LINEAR REUSABLE NEURAL BASES ARCHITECTURE FOR NETWORK COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory constraints remain a critical bottleneck in the deployment of large-scale AI models. Parameter sharing across network depth reduces model storage, but repeatedly applying an identical transformation limits flexibility across layers. Inspired by time–memory trade-offs in classical algorithms, we introduce the Linear Reusable Neural Bases (LRNB) architecture, an RNN-based framework that improves parameter efficiency through parameter reuse at the *neuron level*. Each feedforward residual module is represented as a linear combination of shared neural bases, with depth-specific learnable coefficients and optional shifts providing flexibility across layers. This formulation reduces parameter redundancy across depth and enables the construction of wider and deeper networks within a fixed parameter budget. We further provide a geometric interpretation of the neural bases from a vector-field perspective and extend the framework to linear projection modules. Experiments demonstrate that the LRNB architecture achieves comparable or lower final training loss than independently parameterized baselines while using fewer parameters and maintaining stable training dynamics. These findings support neuron-level reuse as a practical approach to parameter-efficient network design.

1 INTRODUCTION

large-scale AI models outperform smaller ones across a broad range of domains (Brown et al., 2020; Hoffmann et al., 2022) and have attracted significant attention in the machine learning community. As indicated by empirical scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), models with a larger parameter count typically have stronger approximation capabilities. Consequently, modern AI models are often designed with billions or even trillions of parameters, requiring hundreds or thousands of GPUs/TPUs for training and deployment (Brown et al., 2020; Fedus et al., 2022).

Although large-scale AI models have demonstrated remarkable performance, deployment remain challenging, especially for researchers and organizations with limited computational resources. One of the major bottlenecks is the memory requirement. Deploying a large-scale model often requires operating hundreds or thousands of GPUs simultaneously, and the high cost of specialized hardware further increases the financial burden (Narayanan et al., 2021). Overall, these resource and hardware demands hinder many researchers and organizations from deploying large-scale AI models.

A promising approach to addressing this challenge is to integrate a neuron-reuse mechanism into the model, to reduce parameter redundancy and memory requirements. This idea is closely related to recurrent neural network (RNN) models, which repeatedly apply a shared transformation across different time steps or positions (Elman, 1990; Jordan, 1997). Moreover, consistent with the philosophy that *compression leads to intelligence*, models representing information with fewer parameters often have better generalization. Nevertheless, classical RNN models often suffer from optimization difficulties, as repeated transformations can result in unstable gradient propagation, such as vanishing or exploding gradients. As a result, most large-scale AI models are designed with independent modules and integrate only limited reuse mechanisms.

In light of these observations, a natural question arises:

Can we design an architecture that integrates the reuse mechanism at the neuron level to improve parameter efficiency and lower memory costs, while maintaining training stability?

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.

In this paper, we introduce a novel model architecture, the Linear Reusable Neural Bases (LRNB) architecture, to improve parameter efficiency through parameter reuse at the *neuron level*. Inspired by sinusoidal bases in Fourier analysis and linear bases in linear algebra, we introduce the concept of *neural bases*, with an explanation from a vector-field perspective. We represent each neural network module as a linear combination of shared neural bases with allowable shifts, enabling the same neurons to be reused across depths. This reuse mechanism reduces the parameter redundancy along the depth dimension and improves the model compression rate, allowing for the construction of substantially wider and deeper networks under the same parameter budget. Experimental results show that our architecture achieves comparable or lower final loss than classical architectures, while maintaining stable training dynamics and reducing the parameter count.

Organization. The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 provides background on deep learning theory and residual networks. Section 4 introduces the theory of neural bases, the model architecture, and the implementation details. Section 5 discusses the advantages and limitations of the model. Section 6 presents computational experiments to evaluate the model.

Notation. We use $\sigma(\cdot)$ to denote a nonlinear activation function, $\phi(\cdot)$ a neural basis function, and $R(\cdot)$ a residual block. The notation $\langle \cdot, \cdot \rangle$ denotes the inner product. We write d for the embedding dimension, m for the network width (i.e., the hidden dimension or number of neurons), L for the network depth, and T for the token sequence length. For a residual feedforward block, (S, b) denotes the sensor parameters, and P denotes the response parameters. For the j-th neural basis function (or neuron), the corresponding sensor and response parameters are denoted by (s_j, b_j) and p_j, respectively.

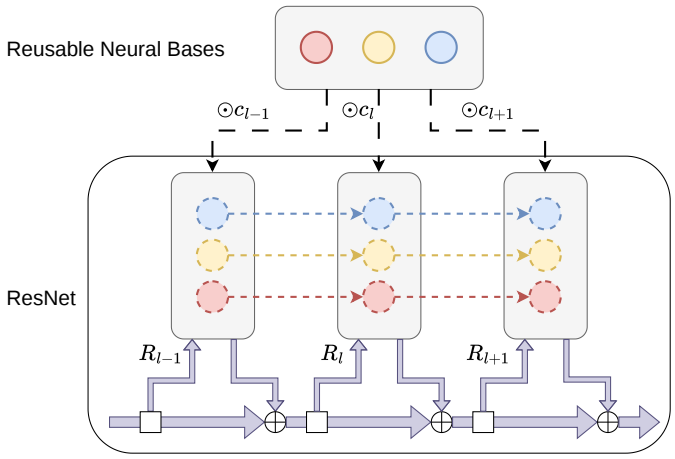


Figure 1: Illustration of the linear reusable neural bases architecture. Residual modules R_{l-1}, R_l, and R_{l+1} are constructed from a shared set of reusable neural bases (matching colors), with depth-specific coefficient vectors c_{l-1}, c_l, and c_{l+1} determining their respective contributions. Colored dashed arrows indicate neuron-reuse mechanism across depths. Purple arrows show feature flow through the ResNet. The symbols $\odot$ and $\oplus$ denote element-wise multiplication and residual addition, respectively.

2 Related Work

Our work relates to three complementary lines of research: recurrent architectures, structured and shared parameterizations, and parameter sharing in Transformers. We review these directions and then clarify how our approach differs in the granularity of reuse.

Recurrent Architectures. Recurrent neural networks (RNNs) provide a classical mechanism for parameter reuse by repeatedly applying shared transformations to evolving hidden states. Representative architectures include Elman networks, Jordan networks, long short-term memory networks (LSTMs), and gated recurrent units (GRUs) (Elman, 1990; Jordan, 1997; Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Such sharing reduces the number of independently learned parameters, but repeated composition can lead to vanishing or exploding gradients during backpropagation through time (Bengio et al., 1994; Hochreiter & Schmidhuber, 1997; Pascanu et al., 2013). LSTMs and GRUs mitigate these difficulties through gating, while other recurrent architectures explore orthogonal transformations (Helfrich et al., 2018) and convolutional structures with connections to recurrent

computation (Bai et al., 2019). These approaches motivate the joint consideration of parameter reuse and optimization stability.

Structured and Shared Parameterizations. Another line of research improves parameter efficiency by imposing structure on network weights or constructing them from shared components. Low-rank approaches compress recurrent transformations (Lu et al., 2016), while tensorized neural networks represent dense transformations using tensor-train decompositions (Novikov et al., 2015). Related methods employ structured matrices to reduce parameter storage (Sindhwani et al., 2015). Beyond factorization, HyperNetworks use auxiliary networks to generate the weights of a target network (Ha et al., 2016). Residual adapters combine shared representations with task-specific parameters (Rebuffi et al., 2017), whereas shared-template methods construct layer weights as learned linear combinations of common parameter templates (Savarese & Maire, 2019). Basis-based sharing (Wang et al., 2025) is also closely related to our approach, although it operates at the layer level rather than directly reusing neuron-level nonlinear features.

Parameter Sharing in Transformers. Parameter sharing has also been extensively explored in Transformer architectures (Vaswani et al., 2017). ALBERT shares parameters across encoder layers to reduce the number of independently learned weights (Lan et al., 2020). The Universal Transformer repeatedly applies shared self-attention and transition blocks across depth (Dehghani et al., 2019), and the Relaxed Recursive Transformer explores a relaxed form of recursive parameter sharing (Bae et al., 2025). The Sparse Universal Transformer combines depth-wise sharing with sparse mixture-of-experts layers (Tan et al., 2023). Sharing can also be restricted to particular components: Pires et al. (2023) investigate redundancy among feed-forward network (FFN) blocks and share a single wide FFN across multiple layers. Closely related looped Transformers repeatedly apply a fixed block, or a small set of blocks, to an evolving hidden representation (Giannou et al., 2023; Yang et al., 2024). These architectures connect depth-wise parameter reuse with iterative computation and the learning of algorithmic tasks.

Relation to Our Work. The LRNB architecture draws on recurrent computation and shared-basis parameterizations, but emphasizes reuse at the *neuron level*. Rather than sharing an entire transformation across depth or combining modules or sub-modules at the weight-matrix level, our approach constructs modules as learned linear combinations of shared nonlinear neural bases. The resulting architecture aims to improve parameter efficiency with flexibility across depth.

3 Background

Before presenting our main results, we briefly review several key concepts and formulations.

3.1 Deep learning

Deep learning theory suggests that the approximation ability of neural networks can be improved by increasing the depth of sequentially composed architectures (LeCun et al., 2015). By hierarchically composing nonlinear transformations, deep neural networks can represent certain functions more efficiently than the shallow networks (Raghu et al., 2017). Consequently, deep networks often exhibit better approximation ability than the shallow networks for complex learning tasks.

Although the sequentially composed architecture of deep neural networks enables models to learn more complex features and representations, it also introduces higher computational and memory costs during training or inference, compared to shallow networks (Rajbhandari et al., 2020). Since each module must be processed sequentially, a large number of parameters and intermediate activations must be retained in memory, even though only a small group of parameters are used in execution during training. This memory requirement grows with network depth, posing a significant burden for researchers with limited computational resources.

3.2 Residual Networks

Residual networks (ResNets) play an important role in deep learning by alleviating the vanishing-gradient problem and enabling the stable training of very deep networks (He et al., 2016). The key

feature of a ResNet is the use of shortcut connections, which allows the input information to be passed directly to deeper layers rather than propagating only through nonlinear layers. Chen et al. (2018) further provide a dynamical-systems perspective by interpreting residual updates as discretized flows of vector fields.

For example, a residual feedforward module in a ResNet can be written as

$$x + R(x),$$

where x represents the shortcut connection and $R(x)$ represents the residual component. A typical residual component takes the form

$$R(x) = P\sigma(Sx + b),$$

where $\sigma(\cdot)$ is an element-wise nonlinear activation function, and (S, b) are the learnable parameters of the input affine layer, and P are the learnable parameters of the output linear layer.[1]

From a more detailed perspective, the residual component can be viewed as a linear combination of a shared collection of nonlinear functions. The parameters (S, b) determine the directions and intercepts (offsets) of these nonlinear functions in the input space, while P specifies the coefficients used to combine them in the output space. We further extend this viewpoint in Section 4 to develop a new model architecture that reuses parameters to reduce memory costs.

4 METHODOLOGY

In this section, we introduce our new framework, the Linear Reusable Neural Bases (LRNB) Architecture, designed to improve parameter efficiency and reduce memory cost. The key idea is to reuse a shared group of neural bases while assigning different coefficients to construct modules at different positions, enabling the same neurons to be reused across depths. In the remainder of this section, we first define neural bases and provide a geometric interpretation from the perspective of vector fields. We then describe the model architecture, training strategy, and key implementation details.

4.1 THE NEURAL BASES

In this subsection, we first introduce our core concept of neural bases and provide a geometric interpretation from the perspective of vector fields.

Definition of Neural Bases. As introduced in Section 3, a residual feedforward block has the form

$$x + P\sigma(Sx + b).$$

Let m denote the hidden dimension of the residual block, and write the rows of S, the entries of b, and the columns of P as

$$S^\top = [s_1, s_2, \ldots, s_m], \qquad b^\top = [b_1, b_2, \ldots, b_m], \qquad P = [p_1, p_2, \ldots, p_m],$$

where $s_j^\top$ is the j-th row of S, b_j is the j-th entry of b, and p_j is the j-th column of P.

For each $j = 1, 2, \ldots, m$, we define the function $\phi_j(x)$ as:

$$\phi_j(x) := \sigma(\langle x, s_j \rangle + b_j)p_j.$$

Then the residual component of the network can be rewritten as a summation of such functions:

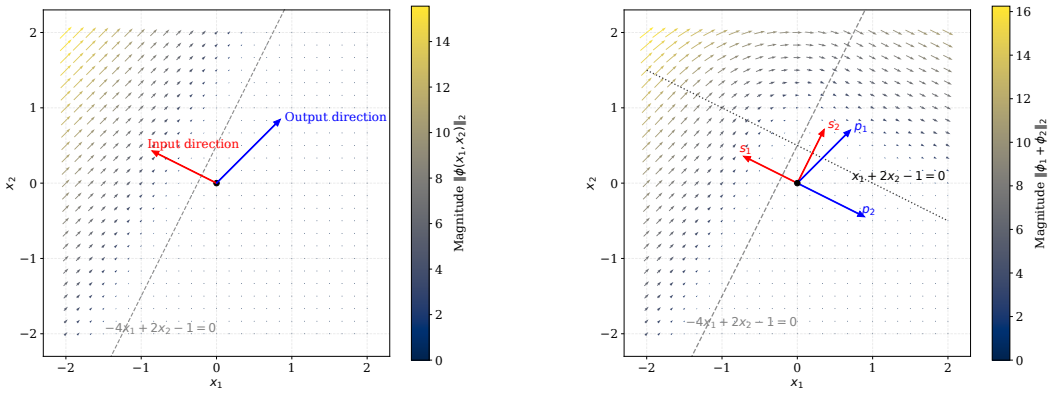
$$P\sigma(Sx + b) = \sum_{j=1}^{m} \sigma(\langle x, s_j \rangle + b_j)p_j = \sum_{j=1}^{m} \phi_j(x).$$

This formulation reveals that the nonlinear function of the residual block can be further decomposed into a group of parallel basic nonlinear functions $\{\phi_j\}_{j=1}^m$. Based on this decomposition, and since each of these functions serves as an *atomic* nonlinear transformation from the input to the output space—in the sense of being the simplest and independent building unit—we refer to each function ϕ_j as a *neuron* or a *neural basis*[2] in the network. Moreover, we refer to the associated parameters (s_j, b_j) for the input space as the *sensor* parameter of neuron ϕ_j and p_j as the *response* parameter of neuron ϕ_j.

[1] For simplicity, we omit the bias term in the output layer. This omission does not affect the nonlinear structure of the residual component, since the output bias contributes only a constant translation.

[2] We use "basis" informally to denote an elementary and independent building unit, analogous to sinusoidal bases in Fourier analysis and basis vectors in linear algebra. However, orthogonality is not assumed.

Superposition of Vector Fields. We provide a geometric interpretation of the neural basis. Specifically, each neural basis can be viewed as a vector field from the input space to the output space. For the j-th neuron, as illustrated in Figure 2a, the sensor parameters (s_j, b_j) determine the direction and bias of the input vector field, and the response vector p_j determines the direction of the push-forward in the output space. A bilinear structure is used to connect the nonlinear activation $\sigma(\langle x, s_j\rangle + b_j)$ with the response vector p_j. Consequently, a residual feedforward module can be interpreted as a *superposition* of many such vector fields, as illustrated in Figure 2b.



(a) Visualization of the vector field defined by a single neural basis.

(b) Visualization of a superposition of two vector fields induced by two neural bases.

Figure 2: Geometric interpretation of neural bases from the perspective of vector fields. **(a)** Visualization of a single neural basis. The function is defined as $\phi(x) = \sigma(\langle x, s\rangle + b)p$, where $s = (-4, 2)$, $b = -1$, $p = (1, 1)$, and σ is the GELU activation function (Hendrycks & Gimpel, 2023). The red and blue arrows illustrate the directions of sensor s and response p in the input and output vector fields, respectively. The dashed line marks the activation boundary induced by the equation $\langle x, s\rangle + b = 0$. The arrows of the vector field show the push-forward effect of the point on the grid. **(b)** A superposition of two vector fields, where the first neural basis is the same as that shown in **(a)** and the second basis has parameters $s_2 = (1, 2)$, $b_2 = -1$, and $p_2 = (2, -1)$. Note that a single basis can only introduce one directional push-forward effort, whereas the combination of two bases can generate multiple directional effects, as exhibited by the rotational dynamics in **(b)**.

Projection Neural Bases. Beyond residual feedforward modules, the concept of neural bases can be extended to other types of neural network modules, such as linear projection modules. For such modules, we introduce a *sensor–response duplex structure* that can parallelize the linear projection modules to form the corresponding *projection neural bases*. Due to space limitations, the detailed formulation is provided in Appendix A.

4.2 Linear Reusable Neural Bases Architecture

Figure 1 illustrates the overall architecture. We first introduce its two main components: reusable neural bases and a coefficient table. We then describe the implementation details and present a sub-neuron reuse technique to further improve parameter efficiency. Throughout this section, we consider a residual network comprising L stacked feedforward blocks.

Model Architecture. Essentially, our architecture is guided by two principles: *neuron reuse* and *linear combination*. These principles give rise to two main components:

1. **A collection of shared neural bases**, denoted by $\{\phi_j\}_{j=1}^{m}$, which serve as shared basic nonlinear functions for constructing residual blocks.
2. **A coefficient table**, denoted by $C = (c_{j,l}) \in \mathbb{R}^{m\times L}$, whose entries determine the contribution of each neural basis to each block.

By using the above components, the residual block at depth l can be constructed as

$$R_l(x) = \sum_{j=1}^{m} c_{j,l}\, \phi_j(x).$$

This formulation reuses a shared collection of neural bases across depths while allowing different blocks to represent distinct transformations through their depth-specific coefficients.

Optionally, we can also add a shift table $\Delta = (\delta_{j,l}) \in \mathbb{R}^{m\times L}$ for the neural bases to provide depth-specific shifts and further increase their flexibility. Here, we treat each coefficient $c_{j,l}$ as an independent, unconstrained real-valued parameter. Alternatively, constraints, such as non-negativity or binary values, may be imposed to control how neural bases contribute to each block, which may influence the topological connectivity of the network, such as sparse or dense structures. We leave the investigation of these constraints to future work.

Remark. The approaches of Savarese & Maire (2019); Wang et al. (2025) share a similar spirit with our framework by representing different layers as linear combinations of a shared set of bases with learned coefficients. However, there are two key distinctions. First, these approaches are primarily designed for CNN-based or attention-based architectures, whereas our framework focuses on FFN-based architectures. Second, they consider reusable transformations at the module or component level, while our framework introduces reuse at the finer-grained neuron level.

Algorithm 1 Forward pass across all depth blocks via reusable neural bases.

for $l \leftarrow 1$ **to** L **do**
  $a \leftarrow \sigma(Sx + b{+}\delta_l)$
  $a \leftarrow a \odot (Mc_l)$
  $x \leftarrow x + Pa$
end for

Algorithm 2 Average accumulated gradients of reusable neural bases.

for each shared parameter p **do**
  $r_p \leftarrow$ number of times p is reused
  Register a gradient hook on p:
  $\mathrm{grad} \mapsto \mathrm{grad}/r_p$
end for

Implementation Details. For initialization, the sensor parameters of the reusable neural bases are initialized using Kaiming initialization (He et al., 2015), while the response parameters are zero-initialized to provide a neutral starting point for stable training, following the guidance of Bachlechner et al. (2021); Zhang et al. (2019). To accelerate the optimization of the coefficient table, we parameterize the effective coefficients as Mc_l instead of c_l in the implementation, where M is a large constant (e.g., $M = 1000$ in our experiments). The parameters of the coefficient table are initialized from a Gaussian distribution $\mathcal{N}(0, 1/\sqrt{M})$. For simplicity, the diagram still uses the effective coefficients c_l.

Our training method mostly follows the standard gradient descent method. However, because the neural bases are reused across multiple blocks, their gradients are accumulated over all blocks for each optimization step. To maintain the magnitude of gradient updates consistent with standard models, we rescale the accumulated gradients of the neural bases by the number of times they are reused. The implementation details are summarized in Algorithm 1 and 2.

Sub-Neuron Reuse. Beyond neuron-level reuse, we further consider reusing *sub-neuron* components, such as sensors or responses, to improve parameter efficiency. One simple approach is to allow multiple neurons to share a single response vector, forming a more complex neuron unit with multiple sensors and a shared response, which we refer to as a *multi-sensor neuron*. Networks employing activation functions such as Maxout (Goodfellow et al., 2013) or SwiGLU (Shazeer, 2020) can be interpreted as examples constructed from such multi-sensor neurons. In addition to improving parameter efficiency, multi-sensor neurons often have larger activation regions, which may help mitigate the dead-neuron issue.

5 Discussion

The neuron-reuse mechanism introduces several new benefits to the model, particularly in terms of network compression and parameter count. We discuss these benefits below, followed by a brief discussion of the limitations.

Network Compression. One important aspect of the proposed architecture is network compression. Compared with classical architectures with independent parameters, the neuron-reuse mechanism allows the same neural bases to be reused across depths. This reuse may enable the network to represent recurrent or repeated structural patterns more intrinsically and effectively, leading to more concise representations. As suggested by the philosophy that compression leads to intelligence, representing a function with fewer parameters may also provide an implicit regularization effect. Therefore, from this perspective, the proposed architecture can offer improved generalization relative to classical architectures with fully independent parameters.

Parameter Count. In classical architectures with independent parameters, the total parameter count is mainly determined by three dimensions of the model: the embedding dimension, width (hidden dimension), and depth. Designing these models requires balancing these three dimensions under a memory budget. In contrast, our architecture reuses a shared collection of neurons across depths. As a result, the total parameter count is primarily determined by the embedding dimension and width, with less dependence on depth.[3] In practice, parameter count does not translate directly into memory usage, which also depends on implementation details and runtime requirements. Nevertheless, under the same parameter budget, our architecture enables the construction of substantially wider and deeper models.

Limitations. A primary limitation of our method is the increased computational cost compared with classical architectures, as each module must be constructed through a linear combination of neural bases. This additional computation may also lead to higher energy consumption.

6 Experiments

We evaluate the compression capability of the proposed architecture on two model types: FFN-based ResNets for regression and decoder-only Transformers for next-token prediction. Both models use GELU (Hendrycks & Gimpel, 2023) as the activation function. For fair comparisons, all models use the same random seed and identical mini-batch sampling order throughout training. All experiments are conducted on a Linux workstation equipped with an Intel Core i7 CPU, 16 GB of RAM, and an NVIDIA RTX 3090 GPU.

6.1 Evaluation of FFN-based ResNet models

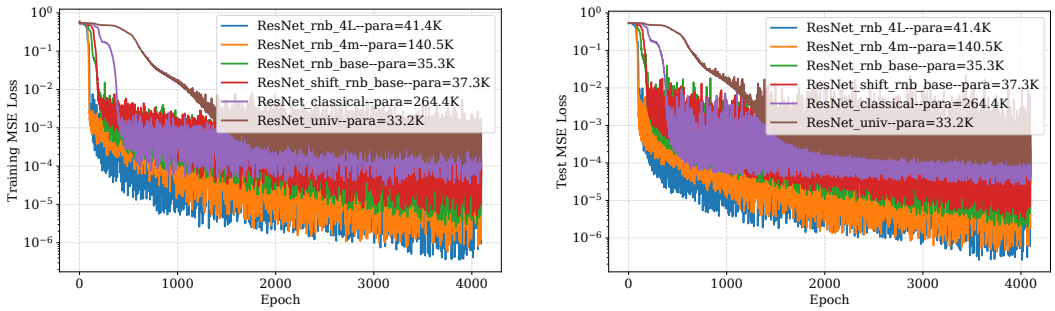
Dataset and Model Architecture. The regression task is conducted on a synthetic dataset generated by the composite function $f(x) = \cos(2\pi \cos(2\pi x))$. We sample $N = 60{,}000$ points randomly on the interval $[-1, 1]$ and split them into training and test sets using a 3:1 ratio.

The default architecture is an FFN-based residual network composed of a sequence of residual blocks, together with input and output linear projection modules. The three dimensions of the model are specified as follows: the embedding dimension is $d = 64$, the width is $m = 4d = 256$ (i.e., the hidden dimension or number of neurons), and the depth is $L = 8$, where the depth counts only the FFN-based residual blocks. To improve training stability, we use the parallelized linear projection modules introduced in Section A, with their corresponding response parameters initialized to zero.

Training Setup. We use mean squared error (MSE) as the loss function and AdamW as the optimizer, with a learning rate of 1×10^{-5}. All models are trained for $4{,}096$ epochs with a batch size of 512. We report the training and test MSE losses, average training time, and visual comparisons between the model predictions and the ground-truth function.

Results. Figure 3(a) shows that the proposed base architecture achieves a final loss comparable to that of the classical architecture while using only approximately $1/7$ of its parameters. It also outperforms the universal architecture with a similar parameter count, in which the same residual module is repeatedly applied across all depths.

[3] Although the coefficient table also introduces a parameter count that grows linearly with depth, this is negligible compared with that of fully independent modules.



(a) Training MSE loss over epochs.

(b) Test MSE loss over epochs.

Figure 3: Evaluation of FFN-based ResNet models on a regression task. **(a)** Training MSE loss over epochs. The proposed architectures achieve comparable or even lower final training loss than the classical ResNet and universal ResNet baselines, while using substantially fewer parameters. **(b)** Test MSE loss over epochs. The test-set results exhibit trends consistent with those observed on the training set. In particular, the proposed models achieve lower test loss than the baselines, while using fewer parameters, suggesting that the neuron-reuse mechanism can yield more compact and parameter-efficient representations.

Furthermore, increasing the width by a factor of 4, denoted by `4m`, leads to a lower final loss while using only about $1/2$ of the parameters of the classical architecture. Similarly, increasing the depth by a factor of 4, denoted by `4L`, also reduces the final loss while requiring only about $1/6$ of the parameters of the classical architecture. These results demonstrate that both width and depth expansion can improve model performance under a reduced parameter budget. The depth expansion can further improve parameter efficiency, although it may incur a higher computational cost and longer training time.

Figure 3(b) illustrates the generalization performance on the test set. The test loss exhibits trends consistent with those observed on the training set. In particular, the proposed models achieve lower test loss while using fewer parameters than the classical ResNet baseline, suggesting that the neuron-reuse mechanism can yield a more compact and parameter-efficient representation.

Due to space limits, results on training time and model predictions are provided in Appendix B.

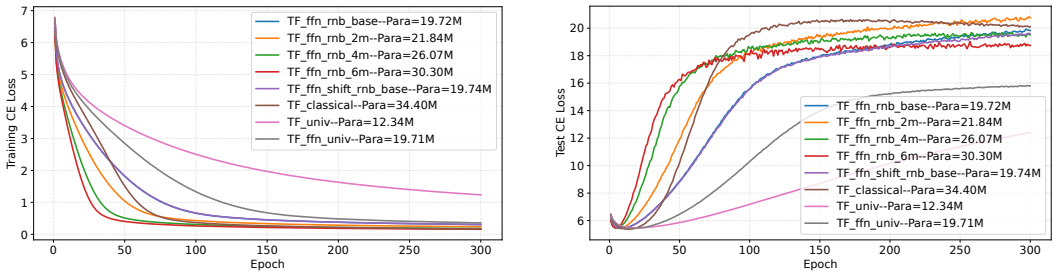
6.2 Evaluation of Decoder-Only Transformer Models

Dataset, Tokenization, and Model Architecture. We use the WikiText-2 dataset (Merity et al., 2016), which contains approximately two million tokens and preserves the original case, punctuation, and document structure, for the next-token prediction task. The text is first tokenized using a byte-level BPE tokenizer with a vocabulary size of $V = 2^{14} = 16{,}384$, including four special tokens: `UNK`, `PAD`, `EOS`, and `SOS`. We then concatenate all sentences into a single token sequence and randomly sample overlapping chunks of length $T = 128$.

For the model architecture, we use decoder-only Transformer models (Vaswani et al., 2017; Radford et al., 2019), including input and output parallelized linear projection blocks. The FFN-based modules share neural bases through the LRNB architecture. The model configuration is specified as follows: embedding dimension $d = 512$, depth $L = 8$, and $H = 8$ attention heads, corresponding to a head dimension of 64. The hidden dimension of each FFN block is $m = 4d = 2048$. No dropout is applied.

Model Variants and Baselines. The classical Transformer uses independently parameterized blocks at each depth. The fully shared baseline, denoted by `TF_univ`, reuses a single Transformer block across all depths. The FFN-shared baseline, denoted by `TF_ffn_univ`, reuses a single FFN module across all depths while retaining independently parameterized attention modules. The shifted LRNB variant, denoted by `TF_ffn_shift_rnb_base`, augments the base LRNB architecture with the learnable depth-specific shifts δ_l introduced in Section 4.

Training Setup. We use cross-entropy (CE) as the loss function and AdamW as the optimizer, with a learning rate of 1.5×10^{-5}. All models are trained for 300 epochs. Each epoch consists of $4,096$ training batches with a batch size of 16, while evaluation is performed using 128 test batches. We report the evolution of the training and test CE losses throughout training, together with the average training time and total parameter count.



(a) Training cross-entropy loss over epochs.

(b) Test cross-entropy loss over epochs.

Figure 4: Evaluation of decoder-only Transformer models on the next-token prediction task. **(a)** Training cross-entropy loss over epochs. Increasing the number of reusable neural bases from `2m` to `4m` and `6m` leads to lower final training loss. In particular, the proposed architecture with an expansion factor of 6 (`6m`) achieves a lower final training loss than the classical Transformer while using approximately 12% fewer parameters (30.30 million versus 34.40 million). **(b)** Test cross-entropy loss over epochs. The classical Transformer yields a higher test CE loss than most LRNB models, with the exception of `2m`. This observation is largely consistent with the philosophy that compression leads to intelligence—models with fewer parameters tend to generalize better on test data.

Results. Figure 4(a) shows the training cross-entropy loss over epochs. Increasing the number of reusable neural bases from `2m` to `4m` and `6m` leads to lower final training loss. In particular, the proposed architecture with an expansion factor of 6 (`6m`) outperforms the classical Transformer in final training loss while using approximately 12% fewer parameters (30.30 million versus 34.40 million). These results demonstrate that the proposed reuse mechanism can improve parameter efficiency while maintaining stable training dynamics.

Figure 4(b) shows that the classical Transformer yields a higher test CE loss than most LRNB models, with the exception of `2m`. This observation is consistent with the philosophy that compression is intelligence—the model with fewer parameters tends to achieve better generalization on test data.

Average training times are provided in Appendix C.

7 Conclusion

In this paper, we introduced the Linear Reusable Neural Bases (LRNB) architecture, a new framework for reducing parameter redundancy and memory costs. By representing each feedforward module as a linear combination of reusable neural bases, the LRNB architecture integrates a parameter-reuse mechanism at the neuron level and provides the flexibility to construct substantially wider and deeper architectures under a fixed parameter budget.

Experiments on FFN-based ResNets and decoder-only Transformers further demonstrate that the proposed framework can outperform classical models with fully independent parameters in terms of both training and test loss, while using fewer parameters and maintaining stable training.

More broadly, integrating neuron reuse, either throughout the network or within selected modules, can yield more compact and parameter-efficient representations and may also improve generalization compared with classical models that use fully independent parameters. These findings highlight neural-basis reuse as a promising direction for memory-efficient deep learning and motivate further investigation into more general reuse structures, coefficient constraints, pruning strategies, and large-scale architectures.

Ethics Statement

This work examines parameter sharing in neural networks and evaluates the approach on a synthetic dataset and a language modeling task based on the WikiText-2 dataset. It does not involve human-subject research or the collection of personal data. Models trained on existing text corpora may inherit biases or produce inaccurate content; the proposed architecture does not directly address these limitations.

Reproducibility Statement

The paper describes the neural-basis parameterization, forward computation, initialization, and gradient-rescaling procedure. The experimental sections specify the datasets, model dimensions, optimizer, learning rates, batch sizes, training duration, and hardware. Additional experimental results are provided in the appendices. Comparisons use a common random seed and identical mini-batch sampling order to control part of the experimental variation. These controls do not establish robustness across random seeds, which remains a limitation of the reported evaluation.

AI Use Statement

Generative AI tools, including ChatGPT, Claude, and DeepSeek, were used to assist with manuscript proofreading, wording revisions, LaTeX troubleshooting, and checking mathematical notation and consistency between experimental descriptions and figures. These tools were also used to improve clarity and readability, assist with manuscript organization, summarize and identify relevant literature, and aid in reference formatting and related writing tasks. We did not use generative AI to directly generate the experimental datasets used in this work. Translation and qualitative or thematic data analysis were not applicable to this study.

All AI-assisted content was reviewed by the authors. In particular, mathematical definitions and derivations were independently checked by the authors; AI-assisted code was reviewed, executed, and tested; numerical results were obtained from the authors' experimental implementations; and references and literature-related information were checked against the original sources where appropriate. Generative AI was not treated as an authoritative source for mathematical or scientific claims.

References

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: fast convergence at large depth. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pp. 1352–1361. PMLR, 27–30 Jul 2021. URL https://proceedings.mlr.press/v161/bachlechner21a.html. 6

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise loRA. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=WwpYSOkkCt. 3

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *International Conference on Learning Representations (ICLR)*, 2019. 3

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. 2

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of International*

Conference on Neural Information Processing Systems, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf. 1

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *International Conference on Neural Information Processing Systems*, pp. 6572–6583, Red Hook, NY, USA, 2018. Curran Associates Inc. 4

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014. URL https://api.semanticscholar.org/CorpusID:5590763. 2

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. 3

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213. 1, 2

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), January 2022. ISSN 1532-4435. 1

Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442, 2023. 3

Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. 6

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016. URL https://arxiv.org/abs/1609.09106. 3

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852. 6, 14

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90. 3

Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1969–1978. PMLR, 07–09 Jul 2018. 2

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415. 5, 7

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 2

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Neural Information Processing Systems*, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088. 1

Michael I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pp. 471–495. North-Holland, 1997. doi: https://doi.org/10.1016/S0166-4115(97)80111-2. URL https://www.sciencedirect.com/science/article/pii/S0166411597801112. 1, 2

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361. 1

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. 3

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. 3

Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5960–5964, 2016. 3

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843. 8

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209. 1

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks, 2015. URL https://arxiv.org/abs/1509.06569. 3

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. III–1310–III–1318. JMLR, 2013. 2

Telmo Pires, António Vilarinho Lopes, Yannick Assogba, and Hendra Setiawan. One wide feed-forward is all you need. In *Proceedings of the Eighth Conference on Machine Translation*, pp. 1031–1044, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.98. URL https://aclanthology.org/2023.wmt-1.98/. 3

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OPENAI, 2019. 8

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2847–2854. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/raghu17a.html. 3

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986. 3

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3

Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgYxn09Fm. 3, 6

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202. 6

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/851300ee84c2b80ed40f51ed26d866fc-Paper.pdf. 3

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer, 2023. URL https://arxiv.org/abs/2310.07096. 3

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. 3, 8

Jingcun Wang, Yu-Guang Chen, Ing-Chao Lin, Bing Li, and Grace Li Zhang. Basis sharing: Cross-layer parameter sharing for large language model compression. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gp32jvUquq. 3, 6

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. 3

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization, 2019. URL https://arxiv.org/abs/1901.09321. 6

A PARALLELIZED PROJECTION MODULES AND PROJECTION NEURAL BASES

In this section, we provide a more detailed formulation of *parallelized projection modules* and *projection neural bases*, together with their decomposition and geometric interpretation.

Sensor–Response Duplex Structure. To parallelize linear projection modules, we represent each projection as the sum of a fixed linear projection and a *sensor–response duplex structure*, with the identity function serving as the activation function. Since both the fixed branch and the duplex branch are affine, their sum is still an affine transformation.

Specifically, we adopt the same sensor–response notation used for the FFN-based ResNet. Let (S, b) denote the sensor parameters and P the response parameters. The linear projection is written as

$$\mathrm{Proj}(x) + P(Sx + b),$$

where $\mathrm{Proj}(x)$ represents the linear projection layer from input space to output space with frozen parameters, using the Kaiming initialization (He et al., 2015).

Let m denote the hidden dimension of the block, and write the rows of S, the entries of b, and the columns of P as

$$S^\top = [s_1, s_2, \ldots, s_m], \qquad b^\top = [b_1, b_2, \ldots, b_m], \qquad P = [p_1, p_2, \ldots, p_m],$$

where $s_j^\top$ corresponds to the j-th row of S, b_j is the j-th entry of b, and p_j is the j-th column of P.

For each $j = 1, 2, \ldots, m$, we define

$$\phi_j(x) := (\langle x, s_j \rangle + b_j) p_j,$$

where the identity function replaces the nonlinear activation function σ for the linear setting.

Decomposition. The duplex structure can then be decomposed as a sum of these elementary functions:

$$P(Sx + b) = \sum_{j=1}^{m} (\langle x, s_j \rangle + b_j) p_j = \sum_{j=1}^{m} \phi_j(x).$$

This formulation shows that the duplex structure of the projection block can be expressed as a superposition of elementary functions $\{\phi_j\}_{j=1}^m$. Based on this decomposition, we refer to each ϕ_j as a *projection neuron*, or equivalently a *projection neural basis*. We similarly refer to (s_j, b_j) as the *sensor* of ϕ_j and p_j as its *response*.

A useful interpretation is that each projection neural basis corresponds to an affine transformation whose linear component has rank at most one. Indeed,

$$\phi_j(x) = (\langle x, s_j \rangle + b_j) p_j = p_j s_j^\top x + p_j b_j,$$

where $p_j s_j^\top$ is a matrix of rank at most one and $p_j b_j$ is the corresponding bias vector. Thus, the trainable duplex component can be viewed as a superposition of multiple low-rank affine transformations.

Geometric Interpretation. We further provide a geometric interpretation of projection neural bases from the perspective of vector fields. We use the same parameter values as in the earlier vector-field visualizations shown in Figure 2.

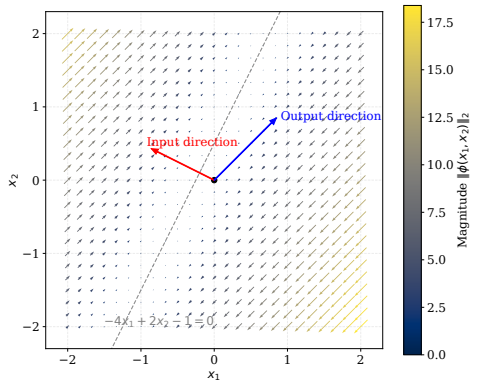
For the j-th projection neuron, as illustrated in Figure 5a, the sensor vector s_j and scalar bias b_j determine how the input position x is mapped to the scalar response

$$\langle x, s_j \rangle + b_j,$$

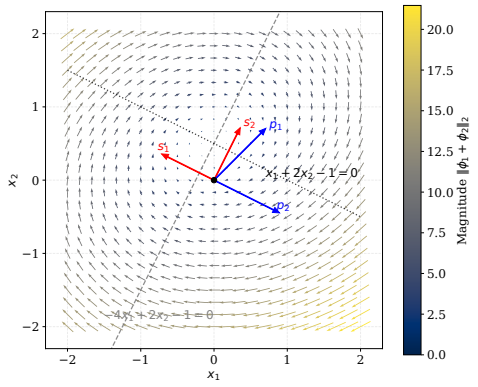
while the vector p_j determines the direction of the resulting output vector. Their interaction therefore produces a vector field of the form

$$\phi_j(x) = (\langle x, s_j \rangle + b_j) p_j.$$

Consequently, the trainable duplex component can be interpreted as a superposition of such vector fields, as illustrated in Figure 5b.



(a) Vector field induced by a single projection neural basis.

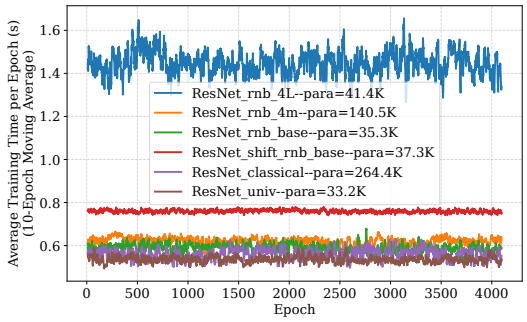


(b) Vector field induced by the superposition of two projection neural bases.

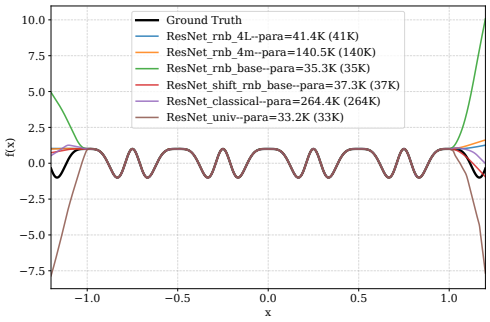
Figure 5: Geometric interpretation of projection neural bases as vector fields. **(a)** A single projection neural basis $\phi(x) = (\langle x, s\rangle + b)p$, with $s = (-4, 2)$, $b = -1$, and $p = (1, 1)$. The red and blue arrows illustrate the sensor and response directions, respectively, while the dashed line indicates the zero line $\langle x, s\rangle + b = 0$. The grid arrows show the output vectors induced at different input locations. **(b)** Superposition of two projection neural bases. The first basis is identical to that in **(a)**, while the second is defined by $s_2 = (1, 2)$, $b_2 = -1$, and $p_2 = (2, -1)$. While a single basis generates vectors along a fixed response direction, the superposition of multiple bases can produce more vector-field structures, such as rotational patterns.

B Additional Experimental Results of FFN-Based ResNets

Results. Figure 6(a) shows the smoothed training time over epochs. The proposed architecture introduces an additional training-time overhead because its residual blocks are dynamically constructed from reusable neural bases and depth-dependent coefficients. Figure 6(b) compares the model predictions with the ground-truth function. All models provide accurate approximations to the target function within $[-1, 1]$.



(a) Smoothed training time over epochs.



(b) Model predictions compared with the ground-truth function over $[-1.2, 1.2]$.

Figure 6: Additional evaluation of the FFN-based ResNet models on the nonlinear regression task. **(a)** Smoothed training time over epochs. The proposed architecture introduces a modest computational overhead due to the dynamic construction of residual blocks from reusable neural bases and depth-dependent coefficients. **(b)** Model predictions compared with the ground-truth function over $[-1.2, 1.2]$. Training and test samples are drawn from $[-1, 1]$; predictions outside this interval illustrate extrapolation. All evaluated models closely approximate the target function over $[-1, 1]$.

C Additional Experimental Results of Transformer models

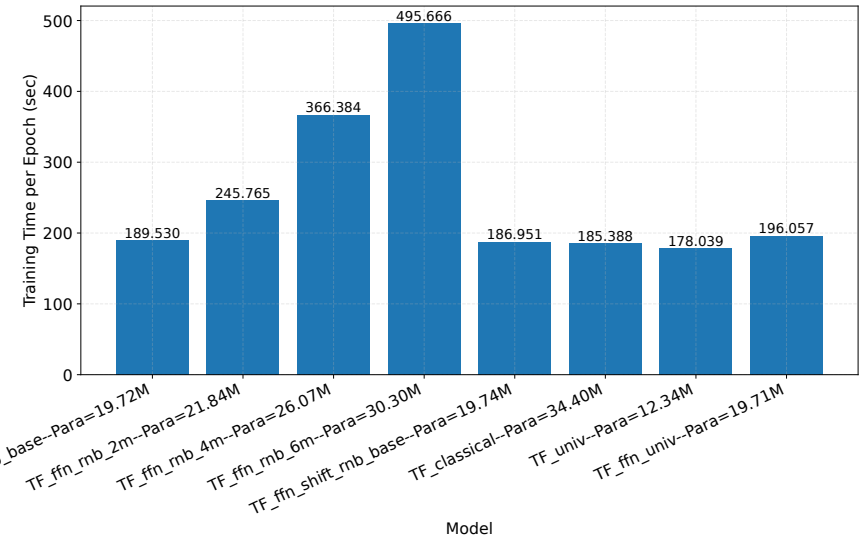


Figure 7: Average training time per epoch. The LRNB base model has a training time per epoch close to that of the classical baseline (189.5 s versus 185.4 s). Increasing the reusable FFN width raises training time, reaching 495.7 s for the 6m variant. These measurements illustrate the parameter–computation trade-off; inference latency and the hardware causes of the overhead were not evaluated here.

Results. Figure 7 shows the average training time per epoch. The LRNB base model has a training time per epoch close to that of the classical baseline (189.5 s versus 185.4 s). Increasing the reusable FFN width raises training time, reaching 495.7 s for the 6m variant. These measurements illustrate the parameter–computation trade-off; inference latency and the hardware causes of the overhead were not evaluated here.