PARAMETER-EFFICIENT LEARNING WITH HYBRID VOLTERRA NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the flexibility of multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs), they learn nonlinear frequency interactions indirectly through successive layers, which can increase parameter requirements. For non-polynomial pointwise activations such as ReLU in these networks, a globally exact finite-order polynomial representation is generally unavailable. To address this limitation, we propose the Hybrid Volterra Neural Network (HVNN), which assigns selected spectral components to an activation-free Volterra branch and sends the remaining spectrum to a neural branch. The Volterra branch directly parameterizes nonlinear responses through finite-order polynomial expansions in the frequency domain, making nonlinear frequency coupling explicit. We further propose spectral interaction sparsification to restrict computation to a budgeted set of valid sum- and difference-frequency pairs, thereby limiting evaluated nonlinear interactions. Thus, HVNN constrains model complexity, targeting improved predictive performance with a modest parameter count. HVNN-MLP and HVNN-CNN variants are evaluated on four nonlinear system-identification, three image-denoising and two image-classification datasets to demonstrate their ability across diverse predictive tasks and to achieve a favorable accuracy–parameter trade-off. Our HVNN-CNN reaches PSNR values of 30.7351 and 26.8766 on MNIST and Fashion-MNIST denoising with 65.6K parameters, compared with 24.1188 and 22.3718 with 79.0K parameters for CNN. The code is available at this link.

1 INTRODUCTION

Parameter efficiency is a central constraint when nonlinear predictors must be deployed under limited budgets (Mostafa & Wang, 2019; Han et al., 2024). This constraint appears in many scenarios such as signal restoration and image recognition, where a model must represent harmonic generation, intermodulation, memory, or local spatial structure without storing a large collection of redundant filters. The relevant objective is therefore not only predictive accuracy, but the accuracy attainable per trainable parameter and per evaluated nonlinear interaction.

Conventional MLPs and CNNs implement nonlinear transformations by alternating learned linear maps or convolutions with pointwise activations such as ReLU or GELU (LeCun et al., 2015). A pointwise nonlinearity does not expose an indexed set of interacting input frequencies: in the Fourier domain, products in the signal domain become frequency combinations, and the corresponding interactions are learned indirectly through subsequent widths, depths, and filters. Fourier analyses further report frequency-dependent learning dynamics and a bias toward lower frequencies in standard neural networks (Rahaman et al., 2019; Xu, 2020); these observations motivate making the relevant frequency structure explicit, but do not by themselves imply a fixed parameter-count reduction. To see the algebraic source of the difficulty, consider a filtered signal $v = h * x$ followed by a finite polynomial nonlinearity $z = \sum_p c_p v^p$ and an output filter $y = g * z$. The Fourier transform of v^p is a p-fold product of input-frequency responses, which is exactly the algebraic structure of a Volterra expansion. For non-polynomial activations, an exact finite-order representation is generally unavailable; for example, applying ReLU to a periodic waveform generally produces a piecewise waveform with an infinite Fourier series. This motivates a direct frequency-product parameterization without asserting that an arbitrary activation network is exactly a finite-order Volterra model.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Volterra series provide such a parameterization (Boyd & Chua, 1985; Roheda et al., 2024a). An order-p response is written as a coefficient-weighted product of p input-frequency components, making nonlinear order and frequency coupling explicit. For piecewise-linear networks, depth and width control the number of linear regions and hence the available representation complexity (Montúfar et al., 2014); this gives a structural reason to treat the capacity used for spectral mixing as a resource in compact models. Low-rank factors can represent many Volterra coefficients with a small number of frequency-dependent vectors. However, a complete high-order Volterra kernel still requires a combinatorial number of frequency tuples, and a finite support can miss dependencies that are useful for a particular task. Thus, the parameter-efficient design problem has two coupled parts: compactly represent the structured spectral interactions, and preserve a small amount of flexibility for the complementary input information.

We address this problem with the *Hybrid Volterra Neural Network* (HVNN). HVNN routes an energy-selected frequency support to an activation-free Volterra branch and routes the complementary spectrum to a compact MLP or CNN branch. The Volterra branch uses low-rank frequency factors, while the neural branch receives only the frequencies not assigned to the structured path. Within the Volterra branch, *spectral interaction sparsification* evaluates a budgeted set of algebraically valid sum- and difference-frequency pairs. Support routing and low-rank factorization primarily control trainable parameter storage; interaction sparsification primarily controls the number of nonlinear products evaluated at inference.

Our contributions are:

- **A parameter-efficient hybrid volterra neural network.** HVNN assigns dominant spectral structure to an activation-free, low-rank Volterra branch and reserves a compact neural branch for the complementary input.
- **Spectral interaction sparsification.** Sparse Pair restricts second-order computation to a budgeted set of valid sum- and difference-frequency interactions. This reduces evaluated interaction count independently of the coefficient parameterization.
- **Accuracy–parameter analysis of HVNN across diverse tasks.** We evaluate system identification, image denoising, and image classification with matched preprocessing and explicit trainable-parameter accounting. The results characterize where the compact representation improves the accuracy–parameter trade-off.

2 Background and Related Work

2.1 Volterra representations

We use the standard Volterra-series definition, in which the response is expanded as a sum of multilinear functionals of the input. For a scalar continuous-time input $x(t)$, the classical order-P Volterra expansion is

$$y(t) = h_0 + \sum_{p=1}^{P} y^{(p)}(t), \qquad y^{(p)}(t) = \int \cdots \int h_p(\tau_1, \ldots, \tau_p) \prod_{j=1}^{p} x(t-\tau_j)\, d\tau_1 \cdots d\tau_p, \tag{1}$$

where h_p is the p-th order Volterra kernel and h_0 is a constant term (Boyd & Chua, 1985). The first-order kernel gives the familiar linear convolution, while higher-order kernels describe interactions among multiple delayed copies of the input (Schetzen, 1980; Worden et al., 2025). For a discrete scalar record, the corresponding finite-memory form is

$$\hat{y}[t] = b + \sum_{p=1}^{P} \sum_{\tau_1, \ldots, \tau_p} h_p[\tau_1, \ldots, \tau_p] \prod_{j=1}^{p} x[t-\tau_j]. \tag{2}$$

Here p is the Volterra order and each τ_j is a delay index. Multi-channel and multi-output systems extend this expression by adding channel and output indices to the input arguments and kernels; the underlying Volterra construction is unchanged. Dense kernels become expensive as both the memory length and p increase. Low-rank Volterra–PARAFAC and tensor-network formulations replace dense kernels by factorized filters or cores (Favier et al., 2012; Batselier et al., 2017), while

regularized nonparametric estimators control conditioning and effective complexity (Birpoutsoukis et al., 2017).

The frequency-domain interpretation follows from the convolution theorem. If $X_c(\omega)$ is the Fourier transform of $x_c(t)$, then an order-p contribution at output frequency ω has the form

$$Y_o^{(p)}(\omega) = \sum_{c_1,\ldots,c_p=1}^{C} \sum_{\omega_1+\cdots+\omega_p\equiv\omega} H_{p,o;c_1,\ldots,c_p}(\omega_1,\ldots,\omega_p) \prod_{j=1}^{p} X_{c_j}(\omega_j). \tag{3}$$

Thus, an order-two response contains sum-frequency and difference-frequency interactions after accounting for conjugate-symmetric negative frequencies. The congruence is taken on the discrete Fourier grid, including the two-dimensional grid used for images. Equation 3 motivates selecting input frequencies and retaining only the interactions that the selected support can generate.

2.2 Pointwise nonlinearities and frequency-product structure

The connection between neural nonlinearities and Volterra operators can be made explicit. Let $v = h * x$ be a filtered signal, let $z = \sigma(v)$, and let $y = g * z$. If the pointwise nonlinearity is represented on the operating range by a finite polynomial, $\sigma(v) \approx c_0 + \sum_{p=1}^{P} c_p v^p$, then the nonconstant response terms satisfy

$$Y(\omega) \approx G(\omega) \sum_{p=1}^{P} c_p \sum_{\omega_1+\cdots+\omega_p\equiv\omega} \prod_{j=1}^{p} H(\omega_j) X(\omega_j). \tag{4}$$

Equation 4 has the same product and frequency-combination structure as Equation 3. It explains why Volterra operators can directly parameterize structured spectral interactions that a conventional network would otherwise learn through successive feature transformations. For non-polynomial activations such as ReLU, a globally exact finite-order polynomial representation is generally unavailable. The statement is therefore a motivation for compact frequency-product parameterization.

2.3 Structured Volterra network models

Trainable Volterra models use low-rank polynomial layers to expose higher-order interactions while controlling kernel cost; recent variants extend this idea to media restoration, multi-kernel representations, and causal active-noise control (Roheda et al., 2024b; Yun et al., 2026; Bai et al., 2025). Volterra Neural Networks make polynomial filters trainable components of a neural architecture (Roheda et al., 2024a). Frequency-domain Volterra pruning has used input-spectrum information to remove weakly excited interactions, while compressive approaches use response-dependent sparsity criteria (Faifer et al., 2025; Laurano et al., 2025). HVNN differs by using the selected support as an input-routing decision: the complement is assigned to a neural branch rather than discarded. The neural branch is used as a general representation learner, while the Volterra branch retains an explicit frequency-interaction structure.

3 Hybrid Volterra Neural Network

3.1 Overview and notation

Volterra models provide direct frequency-product parameterization but incur combinatorial growth in higher-order tuples; neural predictors are flexible but learn these interactions implicitly through learned feature maps. HVNN uses the frequency domain to allocate a limited parameter budget between an activation-free structured branch and a compact neural complement. For an input $x \in \mathbb{R}^{C\times\mathcal{D}}$, the FFT gives $X \in \mathbb{C}^{C\times\Omega}$. For a two-dimensional input of size $H \times W$, Ω is the two-dimensional grid $\{0,\ldots,H-1\} \times \{0,\ldots,W-1\}$, and one shared support is selected across input channels.

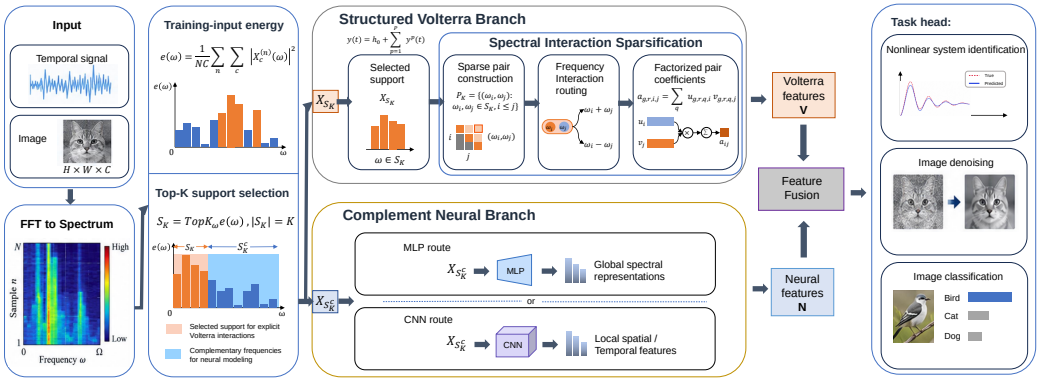


Figure 1: HVNN routes energy-selected Top-K frequencies to an activation-free Volterra branch with spectral interaction sparsification and the complement to a compact neural branch. The Volterra representation provides the main structured feature, and the neural branch supplies a correction.

The HVNN computation is

$$\begin{aligned} x \longrightarrow X \longrightarrow (S_K, S_K^c) \longrightarrow \begin{cases} V = V^{(1)}(X) + V^{(\geq 2)}(X_{S_K}), \\ N = \mathcal{N}(X_{S_K^c}), \end{cases} \\ \longrightarrow \begin{cases} F_V = \Phi_V(V), \\ F_N = \Phi_N(N), \end{cases} \longrightarrow F = F_V + \gamma F_N \longrightarrow \hat{y} = g_{\text{task}}(F) \end{aligned} \tag{5}$$

Here $V^{(1)}$ is the full-spectrum first-order response, while $V^{(\geq 2)}$ contains the routed higher-order interactions. The selected support S_K is used only for the higher-order path; the complement is processed by a compact $\mathcal{N}$, which is an MLP or CNN. The Volterra branch contains no pointwise activation: its nonlinearity is represented by products of frequency responses and learned coefficient factors. The branch maps Φ_V and Φ_N align their feature shapes, after which the neural representation is added as a scaled correction before one shared task head. The fused representation F is the task-level feature passed to the shared head, and $\hat{y}$ is the resulting task prediction. γ controls the correction scale.

3.2 Training-input spectral support via energy Top-K selection

The first design decision is how to divide the input between the structured and neural branches. Using the full spectrum in both branches duplicates input information, whereas discarding all non-selected frequencies deprives the neural branch of potentially useful content. HVNN therefore selects the structured support from training-input energy alone, separating dominant frequency content for explicit higher-order Volterra modeling from the complementary content assigned to the neural branch. For spectrum $X_c^{(n)}(\omega)$ of training example n, define

$$e(\omega) = \frac{1}{NC}\sum_{n=1}^{N}\sum_{c=1}^{C} |X_c^{(n)}(\omega)|^2, \qquad S_K = \text{TopK}_{\omega\in\Omega}\, e(\omega), \qquad |S_K| = K. \tag{6}$$

The inputs are normalized before this calculation, and the support is computed once from the training split without target information. For one-dimensional records, Ω is the temporal-frequency grid; for images, it is the two-dimensional spatial-frequency grid. The same support is shared across input channels. Selection is performed over independent conjugate modes and excludes the DC mode from the nonlinear candidate set; DC remains available to the full-spectrum first-order path. The support is then fixed across orders, factors and channels.

The complement S_K^c feeds the neural branch: its independent real and imaginary coordinates form the MLP input, or it is restored with conjugate symmetry and inverse-transformed for the CNN. Thus, Top-K reduces the structured and complement input dimensions without deleting the information assigned to the neural branch.

3.3 VOLTERRA STRUCTURE AND LOW-RANK PARAMETERIZATION

The second module converts the routed support into a trainable structured operator without introducing a dense high-order kernel. A dense order-p kernel is expensive because every combination of delays and channels receives an independent coefficient. Previous studies have used low-rank factorizations to approximate Volterra kernels (Roheda et al., 2024a; Mohseni et al., 2026). HVNN uses a frequency-domain separable factorization (Wang et al., 2022) so that the parameter count depends on the support and factor ranks rather than on every dense frequency tuple. For order p, selected channel-frequency coefficients are projected into latent factors,

$$z_r(\omega) = \sum_{c=1}^{C} W_{r,c}(\omega) X_c(\omega), \tag{7}$$

and a budgeted p-tuple contributes to the output frequency according to

$$Y^{(p)}_{g,r}(\omega) \mathrel{+}= a^{(p)}_{g,r,\omega_1,\ldots,\omega_p} \prod_{\ell=1}^{p} z_r(\omega_\ell), \qquad \omega_1 + \cdots + \omega_p \equiv \omega, \quad \omega_\ell \in S_K. \tag{8}$$

This subsection defines the general frequency-domain Volterra representation and the latent projection used by the structured branch. Higher-order ablations instantiate Equation 8 with a budgeted tuple set and use the corresponding tuple coefficients. HVNN retains a full-spectrum first-order path for linear information and applies the structured representation only to the routed higher-order path. The time-domain local and full-memory Volterra models use a different low-rank convolutional parameterization and serve as structured references.

3.4 SPECTRAL INTERACTION SPARSIFICATION

The motivation for spectral interaction sparsification is to decide which higher-order interactions should be evaluated, reducing computation without breaking the frequency-combination structure. It is therefore separate from the low-rank parameterization in Section 3.3: sparsification defines the active interaction set, while the factorization supplies coefficients for those active interactions. After Energy Top-K selects S_K, the concrete second-order realization, Sparse Pair, evaluates only valid frequency pairs within this support; the full-spectrum first-order response is retained separately. The construction follows from the Volterra frequency law: an order-p product can contribute to an output frequency only if

$$\omega_1 + \cdots + \omega_p \equiv \omega \quad \text{on the discrete Fourier grid.} \tag{9}$$

For real-valued inputs, conjugate symmetry also gives the corresponding difference-frequency interaction. Sparse Pair therefore converts the selected support into the unordered candidate set

$$\mathcal{P}_K = \{(\omega_i, \omega_j) : \omega_i, \omega_j \in S_K,\ i \le j\}, \qquad |\mathcal{P}_K| = \frac{K(K+1)}{2}. \tag{10}$$

The active set $\mathcal{A}_2 \subseteq \mathcal{P}_K$ is selected using training inputs only. With the energy $e(\omega)$ from Equation 6, each candidate pair is scored by

$$s_{ij} = e(\omega_i) e(\omega_j), \qquad (\omega_i, \omega_j) \in \mathcal{P}_K.$$

For a Pair-retention ratio $\rho \in (0, 1]$, we retain the $\lceil \rho |\mathcal{P}_K| \rceil$ highest-scoring Pairs, with at least one Pair retained. An optional explicit Pair budget can further cap the retained count. The retained set defines $\mathcal{A}_2$.

For every active Pair, the latent projections and factorized coefficients defined in Section 3.3 produce $a^s_{g,r,i,j} = \sum_{q=1}^{Q} u^s_{g,r,q,i} v^s_{g,r,q,j}$, where $s \in \{+, -\}$. $u^s_{g,r,q,i}$ and $v^s_{g,r,q,j}$ are trainable frequency factors for the left and right members of a Pair; i and j index the selected support frequencies, r indexes the latent Volterra factor, g indexes the Pair feature group, q indexes the coefficient factors, and Q is the coefficient factorization rank. The sum and difference branches use separate factor banks, indicated by s, but share the same separable parameterization. The resulting coefficients and latent projections produce

$$Y^{(2)}_{g,r}(\omega_i + \omega_j) \mathrel{+}= a^{+}_{g,r,i,j} z_r(\omega_i) z_r(\omega_j), \qquad Y^{(2)}_{g,r}(\omega_i - \omega_j) \mathrel{+}= a^{-}_{g,r,i,j} z_r(\omega_i) \overline{z_r(\omega_j)}. \tag{11}$$

Here, the sum and difference products are scattered to their corresponding output-frequency bins modulo the Fourier grid, and collisions are accumulated. The scattered spectrum is then completed using the conjugate-symmetry relation of real inputs and transformed back to the signal domain. This second-order response is added to the full-spectrum first-order response, with a projection used when the two feature widths differ.

In summary, support selection and Pair construction determine which interactions are active, whereas the factor ranks determine how the corresponding coefficients are parameterized. The support size K controls the higher-order input and candidate-pair space, the active count J_2 controls evaluated interaction cost, and factorization controls coefficient storage. Thus, Pair sparsification primarily reduces active interaction computation; it is not itself the low-rank parameterization. The full-Pair route is the unpruned control, and budgeted higher-order tuples are used only in controlled ablations.

3.5 Complement neural branches and feature fusion

Support routing preserves complementary information, but the complement may contain dependencies that are not efficiently described by the selected Volterra order. The neural branch models these dependencies without reusing the structured branch's selected-support input. Its hidden width is controlled independently by the neural-branch configuration; the current implementation preserves the configured Volterra feature width and adds a projected neural correction. The MLP receives independent real Fourier coordinates, while the CNN receives the real signal reconstructed from the complement:

$$N_{\mathrm{MLP}} = f_{\mathrm{MLP}}\big(\xi(X_{S_K^c})\big), \qquad N_{\mathrm{CNN}} = f_{\mathrm{CNN}}\big(\mathcal{F}^{-1}(X_{S_K^c})\big), \tag{12}$$

where ξ collects independent real and imaginary coordinates. The MLP learns global coordinate combinations, while the CNN captures local spatial or temporal patterns. The implementation uses a compact bottleneck for the MLP and a position-preserving convolutional mapping for the CNN. For the feature-fusion route, the neural output is mapped to the Volterra feature shape and added before the task head:

$$F = \Phi_V(V) + \gamma \Phi_N(N), \qquad \hat{y} = g_{\mathrm{task}}(F). \tag{13}$$

For classification, Φ_V is the classification-head feature map and Φ_N is either a position-preserving projection or the compact MLP correction aligned with that feature map. For regression and denoising, feature-add maps the neural feature map to the Volterra feature shape before the shared task head. The scalar γ is initialized to 0.1 and learned jointly.

4 Experiments

We evaluate HVNN across nonlinear system identification, image denoising, and image classification. The experiments focus on the predictive value of Sparse Pair Volterra interactions, the complementary feature contribution of the MLP and CNN branches, the parameter-efficiency trade-offs among HVNN and standalone neural baselines, and the effect of Energy Top-K support selection.

4.1 Tasks and datasets

We evaluate (1) nonlinear system identification on four datasets: the asymmetric Duffing oscillator (Duffing), the Van der Pol oscillator (VdP), Wiener–Hammerstein (WH), and Cascaded Tanks (Tanks); (2) image denoising on MNIST, Fashion-MNIST, and CIFAR-10; and (3) image classification on MNIST and Fashion-MNIST. The system-identification datasets are cited by their respective benchmark or dynamical-system sources (Worden et al., 2025; van der Pol, 1926; Schoukens et al., 2009; 2016), and the image datasets by their original dataset references (LeCun et al., 1998; Xiao et al., 2017; Krizhevsky, 2009). Full dataset and corruption details are given in Appendix A.

4.2 Baselines, protocol, and model variants

We compare the core routes with MLP, CNN, and KAN (Liu et al., 2025) references under matched task splits, preprocessing, scoring, and parameter accounting. The neural baselines use the same task-specific implementation and activation convention recorded in Appendix C. Full protocol details are given in Appendix A. The resulting model matrix is shown in Table 1.

Table 1: Model matrix used in the experiments. HVNN routes use the full-spectrum first-order path and a Top-K Sparse Pair higher-order path, together with a complementary neural branch and feature-add fusion.

Model	Volterra branch	Complement branch	Role
Sparse Pair Volterra	Top-K Sparse Pair	None	Structured reference
HVNN-MLP	Top-K Sparse Pair	MLP	Global complement
HVNN-CNN	Top-K Sparse Pair	CNN	Local complement
MLP	None	Full-input MLP	Neural reference
CNN	None	Full-input CNN	Neural reference
KAN	None	Full-input KAN	Neural reference

Table 2: System-identification results. Each cell reports NMSE and trainable parameters. Bold and underlined values indicate the best and second-best NMSE, while bold and underlined parameter counts indicate the smallest and second-smallest models in each dataset column.

Method	Simulated dynamical systems		Measured benchmark systems	
	Duffing	VdP	WH	Tanks
Sparse Pair Volterra	**0.1558 ± 0.0036** 41,973	0.8760 ± 0.0029 41,973	**0.0200 ± 0.0001** 41,973	1.0359 ± 0.0363 11,647
HVNN-MLP	0.1666 ± 0.0130 46,582	1.0371 ± 0.0363 46,582	0.0210 ± 0.0002 46,582	0.9574 ± 0.2010 19,936
HVNN-CNN	0.1562 ± 0.0032 **34,950**	**0.8749 ± 0.0036** **34,950**	0.0220 ± 0.0015 **34,950**	**0.8083 ± 0.0227** **10,768**
MLP	0.9634 ± 0.0333 52,449	1.0131 ± 0.0413 52,449	0.8558 ± 0.0127 52,449	0.8630 ± 0.1104 40,161
CNN	0.9913 ± 0.0125 46,961	1.0186 ± 0.0033 46,961	0.7184 ± 0.0058 46,961	0.9709 ± 0.0703 46,961
KAN	0.8064 ± 0.0293 492,577	0.9475 ± 0.0128 492,577	0.7451 ± 0.0074 492,577	0.9618 ± 0.0634 369,697

4.3 Main results

The main results across the three tasks are reported in Tables 2, 3, and 4. The results support a parameter–accuracy trade-off rather than a universal ranking. Sparse Pair Volterra provides the strongest structured reference for Duffing and Wiener–Hammerstein, while HVNN-CNN is best on Van der Pol and Cascaded Tanks. Sparse Pair uses substantially fewer parameters than KAN on all four system-identification datasets. HVNN-CNN is most effective when the complementary information has local structure: it obtains the best denoising results on MNIST and Fashion-MNIST with approximately 65.6K parameters. For image classification, CNN remains the strongest accuracy baseline, while HVNN-CNN approaches it with 50.7K parameters versus 86.9K. HVNN-MLP is less consistent, indicating that the choice of complement architecture is task-dependent.

The hybrid architecture combines explicit spectral interactions with locally modeled features from the remaining frequencies. HVNN-CNN applies position-preserving convolutions to the complementary spectrum and adds the resulting features to the Volterra representation before the shared task head (Equations 12 and 13). Relative to Sparse Pair Volterra, it lowers NMSE on Van der Pol (0.8749 versus 0.8760) and Cascaded Tanks (0.8083 versus 1.0359), and raises PSNR on MNIST (30.7351 versus 30.6919) and Fashion-MNIST (26.8766 versus 26.8310; Tables 2 and 3). These gains are task-dependent: Sparse Pair Volterra remains stronger on Duffing and WH, while standalone CNN leads classification. These results support the use of a compact structured representation with a local neural complement on some tasks.

Representative task-level efficiency. Figure 2 presents parameter-efficiency trade-offs on one representative dataset from each task: Duffing system identification, MNIST denoising, and Fashion-MNIST classification. KAN is omitted because its parameter count is substantially larger than the compact comparison models; its numerical results remain reported in the tables. Sparse Pair and HVNN-CNN both reach Duffing NMSE near 0.156 with tens of thousands of parameters. On

Table 3: Mean image-denoising results under the specified fixed-level corruption protocol. Each dataset reports PSNR and trainable parameter count in separate columns; bold and underlined values indicate the best and second-best PSNR, while parameter formatting indicates the smallest and second-smallest models.

Method	MNIST		Fashion-MNIST		CIFAR-10	
	PSNR ↑	Parameters ↓	PSNR ↑	Parameters ↓	PSNR ↑	Parameters ↓
Sparse Pair Volterra	<u>30.6919</u>	**63,609**	<u>26.8310</u>	**63,609**	**27.5143**	215,419
HVNN-MLP	30.6265	101,738	26.7115	101,802	26.9862	282,860
HVNN-CNN	**30.7351**	<u>65,610</u>	**26.8766**	<u>65,610</u>	<u>27.4660</u>	217,996
MLP	23.9800	130,849	22.2786	130,849	20.5084	<u>204,099</u>
CNN	24.1188	79,041	22.3718	79,041	20.6167	**80,675**
KAN	24.0199	1,255,457	22.2942	1,255,457	20.5072	1,987,651

Table 4: Image-classification results on MNIST and Fashion-MNIST. Each dataset reports accuracy and trainable parameter count in separate columns; bold and underlined values indicate the best and second-best accuracy, while parameter formatting indicates the smallest and second-smallest models.

Method	MNIST		Fashion-MNIST	
	ACC ↑	Parameters ↓	ACC ↑	Parameters ↓
Sparse Pair Volterra	0.9890 ± 0.0007	<u>61,906</u>	0.9025 ± 0.0020	<u>61,906</u>
HVNN-MLP	0.9783 ± 0.0007	81,315	0.8704 ± 0.0016	81,347
HVNN-CNN	<u>0.9916 ± 0.0010</u>	**50,723**	<u>0.9046 ± 0.0018</u>	**50,723**
MLP	0.9700 ± 0.0007	138,714	0.8747 ± 0.0004	138,714
CNN	$\mathbf{0.9937 \pm 0.0004}$	86,906	$\mathbf{0.9098 \pm 0.0014}$	86,906

MNIST denoising, HVNN-CNN has the highest displayed PSNR with approximately 66K parameters. On Fashion-MNIST classification, HVNN-CNN reaches accuracy close to the CNN baseline with roughly 50K parameters versus 87K for CNN. The bold marker in each panel identifies the better-performing HVNN variant.

4.4 Ablation studies

Support selection. Table 5 reports the main Energy Top-K configuration and the random-K configuration on four system-identification datasets across 10 runs. Under the reported configurations, Energy Top-K has lower mean NMSE on Duffing, VdP, and Tanks.

Pair budget. Table 6 reports the main Energy Top-K result and the effect of varying the active Pair ratio ρ within the selected support across 10 runs. Within the ablation runs, $\rho = 0.01$ reduces the evaluated Pair count from 11,935 to 120 on WH, while retaining or improving NMSE on Duffing, Van der Pol, and Tanks. The full Pair route performs better on WH, and the two settings are similar on Tanks. Thus, Sparse Pair provides strong control over active interaction computation, but its accuracy benefit and its parameter effect remain task- and parameterization-dependent.

4.5 Complexity analysis

Let $P_{\text{trainable}}$ denote the number of trainable parameters, F_Ω the number of independent frequency modes, K the support size, C the input-channel count, R the factor count, and J_2 the number of active order-two pairs. The full-spectrum linear path scales with F_Ω, Top-K factors with $O(RCK)$, and the candidate pair count is $J_{\text{cand}} = \frac{K(K+1)}{2}$.

The active count $J_2 \leq J_{\text{cand}}$ is set by the Pair budget. With frequency-factor rank Q and pair-group count G, the factorized coefficient storage scales as $O(GRQK)$, while the active interaction workload scales with J_2. Thus, support size and factorization primarily control $P_{\text{trainable}}$, whereas Pair sparsification primarily controls active interactions and their arithmetic cost. The hybrid model keeps the configured Volterra feature width and adds a comparatively compact neural branch rather

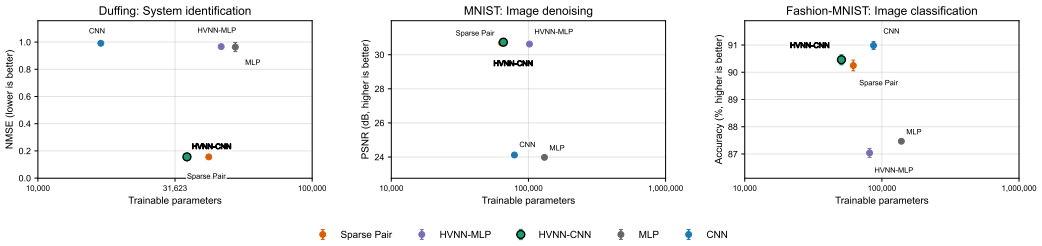


Figure 2: Representative parameter-efficiency trade-offs across the three tasks. KAN is omitted because its parameter count is substantially larger than the displayed compact comparison models. Points show mean predictive results with available standard deviations; lower NMSE is better for Duffing, while higher PSNR and classification accuracy are better for the image tasks. The bold marker identifies the better-performing HVNN variant in each panel; CNN is not bolded.

Table 5: Support-selection configurations. Each cell reports NMSE and trainable parameters.

Configuration	Duffing	VdP	WH	Tanks
Energy Top-K	0.1558 ± 0.0036 41,973	0.8760 ± 0.0029 41,973	0.0200 ± 0.0001 41,973	0.9574 ± 0.2010 11,647
Random-K	0.1693 ± 0.0057 41,973	0.9562 ± 0.0144 41,973	0.0170 ± 0.0009 41,973	1.0044 ± 0.1720 11,647

Table 6: Pair-budget results under Energy Top-K support after 10 runs. Each cell reports NMSE and evaluated Pair count.

Setting	Duffing	VdP	WH	Tanks
Main: $\rho = 0.01$	0.1558 ± 0.0036 120	0.8760 ± 0.0029 120	0.0200 ± 0.0001 120	0.9574 ± 0.2010 8
Ablation: $\rho = 1$	0.1683 ± 0.0024 11,935	0.9548 ± 0.0115 11,935	0.0086 ± 0.0003 11,935	0.9965 ± 0.1833 780
Ablation: $\rho = 0.01$	0.1562 ± 0.0029 120	0.8798 ± 0.0075 120	0.0200 ± 0.0001 120	0.9977 ± 0.1814 8

than a second full-input predictor. Its parameter cost is determined by the complement input dimension, neural hidden width, and feature-alignment map. Because the neural branch receives only S_K^c, it does not duplicate the Volterra branch's selected-support input parameters; its compact bottleneck or convolutional widths further limit the added capacity. The MLP input width is determined by the complementary spectrum and its bottleneck, while CNN parameters are determined by convolutional widths and kernels. This complementary allocation explains why HVNN remains smaller than a standalone neural model or a full hybrid with duplicated inputs.

5 Conclusion

This paper studies parameter-efficient nonlinear prediction through explicit frequency-product structure. HVNN combines Energy Top-K input routing, Sparse Pair Volterra interactions, and a complementary neural feature branch in one task-adaptive architecture. The selected support determines which input frequencies enter the structured branch, the Pair budget determines which algebraically valid interactions are evaluated, and the neural branch corrects the Volterra feature representation from the complementary input before the shared task head. Evaluation across system identification, image denoising, and image classification examines these design mechanisms under different task heads and data structures.

AI USE STATEMENT

In this work, we used generative AI tools for helping develop theoretical models or conceptual frameworks, assisting in the writing of proofs, designing or providing feedback on research methodology or experiments, and assisting with translation. We have not used generative AI tools for generating synthetic data sets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, implementing methods, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, and interpreting results. Proposing or refining hypotheses is not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, summarizing or analysing existing literature, and creating or editing software code. We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey, and LLM-generated code was verified and tested for correctness by all the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lu Bai, Siyuan Lian, Mengtong Li, Yiming He, Li Rao, Xiaofeng Zeng, Ruquan Su, Kai Chen, and Jing Lu. Wavenet-Volterra neural network for active noise control: A fully causal approach, 2025. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5342956.

Kim Batselier, Zhongming Chen, and Ngai Wong. Tensor network alternating linear scheme for MIMO Volterra system identification. *Automatica*, 84:26–35, 2017. doi: 10.1016/j.automatica.2017.06.033. URL https://arxiv.org/abs/1607.00127.

G. Birpoutsoukis, A. Marconato, J. Lataire, and J. Schoukens. Regularized nonparametric Volterra kernel estimation. *Automatica*, 2017. doi: 10.1016/j.automatica.2017.04.014.

Stephen Boyd and Leon O. Chua. Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Transactions on Circuits and Systems*, 32(11):1150–1161, 1985. doi: 10.1109/TCS.1985.1085649.

Marco Faifer, Christian Laurano, Roberto Ottoboni, Sergio Toscani, and Michele Zanoni. Definition of pruned frequency-domain Volterra models based on knowledge about the input spectrum. *IEEE Transactions on Instrumentation and Measurement*, 74:1–12, 2025. doi: 10.1109/TIM.2025.3548761.

Gérard Favier, Alain Y. Kibangou, and Thomas Bouilloc. Nonlinear system modeling and identification using Volterra–PARAFAC models. *International Journal of Adaptive Control and Signal Processing*, 26:30–53, 2012. doi: 10.1002/acs.1272.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/cifar.html. Source of the CIFAR-10 dataset.

Christian Laurano, Sergio Toscani, Paolo Attilio Pegoraro, and Antonio Vincenzo Solinas. Compressive sensing-based pruning of frequency-domain Volterra models. In *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2025. doi: 10.1109/I2MTC62753.2025.11079214.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pp. 2278–2324, 1998. doi: 10.1109/5.726791. Source of the MNIST handwritten-digit dataset.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. KAN: Kolmogorov–Arnold networks. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ozo7qJ5vZi.

Peiman Mohseni, Nick Duffield, and Raymond KW Wong. Revisiting neural processes via fourier transform and volterra series. *arXiv preprint arXiv:2606.01172*, 2026.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *arXiv preprint arXiv:1402.1869*, 2014. URL https://arxiv.org/abs/1402.1869.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 5301–5310. PMLR, 2019. URL https://proceedings.mlr.press/v97/rahaman19a.html.

Siddharth Roheda, Hamid Krim, and Bo Jiang. Volterra neural networks (vnns). *Journal of Machine Learning Research*, 25:1–29, 2024a.

Siddharth Roheda, Amit Unde, and Loay Rashid. MR-VNet: Media restoration using Volterra networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6098–6107, 2024b. doi: 10.1109/CVPR52733.2024.00583.

Martin Schetzen. *The Volterra and Wiener Theories of Nonlinear Systems*. John Wiley and Sons, 1980.

Johan Schoukens, Johan A. K. Suykens, and Lennart Ljung. Wiener–Hammerstein benchmark. In *15th IFAC Symposium on System Identification (SYSID 2009)*, St. Malo, France, 2009.

Maarten Schoukens, Per Mattsson, Torbjörn Wigren, and Jean-Philippe Noël. Cascaded tanks benchmark combining soft and hard nonlinearities, 2016.

Balthasar van der Pol. On relaxation-oscillations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):978–992, 1926. doi: 10.1080/14786442608564127.

Yanjiao Wang, Shihua Tang, and Xiaobo Gu. Parameter estimation for nonlinear volterra systems by using the multi-innovation identification theory and tensor decomposition. *Journal of the Franklin Institute*, 359(2):1782–1802, 2022.

Keith Worden, Timothy Rogers, and Oliver Preston. Machine-learning perspectives on Volterra system identification. *Philosophical Transactions of the Royal Society A*, 383:20240053, 2025. doi: 10.1098/rsta.2024.0053.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. URL https://arxiv.org/abs/1708.07747. Source of the Fashion-MNIST dataset.

Zhi-Qin John Xu. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, 2020. doi: 10.4208/cicp.oa-2020-0085. URL https://doi.org/10.4208/cicp.oa-2020-0085.

H. Yun, Hamid Krim, and Y. Bao. kVNN: Learnable multi-kernel Volterra neural networks, 2026. URL https://arxiv.org/abs/2604.15141.

A Experimental protocol details

The main text summarizes the experimental design; this section gives the complete protocol used to produce the reported comparisons.

System-identification datasets. The two simulated dynamical-system datasets are the asymmetric Duffing oscillator case study and the Van der Pol oscillator. The Duffing benchmark follows the asymmetric nonlinear oscillator setting described in the system-identification literature, while the Van der Pol benchmark uses the classical relaxation-oscillator dynamics (Worden et al., 2025; van der Pol, 1926). The measured benchmarks are Wiener–Hammerstein and Cascaded Tanks. Data are split before window construction, and target suffixes used for scoring remain disjoint from the training windows. Inputs and outputs are normalized using training-split statistics only.

Image datasets and denoising protocol. MNIST, Fashion-MNIST, and CIFAR-10 are used for denoising, whereas only MNIST and Fashion-MNIST are used for classification (LeCun et al., 1998; Xiao et al., 2017; Krizhevsky, 2009). For denoising, clean images $x \in [0,1]^{C\times H\times W}$ are mapped to $[-1,1]$ and corrupted according to

$$y = \mathrm{clip}\Big(2x - 1 + \frac{\sigma}{127.5}\epsilon, -1, 1\Big), \qquad \epsilon \sim \mathcal{N}(0, I). \tag{14}$$

The default target is the residual $r = y - (2x-1)$, and the clean estimate is reconstructed as $\hat{x}_{[-1,1]} = y - \hat{r}$. Fixed-noise experiments use $\sigma \in \{15, 25, 50\}$ on the 8-bit scale; blind experiments sample $\sigma \in [0, 55]$ per image. Grayscale and RGB images use the same residual convention, with RGB channels modeled jointly. MSE and PSNR are computed after reconstruction, using peak range 2 in normalized coordinates. Classification uses the corresponding class labels and cross-entropy.

Baselines and applicability. The reported comparisons include Sparse Pair Volterra, HVNN-MLP, HVNN-CNN, and standalone MLP, CNN, and KAN references. Trainable parameters are counted after model construction, and all reported methods use the same task-specific preprocessing, splits, and scoring protocol.

Metrics and reporting. System identification reports NMSE, denoising reports reconstructed-image MSE and PSNR, and classification reports accuracy and cross-entropy. Unless a table explicitly indicates a single historical run, values are means and standard deviations over the configured repeated runs. Within each dataset column, bold and underlined primary metrics indicate the best and second-best valid results, respectively. Parameter counts are reported independently of predictive ranking.

B Architecture and parameter accounting

This section makes the model-size comparison explicit. All models are instantiated from the configuration recorded with each run; the parameter counts in the tables are therefore measured after construction rather than inferred from a nominal model name. Let Θ denote the trainable tensors of an instantiated model. We report

$$P_{\mathrm{trainable}} = \sum_{\theta\in\Theta} \mathrm{numel}(\theta), \tag{15}$$

where one scalar in a weight, bias, normalization affine parameter, or learned fusion scale contributes one parameter. Non-trainable buffers, including FFT indices, support masks, pair indices, conjugate-partner indices, and cached spectra, are excluded. Optimizer moments, gradients, temporary pair products, batch-normalization running statistics, and input data are also excluded. In code, this is exactly the sum of `parameter.numel()` over parameters with `requires_grad=True`; the exported parameter breakdown additionally groups this sum by top-level module.

Common task heads. The structured models and the standalone neural baselines use the same task-head interface within each task. For classification, a feature map with M channels is mapped by a 1×1 (or 1-D pointwise) projection to A head channels, followed by affine batch normalization, adaptive average pooling to L locations, and a linear classifier with O output classes. Its trainable head count is

$$P_{\text{cls-head}} = (MA + A) + 2A + (AL\,O + O). \tag{16}$$

For regression or denoising, the head uses a pointwise projection from M to A channels and a pointwise output map from A to O channels, giving

$$P_{\text{reg-head}} = (MA + A) + (AO + O). \tag{17}$$

Here L is the number of pooled spatial or temporal locations. These formulas include biases and normalization affine parameters but not running-statistics buffers.

MLP baseline and complement branch. The standalone MLP flattens the input, applies two hidden linear layers with LayerNorm and GELU between them, and reshapes the final linear output into the feature-map shape expected by the common head. If $D = C\prod_d S_d$ is the flattened input width, h is the hidden width, and $R_M = ML$ is the feature output width, the MLP trunk contains

$$P_{\text{MLP-trunk}} = (Dh + h) + (h^2 + h) + (hR_M + R_M) + 4h, \tag{18}$$

where $4h$ accounts for the affine parameters of the two LayerNorm layers. The total count is this trunk plus Equation 16 or Equation 17. In HVNN-MLP, D is replaced by the real independent coordinates of the complementary spectrum $X_{S_K^c}$. For task-residual fusion, the final MLP output has the configured neural target shape; for feature-add fusion, the classification route outputs the flattened feature representation required by the shared head. Thus, the complement branch does not receive the full input in either case.

CNN baseline and complement branch. The standalone CNN uses three convolutional stages: $C \to h$, $h \to h$, and $h \to 2h$, with kernel size k, batch normalization and GELU after each convolution, and pooling after the second and third stages. For a d-dimensional convolution, its convolutional and normalization parameters are

$$\begin{aligned} P_{\text{CNN-trunk}} = {} & (Chk^d + h) + 2h + (h^2k^d + h) + 2h \\ & + (2h^2k^d + 2h) + 4h. \end{aligned} \tag{19}$$

The common task head is then added. The CNN complement branch uses the inverse FFT of $X_{S_K^c}$ and a two-convolution block with widths $C \to h$ and $h \to m$, where m is the configured neural feature width. Feature-add fusion adds a pointwise $m \to M$ projection; task-residual fusion instead adds a pointwise or linear $m \to O$ residual head. The learnable residual scale contributes one scalar.

Time-domain low-rank Volterra. The time-domain Volterra control contains a full linear convolution and one factor convolution for each nonlinear order. For input width C, output width O, memory length L_t, rank R, and order set $\mathcal{P}$, the Volterra block has

$$P_{\text{time-Volterra}} = (OCL_t + O) + \sum_{p\in\mathcal{P}} (RCL_t + R + OR), \tag{20}$$

before its feature projection and task head. The first term is the linear convolution; each summand contains the order-p factor convolution, its bias, and the output mixing matrix β_p. The local and full-memory routes use the same low-rank structure; they differ only in L_t.

Frequency-domain Volterra and Sparse Pair. The full-spectrum frequency-domain control stores frequency-domain linear and low-rank factors over all candidate modes. Sparse Pair uses the same full-spectrum linear path but restricts nonlinear factors to a support of size K. Let F be the number of independent linear modes, C_I the interaction-channel count, M the Volterra feature width, R the input-factor rank, J the active order-two Pair count, G the number of Pair channels, and Q the frequency-factor rank. The main trainable terms are

$$\begin{aligned} P_{\text{linear}} &= 2MCF + M, \\ P_{\text{input-factor}} &= 2RC_IK, \\ P_{\text{Pair}} &\in \{4GRJ,\ 8GRQK\}, \end{aligned} \tag{21}$$

where the first Pair expression is used by explicit, shared, and grouped parameterizations, and the second is used by the factorized parameterization. The factorized expression contains separate real and imaginary sum/difference factor banks. If $G \neq M$, the pointwise Pair projection contributes $MG + M$; higher-order ablations add $2GRJ_p$ for each active order-p tuple set of size J_p. The appropriate task head is then added using Equations 16–17. Consequently, the Pair ratio primarily changes J and the evaluated interaction workload, whereas K, R, Q, and the factorization determine the trainable coefficient count.

Hybrid accounting and fairness. HVNN contains the Sparse Pair Volterra parameters plus only the complement branch and its specified fusion parameters. The Volterra branch is not narrowed when an MLP or CNN complement is added: its feature width M, support K, rank, Pair set, and task head remain those of the corresponding Sparse Pair route. This makes the complementary representation explicit in the parameter breakdown rather than hiding it in a larger backbone. The main tables report natural-configuration comparisons and expose the resulting trainable counts. When a parameter-matched baseline is used, its width is selected before training to match the reference model's instantiated trainable count and is labeled separately. Therefore, no comparison relies on an unreported optimizer state, FFT cache, or temporary interaction tensor as an apparent model parameter, and predictive differences can be interpreted together with the reported architecture and parameter budget.

C Implementation details

All routes use AdamW with learning rate 10^{-4} and weight decay 10^{-6} unless an explicitly reported override is used. For regression and denoising, the default effective order is $P_{\text{eff}} = 2$, the Pair fraction is $\rho = 0.01$, the frequency-factor rank is $Q = 2$, and the regression Pair budget is 250,000 unless overridden. The shared Sparse Pair feature width is 64 and the task-head width is 16 in the default configuration; the hybrid neural hidden width is 32. Higher-order ablations use the configured order list and interaction budget. These settings define the reported routes together with the support selection, frequency-pair construction, and feature-add fusion described in Section 3.

Hardware and common training protocol. The experiments are run on a workstation equipped with four NVIDIA RTX PRO 6000 Blackwell GPUs, an Intel Xeon Silver 4410Y CPU with 48 logical CPUs, and 503 GiB system memory. The software environment uses Python 3.11.13, PyTorch 2.7.1 with CUDA 12.8. Unless explicitly reported otherwise, the dataset-specific default batch sizes are 128 for image classification and denoising, 64 for simulated and most measured one-dimensional system-identification data, and the source-defined batch size for the remaining measured datasets. The spectral support-selection and chunked spectral-cache operations use batch size 256. The default maximum training duration is 200 epochs, with early stopping after 50 consecutive validation epochs without an improvement larger than 10^{-5} in validation loss. The best validation checkpoint is restored before test evaluation. All methods use the same task split, normalization, seed, batch size, maximum epochs, patience rule, AdamW learning rate 10^{-4}, weight decay 10^{-6}, gradient-norm clipping at 5, and validation-checkpoint selection; hybrid-specific parameter groups use the same learning rate unless an explicitly reported override is stated. No automatic hyperparameter sweep is used for the reported default results: configurations are fixed before each comparison, and any manually selected ablation setting is recorded in its run configuration.

For image denoising, the default target parameterization is residual prediction, with $r = y - x_{[-1,1]}$ and clean reconstruction $\hat{x}_{[-1,1]} = y - \hat{r}$. The alternative `--denoise-target clean` directly predicts $x_{[-1,1]}$. The default fixed noise level is $\sigma = 25$ on the 8-bit scale; `--denoise-sigma` selects other fixed levels, and `--denoise-blind` samples a level independently for each image from $[0, 55]$. The corrupted observation is clipped to $[-1, 1]$ as specified in Equation 14; RGB channels are processed jointly. Unless explicitly stated otherwise, MSE and PSNR are calculated on reconstructed clean images with peak range 2 in normalized coordinates.

D Additional reproducibility details

For every run, we archive the input normalization statistics, selected support indices, Pair list or higher-order interaction list, resolved parameter count, P_{eff}, random seed, hardware, precision, optimizer settings, validation checkpoint, and predictions. System-identification windows are created after split assignment, and denoising and classification test targets are never used for support construction. All reported comparisons use the same task-specific scoring masks and output normalization.