h}}^{(\ell)}$ denotes the pre-normalization output of the Sprecher block; if BatchNorm-after-block is enabled then $\mathbf{h}^{(\ell)}$ denotes the post-normalization vector passed to the next layer (otherwise $\mathbf{h}^{(\ell)} = \tilde{\mathbf{h}}^{(\ell)}$).

Remark 6 (On the nature of composition)**.** *Note that in this $L > 1$ composition (Eq. 2), the argument to the inner spline $\phi^{(\ell)}$ is $\mathbf{h}_i^{(\ell-1)}$, the output of the previous layer, not the original input coordinate x_i. This is the fundamental departure from Sprecher's construction and is the defining feature of our proposed deep architecture. The motivation for this compositional structure comes not from Sprecher's work, but from the empirical success of the deep learning paradigm. Each layer processes the complex, transformed output of the layer before it, enabling the network to learn hierarchical representations.*

The composition of these blocks and the final output generation depend on the desired final output dimension $m = d_{\text{out}}$:

(a) Scalar output ($m = 1$): The network consists of exactly L Sprecher blocks. The output of the final block, $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$, is aggregated by summation to yield the scalar output:

$$f(\mathbf{x}) = \sum_{q=0}^{d_L-1} \mathbf{h}_q^{(L)}.$$

If we define the operator for the ℓ-th block (including residuals when enabled) as $T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, where

$$\Big(T^{(\ell)}(z)\Big)_q = \Phi^{(\ell)}\left(\sum_{i=1}^{d_{\ell-1}} \lambda_i^{(\ell)}\, \phi^{(\ell)}\Big(z_i + \eta^{(\ell)}\, q\Big) + \alpha q + \tau^{(\ell)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(\ell)} s_j^{(\ell)}\right) + R^{(\ell)}(z)_q,$$

then the overall function is

$$f(\mathbf{x}) = \sum_{q=0}^{d_L-1} \Big(T^{(L)} \circ T^{(L-1)} \circ \cdots \circ T^{(1)}\Big)(\mathbf{x})_q.$$

This network uses L blocks and $2L$ shared spline functions in total (one pair $(\phi^{(\ell)}, \Phi^{(\ell)})$ per block).

(b) Vector-valued output ($m > 1$): When the target function f maps to $\mathbb{R}^m$ with $m > 1$, the network first constructs the L hidden layers as above, yielding a final hidden representation $\mathbf{h}^{(L)} \in \mathbb{R}^{d_L}$. An *additional* output block (block $L + 1$) is then appended to map this representation $\mathbf{h}^{(L)}$ to the final output space $\mathbb{R}^m$. This $(L+1)$-th block operates *without* a final summation over its output index. It computes the final output vector $\mathbf{y} \in \mathbb{R}^m$ as:

$$y_q = \Big(T^{(L+1)}(\mathbf{h}^{(L)})\Big)_q = \Phi^{(L+1)}\left(\sum_{r=0}^{d_L-1} \lambda_r^{(L+1)}\, \phi^{(L+1)}\Big(\mathbf{h}_r^{(L)} + \eta^{(L+1)}\, q\Big) + \alpha q + \tau^{(L+1)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(L+1)} s_j^{(L+1)}\right) + R^{(L+1)}(\mathbf{h}^{(L)})_q,$$

for $q = 0, \ldots, m-1$. The network output function is then:

$$f(\mathbf{x}) = \mathbf{y} = \Big(T^{(L+1)} \circ T^{(L)} \circ \cdots \circ T^{(1)}\Big)(\mathbf{x}) \in \mathbb{R}^m. \tag{3}$$

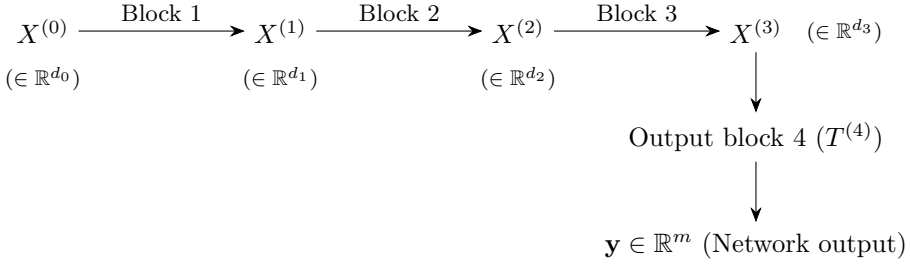
In this configuration, the network uses $L+1$ blocks and involves $2(L+1)$ shared spline functions. The extra block serves as a trainable output mapping layer, transforming the final hidden representation $\mathbf{h}^{(L)}$ into the desired m-dimensional output vector.

In summary: for an architecture with L hidden layers, a scalar-output SN uses L blocks and $2L$ shared splines. A vector-output SN (with $m > 1$) uses $L+1$ blocks and $2(L+1)$ shared splines. This structure provides a natural extension of Sprecher's original scalar formula to the vector-valued setting. In our supervised regression experiments we additionally apply a learned affine head to the final output,

$$f_{\text{final}}(\mathbf{x}) \;=\; s_{\text{out}}\, f(\mathbf{x}) + b_{\text{out}},$$

with $s_{\text{out}}, b_{\text{out}} \in \mathbb{R}$ (2 parameters, shared across output dimensions). We initialize b_{out} to the mean of the training targets (averaged over all outputs when $m > 1$) and set $s_{\text{out}} = 0.1$. In PINN experiments (where no full supervised target set is available), we keep the default initialization $s_{\text{out}} = 1$ and $b_{\text{out}} = 0$.

We illustrate the vector-output case ($m > 1$) for a network architecture $d_0 \to [d_1, d_2, d_3] \to m$ (i.e., $L = 3$ hidden layers). Let $X^{(0)}$ be the input $\mathbf{x}$.



Here, $X^{(\ell)} = \mathbf{h}^{(\ell)}$ denotes the output vector of the ℓ-th Sprecher block (we use both notations interchangeably for clarity in different contexts). Each block $T^{(\ell)}$ internally uses its own pair of shared splines $(\phi^{(\ell)}, \Phi^{(\ell)})$, mixing weights $\lambda^{(\ell)}$, shift $\eta^{(\ell)}$, and optionally lateral mixing parameters $(\tau^{(\ell)}, \omega^{(\ell)})$. The final output block $T^{(4)}$ maps the representation $X^{(3)}$ to the final m-dimensional output $\mathbf{y}$ without subsequent summation.

3.5 Illustrative expansions (scalar output)

To further clarify the compositional structure for the scalar output case ($m = 1$), we write out the full expansions for networks with $L = 1, 2, 3$ hidden layers, omitting optional residual connections and normalization operators for readability. For readability, we omit optional add-ons such as cyclic residual terms and normalization; when enabled, these appear additively (e.g., each block output includes an extra $R^{(\ell)}(\cdot)_q$ term as in (2)).

3.5.1 Single hidden layer ($L = 1$)

For a network with architecture $d_{\text{in}} \to [d_1] \to 1$ (i.e., $d_0 = d_{\text{in}}$), the network computes:

$$f(\mathbf{x}) = \sum_{q=0}^{d_1-1} \mathbf{h}_q^{(1)} = \sum_{q=0}^{d_1-1} \Phi^{(1)} \left(\sum_{i=1}^{d_0} \lambda_i^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, q \Big) + \alpha q + \tau^{(1)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(1)} s_j^{(1)} \right).$$

When lateral mixing is disabled ($\tau^{(1)} = 0$ or $\mathcal{N}(q) = \emptyset$), this reduces to Sprecher's 1965 construction if we choose $d_1 = 2d_0 + 1$, identify $\phi^{(1)} = \phi$, $\Phi^{(1)} = \Phi$, $\lambda_i^{(1)} = \lambda_i$, and set $\alpha = 1$.

3.5.2 Two hidden layers ($L = 2$)

Let the architecture be $d_0 \to [d_1, d_2] \to 1$. The intermediate output $\mathbf{h}^{(1)} \in \mathbb{R}^{d_1}$ is computed as:

$$\mathbf{h}_r^{(1)} = \Phi^{(1)} \Big(\sum_{i=1}^{d_0} \lambda_i^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, r \Big) + \alpha r + \tau^{(1)} \sum_{j \in \mathcal{N}(r)} \omega_{r,j}^{(1)} s_j^{(1)} \Big), \quad r = 0, \ldots, d_1 - 1.$$

The second block computes $\mathbf{h}^{(2)} \in \mathbb{R}^{d_2}$ using $\mathbf{h}^{(1)}$ as input:

$$\mathbf{h}_q^{(2)} = \Phi^{(2)} \Big(\sum_{r=0}^{d_1-1} \lambda_r^{(2)} \, \phi^{(2)} \Big(\mathbf{h}_r^{(1)} + \eta^{(2)} \, q \Big) + \alpha q + \tau^{(2)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(2)} s_j^{(2)} \Big), \quad q = 0, \ldots, d_2 - 1.$$

The final network output is the sum over the components of $\mathbf{h}^{(2)}$: $f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \mathbf{h}_q^{(2)}$.

For the base architecture without optional enhancements, the fully expanded form reveals the nested compositional structure:

$$f(\mathbf{x}) = \sum_{q=0}^{d_2-1} \Phi^{(2)} \left(\sum_{r=0}^{d_1-1} \lambda_r^{(2)} \, \phi^{(2)} \left(\underbrace{\Phi^{(1)} \left(\sum_{i=1}^{d_0} \lambda_i^{(1)} \, \phi^{(1)} \Big(x_i + \eta^{(1)} \, r \Big) + \alpha r \right)}_{\text{Output of first block}} + \eta^{(2)} \, q \right) + \alpha q \right). \tag{4}$$

This nested structure, where the output of $\Phi^{(1)}$ becomes the input to $\phi^{(2)}$, fundamentally departs from Sprecher's original construction and represents our empirically-motivated deep architecture.

Remark 7 (Lateral mixing in expanded form)**.** *When lateral mixing is enabled, the expanded forms become more complex. For instance, in the two-layer case, each $s_r^{(1)}$ term undergoes mixing before the $\Phi^{(1)}$ transformation, and similarly for $s_q^{(2)}$ before $\Phi^{(2)}$. We omit the full expansion for clarity, but note that lateral mixing represents an additional source of inter-dimensional coupling beyond the shifts, potentially enhancing the network's ability to capture correlations between output dimensions.*

3.5.3 Three hidden layers ($L = 3$)

Let the architecture be $d_0 \to [d_1, d_2, d_3] \to 1$. The recursive definition involves:

$$\mathbf{h}_r^{(1)} = \Phi^{(1)}\Big(\sum_{i=1}^{d_0} \lambda_i^{(1)}\, \phi^{(1)}\Big(x_i + \eta^{(1)}\, r\Big) + \alpha r + \tau^{(1)} \sum_{j \in \mathcal{N}(r)} \omega_{r,j}^{(1)} s_j^{(1)}\Big), \quad r = 0, \ldots, d_1 - 1,$$

$$\mathbf{h}_s^{(2)} = \Phi^{(2)}\Big(\sum_{r=0}^{d_1-1} \lambda_r^{(2)}\, \phi^{(2)}\Big(\mathbf{h}_r^{(1)} + \eta^{(2)}\, s\Big) + \alpha s + \tau^{(2)} \sum_{j \in \mathcal{N}(s)} \omega_{s,j}^{(2)} s_j^{(2)}\Big), \quad s = 0, \ldots, d_2 - 1,$$

$$\mathbf{h}_q^{(3)} = \Phi^{(3)}\Big(\sum_{s=0}^{d_2-1} \lambda_s^{(3)}\, \phi^{(3)}\Big(\mathbf{h}_s^{(2)} + \eta^{(3)}\, q\Big) + \alpha q + \tau^{(3)} \sum_{j \in \mathcal{N}(q)} \omega_{q,j}^{(3)} s_j^{(3)}\Big), \quad q = 0, \ldots, d_3 - 1.$$

The network output is $f(\mathbf{x}) = \sum_{q=0}^{d_3-1} \mathbf{h}_q^{(3)}$.

Remark 8. *These expansions highlight the compositional nature where the output of one Sprecher block, which is a vector of transformed values, serves as the input to the next. Each transformation layer involves its own pair of shared splines, learnable parameters, and optionally lateral mixing connections.*

Remark 9 (Necessity of internal shifts)**.** *It is tempting to simplify the nested structures, for instance by removing the inner shift terms like $\eta^{(2)}q$ inside $\phi^{(2)}$. One might hypothesize that the outer splines $\Phi^{(\ell)}$ could absorb this shifting effect (yielding a single composite spline per Sprecher block). However, experiments suggest that these internal shifts $\eta^{(\ell)}q$ applied to the inputs of the $\phi^{(\ell)}$ splines are crucial for the effective functioning of deeper Sprecher Networks. Removing them significantly degrades performance. The precise theoretical reason for their necessity in the multi-layer case, beyond their presence in Sprecher's original single-layer formula, warrants further investigation.*

4 Comparison with related architectures

To position Sprecher Networks accurately, we compare their core architectural features with Multi-Layer Perceptrons (MLPs), networks with learnable node activations (LANs/Adaptive-MLPs), and Kolmogorov-Arnold Networks (KANs).

Table 3 summarizes the key distinctions between these architectures. MLPs learn edge weights with fixed node activations, LANs add learnable node activations to this structure, KANs move learnability entirely to edge splines while eliminating linear weights, and SNs concentrate learnability in shared block-level splines, block-level shifts, mixing weights, and optionally lateral mixing connections. The critical difference for SNs is their use of weight vectors rather than matrices, which fundamentally reduces the dependence on width from quadratic to linear. This architectural choice can be understood through an analogy with convolutional neural networks: just as CNNs achieve parameter efficiency and improved generalization by sharing weights across spatial locations, SNs share weights across output dimensions within each block. In CNNs, spatial shifts provide the necessary diversity despite weight sharing; in SNs, the shifts ηq and the additive term $+\alpha q$, along with lateral mixing when enabled, play this diversifying role. This perspective reframes our architectural constraint not as a limitation, but as a principled form of weight sharing motivated by Sprecher's theorem, suggesting that SNs might be viewed as a "convolutional" approach to function approximation networks. While this weight sharing is theoretically justified for single-layer networks, its effectiveness in deep compositions remains an empirical finding that warrants further theoretical investigation. This combination of choices leads to SNs' distinctive parameter scaling of $O(LN + LG)$ compared to KANs' $O(LN^2G)$ and, for fixed spline resolution G, parameter-memory scaling of $O(LN)$.[1]

Here, we provide a precise comparison between LANs and SNs. While the following proposition shows that SNs can be expressed as special cases of LANs with specific structural constraints, it is important

[1]The $O(LN)$ term includes contributions from input weights (λ), normalization parameters when used, lateral mixing parameters, and (when used) *cyclic* residual connections. Cyclic residuals contribute at most $\max(d_{\text{in}}, d_{\text{out}})$ parameters per block; a learned linear residual projection instead adds $d_{\text{in}}d_{\text{out}}$ parameters when $d_{\text{in}} \neq d_{\text{out}}$ (and only a single scalar gate when $d_{\text{in}} = d_{\text{out}}$), so it is quadratic in width only on dimension-changing blocks.

Table 3: Architectural comparison of neural network families.

Feature	MLP	LAN / Adaptive-MLP	KAN	Sprecher Network (SN)
Learnable Components	Linear Weights (on edges)	Linear Weights + Node Activations	Edge Splines	Block Splines (ϕ, Φ) + Mixing Weights (λ) + Shift Parameter (η) + Lateral Mixing (τ, ω)
Fixed components	Node Activations	—	Node Summation	Node Summation (implicit) + Fixed Shifts $(+\alpha q)$
Location of Non-linearity	Nodes (Fixed)	Nodes (Learnable)	Edges (Learnable)	Blocks (Shared, Learnable)
Node operation	Apply $\sigma(\cdot)$	Apply $\sigma_{\text{learn}}(\cdot)$	$\sum$(inputs)	Implicit in Block Formula
Parameter sharing	None (typically)	Activations? (Maybe)	None (typically)	Splines (ϕ, Φ) per block
Intra-layer mixing	None	None	None	Lateral (cyclic/bidir.) $O(N)$ params
Residual design	Matrix projection $O(N^2)$ per skip	Matrix projection $O(N^2)$ per skip	None (typically)	Cyclic $O(N)$ per skip
Theoretical basis	UAT	UAT	KAT (inspired)	KAS (Sprecher, direct)
Param scaling	$O(LN^2)$	$O(LN^2 + LNG)$ (Approx.)	$O(LN^2G)$	$O(LN + LG)$ (Approx.)
Memory complexity	$O(LN^2)$	$O(LN^2)$	$O(LN^2G)$	$O(LN^2)$ parallel **O(LN + LG)** sequential

Notes: L=depth, N=average width, G=spline grid size/complexity. UAT=Universal Approx. Theorem, KAT=Kolmogorov-Arnold Theorem, KAS=Kolmogorov-Arnold-Sprecher. LAN details often follow KAN Appendix B [9]. Forward working memory refers to peak *forward-pass* intermediate storage plus parameters (excluding optimizer state), treating batch size as a constant; during training, reverse-mode autodiff may store additional activations unless checkpointing is used. The "sequential" mode for SNs processes output dimensions iteratively rather than in parallel, maintaining mathematical equivalence while reducing memory usage. All architectures may optionally include normalization layers; SNs apply normalization to entire block outputs rather than individual activations. For SNs, the $O(LN + LG)$ parameter scaling assumes the default cyclic/node residual; using a full linear residual projection adds an additional $O(\sum_\ell d_{\ell-1} d_\ell)$ parameters. *The parameter scaling notation uses N to denote a typical or average layer width for simplicity, following [9]. For architectures with varying widths d_ℓ, the LN^2 terms should be understood as $\sum_\ell d_{\ell-1} d_\ell$ (MLP, LAN), the LN^2G term for KAN as $(\sum_\ell d_{\ell-1} d_\ell)G$, and the LN term for SN as $\sum_\ell (d_{\ell-1} + d_\ell)$ (since SN uses weight vectors, not matrices), where the sum is over the relevant blocks/layers, for precise counts.*

to note that Sprecher's construction guarantees that this constrained form retains full expressivity in the single-layer case. This suggests that the extreme parameter sharing and structural constraints in SNs may serve as a beneficial inductive bias rather than a limitation.

Definition 2. *A LAN is an MLP with learnable activation. More precisely, the model is defined as:*

$$f(\mathbf{x}) = A^{(L)} \circ \sigma^{(L-1)} \circ A^{(L-1)} \circ \sigma^{(L-2)} \circ \cdots \circ \sigma^{(1)} \circ A^{(1)}(\mathbf{x}),$$

where $A^{(k)} \colon \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ is an affine map, and $\sigma^{(k)} \colon \mathbb{R} \to \mathbb{R}$ is the activation function (applied coordinate-wise). In an MLP, the trainable parameters are the weights $W^{(k)}$ and biases $b^{(k)}$ of $A^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{x} + b^{(k)}$ for $k = 1, \ldots, L$. In a LAN, σ contains additional trainable parameters, e.g., the coefficients of a spline.

Remark 10 (Note on weight structure)**.** *The following proposition uses matrix weights $\lambda_{i,q}^{(\ell)}$ to establish the connection between SNs and LANs. However, the architecture we propose and analyze throughout this paper uses vector weights $\lambda_i^{(\ell)}$ (following Sprecher's original formulation), where the same weights are shared across all output dimensions. This vector version represents an even more constrained special case of the LAN formulation shown below. The practical SN architecture with vector weights corresponds to the constraint that these matrix weights are constant across the output index, i.e., $\lambda_{i,q}^{(\ell)} = \lambda_i^{(\ell)}$ for all q. This constraint is fundamental to maintaining the "true Sprecher" structure and achieving the characteristic $O(LN)$ parameter scaling. Additionally, when lateral mixing is enabled, the structure becomes more complex, requiring additional terms in the LAN representation to capture the intra-block communication.*

Proposition 1. *A matrix-weighted variant of a Sprecher Network (i.e., with per-output mixing weights $\lambda_{i,q}^{(\ell)}$ in each block) and with lateral mixing disabled is a LAN, where:*

- *in odd layers $k = 2\ell - 1$, the weight matrix $W^{(k)} \in \mathbb{R}^{d_\ell d_{\ell-1} \times d_{\ell-1}}$ is fixed to $[I|\cdots|I]^\top$, where I is the $d_{\ell-1} \times d_{\ell-1}$ identity matrix, the bias vector has only one learnable parameter $\eta^{(\ell)}$ and is structured as $b^{(k)} = \eta^{(\ell)}(0, \ldots, 0, 1, \ldots, 1, \ldots, d_\ell - 1, \ldots, d_\ell - 1)^\top \in \mathbb{R}^{d_\ell d_{\ell-1}}$, and the activation is $\sigma^{(k)} = \phi^{(\ell)}$,*
- *in even layers $k = 2\ell$, the learnable weight matrix $W^{(k)} \in \mathbb{R}^{d_\ell \times d_\ell d_{\ell-1}}$ is structured as*

$$\begin{bmatrix} \lambda_{1,0}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},0}^{(\ell)} & 0 & & & \cdots & & & 0 \\ 0 & \cdots & 0 & \lambda_{1,1}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},1}^{(\ell)} & 0 & & \cdots & 0 \\ & & & & & & \ddots & & & \\ 0 & & & \cdots & & & 0 & \lambda_{1,d_\ell-1}^{(\ell)} & \cdots & \lambda_{d_{\ell-1},d_\ell-1}^{(\ell)} \end{bmatrix},$$

  the bias is fixed to $b^{(k)} = \alpha(0, \ldots, d_\ell - 1)^\top \in \mathbb{R}^{d_\ell}$, and the activation is $\sigma^{(k)} = \Phi^{(\ell)}$.

Proof. Follows immediately by inspecting (2) with $\tau^{(\ell)} = 0$ and $R^{(\ell)} \equiv 0$ (no lateral mixing or residual path). □

Remark 11 (Understanding the LAN representation)**.** *The representation of SNs as LANs in Proposition 1 uses an expanded intermediate state space. Each Sprecher block is decomposed into two LAN layers:*

- *The first layer expands the input $\mathbf{h}^{(\ell-1)} \in \mathbb{R}^{d_{\ell-1}}$ to $\mathbb{R}^{d_\ell d_{\ell-1}}$ by creating d_ℓ shifted copies, where the $(q \cdot d_{\ell-1} + i)$-th component contains $\phi^{(\ell)}(h_i^{(\ell-1)} + \eta^{(\ell)} q)$.*
- *The second layer applies the (tied) mixing weights $\lambda_i^{(\ell)}$ (shared across all q) to select and sum the appropriate components for each output q, adds the shift αq, and applies $\Phi^{(\ell)}$.*

This construction shows that while SNs can be expressed within the LAN framework, they represent a highly structured special case with specific weight patterns and an expanded intermediate dimension of $O(d_{\ell-1} d_\ell)$ between each pair of SN layers. The inclusion of lateral mixing would further complicate this representation, requiring additional structure to capture the intra-block communication.

While this proposition shows that SNs are special cases of LANs with specific structural constraints, Sprecher's theorem guarantees that *in the single-layer setting* this highly structured shift-and-sum form is still universal on $[0, 1]^n$. In Sprecher's original construction, η can be chosen as a universal constant rather than a learnable parameter. Thus, universality can already hold in the presence of strongly structured, index-dependent bias patterns; Proposition 1 makes this connection explicit by exhibiting one concrete structured realization.

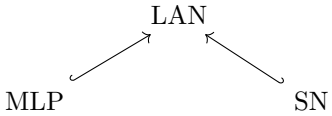


Figure 6: Diagram illustrating the dependencies between the models, in terms of learnable parameters. MLPs are LANs with fixed activation function, while SNs are LANs with a particular parameter structure (Proposition 1).

Remark 12 (Domain considerations)**.** *The above proposition assumes that the spline functions $\phi^{(\ell)}$ can handle inputs outside their original domain $[0, 1]$, which may arise due to the shifts $\eta^{(\ell)} q$. While Sprecher's theoretical construction extends ϕ appropriately, practical implementations typically compute exact theoretical bounds for all spline domains based on the network structure, and use these bounds to update spline knot ranges; any remaining out-of-range queries are handled by the splines' explicit extension/extrapolation rules (constant extension for $\phi^{(\ell)}$ and linear extrapolation for $\Phi^{(\ell)}$).*

5 Parameter counting and efficiency as a trade-off

A key consequence of adhering to the vector-based weighting scheme inspired by Sprecher's formula is a dramatic reduction in parameters compared to standard architectures. This represents a strong architectural constraint that may serve as either a beneficial inductive bias or a limitation, depending on the target function class. The specific design of SNs, particularly the sharing of splines and use of weight vectors rather than matrices, leads to a distinctive parameter scaling that warrants careful analysis.

Let's assume a network of depth L (meaning L hidden layers, thus L blocks for scalar output or $L+1$ for vector output), with an average layer width N. We denote the input dimension as N_{in} when it differs significantly from the hidden layer widths. Let G be the number of knot locations (equivalently, the number of learnable coefficients) used to parameterize each one-dimensional spline; a piecewise-linear spline then has $G-1$ intervals. For simplicity, we approximate the number of parameters per spline as $O(G)$.

The parameter counts for different architectures scale as follows. MLPs primarily consist of linear weight matrices, leading to a total parameter count dominated by these weights, scaling as $O(LN^2)$. LANs (Adaptive-MLPs) have both linear weights ($O(LN^2)$) and learnable activations; if each of the N nodes per layer has a learnable spline, this adds $O(LNG)$ parameters, for a total of $O(LN^2 + LNG)$. KANs replace linear weights with learnable edge splines, with $O(N^2)$ edges between layers. If each edge has a spline with $O(G)$ parameters, the total count per layer is $O(N^2G)$, leading to an overall scaling of $O(LN^2G)$.

Sprecher Networks have a fundamentally different structure. Each block ℓ contains mixing weights $\lambda^{(\ell)}$ with $O(d_{\ell-1})$ parameters, where $d_{\ell-1}$ is the input dimension to that block (crucially, these are vectors, not matrices), shared splines $\phi^{(\ell)}, \Phi^{(\ell)}$ with $2 \times O(G) = O(G)$ parameters per block independent of N, and a shift parameter $\eta^{(\ell)}$ with $O(1)$ parameter per block. When lateral mixing is enabled, each block additionally contains a lateral scale parameter $\tau^{(\ell)}$ with $O(1)$ parameter and mixing weights $\omega^{(\ell)}$ with $O(d_\ell)$ parameters (specifically d_ℓ for cyclic, $2d_\ell$ for bidirectional). When normalization is used, each block may have an associated normalization layer adding $O(N)$ parameters (specifically $2N$ for the affine transformation). When residual connections are used, cyclic residuals add at most $O(N)$ parameters per block (specifically $\max(d_{\ell-1}, d_\ell)$ for block ℓ, or just 1 when dimensions match); a learned linear residual projection instead adds $O(d_{\ell-1}d_\ell)$ parameters when $d_{\ell-1} \neq d_\ell$ (and just 1 when $d_{\ell-1} = d_\ell$). Summing over L (or $L+1$) blocks, the total parameter count scales approximately as $O(LN + LG + L)$, where the linear dependence on N now includes mixing weights, optional normalization parameters, lateral mixing weights, and residual connections.

Beyond parameter efficiency, SNs also admit memory-efficient computation strategies. While standard vectorized implementations materialize a $(B \cdot d_{\text{in}} \cdot d_{\text{out}})$ tensor of shifted inputs, a sequential evaluation strategy avoids this materialization and reduces the largest temporary forward tensor to $O(B \cdot \max(d_{\text{in}}, d_{\text{out}}))$ (Section 7.3). In wide settings where $d_{\text{in}} \approx d_{\text{out}} \approx N$, this yields a peak temporary forward footprint that scales linearly in N rather than quadratically. (As usual, achieving this scaling during training may require checkpointing/recomputation, since the backward pass can retain additional intermediates.)

This scaling reveals the crucial trade-off: SNs achieve a reduction from $O(LN^2)$ to $O(LN)$ in parameter count (and corresponding parameter memory) by using weight vectors rather than matrices; with sequential evaluation (Section 7.3) they also avoid explicitly materializing the $B \times d_{\text{in}} \times d_{\text{out}}$ shifted-input tensor in the forward pass. Additionally, in KANs, the spline complexity G multiplies the N^2 term ($O(LN^2G)$), while in SNs, due to spline sharing within blocks, it appears as an additive term ($O(LG)$). This suggests potential for significant parameter and memory savings, particularly when high spline resolution (G) is required for accuracy or when the layer width (N) is moderate to large.

This extreme parameter sharing represents a fundamental architectural bet: that the structure imposed by Sprecher's theorem, using only weight vectors with output diversity through shifts and lateral mixing, provides a beneficial inductive bias that outweighs the reduction in parameter flexibility. The empirical observation that training remains feasible (albeit sometimes requiring more iterations) with vector weights suggests this bet may be justified for certain function classes. Moreover, viewing this constraint through the lens of weight sharing in CNNs provides a new perspective: both architectures sacrifice parameter flexibility for a structured representation that has proven effective in practice, though for SNs the theoretical justification comes from Sprecher's theorem rather than domain-specific intuitions about spatial invariance. The addition of lateral mixing provides a middle ground, allowing some cross-dimensional communication while maintaining the overall parameter efficiency. Whether this constraint serves as beneficial regularization

or harmful limitation likely depends on the specific problem domain and the alignment between the target function's structure and the inductive bias imposed by the SN architecture.

5.1 Illustrative parameter comparisons (hypothetical examples)

We present parameter count comparisons based on architectures reported in [9] to illustrate the potential parameter efficiency of SNs. Note that these examples are hypothetical; while the KAN architectures are from their paper, the SN results are theoretical projections that require empirical validation. Optimal architectures for SNs may differ from those of KANs due to the shared spline structure.

PDE solving example (KAN Ref § 3.4): KAN architecture $[2, 10, 1]$, reported effective. $N_{in} = 2, N_{hid} = 10, N_{out} = 1$.

- **KAN (est.):** $(2 \times 10 + 10 \times 1) = 30$ edges/splines. KANs use B-splines with $G_{\text{KAN}} = 20$ intervals, $k = 3$ ($G_{\text{KAN}} + k \approx 23$ params/spline). Total KAN params $\approx 30 \times 23 = 690$.
- **SN (hypothetical):** Architecture $2 \to [10] \to 1$. $L = 1$ hidden layer, scalar output means $L = 1$ block total. Using piecewise linear splines with $G = 21$ knots (20 intervals): $G = 21$ params/spline. Shared splines: $2L \times G = 2 \times 21 = 42$. Mixing weights: $d_0 = 2$ (vector weights). Shift $\eta^{(1)}$: 1. If lateral mixing enabled (cyclic): $\tau^{(1)}$: 1, $\omega^{(1)}$: 10. Total SN params $\approx 42 + 2 + 1 + 1 + 10 = 56$ (with lateral mixing) or 45 (without).
- **Potential advantage:** For equivalent structure, theoretical reduction factor $\approx 12 - 15$x. Actual performance requires empirical validation.

Knot theory example (KAN Ref § 4.3): Pruned KAN $[17, 1, 14]$, with B-splines using $G_{\text{KAN}} = 3$ intervals, $k = 3$ ($G_{\text{KAN}} + k \approx 6$ params/spline).

- **KAN (est.):** $(17 \times 1 + 1 \times 14) = 31$ edges/splines. Total KAN params $\approx 31 \times 6 = 186$.
- **SN (hypothetical):** Architecture $17 \to [1] \to 14$. $L = 1$ hidden layer, vector output means $L + 1 = 2$ blocks total. Using piecewise linear splines with $G = 4$ knots (3 intervals): $2(L + 1) \times G = 4 \times 4 = 16$. Mixing weights: $d_0 + d_1 = 17 + 1 = 18$ (vector weights). Shifts $\eta^{(1)}, \eta^{(2)}$: 2. If lateral mixing enabled (cyclic) for both blocks: $\tau^{(1)}, \tau^{(2)}$: 2, $\omega^{(1)}$: 1, $\omega^{(2)}$: 14. Total SN params $\approx 16+18+2+2+1+14 = 53$ (with lateral mixing) or 36 (without).
- **Potential advantage:** Theoretical reduction factor $\approx 3.5 - 5$x. The narrow intermediate dimension $d_1 = 1$ might pose challenges for SN training.

These calculations illustrate the dramatic parameter reduction possible with SNs, but they also highlight a crucial practical consideration: the optimal architecture for SNs likely differs substantially from that of MLPs or KANs. With vector weights providing only linear scaling in width, SNs may require wider or deeper architectures to achieve comparable expressivity. The art of architecture selection for SNs involves balancing the parameter efficiency against the need for sufficient expressivity: a trade-off that remains poorly understood and likely depends strongly on the problem domain. Early empirical evidence suggests that SNs excel when the target function aligns well with their compositional, shift-based structure, but may require more iterations when forced to approximate functions that require truly independent processing of different output dimensions. The addition of lateral mixing partially addresses this limitation by enabling limited cross-dimensional communication.

6 Theoretical aspects and open questions

6.1 Relation to Sprecher (1965) and universality

As shown in Section 3.5.1, a single-layer ($L = 1$) Sprecher Network (SN) reduces exactly to Sprecher's constructive representation (1) when lateral mixing and residuals are disabled (i.e., $\tau = 0$ and no residual

connection), with the additive shift implemented by αq (set $\alpha = 1$ to match (1)). In his 1965 paper, Sprecher proved that for any continuous function $f : [0,1]^n \to \mathbb{R}$ there exist a *single* nondecreasing univariate function ϕ, a continuous univariate function Φ, real constants $\lambda_1, \ldots, \lambda_n$ and η, and an integer upper index $2n$ such that the representation (1) holds [10]. This immediately implies:

Theorem 1 (Universality of single-layer SNs)**.** *For any $n \geq 1$ and any $f \in C([0,1]^n)$, there exists a single-layer SN with $\tau = 0$ and no residual connection that represents f exactly in the form* (1) *(taking $d_{\text{out}} = 2n+1$ and $\alpha = 1$). In particular, for every $\epsilon > 0$ there is a single-layer SN (with $\tau = 0$ and no residual) such that $\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \epsilon$.*

Thus, single-layer SNs inherit (indeed, contain) the family given by Sprecher's representation. In practice, however, we do not fit arbitrary C^2 univariate functions ϕ and Φ exactly; we parameterize them as splines. This motivates the following finite-parameter approximation analysis, which also clarifies how optional *lateral mixing* (controlled by τ and ω) and cyclic residuals affect constants but not rates.

Notation for a single block. A (possibly mixed) Sprecher block $T : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ acts componentwise as

$$[T(\mathbf{z})]_q \;=\; \Phi\Bigg(\underbrace{\sum_{i=1}^{d_{\text{in}}} \lambda_i \, \phi(z_i + \eta q) + \alpha q}_{=:s_q} \;+ \tau \sum_{j \in \mathcal{N}(q)} \omega_{qj} \, s_j \Bigg), \quad q = 0, \ldots, d_{\text{out}} - 1,$$

where $\lambda \in \mathbb{R}^{d_{\text{in}}}$, $\eta, \alpha, \tau \in \mathbb{R}$, $\mathcal{N}(q)$ specifies the (optional) neighborhood used for lateral mixing, and $\omega = (\omega_{qj})$ are the mixing weights. When a cyclic residual connection is present, the block output is $T_{\text{res}}(\mathbf{z}) = T(\mathbf{z}) + R(\mathbf{z})$. The approximation analysis below applies to T; when the residual map R is shared between the target block and its spline approximation, it cancels in the blockwise difference. However, Lipschitz/composition constants for the full block include an additive contribution from R (e.g. $L_{T_{\text{res}}} \leq L_T + L_R$).

Lemma 1 (Single block approximation with piecewise-linear splines)**.** *Fix $B_{\text{in}} > 0$ and assume $\mathbf{z} \in \mathcal{B}_{\text{in}} := [-B_{\text{in}}, B_{\text{in}}]^{d_{\text{in}}}$. Let*

$$I_\phi := \big[-B_{\text{in}} - |\eta|(d_{\text{out}} - 1),\; B_{\text{in}} + |\eta|(d_{\text{out}} - 1)\big], \quad M_\phi := \|\phi\|_{L^\infty(I_\phi)}.$$

Define

$$B_\omega := \sup_q \sum_{j \in \mathcal{N}(q)} |\omega_{qj}|, \qquad R_{\text{mix}} := 1 + |\tau| \, B_\omega, \qquad M_s := \|\lambda\|_1 \, M_\phi + |\alpha| \, (d_{\text{out}} - 1),$$

and set $I_\Phi := [-R_{\text{mix}} M_s,\; R_{\text{mix}} M_s]$. Assume $\phi, \Phi \in C^2$ on neighborhoods of I_ϕ and I_Φ, respectively. Let $\hat{\phi}$ and $\hat{\Phi}$ be the (shape-preserving) piecewise-linear interpolants of ϕ on a uniform grid of $G_\phi \geq 2$ knots over I_ϕ and of Φ on a uniform grid of $G_\Phi \geq 2$ knots over I_Φ; write

$$h_\phi := \frac{|I_\phi|}{G_\phi - 1}, \quad h_\Phi := \frac{|I_\Phi|}{G_\Phi - 1}, \quad M_{\phi''} := \|\phi''\|_{L^\infty(I_\phi)}, \quad M_{\Phi''} := \|\Phi''\|_{L^\infty(I_\Phi)}.$$

Define $\hat{T}$ by replacing (ϕ, Φ) with $(\hat{\phi}, \hat{\Phi})$ in T (keeping the same $\lambda, \eta, \alpha, \tau, \omega, \mathcal{N}$). Then, for all $\mathbf{z} \in \mathcal{B}_{\text{in}}$,

$$\|T(\mathbf{z}) - \hat{T}(\mathbf{z})\|_\infty \;\leq\; L_\Phi \, R_{\text{mix}} \, \|\lambda\|_1 \, \delta_\phi \;+\; \delta_\Phi, \qquad \delta_\phi := \frac{M_{\phi''}}{8} h_\phi^2, \quad \delta_\Phi := \frac{M_{\Phi''}}{8} h_\Phi^2, \tag{5}$$

where $L_\Phi := \|\Phi'\|_{L^\infty(I_\Phi)}$. Equivalently,

$$\|T(\mathbf{z}) - \hat{T}(\mathbf{z})\|_\infty \;\leq\; K_T \cdot \max\{h_\phi^2, h_\Phi^2\} \quad \textit{with} \quad K_T := \frac{1}{8}\Big(L_\Phi \, R_{\text{mix}} \, \|\lambda\|_1 \, M_{\phi''} + M_{\Phi''} \Big).$$

The same bounds hold for a block with a (shared) cyclic residual $T_{\text{res}} = T + R$, since R cancels when T and $\hat{T}$ share the same residual path.

Proof. On each subinterval of the uniform grid for I_ϕ (resp. I_Φ) of length h_ϕ (resp. h_Φ), the classical bound for piecewise-linear interpolation of a C^2 function gives

$$\|\phi - \hat{\phi}\|_{L^\infty(I_\phi)} \le \frac{M_{\phi''}}{8} h_\phi^2 =: \delta_\phi, \qquad \|\Phi - \hat{\Phi}\|_{L^\infty(I_\Phi)} \le \frac{M_{\Phi''}}{8} h_\Phi^2 =: \delta_\Phi.$$

For fixed $\mathbf{z}$ and q, write $s_q = \sum_i \lambda_i \, \phi(z_i + \eta q) + \alpha q$ and $\hat{s}_q$ the analogous quantity with $\hat{\phi}$. Then

$$|s_q - \hat{s}_q| \;\le\; \sum_{i=1}^{d_{\text{in}}} |\lambda_i| \, \|\phi - \hat{\phi}\|_{L^\infty(I_\phi)} \;\le\; \|\lambda\|_1 \, \delta_\phi.$$

Define the post-mixing scalars $\tilde{s}_q := s_q + \tau \sum_{j \in \mathcal{N}(q)} \omega_{qj} s_j$ and $\hat{\tilde{s}}_q := \hat{s}_q + \tau \sum_{j \in \mathcal{N}(q)} \omega_{qj} \hat{s}_j$. Using the triangle inequality and the definition of B_ω,

$$|\tilde{s}_q - \hat{\tilde{s}}_q| \;\le\; |s_q - \hat{s}_q| + |\tau| \sum_{j \in \mathcal{N}(q)} |\omega_{qj}| \, |s_j - \hat{s}_j| \;\le\; R_{\text{mix}} \cdot \sup_j |s_j - \hat{s}_j| \;\le\; R_{\text{mix}} \, \|\lambda\|_1 \, \delta_\phi.$$

Next, since a linear interpolant lies in the convex hull of the function values at the grid endpoints, we have

$$\|\hat{\phi}\|_{L^\infty(I_\phi)} \;\le\; \|\phi\|_{L^\infty(I_\phi)} = M_\phi.$$

Hence $|s_j| \le M_s$ and $|\hat{s}_j| \le M_s$, which implies $|\tilde{s}_q| \le R_{\text{mix}} M_s$ and $|\hat{\tilde{s}}_q| \le R_{\text{mix}} M_s$. Therefore $\tilde{s}_q, \hat{\tilde{s}}_q \in I_\Phi$, and

$$|T(\mathbf{z})_q - \hat{T}(\mathbf{z})_q| = \left|\Phi(\tilde{s}_q) - \hat{\Phi}(\hat{\tilde{s}}_q)\right| \le \underbrace{|\Phi(\tilde{s}_q) - \Phi(\hat{\tilde{s}}_q)|}_{\le L_\Phi |\tilde{s}_q - \hat{\tilde{s}}_q|} + \underbrace{|\Phi(\hat{\tilde{s}}_q) - \hat{\Phi}(\hat{\tilde{s}}_q)|}_{\le \delta_\Phi}.$$

Combining the bounds and taking the maximum over q yields (5). If a residual path R is present and shared between T and $\hat{T}$, then $T_{\text{res}} - \hat{T}_{\text{res}} = (T - \hat{T}) + (R - R) = (T - \hat{T})$, so the same inequalities apply. □

Lemma 2 (Lipschitz constant of a Sprecher block)**.** *Assume ϕ and Φ are Lipschitz on I_ϕ and I_Φ with constants $L_\phi := \|\phi'\|_{L^\infty(I_\phi)}$ and $L_\Phi := \|\Phi'\|_{L^\infty(I_\Phi)}$, respectively. Then T is L_T-Lipschitz on $\mathcal{B}_{\text{in}}$ in the ℓ_∞ norm with*

$$L_T \;\le\; L_\Phi \, R_{\text{mix}} \, \|\lambda\|_1 \, L_\phi,$$

where $R_{\text{mix}} = 1 + |\tau| \, B_\omega$ as in Lemma 1. If a residual path R (shared between the exact and approximated networks) is present with Lipschitz constant L_R on $\mathcal{B}_{\text{in}}$, then $T_{\text{res}} = T + R$ is $(L_T + L_R)$-Lipschitz.

Proof. For $\mathbf{z}, \mathbf{z}' \in \mathcal{B}_{\text{in}}$ and each q,

$$\left|s_q(\mathbf{z}) - s_q(\mathbf{z}')\right| \le \sum_i |\lambda_i| \left|\phi(z_i + \eta q) - \phi(z_i' + \eta q)\right| \le \|\lambda\|_1 \, L_\phi \, \|\mathbf{z} - \mathbf{z}'\|_\infty.$$

Therefore,

$$\left|\tilde{s}_q(\mathbf{z}) - \tilde{s}_q(\mathbf{z}')\right| \le \Big(1 + |\tau| \sum_{j \in \mathcal{N}(q)} |\omega_{qj}|\Big) \, \|\lambda\|_1 \, L_\phi \, \|\mathbf{z} - \mathbf{z}'\|_\infty \le R_{\text{mix}} \|\lambda\|_1 L_\phi \, \|\mathbf{z} - \mathbf{z}'\|_\infty,$$

and hence

$$|T(\mathbf{z})_q - T(\mathbf{z}')_q| \le L_\Phi \, |\tilde{s}_q(\mathbf{z}) - \tilde{s}_q(\mathbf{z}')| \le L_\Phi R_{\text{mix}} \|\lambda\|_1 L_\phi \, \|\mathbf{z} - \mathbf{z}'\|_\infty.$$

Taking the maximum over q gives the claimed bound for T, and adding L_R handles T_{res} by the triangle inequality. □

Lemma 3 (Error composition)**.** *Consider an L-block SN (with optional lateral mixing and cyclic residuals at each block). Let $T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ denote the ℓ-th block map on the exact network and $\hat{T}^{(\ell)}$ its spline-approximated counterpart (same $\lambda, \eta, \alpha, \tau, \omega$ but (ϕ, Φ) replaced by $(\hat{\phi}, \hat{\Phi})$). Assume there exist bounded sets $\mathcal{B}_\ell \subset \mathbb{R}^{d_\ell}$ such that for all inputs $\mathbf{x} \in [0,1]^n$,*

$$\mathbf{h}^{(\ell)}(\mathbf{x}) := T^{(\ell)} \circ \cdots \circ T^{(1)}(\mathbf{x}) \in \mathcal{B}_\ell \quad \textit{and} \quad \hat{\mathbf{h}}^{(\ell)}(\mathbf{x}) := \hat{T}^{(\ell)} \circ \cdots \circ \hat{T}^{(1)}(\mathbf{x}) \in \mathcal{B}_\ell,$$

and suppose each $T^{(\ell)}$ is $L_{T^{(\ell)}}$-Lipschitz on $\mathcal{B}_{\ell-1}$. If $\varepsilon_\ell := \sup_{\mathbf{z}\in\mathcal{B}_{\ell-1}} \|T^{(\ell)}(\mathbf{z}) - \hat{T}^{(\ell)}(\mathbf{z})\|_\infty$, then for $E_\ell := \sup_{\mathbf{x}\in[0,1]^n} \|\mathbf{h}^{(\ell)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell)}(\mathbf{x})\|_\infty$ we have

$$E_\ell \;\le\; \sum_{j=1}^{\ell} \left(\prod_{m=j+1}^{\ell} L_{T^{(m)}} \right) \varepsilon_j, \tag{6}$$

with the empty product equal to 1.

Proof. The case $\ell = 1$ is $E_1 \le \varepsilon_1$. Assume the claim for $\ell - 1$. Then, for any $\mathbf{x}$,

$$\begin{aligned}
\|\mathbf{h}^{(\ell)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell)}(\mathbf{x})\|_\infty &= \|T^{(\ell)}(\mathbf{h}^{(\ell-1)}(\mathbf{x})) - \hat{T}^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x}))\|_\infty \\
&\le \underbrace{\|T^{(\ell)}(\mathbf{h}^{(\ell-1)}(\mathbf{x})) - T^{(\ell)}(\hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x}))\|_\infty}_{\le L_{T^{(\ell)}}\, \|\mathbf{h}^{(\ell-1)}(\mathbf{x}) - \hat{\mathbf{h}}^{(\ell-1)}(\mathbf{x})\|_\infty} + \underbrace{\|T^{(\ell)} - \hat{T}^{(\ell)}\|_{\infty,\mathcal{B}_{\ell-1}}}_{\le \varepsilon_\ell} \\
&\le L_{T^{(\ell)}} E_{\ell-1} + \varepsilon_\ell,
\end{aligned}$$

and the induction hypothesis gives (6). □

Corollary 1 (Global spline approximation rate for SNs with piecewise-linear splines)**.** *Let $f : [0,1]^n \to \mathbb{R}$ be realized by an ideal L-block SN (possibly with lateral mixing and cyclic residuals) with scalar output. For each block $\ell = 1, \dots, L$, let $B_{\text{in}}^{(\ell)} := \sup\{\|\mathbf{z}\|_\infty : \mathbf{z} \in \mathcal{B}_{\ell-1}\}$. Assume:*

(i) *$\phi^{(\ell)}, \Phi^{(\ell)} \in C^2$ on neighborhoods of the intervals*

$$I_\phi^{(\ell)} := \big[-B_{\text{in}}^{(\ell)} - |\eta^{(\ell)}|(d_\ell - 1),\; B_{\text{in}}^{(\ell)} + |\eta^{(\ell)}|(d_\ell - 1)\big], \quad I_\Phi^{(\ell)} := \big[-R_{\text{mix}}^{(\ell)} M_s^{(\ell)},\; R_{\text{mix}}^{(\ell)} M_s^{(\ell)}\big],$$

with $M_s^{(\ell)} := \|\lambda^{(\ell)}\|_1 \|\phi^{(\ell)}\|_{L^\infty(I_\phi^{(\ell)})} + |\alpha|(d_\ell - 1)$ and $R_{\text{mix}}^{(\ell)} := 1 + |\tau^{(\ell)}|\, B_\omega^{(\ell)}$, $B_\omega^{(\ell)} := \sup_q \sum_{j\in\mathcal{N}^{(\ell)}(q)} |\omega_{qj}^{(\ell)}|$;

(ii) *the structural parameters are bounded: $\|\lambda^{(\ell)}\|_1 \le \Lambda_1$, $|\eta^{(\ell)}| \le H$, $|\alpha| \le A$, $|\tau^{(\ell)}| \le T$, and $B_\omega^{(\ell)} \le \Omega$;*

(iii) *there exist bounded sets $\mathcal{B}_\ell$ such that the exact and approximated forward passes both remain in $\mathcal{B}_\ell$ for all inputs in $[0,1]^n$ (bounded propagation).*

Construct $\hat{\phi}^{(\ell)}$ and $\hat{\Phi}^{(\ell)}$ as the piecewise-linear interpolants on uniform grids with $G_\phi^{(\ell)} \ge 2$ and $G_\Phi^{(\ell)} \ge 2$ knots on $I_\phi^{(\ell)}$ and $I_\Phi^{(\ell)}$, respectively, and write $h_\phi^{(\ell)} := |I_\phi^{(\ell)}|/(G_\phi^{(\ell)} - 1)$ and $h_\Phi^{(\ell)} := |I_\Phi^{(\ell)}|/(G_\Phi^{(\ell)} - 1)$. Then, with

$$\delta_\phi^{(\ell)} := \frac{\|\phi^{(\ell)''}\|_{L^\infty(I_\phi^{(\ell)})}}{8} \big(h_\phi^{(\ell)}\big)^2, \qquad \delta_\Phi^{(\ell)} := \frac{\|\Phi^{(\ell)''}\|_{L^\infty(I_\Phi^{(\ell)})}}{8} \big(h_\Phi^{(\ell)}\big)^2,$$

the blockwise error satisfies

$$\varepsilon_\ell := \sup_{\mathbf{z}\in\mathcal{B}_{\ell-1}} \|T^{(\ell)}(\mathbf{z}) - \hat{T}^{(\ell)}(\mathbf{z})\|_\infty \;\le\; L_\Phi^{(\ell)}\, R_{\text{mix}}^{(\ell)}\, \|\lambda^{(\ell)}\|_1\, \delta_\phi^{(\ell)} \;+\; \delta_\Phi^{(\ell)},$$

where $L_\Phi^{(\ell)} := \|\Phi^{(\ell)'}\|_{L^\infty(I_\Phi^{(\ell)})}$. Consequently, by Lemma 3,

$$\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| \;\le\; \sum_{j=1}^{L} \left(\prod_{m=j+1}^{L} L_{T^{(m)}} \right) \Big(L_\Phi^{(j)}\, R_{\text{mix}}^{(j)}\, \|\lambda^{(j)}\|_1\, \delta_\phi^{(j)} + \delta_\Phi^{(j)} \Big),$$

with $L_{T^{(m)}}$ any Lipschitz constant of the m-th block on $\mathcal{B}_{m-1}$ (e.g. from Lemma 2, plus the residual-path constant when present). In particular, if $G_\phi^{(\ell)} = G_\Phi^{(\ell)} = G$ for all ℓ, then with $h := \max_\ell\{|I_\phi^{(\ell)}|, |I_\Phi^{(\ell)}|\}/(G-1)$ we obtain

$$\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| \;=\; \mathcal{O}\big(h^2\big) \;=\; \mathcal{O}\big(G^{-2}\big)\,,$$

with constants depending on $\{\|\lambda^{(\ell)}\|_1, L_{\phi^{(\ell)}}, L_\Phi^{(\ell)}, R_{\text{mix}}^{(\ell)}, \|\phi^{(\ell)''}\|_\infty, \|\Phi^{(\ell)''}\|_\infty, L_{T^{(\ell)}}\}_{\ell=1}^{L}$ and on the bounded-propagation sets $\{\mathcal{B}_\ell\}$.

Remark 13 (Impact of lateral mixing and cyclic residuals). *Lateral mixing appears only inside the argument of Φ and inflates constants via $R_{\text{mix}}^{(\ell)} = 1 + |\tau^{(\ell)}|\, B_\omega^{(\ell)}$ (with $B_\omega^{(\ell)} \le \|\omega^{(\ell)}\|_\infty N_{\max}^{(\ell)}$); it does not change the G^{-2} rate. If a cyclic residual path R is present and shared between $T^{(\ell)}$ and $\hat{T}^{(\ell)}$, it cancels in the blockwise difference and thus does not affect ε_ℓ, but it* does *contribute additively to $L_{T^{(\ell)}}$ in* (6). *For example, when $d_{\text{in}} = d_{\text{out}}$ and $R(\mathbf{z}) = w_{\text{res}}\mathbf{z}$, we may take $L_{T^{(\ell)}} \le L_\Phi^{(\ell)} R_{\text{mix}}^{(\ell)} \|\lambda^{(\ell)}\|_1 L_{\phi^{(\ell)}} + |w_{\text{res}}|$, with $L_{\phi^{(\ell)}} := \|\phi^{(\ell)\prime}\|_{L^\infty(I_\phi^{(\ell)})}$. Analogous bounds hold for broadcast/pooling residuals using the operator norm of the corresponding linear map.*

Remark 14 (Monotone spline parameterization). *Sprecher's construction requires the inner map to be nondecreasing. The piecewise-linear interpolant of a C^2 nondecreasing function on a uniform grid is itself nondecreasing (no oscillatory overshoot), and the $O(h^2)$ error bound used above holds unchanged. Hence the analysis is compatible with the monotone-spline parameterizations of ϕ used in this work.*

Remark 15 (Depth dependence). *The* $\mathcal{O}(G^{-2})$ rate *is dimension-free, while constants accumulate with depth through $\prod_{m=j+1}^{L} L_{T^{(m)}}$ as in* (6). *This mirrors standard error-propagation phenomena in deep models and highlights the practical value of regularizing $\|\lambda^{(\ell)}\|_1$, $L_{\phi^{(\ell)}}$, and $L_{\Phi^{(\ell)}}$, and of controlling lateral mixing $(\tau^{(\ell)}, \omega^{(\ell)})$.*

6.2 Vector-valued functions and deeper extensions

For vector-valued functions $f : [0,1]^n \to \mathbb{R}^m$ with $m > 1$, our construction appends an $(L+1)$-th block without final summation. While intuitively extending the representation, the universality of this specific construction is not directly covered by Sprecher's original theorem. The composition of multiple Sprecher blocks to create deep networks represents a natural but theoretically uncharted extension of Sprecher's construction. While single-layer universality is guaranteed, the expressive power of deep SNs remains an open question with several competing hypotheses. Depth might provide benefits analogous to those in standard neural networks: enabling more efficient representation of compositional functions, creating a more favorable optimization landscape despite the constrained parameter space, or allowing the network to gradually transform inputs into representations that are progressively easier to process. The addition of lateral mixing connections may further enhance these benefits by enabling richer transformations at each layer. Alternatively, the specific constraints of the SN architecture might interact with depth in unexpected ways, either amplifying the benefits of the structured representation or creating new challenges not present in single-layer networks.

Conjecture 1 (Vector-valued Sprecher Representation). *Let $n, m \in \mathbb{N}$ with $m > 1$, and let $f : [0,1]^n \to \mathbb{R}^m$ be any continuous function. Then for any $\epsilon > 0$, there exists a Sprecher Network with architecture $n \to [d_1] \to m$ (using $L = 1$ hidden block of width $d_1 \ge 2n+1$ and one output block), with sufficiently flexible continuous splines $\phi^{(1)}, \Phi^{(1)}, \phi^{(2)}, \Phi^{(2)}$ ($\phi^{(1)}, \phi^{(2)}$ monotonic), appropriate parameters $\lambda^{(1)}, \eta^{(1)}, \lambda^{(2)}, \eta^{(2)}$, and optionally lateral mixing parameters, such that the network output $\hat{f}(\mathbf{x})$ satisfies $\sup_{\mathbf{x}\in[0,1]^n} \|f(\mathbf{x}) - \hat{f}(\mathbf{x})\|_{\mathbb{R}^m} < \epsilon$.*

Furthermore, stacking multiple Sprecher blocks ($L > 1$) creates deeper networks. It is natural to hypothesize that these deeper networks also possess universal approximation capabilities, potentially offering advantages in efficiency or learning dynamics for certain function classes, similar to depth advantages observed in MLPs. The role of lateral mixing in enhancing or modifying these universality properties remains unexplored.

Conjecture 2 (Deep universality). *For any input dimension $n \ge 1$, any number of hidden blocks $L \ge 1$, and any continuous function $f : [0,1]^n \to \mathbb{R}$ (or $f : [0,1]^n \to \mathbb{R}^m$), and any $\epsilon > 0$, there exists a Sprecher Network with architecture $n \to [d_1, \dots, d_L] \to 1$ (or $\to m$), provided the hidden widths $d_1, \dots, d_L$ are sufficiently large (e.g., perhaps $d_\ell \ge 2d_{\ell-1}+1$ is sufficient, although likely not necessary), with sufficiently flexible continuous splines $\phi^{(\ell)}, \Phi^{(\ell)}$, appropriate parameters $\lambda^{(\ell)}, \eta^{(\ell)}$, and optionally lateral mixing parameters $\tau^{(\ell)}, \omega^{(\ell)}$, such that the network output $\hat{f}(\mathbf{x})$ satisfies $\sup_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \epsilon$ (or the vector norm equivalent).*

Proving Conjectures 1 and 2 rigorously would require analyzing the compositional properties and ensuring that the range of intermediate representations covers the domain needed by subsequent blocks, potentially involving careful control over the spline ranges, the effect of the shifts $\eta^{(\ell)}$, and the impact of lateral mixing on the network's expressive power.

7 Implementation considerations

7.1 Trainable splines

For practical implementations, we support both piecewise-linear splines and C^1 cubic Hermite splines with PCHIP (shape-preserving) slopes for both $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. Piecewise-linear splines are the default when only function values are required (high efficiency, simple evaluation); the cubic option is used when higher-order derivatives must flow through the spline (e.g., the Poisson PINN benchmark, which requires second derivatives). In both cases, a spline with G uniformly spaced knot locations (thus $G-1$ intervals) is parameterized by the values at these G knot locations (thus $O(G)$ parameters per spline); for the cubic case, the PCHIP endpoint slopes are computed deterministically from these knot values. For range propagation, piecewise-linear extrema on an interval occur at the interval endpoints and/or knot points, whereas for cubic splines we additionally consider interior stationary points when computing tight bounds (see Lemma 4 and Algorithm 1). Each spline can be defined by a set of knots (x-coordinates) and corresponding coefficients (y-coordinates). A common approach is to use fixed, uniformly spaced knots over the spline's domain, with learnable coefficients. The number of knots is a hyperparameter that affects both expressivity and computational cost.

For the inner spline $\phi^{(\ell)}$, we fix the codomain to $[0,1]$ and enforce *strict* monotonicity of the knot values by parameterizing positive differences. Concretely, let $(v_k)_k$ be unconstrained parameters and set $\Delta_k = \mathrm{softplus}(v_k) > 0$; then the coefficients are

$$u_k \;=\; \sum_{i \le k} \Delta_i, \qquad c_k \;=\; \frac{u_k}{u_{G-1} + \varepsilon},$$

with a tiny $\varepsilon > 0$ (we use $\varepsilon = 10^{-8}$) which yields strictly increasing knot values. Interpolating these knots with either piecewise–linear segments or a shape–preserving monotone C^1 cubic Hermite (PCHIP) scheme yields a monotone non-decreasing spline on its domain. Outside the domain we use constant extension, so $\phi^{(\ell)}(x) = 0$ for inputs below the leftmost knot and $\phi^{(\ell)}(x) = 1$ above the rightmost knot. The increments are initialized nearly uniform so that $\phi^{(\ell)}$ starts close to linear.

The outer spline $\Phi^{(\ell)}$ operates on a domain determined by the block's bounds, and its codomain is parameterized as an interval $[\, c_c^{(\ell)} - c_r^{(\ell)},\, c_c^{(\ell)} + c_r^{(\ell)} \,]$ with trainable center $c_c^{(\ell)}$ and scale (initialized positive) $c_r^{(\ell)}$. We initialize $(c_c^{(\ell)}, c_r^{(\ell)})$ from the computed input domain so that $\Phi^{(\ell)}$ is near identity at initialization, and the parameters then adapt the output range during training. Monotonicity is not required for $\Phi^{(\ell)}$.

7.2 Shifts, weights, and optimization

Each block includes the learnable scalar shift $\eta^{(\ell)}$ and the learnable mixing weight vector $\lambda^{(\ell)} \in \mathbb{R}^{d_{\ell-1}}$. While Sprecher's original construction requires the shift parameter to be positive ($\eta > 0$), practical implementations can relax this constraint. As shown in Lemma 4, the theoretical domain computation handles both positive and negative values of $\eta^{(\ell)}$, allowing it to be trained as an unconstrained parameter initialized to a small positive value.

When lateral mixing is enabled, each block additionally includes a lateral scale parameter $\tau^{(\ell)}$ (typically initialized to a small value like 0.1) and lateral mixing weights $\omega^{(\ell)}$. For cyclic mixing, this involves d_ℓ weights (one per output), while bidirectional mixing requires $2d_\ell$ weights (forward and backward for each output). These weights are typically initialized to small values to ensure training begins with minimal lateral communication, allowing the network to gradually learn the optimal mixing patterns.

All learnable parameters (spline coefficients, $\eta^{(\ell)}$, $\lambda^{(\ell)}$, lateral mixing parameters, and potentially range parameters for $\Phi^{(\ell)}$) are trained jointly using gradient-based optimization methods like Adam [6] or LBFGS. The loss function is typically Mean Squared Error (MSE) for regression tasks. Due to the constrained parameter space and shared spline structure, SNs may require more training iterations than equivalent MLPs or KANs to converge, though the per-iteration computational cost is typically lower due to fewer parameters.

7.3 Memory-efficient forward computation

A key advantage of SNs' vector-based weight structure extends beyond parameter count to memory footprint, in particular to peak forward-intermediate memory. While dense MLP layers require $O(N^2)$ parameters (and thus memory) for their weight matrices alone, SN blocks have only $O(N)$ vector parameters (plus $O(G)$ spline parameters) and can be evaluated without materializing the full $d_{\text{in}} \times d_{\text{out}}$ outer-product intermediate.

The standard forward pass for a Sprecher block naively computes:

$$\text{shifted}_{b,i,q} = x_{b,i} + \eta q \quad \forall b \in [B], i \in [d_{\text{in}}], q \in [d_{\text{out}}]$$

storing the full tensor before applying ϕ, requiring $O(B \cdot d_{\text{in}} \cdot d_{\text{out}})$ memory. However, we can compute the *unmixed* pre-activations

$$s_{b,q} = \sum_{i=1}^{d_{\text{in}}} \lambda_i \phi(x_{b,i} + \eta q) + \alpha q$$

sequentially for $q = 0, \ldots, d_{\text{out}} - 1$ without materializing $\text{shifted}_{b,i,q}$. When lateral mixing is enabled, we then apply

$$\tilde{s}_{b,q} = s_{b,q} + \tau \sum_{j \in \mathcal{N}(q)} \omega_{q,j}\, s_{b,j}$$

to the assembled vector $(s_{b,q})_{q=0}^{d_{\text{out}}-1}$ before evaluating $\Phi(\tilde{s}_{b,q})$ (and adding the residual path).

This reformulation reduces peak *forward-intermediate* memory from $O(B{\cdot}N^2)$ to $O\big(B{\cdot}\max(d_{\text{in}}, d_{\text{out}})\big)$ during computation, while producing *mathematically identical* results. Combined with SNs' $O(N{+}G)$ parameter memory per block, the total (parameters + peak forward intermediates) scales as $O\big(N{+}G{+}B{\cdot}\max(d_{\text{in}}, d_{\text{out}})\big)$; for fixed B and G, this is $O(N)$, compared to dense MLPs' $O(N^2)$ parameter memory per block. (In standard reverse-mode autodiff frameworks, training may require additional saved state for backpropagation; the bound above refers to forward intermediates/inference, or to training with recomputation/checkpointing.)

In practice, this enables training wider architectures under fixed memory budgets by avoiding materialization of the full $(B \times d_{\text{in}} \times d_{\text{out}})$ tensor of shifted inputs. The runtime impact is workload-dependent: sequential evaluation reduces intra-layer parallelism, but can also improve cache behavior and reduce allocator pressure for very large layers.

Remark 16 (Preservation of mathematical structure)**.** *The sequential computation is a pure implementation optimization that is mathematically exact: it produces identical outputs to the naive parallel formulation for any choice of parameters and inputs. It only reduces peak memory by avoiding materialization of the full $(B \times d_{\text{in}} \times d_{\text{out}})$ tensor of shifted inputs; it does not change the function being computed.*

Remark 17 (Parallelism vs. memory trade-off)**.** *Sequential (per-q) evaluation reduces peak activation memory from $O(B\, d_{\text{in}} d_{\text{out}})$ to $O\big(B\, \max\{d_{\text{in}}, d_{\text{out}}\}\big)$, at the cost of less intra-layer parallelism. This is often advantageous on accelerators with ample compute but constrained memory.*

7.4 Theoretical domain computation

One advantage of the structured Sprecher architecture is the ability to compute *provably safe* interval bounds for spline inputs and outputs, which we use to adapt spline domains during training. For a fixed input interval to a spline, the exact range of a piecewise-linear spline on that interval is attained at the endpoints and any interior knots; for shape-preserving cubic (PCHIP) splines, stationary points within knot intervals must also be checked. When these per-spline range computations are propagated through multiple blocks using interval arithmetic, the resulting network-level bounds are generally conservative (since correlations between coordinates are ignored), but are typically much tighter than naive global bounds. Throughout this analysis, we assume inputs lie in $[0, 1]^n$, though the methodology extends naturally to any bounded domain. This assumption enables principled initialization and can aid in debugging and analysis.

Lemma 4 (Domain propagation through Sprecher blocks)**.** *Consider a Sprecher block where inputs lie in a box $\prod_{i=1}^{d_{\text{in}}} [a_i, b_i]$ (and define $a := \min_i a_i$ and $b := \max_i b_i$), with shift parameter η, weights λ_i, lateral mixing parameters τ, ω, and output dimension d_{out}. Let us distinguish between a spline's* domain *(the interval of valid inputs) and its* codomain *(the interval of possible outputs). Then:*

1. *The domain required for ϕ to handle all possible inputs is:*

$$\mathcal{D}_\phi = \begin{cases} [a, b + \eta(d_{\text{out}} - 1)] & \text{if } \eta \geq 0 \\ [a + \eta(d_{\text{out}} - 1), b] & \text{if } \eta < 0 \end{cases}$$

2. *Without lateral mixing, monotonicity of ϕ implies $\phi(x_i + \eta q) \in [\phi(a + \eta q),\ \phi(b + \eta q)]$ for all i and q. Let $\lambda^+ := \sum_{i:\lambda_i \geq 0} \lambda_i$ and $\lambda^- := \sum_{i:\lambda_i < 0} \lambda_i$. Then for each head index q,*

$$s_q^{\min,\text{unmixed}} = \lambda^+ \phi(a + \eta q) + \lambda^- \phi(b + \eta q) + \alpha q, \qquad s_q^{\max,\text{unmixed}} = \lambda^+ \phi(b + \eta q) + \lambda^- \phi(a + \eta q) + \alpha q.$$

A valid domain for Φ is therefore $\mathcal{D}_\Phi = \left[\min_q s_q^{\min,\text{unmixed}},\ \max_q s_q^{\max,\text{unmixed}}\right]$. (As a coarse bound, one may use only that $\phi(\cdot) \in [0, 1]$ and thus replace the unknown values $\phi(a + \eta q)$ and $\phi(b + \eta q)$ by 0 and 1, respectively.)

3. *With lateral mixing, for cyclic mixing with scale τ and weights ω, the domain for Φ must account for sign-aware neighbor contributions. Using the* unmixed *bounds from the previous item, for each output q the mixed pre-activation bounds are:*

$$s_q^{\min} = s_q^{\min,\textit{unmixed}} + (\tau\omega_q)^+ s_{(q+1) \bmod d_{\text{out}}}^{\min,\textit{unmixed}} + (\tau\omega_q)^- s_{(q+1) \bmod d_{\text{out}}}^{\max,\textit{unmixed}}$$
$$s_q^{\max} = s_q^{\max,\textit{unmixed}} + (\tau\omega_q)^+ s_{(q+1) \bmod d_{\text{out}}}^{\max,\textit{unmixed}} + (\tau\omega_q)^- s_{(q+1) \bmod d_{\text{out}}}^{\min,\textit{unmixed}}$$

where $a^+ = \max(a, 0)$ and $a^- = \min(a, 0)$. For bidirectional mixing, apply the analogous sign-split bound to each neighbor contribution $(\tau\omega_{q,fwd})\, s_{(q+1) \bmod d_{\text{out}}}$ and $(\tau\omega_{q,bwd})\, s_{(q-1) \bmod d_{\text{out}}}$, and sum the contributions.

4. *Tight output bounds require checking all candidate extrema of Φ over the interval. For piecewise linear Φ, extrema occur at endpoints and knot points, so it suffices to evaluate Φ at interval endpoints and all knots within the interval. For cubic Φ (PCHIP), extrema may also occur at internal critical points where Φ' is zero, requiring evaluation at those points. (If an affine codomain reparameterization (c_c, c_r) is used, it is an affine map of the underlying spline values; it does not change the set of candidate points for extrema, only rescales and shifts the resulting values.) This yields correct range propagation even for oscillatory Φ.*

5. *When residual connections are present, the output range must be adjusted by the residual contribution. For a scalar residual weight w (the $d_{\text{in}} = d_{\text{out}}$ case), the residual term in coordinate q is $r_q = wx_{q+1}$ and lies in $[\min\{wa_{q+1}, wb_{q+1}\}, \max\{wa_{q+1}, wb_{q+1}\}]$. For a linear residual projection matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ (the $d_{\text{in}} \neq d_{\text{out}}$ case), the residual term in coordinate q is $r_q = \sum_{i=1}^{d_{\text{in}}} W_{i,q} x_i$ and lies in:*

$$r_q^{\min} = \sum_{i=1}^{d_{\text{in}}} \left(W_{i,q}^+ a_i + W_{i,q}^- b_i\right), \qquad r_q^{\max} = \sum_{i=1}^{d_{\text{in}}} \left(W_{i,q}^+ b_i + W_{i,q}^- a_i\right),$$

where $W^+ = \max(W, 0)$ and $W^- = \min(W, 0)$ elementwise. These adjustments can be applied elementwise. For Cyclic residuals:

- ***Broadcast**: Each output q receives $w_q^{bcast} \cdot x_{1+(q \bmod d_{\text{in}})}$*
- ***Pooling**: With assignment $i \mapsto q(i)$ and weights w_i, the contribution to output q is:*

$$r_q^{\min} = \sum_{i:q(i)=q} \min\{w_i a_i, w_i b_i\}, \quad r_q^{\max} = \sum_{i:q(i)=q} \max\{w_i a_i, w_i b_i\}$$

6. ***Out-of-domain extension:** When inputs fall outside the spline's domain:*

- *ϕ (monotonic): Uses* constant *extension (zero slope) outside its domain: for $x < \min \mathcal{D}_\phi$, set $\phi(x) = 0$; for $x > \max \mathcal{D}_\phi$, set $\phi(x) = 1$. This preserves the codomain $[0, 1]$ and aligns with the monotone-increments parameterization.*

- Φ *(general): Uses* linear *extrapolation based on the boundary* derivative *of the spline interpolation (piecewise-linear: slope of the first/last segment; cubic PCHIP: endpoint derivative from the PCHIP slope rule). If Φ's codomain is parameterized by (c_c, c_r) with* $\mathrm{cod}(\Phi) = [c_c - c_r,\, c_c + c_r]$, *we first evaluate the* unnormalized *spline $\tilde{\Phi}$ (including its linear extrapolation) in coefficient space, then map its value $t = \tilde{\Phi}(x)$ by*

$$t \;\mapsto\; c_c - c_r \;+\; 2c_r \cdot \frac{t - t_{\min}}{t_{\max} - t_{\min} + \epsilon}\,,$$

where $t_{\min} = \min_k \tilde{\Phi}(u_k)$ and $t_{\max} = \max_k \tilde{\Phi}(u_k)$ are the minimum/maximum unnormalized knot values (with $\{u_k\}$ the knot locations) and $\epsilon > 0$ is a small constant. The boundary derivatives are scaled by the same factor $2c_r/(t_{\max} - t_{\min} + \epsilon)$.

Corollary 2 (Per-dimension input intervals)**.** *When per-dimension intervals $x_i \in [a_i, b_i]$ are available, tighter bounds can be computed:*

$$s_q^{\min} = \sum_i \big(\lambda_i^+ \phi(a_i + \eta q) + \lambda_i^- \phi(b_i + \eta q)\big) + \alpha q$$

$$s_q^{\max} = \sum_i \big(\lambda_i^+ \phi(b_i + \eta q) + \lambda_i^- \phi(a_i + \eta q)\big) + \alpha q$$

where $\lambda_i^+ = \max(\lambda_i, 0)$ and $\lambda_i^- = \min(\lambda_i, 0)$. These per-$q$ bounds then undergo sign-aware lateral mixing as in Lemma 4.

This lemma enables a forward propagation algorithm for computing all spline domains throughout the network. The algorithm can optionally apply a small relative domain safety margin (e.g., 1%) to reduce edge hits during training; setting the margin to 0 yields the tightest domains. Crucially, the algorithm's efficiency relies on the fact that spline extrema over compact intervals can be found by evaluating a small, explicit candidate set: for piecewise-linear splines, the interval endpoints and any knots inside the interval; for piecewise-cubic Hermite/PCHIP splines, the same candidates plus any interior critical points where the derivative vanishes (one quadratic solve per segment).

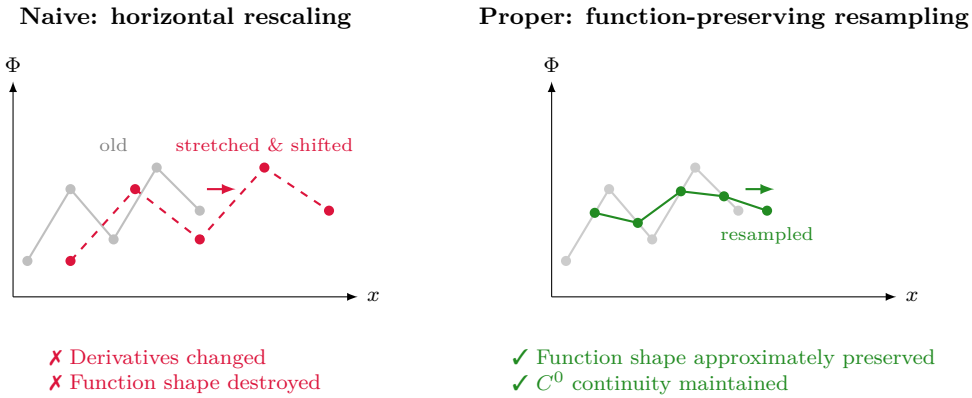


Figure 7: Spline domain resampling during training (applied to the outer spline Φ). As parameters λ and η evolve, induced spline input ranges shift dynamically. **Left:** Naive horizontal rescaling distorts the learned function by stretching and shifting the shape to fit the new domain. **Right:** Our resampling evaluates the previous spline at the new knot positions (for piecewise-linear Φ, query points are clamped to the previous domain when a new knot lies outside it; for cubic/PCHIP Φ, the previous spline's linear extrapolation is used), approximately preserving the function values at the new knots that lie within the previous domain; the updated Φ still uses linear extrapolation outside its *new* domain.

Remark 18 (Dynamic spline updates and function preservation)**.** *A critical challenge in training Sprecher Networks is that the domains of the splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ evolve as the parameters $\eta^{(\ell)}$, $\lambda^{(\ell)}$, and lateral mixing parameters are updated. To maintain theoretical fidelity to Sprecher's formula while adapting to evolving domains, we employ a selective function-preserving resampling strategy:*

Algorithm 1 Forward domain propagation with lateral mixing and per-dimension tracking

Input: Network parameters $\{\lambda^{(\ell)}, \eta^{(\ell)}, \phi^{(\ell)}, \Phi^{(\ell)}, \tau^{(\ell)}, \omega^{(\ell)}, R^{(\ell)}\}_{\ell=1}^{L_{\text{blocks}}}$, input domain $[0,1]^n$
(Here L_{blocks} is the number of Sprecher blocks: $L_{\text{blocks}} = L$ for scalar outputs, and $L_{\text{blocks}} = L{+}1$ for vector outputs.)
Output: Domains and ranges for all splines
Initialize current range $\mathcal{R}_0 \leftarrow [0,1]^n$ (per-dimension when available)
for each block $\ell = 1, \ldots, L_{\text{blocks}}$ **do**
    if normalization is applied *before* block ℓ **then**
        Apply normalization bounds to $\mathcal{R}_{\ell-1}$ (training: conservative shared bound derived from standardized $[-4,4]$; eval: use running stats and affine)
    end if
    $\mathcal{D}_{\phi}^{(\ell)} \leftarrow$ apply Lemma 4 part (1) to the current input range
    if per-dimension intervals available **then**
        Compute per-q bounds using the corollary (sign-splitting via $\lambda_i^{\pm}$ and $\phi(a_i + \eta q), \phi(b_i + \eta q)$)
    else
        Compute coarse per-q bounds $s_q^{\min,\text{unmixed}}, s_q^{\max,\text{unmixed}}$ via Lemma 4 part (2)
    end if
    Apply sign-aware lateral mixing (part 3) to get final $s_q^{\min}, s_q^{\max}$
    $\mathcal{D}_{\Phi}^{(\ell)} \leftarrow [\min_q s_q^{\min}, \max_q s_q^{\max}]$
    Evaluate $\Phi^{(\ell)}$ on each interval $[s_q^{\min}, s_q^{\max}]$ at (i) the endpoints, (ii) all Φ-knots lying inside the interval, and (iii) for cubic/PCHIP Φ, all interior critical points where $\frac{d}{dx}\Phi(x) = 0$
    Let $[y_q^{\min}, y_q^{\max}]$ denote the range of $\Phi^{(\ell)}$ on $[s_q^{\min}, s_q^{\max}]$
    if block ℓ has residual connections **then**
        Adjust each $[y_q^{\min}, y_q^{\max}]$ according to Lemma 4 part (5)
    end if
    if $\ell = L_{\text{blocks}}$ and the network output is scalar (sum over q) **then**
        $\mathcal{R}_\ell \leftarrow \left[\sum_{q=0}^{d_\ell - 1} y_q^{\min}, \sum_{q=0}^{d_\ell - 1} y_q^{\max}\right]$
    else
        $\mathcal{R}_\ell \leftarrow \prod_{q=0}^{d_\ell - 1} [y_q^{\min}, y_q^{\max}]$
    end if
    if normalization is applied *after* block ℓ **then**
        Apply normalization bounds (training: conservative shared bound derived from standardized $[-4,4]$; eval: use running stats and affine)
    end if
end for

General Splines ($\Phi^{(\ell)}$): *When the domain of a $\Phi^{(\ell)}$ spline changes, new knot locations are established (typically uniformly spaced across the new computed domain). The original spline is treated as a continuous function and evaluated at these new knot locations to yield updated knot coefficients (for piecewise-linear $\Phi^{(\ell)}$, query points are clamped to the previous domain so any new knot outside the old domain uses the nearest boundary value; for cubic/PCHIP $\Phi^{(\ell)}$, the previous spline's linear extrapolation is used outside the old domain). This effectively "resamples" the learned shape onto the new domain: it preserves the* values *of the old spline at the new knot locations that lie within the previous domain (and therefore preserves the function exactly on the overlap whenever the old spline is linear on each new knot interval, e.g. when the new knot set contains the old knots). For piecewise-linear $\Phi^{(\ell)}$, this procedure is exact whenever the new knot set contains the old knots; for cubic (PCHIP) $\Phi^{(\ell)}$, it matches the old spline at the resampled knots but can (slightly) change derivatives between knots, while still avoiding abrupt resets. Optionally, one may additionally apply a cheap post-update refresh that adjusts knot* locations *without resampling coefficients; this is not function-preserving (it amounts to a mild horizontal rescaling), so it is best reserved for small domain changes where the primary goal is to keep bound propagation synchronized after parameter updates.*

Monotonic Splines ($\phi^{(\ell)}$): *For the monotonic splines, whose purpose is to provide an increasing map to $[0, 1]$, complex resampling is not required. Their learnable parameters define relative increments between knots, not an arbitrary shape. Therefore, a straightforward update of the knot positions to the new theoretical domain is sufficient and computationally efficient.*

This targeted approach avoids the most direct sources of information loss and instability from domain changes. In engineering ablations on Toy-2D (Table 4), disabling Φ-resampling typically led to oscillatory or unstable training as domains shifted, while disabling dynamic domain updates caused spline inputs to drift far outside their learned ranges (forcing ϕ into saturation and Φ into extrapolation). We therefore treat dynamic domains and Φ-resampling as default components unless explicitly stated otherwise (e.g., the PINN setting in Sec. 8). The fundamental challenge of optimizing splines within dynamically shifting domains remains, but this mitigation strategy has proven effective in practice.

One particularly useful application of domain computation is the initialization of $\Phi^{(\ell)}$ splines. When Φ codomain scaling $(c_c^{(\ell)}, c_r^{(\ell)})$ is enabled, we compute each $\Phi^{(\ell)}$'s theoretical domain before training and (i) set its knot domain to this interval, (ii) initialize its knot coefficients to a linear ramp over that same interval so that the underlying spline represents the identity map on the interval (for both linear and cubic/PCHIP interpolation), and (iii) initialize $(c_c^{(\ell)}, c_r^{(\ell)})$ to the domain center and radius so that the affine codomain reparameterization preserves this identity map. This provides a principled initialization strategy that ensures the initial network performs a series of near-identity transformations.

Remark 19 (Practical benefits)**.** *The ability to compute* provably safe *(typically conservative) domain and range bounds through interval arithmetic provides several practical benefits: (i) it enables theoretically grounded initialization without arbitrary hyperparameters, (ii) it can help diagnose training issues by detecting when values fall outside expected ranges, (iii) it promotes numerical stability by encouraging spline evaluations to remain within their well-resolved knot regions (with explicit extension/extrapolation behavior outside the knot domain), and (iv) it allows for adaptive domain adjustments that account for lateral mixing dynamics. These benefits distinguish Sprecher Networks from architectures with less structured internal representations.*

8 Empirical demonstrations and case studies

While comprehensive benchmarking remains future work, we provide initial empirical demonstrations to illustrate the feasibility and characteristics of SNs, including the impact of lateral mixing connections.

Benchmarks used in this work. We evaluate Sprecher Networks on synthetic supervised regression tasks and on a physics-informed Poisson problem.

Component ablated	Typical observed effect
Disable domain resampling	Training can become unstable when pre-activations drift beyond initial spline domains, increasing extrapolation error.
Disable domain updates	Intermediate pre-activations may drift outside predicted bounds, reducing effective capacity via saturation/clipping effects.
Remove normalization	Pre-activations become poorly scaled, making spline-domain management harder and slowing optimization.
Remove Φ codomain scaling (c_c, c_r)	Forces Φ to learn both scale and shape, often slowing convergence and degrading final accuracy.
Remove cyclic residuals	Deep networks become harder to train, with degraded accuracy and occasional divergence.
Remove lateral mixing	Wide shallow networks can plateau due to shared-weight symmetries (cf. Remark above).

Table 4: Qualitative summary of component ablations observed across our experiments. These effects align with each component's motivation: domain management for numerical stability, normalization and Φ codomain scaling for conditioning, residual connections for depth, and lateral mixing for breaking shared-weight symmetries.

Poisson PINN benchmark (manufactured solution). We solve the Dirichlet problem

$$\Delta f(\mathbf{x}) = g(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega, \qquad f(\mathbf{x}) = 0 \quad \text{for } \mathbf{x} \in \partial\Omega,$$

with $\Omega = [-1, 1]^2$, using the manufactured solution

$$u(x, y) = \sin(\pi x)\, \sin(\pi y^2),$$

so that $g = \Delta u$ is known analytically. We train with collocation sets $S_{\text{int}} \subset \Omega$ and $S_\partial \subset \partial\Omega$ using

$$\mathcal{L} \;=\; \frac{1}{|S_{\text{int}}|} \sum_{\mathbf{x} \in S_{\text{int}}} \frac{\big(\Delta f(\mathbf{x}) - g(\mathbf{x})\big)^2}{\mathbb{E}_{\mathbf{x} \in S_{\text{int}}}[g(\mathbf{x})^2]} \;+\; \frac{1}{|S_\partial|} \sum_{\mathbf{x} \in S_\partial} f(\mathbf{x})^2.$$

Unless stated otherwise, we draw a fixed set of $|S_{\text{int}}| = 2048$ interior points i.i.d. uniformly at random in Ω (sampled once at the start of training), and $|S_\partial| = 4 \cdot 257$ boundary points are taken on a uniform grid along each edge (including the corners on each edge, hence corners are duplicated).

Model variants for the PINN benchmark. To isolate the spline parameterization itself, we compare a **barebones Sprecher Network** (no lateral mixing, no residuals, no normalization, and **fixed** spline domains; i.e. no domain updates) against a **barebones KAN** baseline with spline-only edge activations (no SiLU/base activation and no grid updates). Because the Poisson residual involves second derivatives, both models use shape-preserving cubic (PCHIP) splines in this benchmark (piecewise-linear splines yield $\Delta f_\theta = 0$ almost everywhere). For the Sprecher Network, inputs are linearly rescaled from $[-1, 1]^2$ to $[0, 1]^2$ before the first spline evaluation, matching the $[0, 1]$ spline-domain convention used for SNs in this work; the KAN baseline operates directly on $[-1, 1]^2$. In both cases, derivatives are taken with respect to the original coordinates via automatic differentiation (for SN this is through the rescaling map).

Optimization and evaluation. We train in `torch.float64` using Adam with 10^{-3} learning rate, cosine learning rate decay to 10^{-5}, and gradient-norm clipping 1.0. We report two parameter budgets: an ≈1200-parameter model trained for 5,000 epochs (mean ± std over 5 seeds), and an ≈3000-parameter model trained for 20,000 epochs (1 seed). We evaluate both models on a uniform 101×101 grid over $\Omega = [-1, 1]^2$ and report: (i) PDE residual MSE (interior grid points), (ii) boundary MSE (boundary grid points), and (iii) L^2 MSE against the manufactured solution (full grid).

8.1 Basic function approximation

We train SNs on datasets sampled from known target functions f. The network learns the parameters ($\eta^{(\ell)}$, $\lambda^{(\ell)}$), spline coefficients ($\phi^{(\ell)}$, $\Phi^{(\ell)}$), and when enabled, lateral mixing parameters ($\tau^{(\ell)}$, $\omega^{(\ell)}$) via gradient descent (Adam optimizer) using an MSE objective.

Table 5: **Poisson PINN results (manufactured solution).** Metrics are evaluated on a uniform 101×101 grid over $\Omega = [-1, 1]^2$: PDE residual MSE is computed on the interior grid points, boundary MSE on the boundary grid points, and L^2 MSE against the manufactured solution on the full grid (all lower is better). For the ≈1200-parameter setting we report mean ± std over 5 seeds; the ≈3000-parameter setting is reported for 1 seed due to cost.

Model	PDE MSE	Bndry MSE	L^2 MSE
≈1200 params, 5,000 epochs (5 seeds)			
SN	$\mathbf{1.36 \times 10^2 \pm 3.69}$	$7.85 \times 10^{-2} \pm 7.22 \times 10^{-3}$	$\mathbf{6.55 \times 10^{-2} \pm 1.75 \times 10^{-3}}$
KAN	$2.41 \times 10^3 \pm 1.78 \times 10^3$	$\mathbf{2.06 \times 10^{-4} \pm 4.53 \times 10^{-5}}$	$1.70 \times 10^{-1} \pm 4.33 \times 10^{-3}$
≈3000 params, 20,000 epochs (1 seed)			
SN	$\mathbf{1.38 \times 10^2}$	4.76×10^{-2}	$\mathbf{1.05 \times 10^{-1}}$
KAN	1.09×10^4	$\mathbf{8.42 \times 10^{-4}}$	1.32×10^{-1}

For 1D functions $f(x)$ on $[0, 1]$, an SN like $1 \to [W] \to 1$ (one block) learns $\phi^{(1)}$ and $\Phi^{(1)}$ that accurately approximate f, effectively acting as a learnable spline interpolant structured according to Sprecher's formula. Figure 8 provides an example where an SN is trained on data generated from a known Sprecher-style superposition. While the network achieves a very accurate approximation of the overall function $f(x)$, the learned components (splines $\hat{\phi}, \hat{\Phi}$, weights $\hat{\lambda}$, shift $\hat{\eta}$) only partially resemble the ground truth functions and parameters used to generate the data. This is expected: the internal decomposition is generally not identifiable, and in particular the learned outer spline $\hat{\Phi}$ can become strongly oscillatory/jagged even when the ground-truth outer function Φ is smooth.

Moving to multivariate functions, consider the 2D scalar case $f(x, y) = \exp(\sin(11x)) + 3y + 4\sin(8y)$. A network like $2 \to [5, 8, 5] \to 1$ (3 blocks) can achieve high accuracy. Figure 9 shows the interpretable layerwise spline plots and the final fit quality. When lateral mixing is enabled for this architecture, we often observe a different allocation of complexity across channels and layers; however, the learned $\Phi^{(\ell)}$ splines can still be highly oscillatory, so we treat the spline plots as qualitative rather than expecting smoothness.

For 2D vector-valued functions $f(x, y) = (f_1(x, y), f_2(x, y))$, we use an SN of the form $2 \to [20, 20] \to 2$ (2 hidden layers plus a final block for the vector-valued output; 3 blocks total). Figure 10 illustrates the learned splines and the approximation of both output surfaces. Lateral mixing can be beneficial here by breaking output symmetry across channels, so we enable it in this example.

These examples demonstrate the feasibility of training SNs and the potential interpretability offered by visualizing the learned shared splines $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ for each block, as well as the impact of lateral mixing on spline smoothness and approximation quality.

8.2 Impact of lateral mixing

To evaluate the contribution of lateral mixing, we conducted ablation studies comparing networks with and without this mechanism across various tasks:

Scalar outputs: For many scalar-output regression tasks with moderate widths, lateral mixing yields only modest gains. However, scalar-output SNs can still benefit substantially in regimes where the shared-weight constraint induces strong symmetry across channels (e.g., very wide shallow networks), where lateral mixing provides lightweight cross-channel interaction that can break optimization plateaus (see the $2 \to [120] \to 1$ example above). Note that the final summation is a fixed linear readout *after* applying Φ channelwise; it does not provide pre-Φ cross-channel communication, so lateral mixing remains a distinct (and sometimes crucial) mechanism even when the network output is scalar.

Vector outputs: On vector-valued regression tasks, networks with lateral mixing (cyclic variant) often achieved lower RMSE with only a marginal increase in parameters (one additional per-channel weight plus a shared scale). The improvement was most pronounced for functions where output dimensions exhibit strong

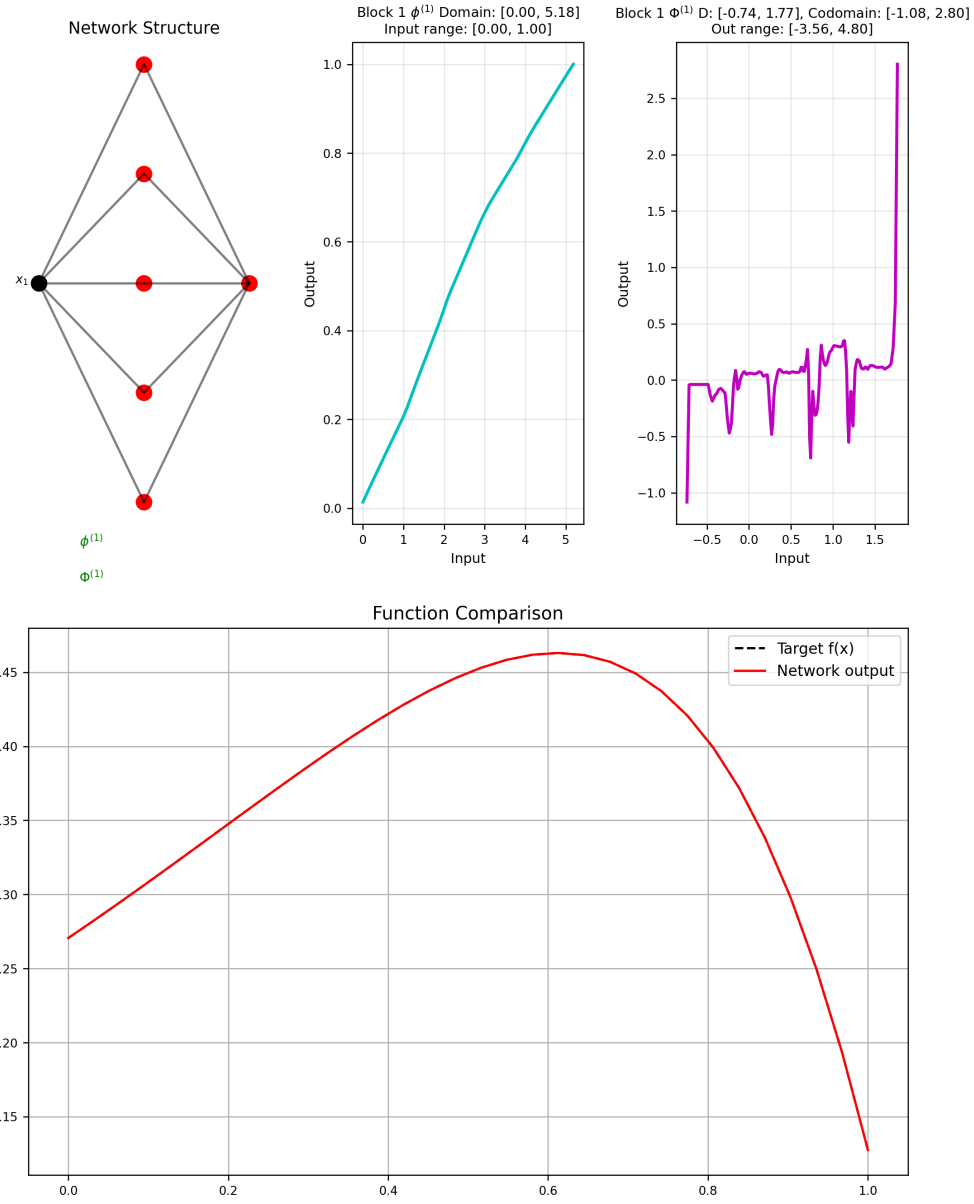


Figure 8: Visualization of a trained Sprecher Network with architecture $1 \to [5] \to 1$ (one block) trained on data sampled from $f(x) = \sum_{q=0}^{4} \Phi(a_q \phi(x + \eta q) + q)$, where the ground truth functions were $\phi(x) = (e^x - 1)/(e-1)$ and $\Phi(x) = \sin x$, with shift $\eta = 1/10$ and channel coefficients $\{a_q\}_{q=0}^{4} = \{1/2, -4/5, 1, 1/5, -6/5\}$. Top row: Network structure, learned monotonic spline $\phi^{(1)}$ (cyan), learned general spline $\Phi^{(1)}$ (magenta). Bottom row: Comparison between the target function $f(x)$ (dashed black) and the network output (solid red). Even though the ground-truth outer function Φ here is smooth, the learned outer spline $\Phi^{(1)}$ can become highly oscillatory/jagged while still yielding an accurate composite approximation, consistent with the role of Φ in Sprecher-style constructions. (This run uses the default cyclic lateral mixing; for 1D targets it typically has only a minor effect.)

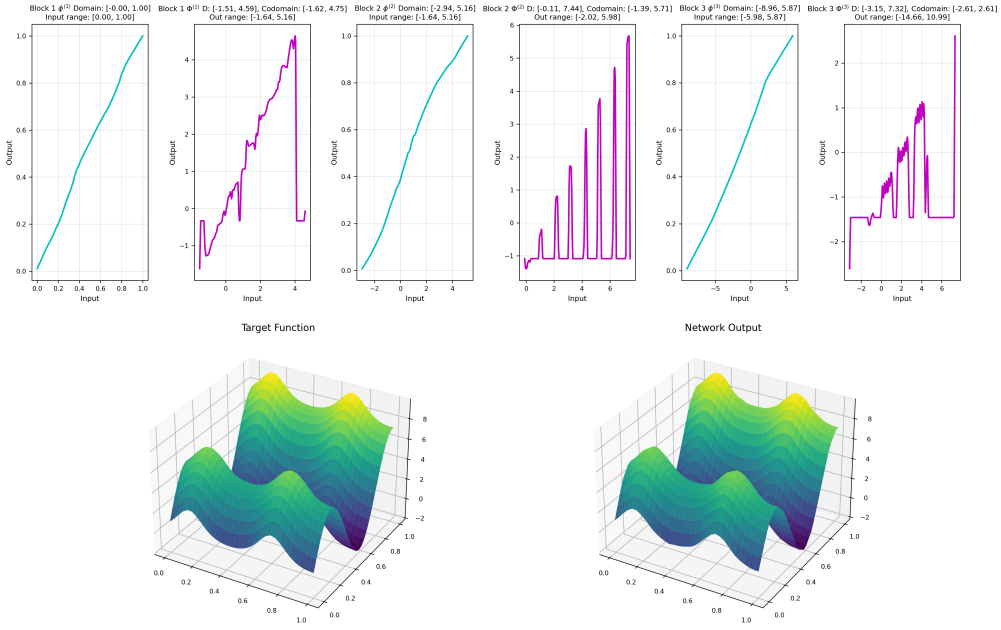


Figure 9: Example of SN approximating the scalar 2D target function $z = f(x, y) = \exp(\sin(11x)) + 3y + 4\sin(8y)$ on $(x, y) \in [0, 1]^2$ using architecture $2 \to [5, 8, 5] \to 1$ (3 blocks). Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta). Bottom row: Comparison between the target function surface (left) and the network approximation (right). This run uses cyclic lateral mixing (default); the learned outer splines $\Phi^{(\ell)}$ can be highly oscillatory/jagged even when the overall fit is accurate, so the per-layer spline plots should be interpreted qualitatively.

correlations, suggesting that lateral mixing can help the network capture cross-output dependencies despite the constraint of shared splines.

Convergence speed: Networks with lateral mixing often converged faster during training in our ablations, reaching the same loss threshold in fewer iterations. This suggests that lateral connections can provide beneficial gradient pathways that accelerate optimization, although the effect is task-dependent.

Spline behavior: Visual inspection of learned splines suggests that lateral mixing can change how complexity is distributed across output channels. In particular, the learned outer splines $\Phi^{(\ell)}$ can remain highly oscillatory even when the overall fit is accurate, which is consistent with Sprecher's original construction. KANs are often motivated by interpretability because each edge carries an explicit learned univariate function [9]; however, as Figure 11 illustrates, these learned edge functions can also be jagged/noisy under parameter parity, so visual interpretability does not automatically imply smoothness.

8.3 Baseline apples-to-apples comparisons: SNs vs. KANs

Rationale. The goal of this subsection is to provide simple, controlled, apples-to-apples tests where Sprecher Networks (SNs) and Kolmogorov–Arnold Networks (KANs) are trained under the same budget and with matched modeling capacity, so that we can isolate architectural inductive biases rather than hyperparameter tuning. We do *not* claim that SNs dominate KANs on every task; rather, we highlight representative settings where SNs perform as well or better, sometimes substantially so, while keeping the comparison fair.

Protocol (fairness constraints). Unless otherwise stated (e.g., Figure 11), each head-to-head uses exactly 4000 training epochs for both architectures. Both models use the same *linear* residual style: a learned

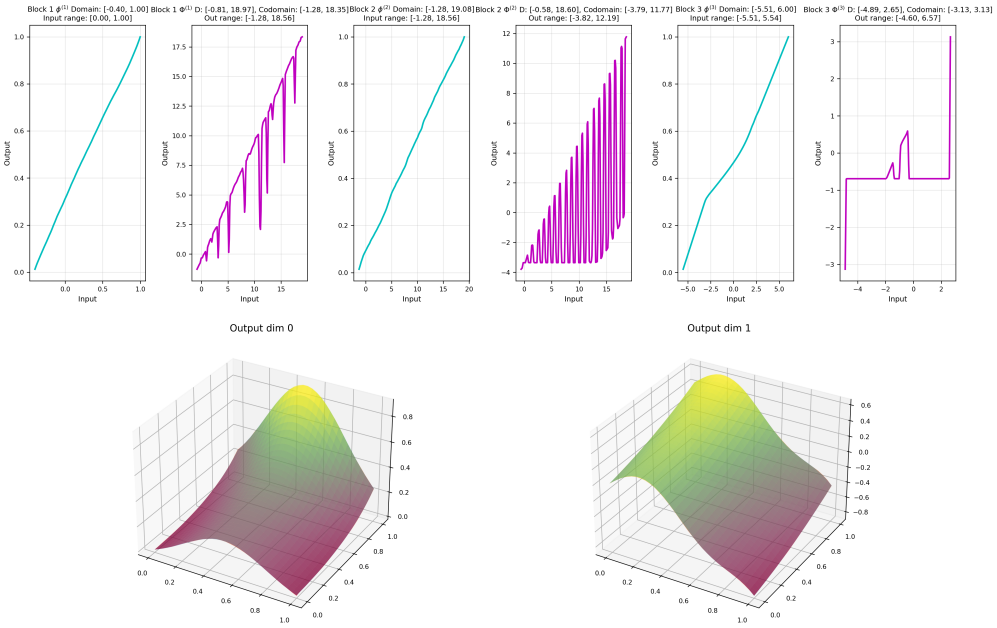


Figure 10: Example of SN approximating a 2D vector-valued function $f : [0,1]^2 \to \mathbb{R}^2$ with components $f_1(x,y) = \frac{\exp(\sin(\pi x)+y^2)-1}{7}$ and $f_2(x,y) = \frac{1}{4}y + \frac{1}{5}y^2 - x^3 + \frac{1}{5}\sin(7x)$. The network architecture is $2 \to [20,20] \to 2$ (2 hidden layers plus an additional output block; 3 blocks total). The first row shows the learned monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta) for each block. The second row shows, for each output component (left: f_1, right: f_2), the target surface overlaid with the corresponding network output surface. Lateral mixing was enabled (cyclic in this run); as in other examples, the learned $\Phi^{(\ell)}$ splines can be highly oscillatory/jagged even when the overall fit is accurate.

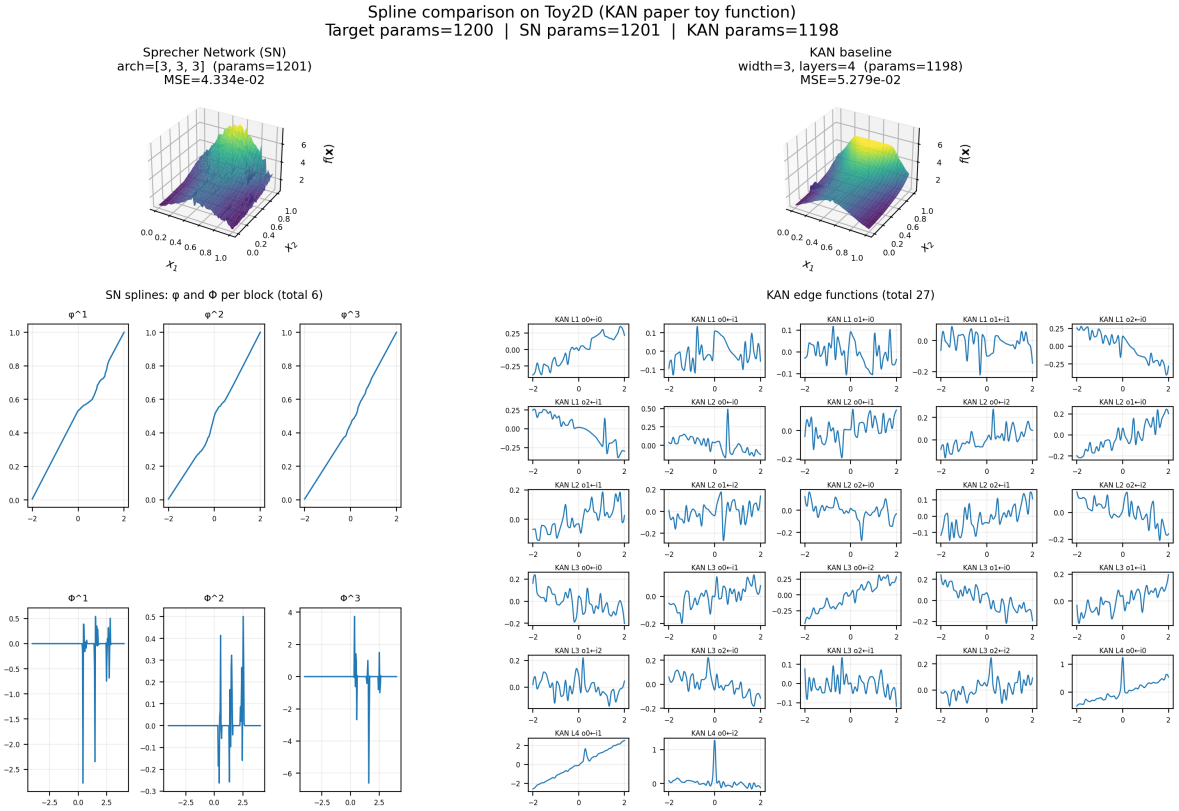


Figure 11: Side-by-side comparison of learned functions and internal univariate splines for an SN and a KAN on the Toy-2D benchmark ($f(x, y) = \exp(\sin(\pi x) + y^2)$) under parameter parity (≈ 1200 parameters). For this figure we use a spline-isolation setting (no residual connections, no normalization, no lateral mixing, fixed spline domains) to make the learned univariate functions directly comparable. In the representative run shown, the SN (left; arch $= [3, 3, 3]$, 1201 params) attains MSE 4.334×10^{-2} and the KAN (right; width=3, layers=4, 1198 params) attains MSE 5.279×10^{-2}. KANs are often motivated by interpretability because each edge carries an explicit learned univariate function [9], yet the learned edge functions (27 total here, one per edge) can still be jagged/noisy in practice. In contrast, the SN achieves a comparable fit using one shared pair $(\phi^{(\ell)}, \Phi^{(\ell)})$ per block (6 shared splines total across three blocks) with similarly oscillatory outer splines $\Phi^{(\ell)}$, consistent with the role of Φ in Sprecher-style superpositions.

scalar gate when $d_{\text{in}} = d_{\text{out}}$, and a learned projection matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ when $d_{\text{in}} \neq d_{\text{out}}$. KANs use shape-preserving cubic Hermite (PCHIP; deg = 3) edge splines. SNs use domain warmup early in training (300–400 epochs depending on the task; relative domain safety margin 0.01) and then freeze the learned domains; lateral mixing is disabled (i.e., $\tau^{(\ell)} = 0$) in these parity runs. We match parameter counts by choosing the KAN spline size K (the number of learnable knot values / spline coefficients per univariate edge spline) to be as close as possible (preferring $\leq$) to the SN parameter count, and we keep batch-normalization (BN) semantics aligned at test time with the training semantics. Seeds, datasets, and train/test splits are identical across models. Training is full-batch (no mini-batching): each epoch uses the entire training set. Primary metrics are test set RMSE or MSE (reported per task below); for multi-head monotone problems we also report a monotonicity-violation rate and a correlation-structure error.

Reporting conventions (and what we *do not* report). To avoid confounds from implementation-level vectorization (e.g., a highly vectorized KAN forward vs. a less vectorized SN implementation), we *do not* treat wall-clock time as a primary figure of merit here. We record and report it for completeness, but treat it as secondary and highly implementation-dependent; we therefore do not draw algorithmic conclusions from wall-clock time.

Structure of this subsection. We present each comparison task as its own *subsubsection*, with (i) a brief problem description, (ii) a one-paragraph explanation for why the task might reward SN inductive biases (if applicable), and (iii) a compact result table for that task.

8.3.1 Monotone single-index with quantiles (MQSI; 20D, m=9 heads)

Setup. We sample inputs $\mathbf{x} \sim \text{Unif}([0,1]^D)$ with $D = 20$ and define $m = 9$ heads at quantile levels $\tau_j = \text{linspace}(0.1, 0.9)$, with $z_j = \Phi_{\mathcal{N}}^{-1}(\tau_j) = \sqrt{2}\,\text{erf}^{-1}(2\tau_j - 1)$, where $\Phi_{\mathcal{N}}$ is the standard normal CDF (to avoid confusion with the outer spline Φ). For each coordinate, define a monotone sigmoid feature

$$h_i(x_i) = \text{sigmoid}\big(a_i(x_i - c_i)\big),$$

where $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$, $a_i \sim \text{Unif}[2, 8]$, and $c_i \sim \text{Unif}[0.2, 0.8]$. Draw raw positive weights $w_i^{\text{raw}}, v_i^{\text{raw}} \sim \text{Unif}[0, 1]$ and normalize $w_i = w_i^{\text{raw}} / \sum_k w_k^{\text{raw}}$ and $v_i = v_{\text{scale}}\, v_i^{\text{raw}} / \sum_k v_k^{\text{raw}}$ with $v_{\text{scale}} = 0.25$. Let $s_0 = 0.15$ and define

$$\sigma(\mathbf{x}) = s_0 + \max\Big(10^{-5}, \sum_{i=1}^{D} v_i\, h_i(x_i)\Big).$$

To introduce controlled head correlations while preserving monotonicity, we add positive pairwise terms: choose $n_{\text{pairs}} = \text{round}(0.15D)$ disjoint index pairs (i, j) (implemented by sampling $2n_{\text{pairs}}$ distinct indices without replacement and grouping them into pairs) and set

$$\mu(\mathbf{x}) = \sum_{i=1}^{D} w_i\, h_i(x_i) \;+\; 0.08 \sum_{(i,j)} h_i(x_i)\, h_j(x_j).$$

Finally, outputs are

$$y_j = \tanh\big(\beta(\mu(\mathbf{x}) + \sigma(\mathbf{x}) z_j)\big), \qquad \beta = 0.8,$$

with no additional observation noise. We train on a fixed i.i.d. training set of size $n_{\text{train}} = 1024$ and evaluate on a held-out i.i.d. test set of size $n_{\text{test}} = 50{,}000$.

Metrics. Primary: *mean RMSE across heads* (lower is better). Secondary: (i) correlation-structure error $\|\text{Corr}(\widehat{\mathbf{Y}}) - \text{Corr}(\mathbf{Y})\|_{\text{F}}$ where $\text{Corr}(\cdot)$ denotes the Pearson correlation matrix across head dimensions computed over the test set, and (ii) the monotonicity-violation rate

$$\text{viol} = \frac{1}{n_{\text{test}}} \sum_{n=1}^{n_{\text{test}}} \mathbf{1}[\exists\, j \in \{0, \ldots, m-2\} :\; \widehat{y}_{n,j+1} - \widehat{y}_{n,j} < 0]\,,$$

i.e. the fraction of test inputs for which the predicted head sequence is not monotone (lower is better).

Results (10 seeds). Best seed (minimum over seeds of mean RMSE): SN = 4.47×10^{-3}, KAN = 6.58×10^{-3} (ratio 1.47× in favor of SN). Averaged over the 10 seeds: SN = 8.31×10^{-3} vs. KAN = 1.280×10^{-2} (1.54×), with SN winning in 8/10 seeds. Correlation–structure fidelity strongly favored SN: mean Frobenius error ≈ 0.0124 (SN) vs. 0.124 (KAN). Monotonicity violations were negligible for both (SN had two seeds with $O(10^{-5})$ rates; KAN was 0). Wall clock (train, CPU; 4,000 epochs): KAN 393 s (avg; min 88.6 s, max 469.1 s) vs. SN 589 s (avg; min 231.5 s, max 732.4 s); SN timing includes warm-up + domain-freeze.

Discussion. This MQSI family is structurally aligned with an SN block $s_q = \sum_i \lambda_i\,\phi(x_i+\eta q)+\cdots$ followed by Φ, so the learned q-shift and monotone ϕ produce heads that are smooth shifts of a shared latent index; Φ handles squashing. KANs can approximate this but lack that inductive bias and rely on fixed grids. Under parameter parity and matched BN semantics, the bias yields lower head-averaged error and markedly better preservation of cross-head correlations, with no practical loss of monotonicity. KAN trained faster on CPU in this setting but at higher error.

8.3.2 Dense-head shifts (DenseHeadShift; 12D, m=64): SN vs. KAN

Task. Multi-head regression in which each output head is a smoothly shifted version of a shared latent index (we denote the *task*'s head-shift coefficient by a_{shift} to avoid confusion with the SN channel-spacing constant α):

$$y_j(x) \;=\; \tanh\Big(\beta\big[\mu(x)+\sigma(x)z_j + a_{\text{shift}}\,q_j + q_{\text{bias}}\,q_j\big]\Big),$$

with $x\in[0,1]^D$, $j=0,\dots,m-1$, $q_j\in[-1,1]$ on a uniform grid, and $z_j=\Phi_{\mathcal{N}}^{-1}(\tau_j)$. Here we set $D=12$ and $m=64$ and construct $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ as follows. For each coordinate,

$$h_i(x_i)=\text{sigmoid}\big(a_i(x_i-c_i)\big),$$

where $\text{sigmoid}(z)=\frac{1}{1+e^{-z}}$, $a_i\sim\text{Unif}[2,8]$, and $c_i\sim\text{Unif}[0.2,0.8]$. Draw raw positive weights $w_i^{\text{raw}},v_i^{\text{raw}}\sim\text{Unif}[0,1]$ and normalize $w_i=w_i^{\text{raw}}/\sum_k w_k^{\text{raw}}$ and $v_i=v_{\text{scale}}\,v_i^{\text{raw}}/\sum_k v_k^{\text{raw}}$ with $v_{\text{scale}}=0.25$. Let $\mathcal{P}$ be a set of $\max(1,\text{round}(0.15D))$ random distinct pairs (i,k) with $1\le i<k\le D$, and set $s_0=0.15$. Then

$$\sigma(\mathbf{x})=s_0+\max\Big(10^{-5},\sum_{i=1}^{D}v_ih_i(x_i)\Big),$$

$$\mu(\mathbf{x})=\sum_{i=1}^{D}w_ih_i(x_i)+0.08\sum_{(i,k)\in\mathcal{P}}h_i(x_i)h_k(x_k).$$

We use $\tau_j=\text{linspace}(0.1,0.9)$ and q_j as a uniform grid on $[-1,1]$. We use the same evaluation protocol throughout this subsection: all metrics are computed at test time; when BatchNorm is enabled, we use per-batch normalization statistics while freezing the running-statistics buffers.

Setup. D=12, m=64, a_{shift}=0.7, β=0.8, q_{bias}=0.6; train for 4000 epochs on CPU on a fixed i.i.d. training set of n_{train}=1024 samples; n_{test}=50,000. BN evaluation uses per-batch statistics while freezing the running-statistics buffers (i.e., no running-mean/variance updates during evaluation). Parameter parity was enforced by choosing the KAN basis size K to be as close as possible to (and not exceeding) the SN parameter count, yielding #params $\approx$ 3072 for both models; the resulting KAN basis size was K=4. The specific model configurations were:

- **SN:** Two hidden Sprecher blocks of width 32 (architecture $12\to32\to32\to64$), with 60 knots for each inner spline ϕ and each outer spline Φ; BatchNorm after each block except the first; linear residual connections; no lateral mixing; and dynamic spline-domain updates for the first 400 epochs, frozen thereafter.
- **KAN baseline:** A lightweight edge-wise KAN with hidden widths $(3,24,4)$ (architecture $12\to3\to24\to4\to64$), degree-3 splines with K=4 learnable spline coefficients per edge; BatchNorm after each layer except the first; linear residual connections; and linear extrapolation outside the knot interval.

Metric. Mean per-head RMSE over the m=64 heads on the held-out test set (we report both best-of-10 seeds and across-seed aggregates).

Results (10 seeds).

- *Best-of-10 (lower is better).* SN: $\mathbf{2.546 \times 10^{-3}}$ (seed 2) vs. KAN: 5.191×10^{-3}; ratio = 2.04× (KAN/SN).
- *Across seeds (means).* SN: 5.47×10^{-3}; KAN: 6.94×10^{-3} (i.e., KAN is ≈ 1.27× higher RMSE on average).
- *Monotonicity (fraction of test points violating head-wise order).* KAN: ≈ 0 across runs (all ten had 0.0). SN: varied by seed, with notable high-violation outliers (e.g., 0.276 and 0.476 on some seeds). This reflects that the SN here does not impose head-wise monotonicity explicitly, whereas the KAN runs empirically preserved the ordering under this configuration.
- *Wall-clock (train time; CPU).* Averaging the ten recorded wall-clock training times: KAN ≈ **476 s**, SN ≈ **1,433 s**. For reference, KAN runs were in the $\sim [338, 624]$ s range and SN runs in the $\sim$ [1,241, 1,607] s range.

Takeaways. On this dense-head shift task, whose structure matches SN's shared-ϕ/Φ plus head-wise shift inductive bias, SN attains markedly lower error (best-of-10 2.04× better, and ≈ 21% lower RMSE on average (equivalently, KAN is ≈ 27% higher)) under strict parameter parity (∼3k parameters each, KAN K=4). KAN trains ≈ 3× faster on CPU but shows consistently higher RMSE in this setting. The occasional SN monotonicity violations suggest that adding an explicit head-order regularizer or constraint would likely remove those outliers without changing the overall accuracy picture.

Summary across baseline tasks. Table 6 collates the head-to-heads above. In both cases we observe SNs outperform KANs at matched parameter budgets and training epochs. The MQSI task, which directly matches the SN compositional/shift bias, shows the clearest gap in correlation-structure fidelity; the DenseHeadShift task is closer, with SNs typically ahead by a modest margin.

Task	Metric	SN (mean)	KAN (mean)	Ratio (KAN/SN)	Time (KAN/SN) (s)
MQSI (20D, 9 heads)	Mean RMSE ↓ (10 seeds)	8.31×10^{-3}	1.280×10^{-2}	1.54×	393 / 589 (avg)
DenseHeadShift (12D, 64 heads)	Mean RMSE ↓ (10 seeds)	5.47×10^{-3}	6.94×10^{-3}	1.27×	476 / 1,433 (avg)

Table 6: Summary across baseline tasks at matched parameter counts and 4000 epochs; wall-clock time shown for completeness. Times are per-task averages across seeds.

Reproducibility checklist (concise).

- **Epochs/optimizer:** 4000 epochs for both models. We use Adam with gradient-norm clipping 1.0. For SNs we use learning rate 3×10^{-4} for the main parameters, 5×10^{-4} for lateral-mixing parameters (when enabled), and 10^{-3} for (c_c, c_r) codomain parameters (when enabled); weight decay 10^{-7}. For KANs we use learning rate 10^{-3} and weight decay 10^{-6} (no other tuning beyond the fairness knobs).
- **Residuals/BN:** Linear residuals in both; BatchNorm layers placed "after" each layer/block, and evaluated using per-batch statistics without updating running-mean/variance buffers.
- **Parity:** Parameter matching via a closed-loop count of KAN parameters as a function of K (prefer $\leq$).
- **Data:** Identical $(x_{\text{train}}, y_{\text{train}})$ and test sets across models per task; fixed seeds.

Takeaways. These controlled baselines demonstrate that there exist natural tasks, particularly those exhibiting shared-index families or smooth saturations, on which SNs achieve lower test error than parameter-matched KANs under equal training budgets. The observed margins range from modest to substantial (especially on structured multi-head problems); we expect this picture to sharpen as we expand the suite with ∼8 additional tasks in the same format.

9 Limitations and future work

The primary limitation of this work is the gap between our theoretically-grounded single-layer model and our empirically-driven deep architecture. While single-layer SNs inherit universal approximation properties directly from Sprecher's theorem, the universality and approximation bounds of deep, compositional SNs remain open theoretical questions (Conjectures 1 and 2). The role of lateral mixing in these theoretical properties is particularly unclear, while it empirically improves performance, its theoretical justification within the Sprecher framework remains elusive.

The Sprecher block design imposes strong constraints: forcing all feature interactions through two shared splines and using weight vectors rather than matrices heavily restricts expressive power compared to standard architectures. While lateral mixing partially alleviates this constraint by enabling limited cross-dimensional communication, it represents an ad-hoc enhancement rather than a principled extension of Sprecher's theory. This represents a fundamental trade-off between parameter efficiency and flexibility that may limit performance on functions not aligned with this compositional structure.

Current implementations may require more training iterations than MLPs for certain tasks, though the per-iteration computational cost is typically lower due to fewer parameters. When memory-efficient sequential computation is employed to enable training of wider architectures, wall-clock training time increases, representing a fundamental trade-off between memory usage and computational efficiency. The sequential computation mode proves most beneficial for architectures with individual layers exceeding 128-256 units in width, while offering minimal advantage for very deep networks with modest layer widths where the memory bottleneck lies in storing activations across many layers rather than within-block computations. The development of adaptive knot placement strategies that concentrate resolution where data lives while maintaining fixed parameter counts could improve both efficiency and interpretability.

The lateral mixing mechanism, while empirically beneficial, lacks theoretical justification within the Sprecher framework. Understanding whether this enhancement can be connected to the underlying mathematical structure or represents a purely empirical improvement remains an open question. Future work could explore adaptive mixing topologies beyond cyclic patterns, potentially learning the neighborhood structure $\mathcal{N}(q)$ itself, or investigating connections to graph neural networks where the mixing pattern could be viewed as a learnable graph structure over output dimensions.

Beyond addressing these immediate challenges, SNs open several intriguing research directions. The weight-sharing structure combined with lateral mixing raises fundamental theoretical questions about potential equivariance properties. Just as CNNs exhibit translation equivariance through spatial weight sharing, SNs' sharing across output dimensions with structured mixing may satisfy a related form of equivariance, possibly connected to permutations of output indices or transformations of the function domain. Understanding such properties could provide deeper insight into when and why the architecture succeeds.

The architecture's properties suggest unique opportunities in scientific machine learning where both interpretability and parameter efficiency are valued. One particularly compelling possibility is dimensionality discovery: the architecture's sensitive dependence on input dimension could enable inference of the intrinsic dimensionality of data-generating processes. By training SNs with varying d_{in} and using model selection criteria that balance fit quality against complexity, researchers might determine the true number of relevant variables in systems where this is unknown, a valuable capability in many scientific domains. The lateral mixing patterns learned by the network could additionally reveal structural relationships between output dimensions, potentially uncovering hidden symmetries or conservation laws in physical systems. Furthermore, the explicit structure of learned splines could enable integration with symbolic regression tools to extract closed-form expressions for the learned $\phi^{(\ell)}$ and $\Phi^{(\ell)}$ functions, potentially revealing underlying mathematical relationships in data.

Architectural enhancements also merit exploration. Unlike KANs where edge pruning can significantly reduce parameters, SNs' vector structure suggests different optimization strategies. Automatic pruning of entire Sprecher blocks based on their contribution to network output could yield more compact architectures. The learned lateral mixing weights could guide this pruning: blocks with near-zero mixing weights might be candidates for removal. For very high-dimensional problems where even $O(N)$ scaling becomes prohibitive, hybrid approaches using low-rank approximations for $\lambda^{(\ell)}$ or replacing splines with small neural sub-networks in certain blocks could maintain efficiency while improving expressivity. Alternatively, hierarchical lateral mixing patterns (e.g., mixing within local groups before global mixing) could provide a middle ground

between full connectivity and the current nearest-neighbor approach.

The interaction between lateral mixing and other architectural components deserves systematic investigation. How do different mixing topologies (cyclic, bidirectional, or more complex patterns) interact with network depth? Can theoretical guarantees be established for specific mixing patterns? Is there an optimal ratio between the lateral scale τ and the main transformation strength? These questions highlight how theorem-inspired architectures enhanced with empirical innovations can open new research avenues beyond simply providing alternative implementations of existing methods.

10 Conclusion

We have introduced Sprecher Networks (SNs), a trainable architecture built by re-imagining the components of David Sprecher's 1965 constructive proof as the building blocks for a modern deep learning model. By composing functional blocks that utilize shared monotonic and general splines, learnable mixing weights, explicit shifts, and optionally lateral mixing connections, SNs offer a distinct approach to function approximation that differs fundamentally from MLPs, KANs, and other existing architectures.

The key contributions are demonstrating that Sprecher's shallow, highly-structured formula can be extended into an effective deep architecture with remarkable parameter efficiency: $O(LN + LG)$ compared to MLPs' $O(LN^2)$ or KANs' $O(LN^2G)$, and enabling $O(LN)$ *forward working memory* (peak intermediate storage) through sequential computation strategies compared to MLPs' $O(LN^2)$. This dual efficiency in both parameters and memory comes from adhering to Sprecher's use of weight vectors rather than matrices, representing a strong architectural constraint that may serve as either beneficial inductive bias or limitation depending on the problem domain. The addition of lateral mixing connections provides a parameter-efficient mechanism for intra-block communication, partially addressing the limitations of the constrained weight structure while maintaining the overall efficiency of the architecture.

Our initial demonstrations show SNs can successfully learn diverse functions and achieve competitive performance across synthetic regression benchmarks and PDE-constrained learning problems, with the added benefit of interpretable spline visualizations. The lateral mixing mechanism proves particularly valuable for vector-valued outputs and deeper networks, often improving optimization and convergence behavior even though the learned Φ splines can remain highly oscillatory. The sequential computation mode enables training of architectures with wide layers that would otherwise exhaust available memory, making SNs particularly suitable for exploring high-capacity models under memory constraints. However, the need for more training iterations in some cases and theoretical gaps regarding deep network universality remain open questions. The theoretical status of lateral mixing — whether it represents a principled extension of Sprecher's construction or merely an empirical enhancement — requires further investigation.

Whether SNs prove broadly useful or remain a fascinating special case, they demonstrate the value of mining classical mathematical results for architectural inspiration in modern deep learning. The successful integration of lateral mixing shows how theorem-inspired designs can be enhanced with empirical innovations while maintaining their core theoretical structure. The ability to evaluate SN layers with $O(LN)$ *forward working memory* (peak intermediate storage) distinguishes SNs from other major architectures, suggesting applications in memory-constrained settings and when exploring extremely wide output layers. This synthesis of rigorous mathematical foundations with practical deep learning techniques points toward a promising direction for developing novel architectures that balance theoretical elegance with empirical effectiveness.

References

[1] Arnold, V. I. (1963). "On functions of three variables," *Doklady Akademii Nauk SSSR*, **48**.

[2] Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, **2**(4), 303–314.

[3] Goyal, M., Goyal, R., Lall, B. (2019). "Learning activation functions: A new paradigm for understanding neural networks," arXiv preprint arXiv:1906.09529.

[4] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[5] Hornik, K., Stinchcombe, M., White, H. (1989). "Multilayer feedforward networks are universal approximators," *Neural networks*, **2**(5), 359–366.

[6] Kingma, D. P., Ba, J. (2014). "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980.

[7] Kolmogorov, A. N. (1957). "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Doklady Akademii Nauk SSSR*, **114**(5), 953–956.

[8] Köppen, M. (2002). "On the training of a Kolmogorov Network," in *Artificial Neural Networks—ICANN 2002: International Conference, Madrid, Spain, August 28–30, 2002 Proceedings 12*, pp. 474–479. Springer.

[9] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., Tegmark, M. (2025). "KAN: Kolmogorov-Arnold Networks," *ICLR 2025 (to appear)*. arXiv preprint arXiv:2404.19756.

[10] Sprecher, D. A. (1965). "On the Structure of Continuous Functions of Several Variables," *Transactions of the American Mathematical Society*, **115**, 340–355.

[11] Zhang, S., Shen, Z., Yang, H. (2022). "Neural network architecture beyond width and depth," *Advances in Neural Information Processing Systems*, **35**, 5669–5681.