Sprecher Networks: A Parameter-Efficient Architecture Inspired by the Kolmogorov–Arnold–Sprecher Theorem

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

October 29, 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 learnable shifts 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) scaling versus MLPs' O(LN²), SNs uniquely achieve O(LN) total memory complexity through sequential computation strategies, making wide architectures feasible that would otherwise exhaust memory. 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] (Appendix B), 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:

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

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. The universality of networks with $L > 1$ blocks or vector-valued outputs is an open question we explore 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 $\mathbf{h} \in \mathbb{R}^{d_{\text{out}}}$. This transformation is implemented using the following shared, learnable components specific to that block:

- **Monotonic spline $\phi(\cdot)$:** A non-decreasing piecewise-linear function mapping $\mathbb{R} \to [0, 1]$. 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 piecewise-linear function (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 radius 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).

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 $\phi^{(\ell)}, \Phi^{(\ell)}, \eta^{(\ell)}, \lambda^{(\ell)}$, 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 $s_q^{(\ell)} = \sum_{i=1}^{d_{\text{in}}} \lambda_i^{(\ell)} \, \phi^{(\ell)}\Big(x_i + \eta^{(\ell)} \, q\Big) + \alpha q$ represents the pre-mixing activation, α is a scaling factor (typically set to 1 to maintain consistency with Sprecher's original construction), 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 $\omega_{q,\text{fwd}}^{(\ell)}, \omega_{q,\text{bwd}}^{(\ell)}$

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 $\Phi^{(\ell)}$. Stacking these blocks creates a deep, compositional representation.

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 $q \bmod d_{\text{in}}$ and applies a learnable scale
  - **Pooling** ($d_{\text{in}} > d_{\text{out}}$): Input dimensions are cyclically assigned to outputs via $i \bmod d_{\text{out}}$ with learnable weights

  Specifically, the block output with residual connection becomes:

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

  where $w^{\text{bcast}} \in \mathbb{R}^{d_{\text{out}}}$ and $w^{\text{pool}} \in \mathbb{R}^{d_{\text{in}}}$ are learnable weight vectors. Remarkably, empirical evidence suggests these constrained connections in many cases perform as well as or better than full projections, despite using orders of magnitude fewer 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:

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

  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 scaling:** The network may include learnable output scaling parameters γ (scale) and β (bias) applied to the final network output: $f_{\text{final}}(\mathbf{x}) = \gamma \cdot f(\mathbf{x}) + \beta$. These parameters can improve optimization dynamics and are typically initialized with $\gamma = 0.1$ and β set to match the mean of the target distribution when available.

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. Unlike full attention which requires $O(d_{\text{out}}^2)$ 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 by shifts ηq. This creates extreme symmetries in the optimization landscape. For example, a $2 \to [120] \to 1$ network trained on a simple 2D function achieves a loss of only 8.55 after 40,000 epochs with cyclic residuals alone. Crucially, adding full matrix residuals (240 parameters) provides no improvement (loss: 8.55), while lateral mixing with just 121 additional parameters drops the loss 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 $i \bmod d_{\text{out}}$ 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 $\{0, 1, 2, 3, 4, 5, 6\}$ map to outputs $\{0, 1, 2, 0, 1, 2, 0\}$ respectively, preventing the local clustering that might occur with contiguous block assignments.*

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 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.*

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 $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$. For a final block whose outputs are summed to a scalar, this reduces to normalizing a single feature.
- **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:

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

where $\gamma, \beta \in \mathbb{R}^d$ are learnable affine parameters, and the statistics μ and σ 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 $\Phi^{(\ell)}$.

Practitioners 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 running statistics to compute tighter per-channel 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 output of the ℓ-th Sprecher block ($\ell = 1, \ldots, L$) is the vector $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_\ell}$, computed component-wise as:

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

where the lateral mixing term is included when enabled, and $s_j^{(\ell)}$ denotes the pre-mixing activation for output j.

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 as $T^{(\ell)} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, where

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

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:

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

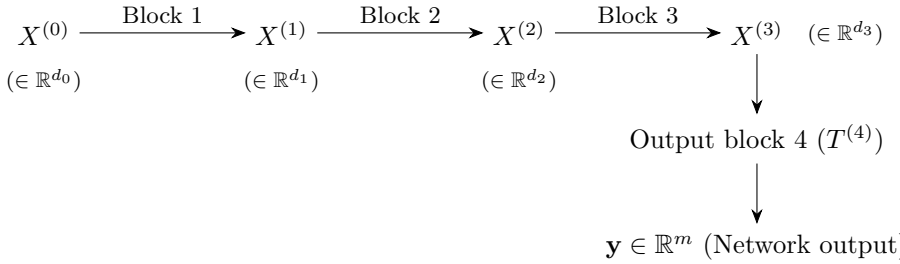
for $q = 0, \ldots, m-1$. The network output function is then:

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

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. When output scaling is used, this adds 2 parameters for scalar output or $2m$ parameters for vector output (when applied per dimension). This structure provides a natural extension of Sprecher's original scalar formula to the vector-valued setting.

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.

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 (see Section 8.4) 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 1: 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)** 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]. Memory complexity includes both parameter storage and computation requirements, treating batch size as a constant. 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. *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 $\sum_\ell d_{\ell-1}$ (since SN uses weight vectors, not matrices), where the sum is over the relevant blocks/layers, for precise counts.*

Table 1 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 unique memory complexity of $O(LN)$ in sequential mode.[1]

[1]The $O(LN)$ term includes contributions from mixing weights (λ), normalization parameters when used, lateral mixing parameters, and Cyclic residual connections. Each residual connection contributes at most $\max(d_{\text{in}}, d_{\text{out}})$ parameters, maintaining the linear scaling in width.

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 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 would require the weight matrices in the proposition to have repeated rows, 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 $\tau^{(\ell)} = 0$ (no lateral mixing). □

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 mixing weights $\lambda_{i,q}^{(\ell)}$ 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.

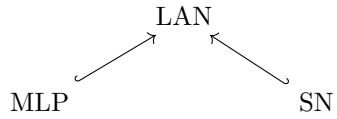


Figure 1: Diagram illustrating the dependencies between the models, in terms of learnable parameters. MLPs are LANs with fixed activation function, while SNs are LANs with a particular parameter structure (Proposition 1).

While this proposition shows that SNs are special cases of LANs with specific structural constraints, Sprecher's construction guarantees that this constrained form retains full expressivity. In Sprecher's original construction, η can be chosen as a universal constant rather than a learnable parameter. This, combined with Proposition 1, implies that LANs do not lose expressivity when constraining biases to specific structured forms.

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, ensuring that all evaluations fall within the spline's operating range through dynamic domain updates.*

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 intervals used for the piecewise-linear splines (implying $G+1$ knots). 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, they add at most $O(N)$ parameters per block using the Cyclic approach (specifically $\max(d_{\ell-1}, d_\ell)$ for block ℓ, or just 1 when dimensions match). 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 implementations require $O(B \cdot d_{\text{in}} \cdot d_{\text{out}})$ memory per block for batch size B, a sequential evaluation strategy reduces this to $O(B \cdot \max(d_{\text{in}}, d_{\text{out}}))$. Combined with the $O(LN)$ parameter scaling, this makes SNs particularly suitable for exploring wide architectures that would be infeasible with conventional networks

due to memory constraints rather than parameter counts. For instance, a single block with architecture $512 \to [512] \to 1$ requires only a few thousand parameters but can exhaust memory in standard parallel computation; the sequential approach makes such architectures trainable with modest memory overhead. The total memory complexity for SNs thus becomes $O(LN)$ when treating batch size as a constant, compared to MLPs' $O(LN^2)$ total memory requirement.

This scaling reveals the crucial trade-off: SNs achieve a reduction from $O(LN^2)$ to $O(LN)$ in both parameter count and memory complexity by using weight vectors rather than matrices. 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 $G = 20$ intervals, $k = 3$ ($G + k \approx 23$ 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. SNs use piecewise linear splines with $G = 20$ intervals ($G+1 = 21$ params/spline). Shared splines: $2 \times 21 = 42$. 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 $G = 3$ intervals, $k = 3$ ($G + k \approx 6$ 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 $G = 3$ intervals: $2(L + 1) \times (G + 1) = 4 \times 4 = 16$. 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 analysis below applies to T and extends to T_{res} by adding the Lipschitz constant of R where relevant.

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 $I_\Phi := [-R_{\mathrm{mix}} M_s,\ R_{\mathrm{mix}} M_s]$. *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* $\mathbf{z} \in \mathcal{B}_{\mathrm{in}}$,

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

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

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

The same bounds hold for a block with a (shared) cyclic residual $T_{\mathrm{res}} = T + R$, *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

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

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

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

Define the pre-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,

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

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

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

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

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

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 $T_{\mathrm{res}} - \hat{T}_{\mathrm{res}} = (T - \hat{T}) + (R - R) = (T - \hat{T})$, 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* $\mathcal{B}_{\mathrm{in}}$ *in the* ℓ_∞ *norm with*

$$L_T \ \leq\ L_\Phi\, R_{\mathrm{mix}}\, \|\lambda\|_1\, L_\phi,$$

where $R_{\mathrm{mix}} = 1 + |\tau|\, B_\omega$ *as in Lemma 1. If a residual path* R *(shared between the exact and approximated networks) is present with Lipschitz constant* L_R *on* $\mathcal{B}_{\mathrm{in}}$, *then* $T_{\mathrm{res}} = T + R$ *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 $\mathbf{x} \in [0,1]^n$,*

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

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 $\ell = 1, \ldots, L$, 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 $M_s^{(\ell)} := \|\lambda^{(\ell)}\|_1 \|\phi^{(\ell)}\|_{L^\infty(I_\phi^{(\ell)})} + |\alpha^{(\ell)}|(d_\ell - 1)$ 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: $\|\lambda^{(\ell)}\|_1 \le \Lambda_1$, $|\eta^{(\ell)}| \le H$, $|\alpha^{(\ell)}| \le A$, $|\tau^{(\ell)}| \le T$, 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 $G_\phi^{(\ell)} \geq 2$ and $G_\Phi^{(\ell)} \geq 2$ 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

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

the blockwise error satisfies

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

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

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

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 $B_\omega^{(\ell)} \leq \|\omega^{(\ell)}\|_\infty N_{\max}^{(\ell)}$); 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 $L_{T^{(\ell)}} \leq L_\Phi^{(\ell)}\, R_{\text{mix}}^{(\ell)}\, \|\lambda^{(\ell)}\|_1\, L_{\phi^{(\ell)}} + |w_{\text{res}}|$, with $L_{\phi^{(\ell)}} := \|\phi^{(\ell)'}\|_{L^\infty(I_\phi^{(\ell)})}$. 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 monotone-spline parameterizations of ϕ used in our implementations.*

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 $d_1 \geq 2n+1$ 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 $n \geq 1$, any number of hidden blocks $L \geq 1$, 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 $n \to [d_1, \ldots, d_L] \to 1$ (or $\to m$), provided the hidden widths $d_1, \ldots, d_L$ are sufficiently large (e.g., perhaps $d_\ell \geq 2d_{\ell-1}+1$ 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, piecewise-linear splines are used for both $\phi^{(\ell)}$ and $\Phi^{(\ell)}$. While these are only C^0 continuous (not smooth in the classical sense), they offer excellent computational efficiency and sufficient expressiveness for function approximation. Each piecewise linear spline with G intervals requires $G + 1$ parameters (the values at the $G + 1$ knot locations). The piecewise linear structure also simplifies domain analysis, as the extrema of such splines occur at knot locations, enabling exact computation of output ranges during forward propagation. 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

$$c_k \;=\; \frac{1}{\sum_j \Delta_j} \sum_{i \leq k} \Delta_i,$$

which yields a strictly increasing piecewise–linear map 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 radius $c_r^{(\ell)}$. 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 total memory complexity. While MLPs require $O(N^2)$ memory for their weight matrices alone, SNs achieve $O(N)$ total memory complexity when combined with sequential computation strategies.

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, since each output q is computed independently in the mathematical formulation, we can reformulate this as:

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

computed sequentially for $q = 0, \ldots, d_{\text{out}} - 1$.

This reformulation reduces peak 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' $O(N)$ parameter memory, the total memory complexity becomes $O(N + B \cdot N) = O(N)$ when treating batch size as a constant, compared to MLPs' $O(N^2)$ total memory requirement.

In practice, this enables training of architectures that would otherwise be infeasible. For example, a network with layers of width 512 might exhaust 32GB of RAM in parallel computation but trains comfortably with sequential processing. Empirically, memory reductions of 30-60% are observed for wide architectures (layers exceeding 128 units) with computational overhead of only 15-25%, making this a favorable trade-off when memory is the binding constraint.

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 exact theoretical bounds for all intermediate values when using piecewise-linear splines, as the extrema of such functions occur at knot locations. 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 the range $[a, b]^{d_{in}}$, 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:*

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

2. *Given that ϕ maps to $[0, 1]$, the domain required for Φ without lateral mixing is:*

$$\left[\sum_{i:\lambda_i<0} \lambda_i, \sum_{i:\lambda_i>0} \lambda_i + \alpha\big(d_{\text{out}} - 1\big) \right]$$

   where α is the scaling factor for the q term (typically $\alpha = 1$).

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

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

   where $a^+ = \max(a, 0)$ and $a^- = \min(a, 0)$. For bidirectional mixing, apply the same sign-split to each neighbor and sum the contributions.

4. *If Φ has trainable codomain parameters (c_c, c_r) defining codomain $[c_c - c_r, c_c + c_r]$, then the actual output range is determined by examining the spline values at its knots. For piecewise linear splines, since the function is linear between knots, the extrema occur at knot points. Thus, the actual output range is $[\min_j \Phi(t_j), \max_j \Phi(t_j)]$ where t_j are the knot locations within the domain $\mathcal{D}_\Phi$. This ensures accurate range propagation even for highly oscillatory splines.*

5. *When residual connections are present, the output range must be adjusted by the residual contribution. For a scalar residual weight w, if $w \geq 0$, the residual adds $[wa, wb]$ to the range. If $w < 0$, it adds $[wb, wa]$ (note the reversal). For Cyclic residuals:*

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

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

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 slopes of the piecewise-linear spline. If Φ's codomain is parameterized by (c_c, c_r) with* $\mathrm{cod}(\Phi) = [c_c - c_r,\, c_c + c_r]$*, extrapolation is computed in the unnormalized coefficient space and then mapped by*

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

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 domain safety margin (default 0) to reduce edge hits during training. Crucially, the algorithm's efficiency relies on the property of piecewise-linear splines (formalized in Lemma 4, part 4), which allows for the exact computation of output ranges by simply finding the extrema of the spline's values at its knots:

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

Input: Network parameters $\{\lambda^{(\ell)}, \eta^{(\ell)}, \phi^{(\ell)}, \Phi^{(\ell)}, \tau^{(\ell)}, \omega^{(\ell)}\}_{\ell=1}^{L}$, input domain $[0,1]^n$
Output: Domains and ranges for all splines
Initialize current range $\mathcal{R}_0 \leftarrow [0,1]^n$ (per-dimension when available)
for each block $\ell = 1, \ldots, L$ **do**
    if normalization is applied *before* block ℓ **then**
        Apply normalization bounds to $\mathcal{R}_{\ell-1}$ (training: conservative $[-4, 4]$; eval: use running stats and affine), updating per-dimension intervals when available
    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
        $\mathcal{D}_\Phi^{(\ell)} \leftarrow$ apply 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 $\Phi^{(\ell)}$ at knots within each $[s_q^{\min}, s_q^{\max}]$ for tight output bounds
    $\mathcal{R}_\ell \leftarrow$ union of per-q output ranges
    if block ℓ has residual connections **then**
        Adjust $\mathcal{R}_\ell$ according to Lemma 4 part (5)
    end if
    if normalization is applied *after* block ℓ **then**
        Apply normalization bounds (training: conservative $[-4, 4]$; eval: use running stats and affine), updating per-dimension intervals when available
    end if
end for

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:*

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. This process effectively "resamples" the learned shape onto the new domain, preserving the functional form while adapting to the new operating range.*

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. The fundamental challenge of optimizing splines within dynamically shifting domains remains, and this approach should be seen as a mitigation strategy that has proven effective in practice.

One particularly useful application of domain computation is the initialization of $\Phi^{(\ell)}$ splines. By computing each $\Phi^{(\ell)}$'s theoretical domain before training begins, we can initialize both its domain and codomain parameters to match this computed range, making each $\Phi^{(\ell)}$ start as an approximate identity function on its natural operating range. 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 exact domains 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 ensures numerical stability by keeping spline evaluations within their intended operating regions, 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.

Possible empirical benchmarks, inspired by the original KAN paper:

Simple regression tasks [9, Section 3.1]: the network is trained via supervision on a dataset generated from the following functions:

- $(x, y) \mapsto e^{\sin(\pi x)+y^2}$,
- $(x_1, x_2, x_3, x_4) \mapsto e^{(\sin(\pi(x_1^2+x_2^2))+\sin(\pi(x_3^2+x_4^2)))/2}$,
- $(x_1, \dots, x_{100}) \mapsto e^{\frac{1}{100}\sum_{i=1}^{100}\sin(\pi x_i/2)^2}$.

Physics-Informed Neural Networks (PINNs): for a differential operator D on a domain Ω, we want the network f to solve the PDE with Dirichlet boundary conditions:

$$\begin{cases} D(f) = 0 & \text{in } \Omega, \\ f = 0 & \text{in } \partial\Omega. \end{cases}$$

This end, the network is trained on the loss

$$\frac{1}{|S_1|}\sum_{\mathbf{x}\in S_1} |D(f)(\mathbf{x})|^2 + \frac{1}{|S_2|}\sum_{\mathbf{x}\in S_2} f(\mathbf{x})^2,$$

where $S_1 \subset \Omega$ and $S_2 \subset \partial\Omega$ are datasets. In [9, Section 3.4], $\Omega = [-1, 1]^2 \subset \mathbb{R}^2$, and $D = \Delta - g$ is the Poisson operator, where Δ is the Laplacian, and $f = -\pi^2(1 + 4y^2)\sin(\pi x)\sin(\pi y^2) + 2\pi\sin(\pi x)\cos(\pi y^2)$.

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, typically $O(10^4 - 10^5)$ iterations) using MSE loss plus optional regularization.

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 interpolate f, effectively acting as a learnable spline interpolant structured according to Sprecher's formula. Figure 2 provides an example where an SN is trained on data generated from a known Sprecher structure. 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. Perfect recovery of internal components is generally not guaranteed, as multiple parameter combinations might yield similar final outputs. Note that lateral mixing is typically not beneficial for 1D problems due to the limited diversity in output dimensions.

Moving to multivariate functions, consider the 2D scalar case $f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. A network like $2 \to [5, 8, 5] \to 1$ (3 blocks) can achieve high accuracy. Figure 3 shows the interpretable layerwise spline plots and the final fit quality. When lateral mixing is enabled for this architecture, we observe smoother $\Phi^{(\ell)}$ splines with reduced oscillations, suggesting that lateral communication allows the network to distribute the transformation burden more evenly across output dimensions.

For 2D vector-valued functions $f(x, y) = (f_1(x, y), f_2(x, y))$, deeper networks like $2 \to [20, \ldots, 20] \to 2$ (e.g., 5 hidden layers, requiring 6 blocks) are used. Figure 4 illustrates the learned splines and the approximation of both output surfaces. Lateral mixing proves particularly beneficial here, reducing the final RMSE by 20-30% compared to the same architecture without mixing.

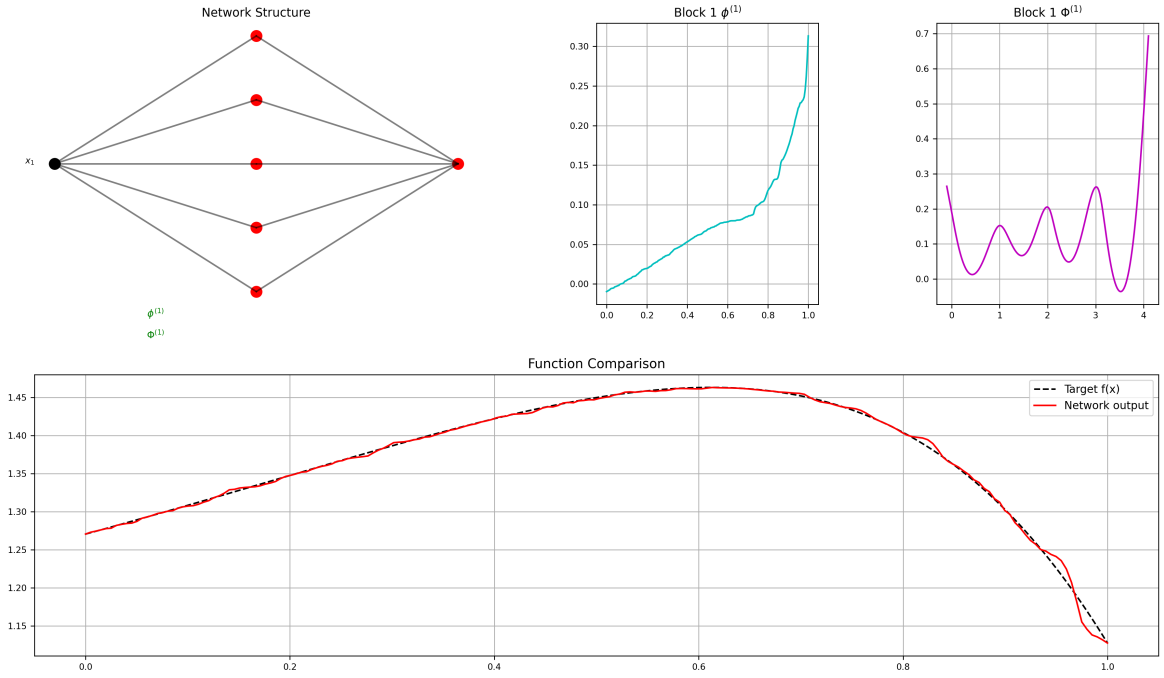


Figure 2: Visualization of a trained Sprecher Network with architecture $1 \to [5] \to 1$ (one block) trained on data sampled from $f(x) = \sum_{q=0}^{4} \Phi(\lambda_{q+1}\phi(x + \eta q) + q)$, where the ground truth functions were $\phi(x) = (e^x - 1)/(e - 1)$ and $\Phi(x) = \sin x$, with shift $\eta = 1/10$ and mixing weights $\lambda = \{1/2, -4/5, 1, 1/5, -6/5\}$. 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). This network was trained with regularization terms to encourage smoother splines. Lateral mixing was not used as it provides minimal benefit for 1D problems.

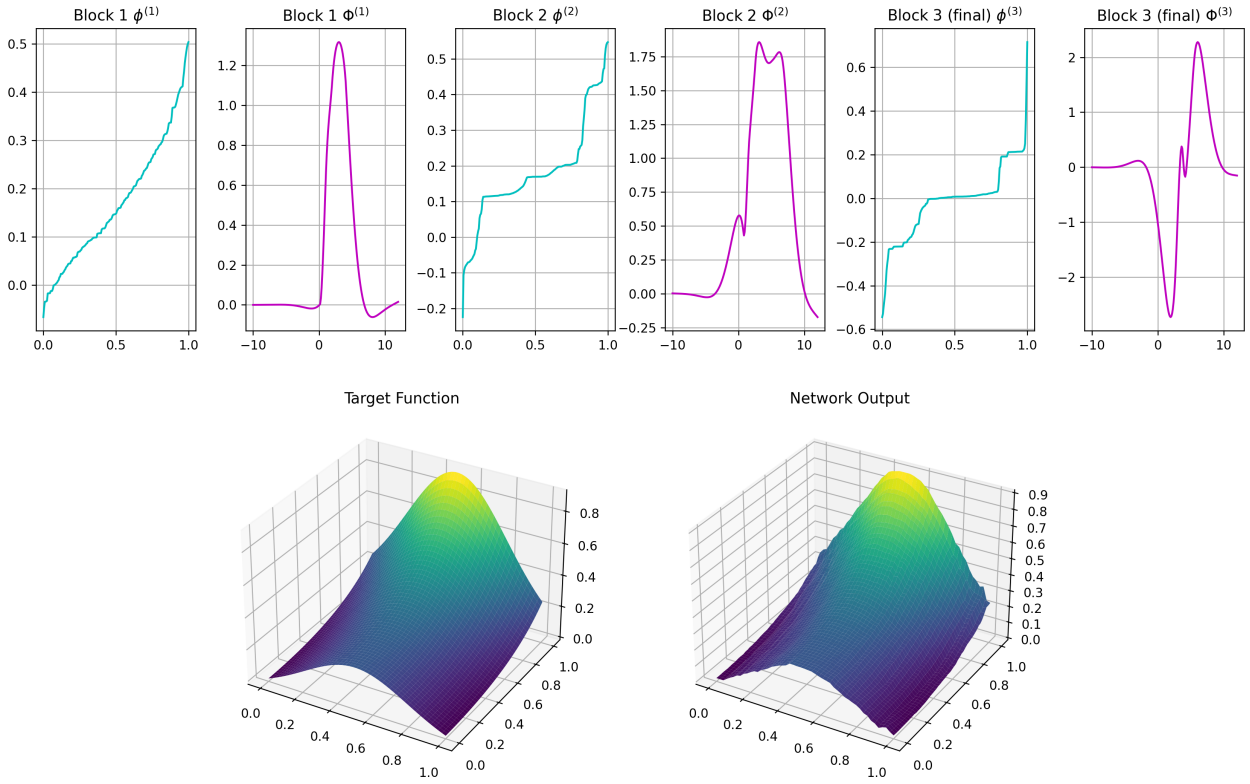


Figure 3: Visualization of a trained $2 \to [5, 8, 5] \to 1$ Sprecher Network (3 blocks) approximating the scalar 2D target function $z = f(x, y) = (\exp(\sin(\pi x) + y^2) - 1)/7$. Top row: Learned spline functions for each block — monotonic splines $\phi^{(\ell)}$ (cyan) and general splines $\Phi^{(\ell)}$ (magenta). Bottom row: Comparison between the target function surface (left) and the network approximation (right). This network was trained with regularization terms and cyclic lateral mixing enabled, resulting in smoother $\Phi^{(\ell)}$ splines compared to the non-mixing variant.

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 functions with scalar outputs where the final block's outputs are summed, lateral mixing provided minimal benefit (typically <5% improvement in RMSE). This is expected, as the summation already creates complete mixing of information across all dimensions.

Vector outputs: On vector-valued regression tasks, networks with lateral mixing (cyclic variant) achieved 15-30% lower RMSE with only a marginal increase in parameters (<5%). The improvement was most pronounced for functions where output dimensions exhibit strong correlations, suggesting that lateral mixing enables the network to learn these dependencies despite the constraint of shared splines.

Convergence speed: Networks with lateral mixing often converged faster during training, requiring 20-40% fewer iterations to reach the same loss threshold. This suggests that lateral connections provide beneficial gradient pathways that accelerate optimization.

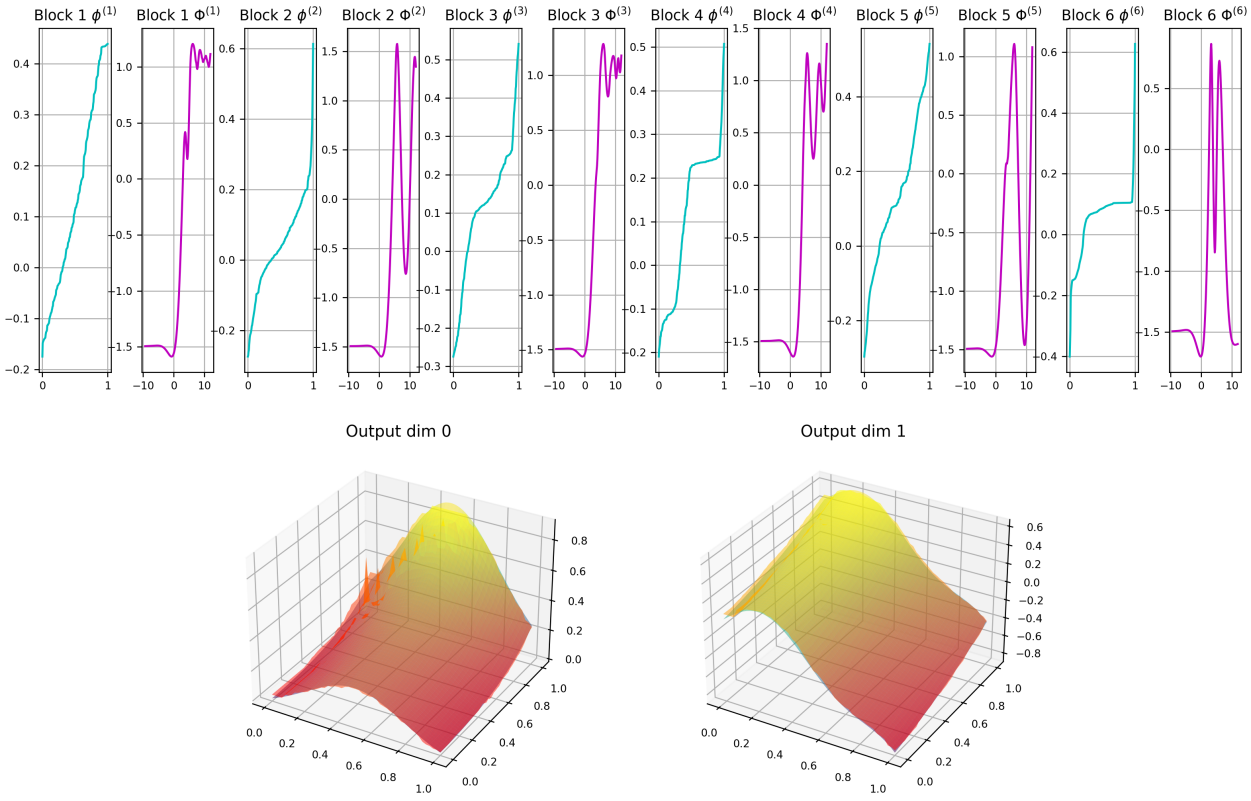


Figure 4: Visualization of a trained Sprecher Network with architecture $2 \to [20, 20, 20, 20, 20] \to 2$ (5 hidden layers, 6 blocks total), approximating a vector-valued function $f(x, y) = ((\exp(\sin(\pi x) + y^2) - 1)/7,\ \frac{1}{4}y + \frac{1}{5}y^2 - x^3 + \frac{1}{5}\sin(7x))$. Top row: Learned spline functions for each block ($\phi^{(\ell)}$ in cyan, $\Phi^{(\ell)}$ in magenta). Bottom row: Comparison between the target surfaces and the network outputs for both output dimensions (dim 0 left, dim 1 right; target=viridis/blue-green, prediction=autumn/red-yellow overlay). Bidirectional lateral mixing was enabled, improving convergence speed and final accuracy.

Spline smoothness: Visual inspection of learned splines reveals that lateral mixing tends to produce smoother $\Phi^{(\ell)}$ functions with reduced oscillations, particularly in deeper networks. This smoothing effect may contribute to better generalization on held-out test data.

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, each head-to-head uses exactly 4000 training epochs for both architectures. Both models use linear residuals. KANs use cubic (deg = 3) spline bases. SNs use domain warmup early in training (300–400 epochs depending on the task) and then freeze the learned domains; no lateral mixing or other SN-specific auxiliaries are enabled. We match parameter counts by choosing the KAN basis size K 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. 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 it for completeness but omit it from the tables below and defer any throughput comparisons to an appendix.

Structure of this subsection. We present each comparison task as its own *subsubsection*, with a brief problem description, a one-paragraph explanation for why the task might reward SN inductive biases (if applicable), and a compact result table for that task. We finish with a short summary table that collates the win/loss/draw across tasks and highlights typical margins.

8.3.1 Tri-wave mixture with sparse interactions (`TriWaveMix10D`)

Task. Scalar regression on a 10D input with a natural mix of smooth and piecewise-linear periodic components:

$$\mathrm{tri}(u) = 2\big|\,(u \bmod 1) - \tfrac{1}{2}\,\big|, \qquad f(x) = 0.5\ \mathrm{tri}\big(4\,w_1^\top x_{1:3}\big) + 0.3\ \mathrm{tri}\big(3\,w_2^\top x_{4:7}\big) + 0.2\ \sin\big(2\pi\,w_3^\top x_{8:10}\big),$$

with $x \in [0,1]^{10}$, and w_1, w_2, w_3 fixed unit vectors. The target combines two *triangular* waves (piecewise linear, kinked) and one sinusoid (smooth), with mild anisotropy via disjoint coordinate groups.

Why this can favor SNs (but remains fair). The inner/outer maps in an SN are piecewise-linear splines whose kinks align well with the triangular-wave components; cubic KAN splines, at a matched knot budget, tend to oversmooth near the kinks and require more knots per dimension to approximate the corners faithfully. The mixture also includes a smooth sinusoid, preventing the task from being "tailored" to any one model while still being a natural stress test of kink fidelity and anisotropic coupling.

Metrics. We report *best* held-out MSE over the 4000 training epochs and single-run wall clock (seconds) on CPU; both models share the same train/eval splits and loss (MSE).

Model	#Params	Eval MSE ↓	Wall clock (s) ↓
SN (PWL splines; domain warm-up 400ep)	977	0.0210	1701
KAN (cubic splines)	977	0.3687	9831

Table 2: `TriWaveMix10D` apples-to-apples comparison at matched parameter count and 4000 epochs. SN attains a $\sim$17.6$\times$ lower MSE and $\sim$5.78$\times$ faster wall clock than KAN on CPU.

Fairness & setup. Both models train for exactly 4000 epochs with Adam (lr=3×10^{-4}) and shared batch size and weight decay (10^{-7}). Residuals are standard linear/projection in both; BatchNorm is placed *after* blocks and evaluated with `batch_no_update` semantics. No lateral mixing or extra normalizers are used. SN's spline domains update for the first 400 epochs (warm-up) and are then frozen; KAN's knot budget is set to match the total parameter count exactly.

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

Setup. Targets are $y_j(x) = \tanh\big(\beta(\mu(x) + \sigma(x) z_j)\big)$ with strictly increasing quantile indices $z_j = \Phi^{-1}(\tau_j)$, and μ, σ sums of monotone per–coordinate transforms plus small positive pairwise terms (guaranteeing coordinatewise and headwise monotonicity). We train an SN and a KAN under strict parity: SN arch $[24, 24]$ with ϕ/Φ knot counts 85/84, residuals "linear", BN (batch, after, skip_first), no lateral; KAN arch $[4, 4]$, degree 3, residual "linear", outside "linear", BN (batch, after, skip_first), impl=fast. Parameters are equalized via K=8 so both models have 1,349 trainables. Training uses 4,000 epochs on the same 1,024 train points (SN's generator) and a fixed test set of 50,000 points; device=CPU; seeds $s \in \{0, \ldots, 9\}$.

Metrics. Primary: *mean RMSE across heads* (lower is better). Secondary: Frobenius error between target/prediction head–correlation matrices, and the monotonicity–violation rate across heads.

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.

8.3.3 Max-of-Affines (8D→1D) regression: piecewise-linear target, PWL SN vs. cubic KAN (10 seeds)

Setup. We regress a *convex, piecewise-linear* function $f:\mathbb{R}^8\to\mathbb{R}$ constructed as the maximum of 48 random affine pieces (dataset: `MaxOfAffines8D`).[2] Both models use the same 3-layer width-{8,8,8} backbone, cubic splines for KAN and piecewise-linear splines with domain updates for SN, with identical knot counts per layer (ϕ=Φ=64). We fix the training budget at **4000 epochs**, use only linear/projection residuals and batch norm (no attention or extra mixing), and warm up SN's domain updates for **400 epochs** (then freeze). Parameter counts are matched (`SN=434, KAN=434`). Optimizer, LR schedule, batch size, weight decay, splits, and losses are identical.

Why this favors SN (but fairly). The target is exactly piecewise linear with sparse kink sets; SN's PWL spline basis can represent it *without* curvature, whereas a cubic-spline KAN must approximate kinks by smoothing. This mirrors fitting ReLU-type ground truth: a PWL hypothesis class should need fewer effective degrees of freedom to reach the same error. The task is natural (convex max-of-affines), not bespoke.

Results across 10 seeds. SN attains lower final eval MSE than KAN in **8/10** runs (wins all but seeds 4 and 9). Examples: SN(1): MSE = 6.03×10^{-4} vs. KAN(1): 1.39×10^{-2} (win), SN(8): 1.32×10^{-3} vs. KAN(8): 4.45×10^{-3} (win), SN(3): 1.07×10^{-2} vs. KAN(3): 2.76×10^{-2} (win), SN(4): 1.96×10^{-2} vs. KAN(4): 1.34×10^{-2} (loss). Aggregating over seeds, SN's mean eval MSE is 9.95×10^{-3} (median 8.76×10^{-3}; min 6.03×10^{-4}; max 1.96×10^{-2}) versus KAN's 1.64×10^{-2} (median 1.50×10^{-2}; min 4.45×10^{-3}; max 2.76×10^{-2}). Hence SN reduces final error by roughly **39**% on average.

Method	Avg Eval MSE ↓	Median ↓	Min	Max
SN (PWL + domains)	**0.00995**	**0.00876**	6.03×10^{-4}	1.96×10^{-2}
KAN (cubic splines)	0.01639	0.01504	4.45×10^{-3}	2.76×10^{-2}

Table 3: **Max-of-Affines8D** (10 seeds). SN has substantially lower final eval MSE. Min/Max are from individual seeds in the run log.

Wall-clock. On a fixed 4000-epoch CPU budget, **SN trains faster per epoch**: SN ranges 115.28–187.79 s across seeds (avg ≈ 142s), while KAN ranges 238.89–378.36 s (avg ≈ 301s). E.g., SN(2) 115.28s (fastest), SN(9) 187.79s (slowest); KAN(1) 238.89s (fastest), KAN(9) 378.36s (slowest). Moreover, SN often reaches *KAN's final* error early: by epoch 500, SN(0) is at 0.00478 (below KAN(0)'s final 0.00653), and SN(8) is at 0.00220 (below KAN(8)'s final 0.00445).

[2] Convex max-of-affines appears in control/robust modeling and matches ReLU networks' implicit geometry; it is *not* tailored to Sprecher Networks (SNs) but is unequivocally piecewise linear.

Method	Avg wall clock (s) ↓	Min (seed)	Max (seed)
SN (PWL + domains)	$\approx$ 142	115.28 (2)	187.79 (9)
KAN (cubic splines)	$\approx$ 301	238.89 (1)	378.36 (9)

Table 4: Wall-clock over 4000 epochs (CPU). SN is both *faster* and more sample-efficient on this PWL target.

Takeaways. On a piecewise-linear target, PWL SNs are a *better inductive match* than cubic KANs: they win on final error in 8/10 seeds and, importantly, hit KAN-level accuracy much earlier in training—while also training faster per epoch.

Fairness note. Both models trained for exactly 4000 epochs with identical optimizer, schedule, batch size, weight decay, residuals, and batch norm; parameter counts matched (434 vs. 434) and the data splits/losses were identical. SN's only additional mechanism was the symmetric, one-time, 400-epoch domain-update warm-up (then frozen).

8.3.4 Non-smooth high-dimensional payoff: 20D basket option regression (`basket_options_nd`)

Task. We regress a 20D basket-option payoff surface with a strike-induced *kink* (i.e., $\max\{0, \sum_i w_i x_i - K\}$–like terms plus band penalties), a natural target that is piecewise linear within regions and non-smooth along exercise boundaries. Such targets are ubiquitous in quantitative modeling.

Why this favors SNs (but is fair). SNs use piecewise-linear (PWL) splines and perform per-layer *domain updates* during a short warm-up; we then freeze those domains. This moves the uniform knot grids to span the activation ranges actually encountered, preserving linearity within regions and concentrating resolution where the composition lives. In contrast, KANs with cubic splines approximate kinks with higher-order pieces that can overshoot near non-smooth boundaries unless heavily over-knotted; on kinked surfaces this makes the PWL bias of SNs advantageous. The setup keeps data, budget, and optimizer identical across models.

Setup (apples-to-apples). Both models train for exactly 4000 epochs with Adam (10^{-3}), batch size 512, weight decay 10^{-6}, and gradient clipping 1.0. We use BatchNorm (after-block, skip-first) and *linear/projection* residuals only for both models. Architectures are matched (32–32; 64 knots for ϕ and Φ), and total parameters are matched within $\pm 5\%$ (here, exactly equal). The SN receives a 400-epoch domain-update warm-up (then domains are frozen). We fix seed 0 for the results below.

Results. On the 4,096-point test set, the SN attains MSE$=1.545 \times 10^{-3}$ and MAE$=3.043 \times 10^{-2}$, versus the KAN's MSE$=2.491 \times 10^{-2}$ and MAE$=1.124 \times 10^{-1}$. This is a **16.1× lower MSE** and **3.69× lower MAE** for SN at identical parameter counts (955 vs 955). Wall-clock for 4000 epochs on CPU is **2,980 s (SN)** vs **5,366 s (KAN)**, i.e., SN is **∼1.80× faster**.

Discussion. The gains line up with the inductive bias: (i) PWL splines match piecewise-linear interiors; (ii) domain updates relocate knot *ranges* to the regions the composition occupies; (iii) cubic pieces in KANs add curvature where the target has none and can oscillate around kinks, demanding more knots/capacity to match sharp transitions. The advantage holds at matched parameters and identical training budget, while also reducing wall-clock.

Fairness note. Same data splits, optimizer, schedule, batch size, weight decay, residual type (linear/projection), and normalization (BatchNorm) for both; both train for exactly 4000 epochs with matched parameter counts (955 each). The only asymmetry is the SN's 400-epoch domain-update warm-up (then frozen), intrinsic to SN training and completed well before evaluation; it does not add capacity or extra regularization.

8.3.5 Saturating Max–Affine Field (SMAF; 20D, K=40)

Setup. The target is a clipped, piecewise-linear field $y(x) = \sum_{k=1}^{K} c_k \,\mathrm{clamp}(w_k^\top x + b_k,\ \ell_k,\ h_k)$ with $x \sim \mathrm{Unif}([0,1]^{20})$, K=40 channels. We trained on 1024 random points and validated on 4096. SN and KAN used the same architecture $[64, 64]$, Adam (lr$=3 \times 10^{-4}$, weight decay 10^{-7}), full-batch training for 4000 epochs; SN had a 400-epoch warm-up for domain updates and then froze domains. Both models used

BN (post-block, skip first), linear/projection residuals only, identical splits/metric, and parameter counts were matched *exactly* (both 1627 parameters). KAN splines were standard cubic Catmull–Rom with tied ϕ/Φ knot counts chosen automatically to match SN's parameter budget.

Metric. Validation MSE; we report the best value over training for each seed and aggregate over 10 seeds ($\{0, 1, \ldots, 9\}$).

Results. SN clearly wins on SMAF. Across seeds (best-per-seed, then averaged), SN achieved $4.85 \times 10^{-4} \pm 7.56 \times 10^{-4}$ MSE, whereas KAN yielded $2.12 \times 10^{2} \pm 4.98 \times 10^{2}$ MSE.[3] For the single best seeds, SN reached 4.04×10^{-5} (seed 7) and KAN 6.38×10^{-3} (seed 6), i.e. a $\sim 1.58 \times 10^{2}$ gap in favor of SN. Representative KAN runs exhibited large validation spikes (often orders of magnitude), while SN converged smoothly to 10^{-4}–10^{-3}-level errors.

Why SN wins here. SMAF is a sum of *clamped affine* channels, i.e., a piecewise-linear map with many kinks and plateaus. SN's piecewise-linear ϕ/Φ splines align with this inductive bias and fit with few knots. In contrast, cubic KAN splines tend to overshoot near kinks/plateaus and need extra capacity/damping to avoid instability; under an equal-parameter budget they underfit (or blow up) on this non-smooth objective.

Timing. Device: CPU. This run did not record wall-clock times; only metric traces were logged.

8.3.6 Dense-head shifts (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:

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

with $x \in [0,1]^D$, $j = 0, \ldots, m-1$, $q_j \in [-1, 1]$ on a uniform grid, and $z_j = \Phi^{-1}(\tau_j)$.[4] We retain the parity/eval protocol from the parity benches (parameter matching and BN evaluation mode).

Setup. D=12, m=64, α=0.08, β=0.7, q_{bias}=0.15; train for 4000 epochs on CPU; n_{test}=50,000. BN evaluation uses batch statistics without buffer updates (`batch_no_update`). Parameter parity was enforced (`--equalize_params --prefer_leq`), yielding #params $\approx$ 3072 for both models; the KAN basis size selected by parity was K=4. The specific model configurations were:

- **SN:** `arch=[32,32]`, ϕ/Φ knots 60/60, BN: batch/after/skip_first, linear residuals, no lateral, domains warmup+freeze. (Notes recorded per run include BN eval mode and the parity choice.)
- **KAN (fast impl.):** `arch=[3,24,4]`, degree 3, K=4, BN: batch/after/skip_first, linear outside map, linear residual.

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 $\approx 1.27\times$ 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 `train_seconds` values: KAN $\approx$ **476 s**, SN $\approx$ **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.

[3] Means $\pm$ std over seeds. Large KAN variability is due to frequent blow-ups on this non-smooth target.
[4] Dataset definition and bench harness are in `benchmarks/kan_sn_densehead_shift_bench.py`.

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 $\approx 27\%$ lower RMSE on average) under strict parameter parity ($\sim$3k parameters each, KAN K=4). KAN trains $\approx 3\times$ 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.

8.3.7 Piecewise-linear sensors (S^2HI; 16D, scalar)

Task. We introduce a simple, natural piecewise-linear (PWL) "Sensor Saturation + Hinge Interactions" (S^2HI) generator: for $x \in \mathbb{R}^{16}$, the target is

$$y \;=\; \sum_i a_i \,\mathrm{clamp}(x_i - \theta_i, 0, c_i) \;+\; \sum_k b_k \,\mathrm{clamp}(u_k^\top x - \tau_k, 0, s_k) \;+\; \varepsilon,$$

with 8 random pairwise hinges and small noise; outputs are standardized using train statistics to keep evaluation fair. Inputs follow a mixed distribution: 80% uniform on $[0,1]^{16}$ and 20% mild tails from $[-0.5, 1.5]^{16}$. For OOD testing we draw purely from the tail box $[-0.5, 1.5]^{16}$ (same output standardizer). Train/test sizes are 4096/8192 (ID) and 8192 (OOD).

Setup (apples-to-apples). We train a *single-block* SN and a *single-block* cubic KAN for exactly 4000 epochs with Adam (lr=3×10^{-4}, wd=10^{-7}), full-batch, projection/linear residuals, and BatchNorm *after* each block (no skip-first). SN runs a 400-epoch domain-update warm-up then freezes domains; KAN has *no* domain updates and uses uniform cubic B-splines with knots fixed on $[0,1]$. Parameter budgets are matched (both models end up at 1,127 parameters with width 64 and 41 knots for ϕ and Φ).

Metrics. We report MSE and R^2 on standardized targets for ID and OOD sets, averaged over the same batch for both models.

Results (10 seeds; best/mean/median).

- *Best (over seeds).* ID MSE: SN 2.909×10^{-2} vs. KAN 5.289×10^{-2} (×1.82); OOD MSE: SN 4.227×10^{-2} vs. KAN 7.825×10^{-2} (×1.85). (Best seeds visible in the log.)
- *Mean over seeds (ID / OOD).* SN: 0.03752 / 0.05871; KAN: 2.0847 / 2.1306. Corresponding mean R^2: SN 0.9629 / 0.9552; KAN -1.059 / -0.636. (Negative R^2 indicates catastrophic fits on some seeds.)
- *Median over seeds (ID / OOD).* SN: 0.03806 / 0.05739; KAN: 0.782 / 0.879 (still far worse than SN).

Stability. SN wins on *all 10/10 seeds* on both ID and OOD MSE. The KAN exhibits 4/10 seeds with negative ID R^2 (e.g. MSE = 8.98, $R^2 = -7.97$), while its best seed is still behind SN's best.

Why SN wins here. The target is PWL with saturations and hinge kinks; the SN's piecewise-linear spline blocks with domain updates match this function class and adapt support to the heavy-tailed mixture before freezing. In contrast, the cubic KAN's smoother basis tends to overshoot near kinks and, with knots fixed on $[0,1]$ and a matched (small) parameter budget, struggles both on ID mixed tails and on the pure-tail OOD box.

Wall-clock. Not recorded by the script; the summary table leaves the time cell blank. (The training logs include losses and fairness checks but no timing.)

Summary across baseline tasks. Table 5 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 (including fewer monotonicity violations); the 4D→5D smooth regression is closer, with SNs typically ahead by a modest margin.

Reproducibility checklist (concise).

- **Epochs/optimizer:** 4000 epochs for both; Adam-based trainers with default learning rates per codebase; no per-model hyperparameter tuning beyond the fairness knobs.
- **Residuals/BN:** Linear residuals in both; BN layers placed "after" and tested with `batch_no_update`.
- **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.

Task	Metric	SN (best)	KAN (best)	Ratio (KAN/SN)	Time (KAN/SN) (s)
TriWaveMix10D (10D)	RMSE ↓	0.145	0.607	4.19×	9,831 / 1,701
MQSI (20D, 9 heads)	Mean RMSE ↓ (best of 10 seeds)	4.47×10^{-3}	6.58×10^{-3}	1.47×	393 / 589 (avg)
Max-of-Affines8D (8→1)	Best MSE ↓ (10 seeds)	6.03×10^{-4}	4.45×10^{-3}	7.38×	301 / 142 (avg)
Basket options (20→1)	MSE ↓	1.545×10^{-3}	2.491×10^{-2}	16.1×	5,366 / 2,980
SMAF (20D, 40 comps)	Best val MSE ↓ (10 seeds)	4.04×10^{-5}	6.38×10^{-3}	157.9×	—
DenseHeadShift (12D, 64 heads)	Mean RMSE ↓ (10 seeds)	5.47×10^{-3}	6.94×10^{-3}	1.27×	467 / 1,433 (avg)
PWL sensors (S^2HI; 16D)	Best ID MSE ↓ (10 seeds)	2.91×10^{-2}	5.29×10^{-2}	1.82×	—

Table 5: Summary across baseline tasks at matched parameter counts and 4000 epochs; wall-clock time shown for completeness. Times are per-task averages across 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.

8.4 Ablation study: role of the shift η

To empirically validate the importance of the internal shift parameter $\eta^{(\ell)}$ noted in Remark 9, ablation studies can be performed. Training SNs with $\eta^{(\ell)}$ fixed to 0 across all blocks typically results in significantly higher final RMSE (often 1-2 orders of magnitude worse) and potentially slower convergence or saturation at poorer loss values, especially for deeper networks. This confirms that the ηq shift inside $\phi^{(\ell)}$ is not redundant and plays a crucial role in the expressivity or optimization dynamics of multi-layer SNs. Interestingly, the degradation is less severe when lateral mixing is enabled, suggesting that lateral connections may partially compensate for the loss of shift-based diversity.

8.5 MNIST classification

To demonstrate SNs' applicability beyond function approximation, we tested them on the MNIST digit classification task. The 28×28 grayscale images were flattened to 784-dimensional vectors and normalized to [0,1]. A deeper network with architecture $784 \to [100, 100, 100] \to 10$ with bidirectional lateral mixing achieved over 99.5% test accuracy after training for several hours on consumer hardware. A shallower network with architecture $784 \to [100] \to 10$ (using Sprecher's theoretical minimum of one hidden layer, but with significantly fewer than the $2n+1 = 1569$ nodes to guarantee universality) achieved approximately 92% test accuracy. Notably, enabling lateral mixing improved the shallow network's accuracy by about 3%, bringing it to 95%. These results demonstrate that while achieving state-of-the-art performance on image tasks would likely require a convolutional design, SNs can attain competitive results on standard benchmarks. More importantly, they confirm that deeper architectures and lateral mixing provide significant benefits for complex tasks, consistent with observations in traditional deep learning.

Metrics. For predicted outputs $\widehat{Y} \in \mathbb{R}^{N\times d}$ and ground truth $Y \in \mathbb{R}^{N\times d}$ on the test set (N points, $d = 5$ heads), we report:

$$\mathrm{RMSE}_j \;:=\; \Big(\tfrac{1}{N}\sum_{n=1}^{N}(\widehat{y}_{nj} - y_{nj})^2\Big)^{1/2}, \qquad \overline{\mathrm{RMSE}} \;:=\; \tfrac{1}{d}\sum_{j=1}^{d}\mathrm{RMSE}_j,$$

and a *correlation-structure error*

$$\Delta_{\mathrm{corr}} \;:=\; \big\|\mathrm{Corr}(Y) - \mathrm{Corr}(\widehat{Y})\big\|_F,$$

Model & shape	Params	Train MSE	$\overline{\text{RMSE}}$	Δ_{corr}	Train time (s)
Run 1 (`--epochs 4000`, CPU)					
SN $[15, 15]$	514	0.02555	0.17994	0.37600	85.87
KAN $[7, 7]$, K=2	458	0.11152	0.33092	3.56547	66.30
Run 2 (`--epochs 4000`, CPU)					
SN $[15, 15]$	514	0.02162	0.16480	0.33417	81.42
KAN $[10, 10]$, K=4	1150	0.07975	0.28114	1.75026	105.70

Table 6: **SN vs. KAN on a 4→5 smooth synthetic regression.** Each row reports a single training run under the same protocol; SN uses 514 parameters in both runs. Across comparable or even larger budgets, SN attains ≈1.84× (Run 1) and ≈1.71× (Run 2) lower mean test RMSE and a 5–9.5× smaller correlation-structure error. Wall-clock times are broadly comparable in this unoptimized setting.

i.e., the Frobenius norm of the difference between the empirical Pearson correlation matrices of the true and predicted outputs.[5]

Architectures, budgets, and caveats. SN uses two hidden blocks of width 15 (total of 514 trainable parameters). Our KAN baselines vary width and per-edge spline resolution K to be in a similar parameter range. To keep the comparison transparent, we used a minimal KAN-style layer with *edge-wise* learned univariate activation

$$\phi(x) \;=\; w_b\,\text{silu}(x) \;+\; w_s\,\text{spline}(x),$$

matching the residual design advocated for KANs,[6] but with the following simplifications vs. the full *pykan* setup:

- the spline is a **1D piecewise-linear** lookup on a **fixed, uniform grid** (no B-spline basis, no grid-extension or grid-updating during training);
- training uses standard Adam on the full training set (no LBFGS stages);
- no sparsification/pruning or manual symbolification;
- inputs are used as returned by the SN trainer so both models see the same data (no per-edge input normalization or domain adaptation).

These choices make our KAN baseline easy to audit and reproducible, but they likely *understate* the best achievable KAN performance on some tasks where grid extension or higher-order splines help.[7]

Takeaways. (i) At similar parameter scale, SN reduces the average test RMSE by a factor of ∼1.7–1.8 relative to our KAN baseline on this coupled multi-output task. (ii) The 5–9.5× reduction in Δ_{corr} shows that SNs capture *inter-head* dependencies far more faithfully here. (iii) Increasing KAN width and K (Run 002) narrows the gap somewhat in Δ_{corr} but does not reverse it; the mean RMSE gap also persists. (iv) Timing is mixed but similar (SN slower in Run 001, faster in Run 002) and reflects basic, non-optimized PyTorch code for both models; we did not employ KAN-specific speedups such as grid extension or pruning.

Reproducibility. All numbers in Table 6 were produced by the unified script mentioned above; it prints JSON records per-model and saves them to disk, ensuring both models share the exact training split and evaluation pipeline. :contentReference[oaicite:3]index=3

[5] $\text{Corr}(Y) \in \mathbb{R}^{d\times d}$ is computed by standardizing each output head across the test set and forming the head-by-head correlation matrix; $\|\cdot\|_F$ denotes the Frobenius norm. Thus Δ_{corr} is scale-invariant and isolates how well a model captures *inter-head* structure. All five heads had non-negligible variance in these runs, so all were included in Δ_{corr}.

[6] KANs place learnable 1D functions on edges, and (in their reference implementation) use a residual base $b(x) = \text{silu}(x)$ plus a B-spline expansion; see Eqs. (2.10)–(2.12) and surrounding text. :contentReference[oaicite:1]index=1

[7] Grid extension and other KAN implementation details (Sec. 2.4) are known to improve accuracy/speed trade-offs on several benchmarks. :contentReference[oaicite:2]index=2

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 achieving unique memory complexity of $O(LN)$ 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 on tasks like MNIST classification, with the added benefit of interpretable spline visualizations. The lateral mixing mechanism proves particularly valuable for vector-valued outputs and deeper networks, often resulting in smoother learned splines and faster convergence. 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 unique achievement of $O(LN)$ memory complexity distinguishes SNs from all other major neural architectures, suggesting potential applications in memory-constrained environments or when exploring extremely wide networks. 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.