HIGHLY EXPRESSIVE ACTIVATION FUNCTIONS BASED ON LEARNABLE SOFT QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

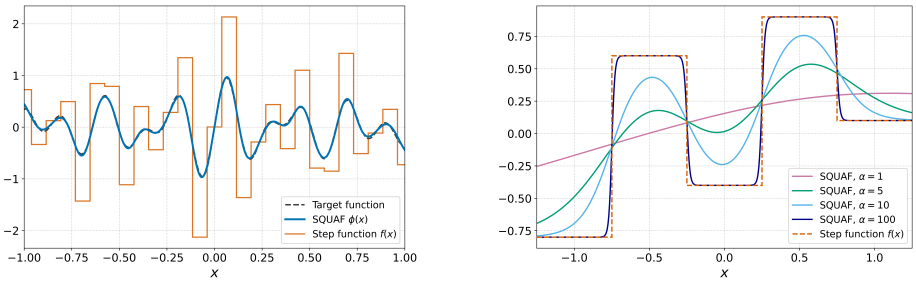
Activation functions (AFs) provide the crucial nonlinearity that underpins the expressiveness of deep neural networks (DNNs). However, widely used AFs such as ReLU and GELU are simple and non-adaptive, limiting the flexibility of DNNs to capture intricate structures in data. In this paper, we introduce Soft Quantization Activation Functions (SQUAFs), a family of trainable, highly expressive AFs constructed as differentiable relaxations of generalized scalar quantizers. We show theoretically that SQUAFs can approximate any continuous 1D function defined over a closed interval with arbitrary precision. Extensive evaluations on standard image classification benchmarks show that SQUAF yields consistent gains over strong fixed and trainable AFs. Moreover, the advantage of SQUAFs is especially pronounced when a task involves modeling complex signals. For partial differential equation (PDE) solving, Physics-Informed Neural Networks (PINNs) equipped with SQUAFs attain the lowest relative error across nine out of ten PDE problems compared to strong baseline AFs. For implicit neural representations (INRs), SQUAF also consistently improves upon existing INR methods across sixteen 2D image representation cases and four 3D shape representation cases, preserving fine-scale details in the fitted signals. Additionally, we show, through knowledge distillation and model depth reduction experiments, that SQUAFs enable smaller models to outperform larger ones trained with conventional AFs, suggesting that increased AF expressiveness can compensate for reduced network size, thus achieving model compression. These results establish SQUAF as a general-purpose, parameter-efficient AF with particular strength in modeling complex signals.

1 INTRODUCTION

Deep neural networks (DNNs) derive their modeling power from composing linear transformations with nonlinear operations. Although nonlinearity may also arise from architecture-specific operations, such as attention (Vaswani et al., 2017), normalization (Ioffe & Szegedy, 2015; Ba et al., 2016), and max pooling, activation functions (AFs) are ubiquitous across all major DNN architectural families, including multilayer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, serving as the primary architecture-agnostic mechanism through which nonlinearity is introduced. As a result, AFs underpin the expressiveness of DNNs and the success of deep learning (Hornik et al., 1989).

Despite their critical role, most established DNN architectures rely on simple, fixed AFs, including the Rectified Linear Unit (ReLU) (Nair & Hinton, 2010), Leaky ReLU (LReLU) (Maas et al., 2013), the Gaussian Error Linear Unit (GELU) (Hendrycks & Gimpel, 2016), and SiLU (Elfwing et al., 2018). While computationally efficient, their limited and static forms of nonlinearity constrain the expressiveness attainable by a network of fixed architecture and size. With fixed AFs, increasing a DNN's nonlinear modeling capacity then generally relies on adding more trainable parameters to linear modules by expanding the depth and/or width of the underlying DNN architecture. Trainable AFs offer a complementary approach: PReLU (He et al., 2015), Swish (Ramachandran et al., 2017), and CRReLU (Sun et al., 2025) introduce a small number of adaptive parameters to AFs, while recent methods such as PolyReLU, PolyNorm (Zhuo et al., 2025) and DiTAC (Chelly et al., 2024) leverage more complex parameterizations and compositional forms. Nevertheless, most existing

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.



(a) (b)

Figure 1: (a) Illustration of a SQUAF fitted to a 1D function, and its underlying step function. (b) Fixing $(\boldsymbol{y}, \boldsymbol{z})$, as α increases, the SQUAF in Example 3 approaches its underlying step function.

trainable AFs remain confined to predefined functional families, limiting the nonlinear responses they can learn and, consequently, the effective expressiveness of the resulting network.

This limitation becomes especially consequential when DNNs are used to model complex signals. For example, standard physics-informed neural networks (PINNs) (Raissi et al., 2019a) face challenges in handling partial differential equations (PDEs) involving complex geometries, multi-scale phenomena, and nonlinear behaviors (Hao et al., 2024). Similarly, coordinate-based MLPs with conventional AFs have difficulty capturing high-frequency components in natural images and 3D objects, a phenomenon known as spectral bias, which limits their effectiveness as implicit neural representations (INRs) (Tancik et al., 2020; Sitzmann et al., 2020). These tasks, i.e., PDE solving and implicit neural representation, further highlight the need for more flexible AFs that can enhance the capability of DNNs to model complex, high-frequency, and highly nonlinear signals.

To this end, we introduce Soft Quantization Activation Functions (SQUAFs), a family of highly expressive AFs based on learnable soft quantization. SQUAFs replace the hard interval assignments within generalized scalar quantizers, i.e., step functions, with probabilistic assignments, yielding soft and differentiable relaxations of step functions. Both the partitioning of the input domain and the output amplitudes associated with the resulting intervals are controlled by trainable parameters. This flexible construction allows SQUAFs to include a rich variety of functional forms. In fact, we show theoretically that SQUAFs can approximate any continuous 1D function defined over a closed interval with arbitrary precision. SQUAFs can therefore overcome the limitations of conventional AFs and endow DNNs with increased expressiveness.

We evaluate SQUAF with extensive experiments across various model architectures, learning tasks and datasets. On standard image classification benchmarks, SQUAF consistently improves the performance of a variety of CNNs and transformer-based models, demonstrating its effectiveness as an architecture-agnostic drop-in AF. Unlike most conventional AFs which perform decently on standard learning tasks but struggle to model complex signals, SQUAF extends its effectiveness to those challenging tasks. SQUAF-equipped PINNs attain the lowest relative error on nine out of ten representative PDE problems, and SQUAF consistently improves over strong INR methods across sixteen 2D images and four 3D shapes. More interestingly, SQUAFs are parameter-efficient: in all of the experiments above, the additional number of trainable parameters required for SQUAFs is less than 0.13%, and can be as low as 0.0003% of the original model parameter count. Furthermore, we show that SQUAFs can actually facilitate model compression. Under knowledge distillation (KD) (Hinton et al., 2015), it is demonstrated that a 3.2$\times$ smaller distilled student with SQUAFs outperforms its teacher with conventional AFs. Even when trained from scratch without distillation, an 8-block vision transformer (ViT) equipped with SQUAFs outperforms a standard 12-block ViT using GELU. Together, our findings position SQUAF as a general-purpose, parameter-efficient AF with particular strength in modeling complex signals. Our main contributions are summarized as follows:

- We propose SQUAFs, a novel family of trainable, highly expressive AFs.
- We provide theoretical results showing that SQUAFs can approximate any continuous 1D function defined over a closed interval with arbitrary precision, and derive their closed-form partial derivatives with respect to both inputs and parameters for efficient training.
- We demonstrate the effectiveness of SQUAF on standard image classification benchmarks across diverse DNN architectures.

- We show that SQUAF is particularly effective for complex signal modeling tasks, such as PDE solving and implicit neural representation.
- We show additional benefits of SQUAFs in model compression.

2 Background and Related Work

A comprehensive review of related work on AFs is provided in Appendix A.1, where we carefully position our work among the relevant literature. In the following, we briefly discuss some background information to facilitate understanding of our proposed method.

2.1 Scalar Quantizers

A scalar quantizer is a function Q that maps each $x \in \mathbb{R}$ into a finite reproduction alphabet

$$\hat{\mathcal{A}} = \{y_1, y_2, \cdots, y_L\} \subset \mathbb{R}.$$

When $Q(x) = y_i$, x is said to be quantized to y_i. The quantizer Q is said to be uniform if

$$\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}$$

and each x is quantized to the nearest point in $\hat{\mathcal{A}}$, where k is a positive integer and q is a positive real number called the quantization step size.

2.2 Probabilistic and Soft Deterministic Quantizers

Let $P(\cdot|x)$ be a conditional probability mass function (CPMF) over $\hat{\mathcal{A}}$ or equivalently its index set $\mathcal{A} = \{1, 2, \cdots, L\}$ (or $\{0, \pm 1, \pm 2, \ldots, \pm k\}$ in the case of uniform quantization), given $x \in \mathbb{R}$. That is, for each $x \in \mathbb{R}$, $P(y_i|x) \geq 0, \forall y_i \in \hat{\mathcal{A}}$ and $\sum_{i=1}^{L} P(y_i|x) = 1$. With $P(\cdot|x)$, each x can be quantized randomly to $y_i \in \hat{\mathcal{A}}$ with probability $P(y_i|x)$. Denote this random mapping by $Q_p(x)$, which is a probabilistic quantizer.

Given x, $Q_p(x)$ is a random variable taking values in $\hat{\mathcal{A}}$. Consider the expectation of $Q_p(x)$:

$$Q_d(x) = \mathbb{E}[Q_p(x)] = \sum_{i=1}^{L} y_i \cdot P(y_i|x). \tag{1}$$

$Q_d(x)$ is the soft deterministic quantizer corresponding to $Q_p(x)$. $Q_d(x)$ and $Q_p(x)$ were introduced by Yang & Hamidi (2025) to formulate coded deep learning.

Many types of CPMFs can be used for $Q_d(x)$ and $Q_p(x)$. Let $\alpha > 0$ be a trainable parameter. The following are two well-known examples.

Example 1: For any x,

$$P(y_i|x) = \frac{e^{-\alpha(x-y_i)^2}}{\sum_{j=1}^{L} e^{-\alpha(x-y_j)^2}}, \tag{2}$$

which is the discrete counterpart of a conditional Gaussian distribution.

Example 2: For any x,

$$P(y_i|x) = \frac{e^{-\alpha|x-y_i|}}{\sum_{j=1}^{L} e^{-\alpha|x-y_j|}}, \tag{3}$$

which is the discrete counterpart of a conditional Laplacian distribution.

2.3 Step Functions

Let $\{C_i\}_{i=1}^{L}$ be an interval partition of the whole real line $\mathbb{R}$. That is, each C_i is an interval; they are pairwise disjoint, $C_i \cap C_j = \emptyset, \forall i \neq j$; and together they cover the whole real line, $\cup_{i=1}^{L} C_i = \mathbb{R}$. A step function $f(x)$ can be written as

$$f(x) = \sum_{i=1}^{L} z_i \mathcal{X}_{C_i}(x), \tag{4}$$

where $z_i \in \mathbb{R}$ is the amplitude value corresponding to C_i, and $\mathcal{X}_C(x)$ is the indicator function of the set C, which is equal to 1 if $x \in C$ and 0 otherwise.

Compared with scalar quantizers, the step function $f(x)$ in equation 4 can also be regarded as a generalized scalar quantizer which maps x to z_i if and only if $x \in C_i$. The difference between the step function $f(x)$ and the scalar quantizer Q lies in that x and $Q(x) = y_i$ generally belong to the same partition interval $\{y : Q(y) = y_i\}$, whereas x and $f(x) = z_i$ have no such constraint.

3 SQUAFs

Inspired by probabilistic and soft deterministic quantizers, we now introduce general CPMFs into step functions, yielding our proposed soft quantization activation functions.

3.1 Definitions

Recall that $\hat{\mathcal{A}} = \{y_1, y_2, \cdots, y_L\} \subset \mathbb{R}$. For convenience, we also regard $\hat{\mathcal{A}}$ as a vector of dimension L denoted by $\boldsymbol{y} := [y_i]_{i=1}^L$. Likewise, write $\boldsymbol{z} := [z_i]_{i=1}^L$. Based on $\boldsymbol{y}$ and a general CPMF $P(\cdot|x)$ over $\hat{\mathcal{A}}$, we replace the indicator functions $\mathcal{X}_{C_i}(x)$ in equation 4 with $P(y_i|x)$, resulting in a soft quantization activation function (SQUAF)

$$\phi(x) = \sum_{i=1}^{L} z_i \cdot P(y_i|x). \tag{5}$$

Example 3: When the CPMF $P(\cdot|x)$ is the discrete conditional Gaussian distribution in Example 1, $\phi(x)$ can be conveniently expressed as

$$\phi(x) = \phi(x|\boldsymbol{y}, \boldsymbol{z}, \alpha) = \sum_{i=1}^{L} z_i \frac{e^{-\alpha(x-y_i)^2}}{\sum_{j=1}^{L} e^{-\alpha(x-y_j)^2}} \tag{6}$$

in the general case, and as

$$\phi(x) = \phi(x|q, \boldsymbol{z}, \alpha) = \sum_{i=-k}^{k} z_i \frac{e^{-\alpha(x-iq)^2}}{\sum_{j=-k}^{k} e^{-\alpha(x-jq)^2}} \tag{7}$$

when $\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}$.

Remark. **(1)** $\phi(x)$ in Example 3 is differentiable with respect to $(x, \boldsymbol{y}, \boldsymbol{z}, \alpha)$ or $(x, q, \boldsymbol{z}, \alpha)$ as the case may be. The analytical partial derivatives are presented in Appendix A.2. **(2)** The role of $\boldsymbol{z}$ is to approximate the amplitude of a desirable nonlinear activation function, while the role of $\boldsymbol{y}$ or q in Example 3 is to partition the domain of the desirable nonlinear activation function. **(3)** $(\boldsymbol{y}, \boldsymbol{z}, \alpha)$ or $(q, \boldsymbol{z}, \alpha)$ in Example 3 are trainable parameters, and α can be regarded as a horizontal scaling factor. **(4)** The relationship between $\phi(x)$ in Example 3 and $f(x)$ is analogous to that between $Q_d(x)$ and $Q(x)$. Given $(\boldsymbol{y}, \boldsymbol{z})$ or $(q, \boldsymbol{z})$, if we determine $\{C_i\}_{i=1}^L$ through the nearest neighbor quantization of x w.r.t. $\hat{\mathcal{A}}$, we obtain the underlying step function $f(x)$ (given in equation 4) of $\phi(x)$. As shown in Fig. 1(a), $\phi(x)$ is a soft and differentiable relaxation of its underlying step function $f(x)$. **(5)** Given fixed $(\boldsymbol{y}, \boldsymbol{z})$ or $(q, \boldsymbol{z})$, as $\alpha \to \infty$, $\phi(x)$ in Example 3 approaches its underlying step function, as shown in Fig. 1(b). **(6)** For readers who are not familiar with quantization, but familiar with the attention operation used in transformers (Vaswani et al., 2017), $\phi(x)$ in Example 3 can be regarded as an attention operation in a broad sense, with the query being x, the key being $\boldsymbol{y}$, the value being $\boldsymbol{z}$, and the similarity metric between the query and key being negative squared distance instead of dot product.

3.2 Universal Approximation

As illustrated in Example 3, let us introduce a horizontal scaling factor α into the general CPMF $P(\cdot|x)$. Write $P(\cdot|x)$ as $P_\alpha(\cdot|x)$ and $\phi(x)$ in equation 5 as $\phi(x|\boldsymbol{y}, \boldsymbol{z}, \alpha)$ to explicitly show dependence on α. Assume $L \geq 2$. For each x, let $y_{i(x)}$ and $y_{j(x)}$ be the two nearest points of x in $\hat{\mathcal{A}}$.

In order for $\phi(x|\boldsymbol{y}, \boldsymbol{z}, \alpha)$ to facilitate the learning process during DNN training and be a universal approximator for 1D functions, we further assume that the CPMF $P_\alpha(\cdot|x)$ over $\hat{\mathcal{A}}$ satisfies the following properties:

- **P1** $P_\alpha(\boldsymbol{y}|x) = [P_\alpha(y_1|x), P_\alpha(y_2|x), \cdots, P_\alpha(y_L|x)]$, as a function of $\boldsymbol{y}$ and x, is differentiable almost everywhere.
- **P2** As α goes to ∞, $\sup_x \min_{0\le\lambda\le1} |\phi(x|\boldsymbol{y}, \boldsymbol{z}, \alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)}| \to 0$.

Property P2 is a localization property. It implies that when α is large, $\phi(x)$ depends mainly on $z_{i(x)}$ and $z_{j(x)}$. In other words, each amplitude value in $\boldsymbol{z}$ has mainly a local effect on the shape of $\phi(x)$, which is advantageous during optimization of $\boldsymbol{z}$ in the training process. With Properties P1 and P2, $\phi(x)$ can approximate any continuous function on a closed interval with arbitrary precision. We formalize this in the following theorem, which is proved in Appendix A.3.

Theorem 1 (Universal approximation)**.** *Assume that the CPMF $P_\alpha(\cdot|x)$ has Properties P1 and P2. Let g be any continuous function defined on a closed interval $[a, b]$. For every $\varepsilon > 0$, there exist $L \in \mathbb{N}$, and $(\boldsymbol{y}, \boldsymbol{z}, \alpha)$ such that the SQUAF $\phi(x)$ as in equation 5 satisfies $\sup_{x\in[a,b]} |\phi(x) - g(x)| \le \varepsilon$.*

Many types of CPMFs satisfy Properties P1 and P2. In particular, the discrete conditional Gaussian and Laplacian distributions in Examples 1 and 2 have Properties P1 and P2, as shown in the following theorem, which is proved in Appendix A.3 as well.

Theorem 2. *If $P_\alpha(\cdot|x)$ is given by equation 2 or equation 3, then it satisfies Properties P1 and P2.*

3.3 Integration of SQUAFs to DNNs

We consider conventional DNN architectures with T hidden transforms separated by $T-1$ AFs. A broad class of DNNs, including MLPs, CNNs, and transformers, can be expressed as

$$f_\Theta(\boldsymbol{u}) = (h_T \circ \varphi_{T-1} \circ h_{T-1} \circ \cdots \circ \varphi_1 \circ h_1)(\boldsymbol{u}), \tag{8}$$

where $\boldsymbol{u}$ denotes the input, Θ denotes all trainable parameters in the network, $\{\varphi_1, \ldots, \varphi_{T-1}\}$ are AFs, and $\{h_1, \ldots, h_T\}$ are hidden transforms that may include fully connected layers, convolutional layers, attention layers, normalization layers, etc.

We obtain a SQUAF-DNN by setting each AF φ_t to a SQUAF:

$$\varphi_t(\boldsymbol{x}_t) = \phi(\boldsymbol{x}_t|\boldsymbol{y}_t, \boldsymbol{z}_t, \alpha_t), \;\; t = 1, 2, \ldots, T-1, \tag{9}$$

where the scalar function ϕ is applied to pre-activations $\boldsymbol{x}_t$ in an element-wise manner. I.e., the same SQUAF instance with its parameters $\boldsymbol{y}_t := [y_i^t]_{i=1}^L$, $\boldsymbol{z}_t := [z_i^t]_{i=1}^L$, and α_t is shared across all elements of $\boldsymbol{x}_t$. For a uniform SQUAF construction such as equation 7, $\boldsymbol{y}_t$ is further replaced by q_t. In general, these SQUAF parameters are jointly optimized with other DNN parameters W in $\{h_t\}_{t=1}^T$ using the original task loss, although some of them can also be frozen during training. Since the number of SQUAF parameters depends only on L and T, but not the widths of the hidden transforms, only a negligible number of trainable parameters are added to the original DNN.

4 Experiments

Throughout this section, SQUAF refers to $\phi(x)$ in the form of equation 7.

4.1 Function Fitting

We consider fitting two sinusoidal functions of increasing complexity. The first is a 1D function $g(x) = 0.4\sin(19x) + 0.2\sin(23x) + 0.3\sin(29x) + 0.1\sin(31x)$ within domain $[-1, 1]$, and the second is a 2D function $g(x_1, x_2) = 0.4\sin(9x_1 - 7x_2) + 0.1\sin(-9x_1 + 11x_2) + 0.15\sin(3x_1 + 13x_2) + 0.15\sin(9x_1 + 9x_2) + 0.1\sin(13x_1 + 5x_2) + 0.1\sin(3x_1 + 19x_2)$ within domain $[-1, 1]^2$. We use a $[1, 64, 1]$ MLP (i.e., a 3-layer MLP with one input neuron, 64 hidden neurons, and one output neuron) for the 1D case, and a $[2, 50, 1]$ MLP for the 2D case, both equipped with SQUAF or a compared AF. For SQUAF, we set $k = 2$. Following Chelly et al. (2024), all models are trained for 40K iterations using Adam (Kingma & Ba, 2014). More implementation details are provided in Appendix A.4.

Table 1: Results for function fitting. We report the number of trainable parameters, MSE, and R^2 score for each model. In the last two rows, we report results for 1D function fitting solely based on SQUAFs. We highlight the best results in bold, and the second best results with underscores.

AF	1D Function			2D Function		
	Param.	MSE↓	R^2 ↑	Param.	MSE↓	R^2 ↑
ReLU	193	1.0E-2	93.3	201	2.3E-2	88.9
GELU	193	1.1E-1	9.7	201	5.4E-2	65.3
LReLU	193	8.0E-3	94.7	201	1.1E-2	93.1
PReLU	194	1.0E-2	93.1	202	8.0E-3	96.1
SiLU	193	1.2E-1	2.2	201	1.1E-1	11.4
Swish	194	4.0E-3	97.5	202	1.3E-1	25.5
CRReLU	194	9.7E-3	93.7	202	1.3E-2	90.4
PolyReLU	197	7.1E-3	95.1	205	8.1E-3	93.4
PolyNorm	197	6.4E-2	56.9	205	7.0E-2	39.9
DiTAC	202	<u>1.0E-3</u>	<u>99.3</u>	210	<u>4.0E-3</u>	<u>98.4</u>
KAF	200	1.2E-1	1.9	208	1.1E-1	6.4
SQUAF	200	**4.0E-5**	**100.0**	208	**4.0E-4**	**99.7**
SQUAF-only	192	**2.0E-7**	**100.0**	n/a	n/a	n/a
SQUAF-only	**28**	1.0E-3	99.3	n/a	n/a	n/a

As shown in Tab. 1, SQUAF achieves the best performance with a comparable number of trainable parameters, reducing mean squared error (MSE) by 1∼4 orders of magnitude and reaching almost perfect R^2 scores. Note that KAF (Scardapane et al., 2019) is originally element-dependent; i.e., each hidden neuron is connected to a KAF with distinct parameterization. Here, we make it element-independent as a conventional AF for a fair comparison with comparable parameter count, under which it performs generally the worst. We therefore exclude KAF from subsequent comparisons. Additionally, we fit the 1D function using a SQUAF alone, without any linear layers, as a direct empirical demonstration of Theorem 1. With 192 trainable parameters, the SQUAF-only model further reduces the MSE of SQUAF-MLP by 200×; with only 28 parameters, it matches DiTAC while using 7.2× fewer parameters. The latter is visualized in Fig. 1(a). This SQUAF-only fitting procedure is also used for SQUAF initialization in subsequent experiments, with details provided in Appendix A.4.2. Model sizes are reported in Tab. 1 because AF parameters are non-negligible for these small MLPs, but omitted thereafter because the base DNNs are much larger and SQUAF adds at most 0.13% parameters; detailed counts are provided in Appendix A.9.

4.2 Image Classification

CIFAR-100. We perform experiments across 6 model architectures, namely ShuffleNetV1 (Zhang et al., 2018), VGG-8, VGG-13 (Simonyan & Zisserman, 2014), ResNet-56 (He et al., 2016), ViT-T (Touvron et al., 2021a) and EfficientFormer-L1 (Li et al., 2022), with the first 4 being CNNs and the last 2 being transformer-based networks. For all experiments on CNNs, we follow the training recipe proposed by Tian et al. (2019), and for those on transformer-based networks, we follow the training recipe proposed by Xu et al. (2023). See Appendix A.4 for implementation details. Results on CIFAR-100 are shown in Tab. 3, where SQUAF consistently outperforms all compared AFs, achieving up to 2.24% accuracy gain.

ImageNet-1K. We perform experiments across 4 model architectures, i.e., ResNet-18, ResNet-34, ViT-T and Swin-T (Liu et al., 2021), with the first 2 being CNNs and the last 2 being transformer-based. For CNNs and Swin-T, we follow the standard PyTorch (Paszke et al., 2019) training recipe, and for ViT-T, we follow the training recipe proposed by Xu et al. (2023). Results on ImageNet-1K are shown in Tab. 2, where SQUAF consistently outperforms compared AFs, achieving up to 1.13% gain over the baseline. Please refer to Appendix A.14 for visualization of SQUAFs.

Table 2: Top-1 accuracy (%) on ImageNet-1K.

AF	ResNet-18	ResNet-34	ViT-T	Swin-T
	69.76	73.31	73.90	81.28
ReLU	69.76 (+0.00)	73.31 (+0.00)	73.43 (-0.47)	80.70 (-0.58)
GELU	70.66 (+0.90)	73.71 (+0.40)	73.90 (+0.00)	81.28 (+0.00)
SiLU	70.51 (+0.75)	72.76 (-0.55)	73.70 (-0.20)	80.29 (-0.99)
SQUAF	**70.88** (+1.12)	**74.44** (+1.13)	**75.00** (+1.10)	**81.77** (+0.49)

Note that the performance of the baseline models are obtained using models' original AFs: ReLU for CNNs, GELU for ViT-T and Swin-T, while EfficientFormer-L1 uses its original hybrid AFs.

4.3 PDE Solving

We evaluate SQUAF on ten PDE problems from the PINNacle (Hao et al., 2024) benchmark, covering the Burgers' equation, the Poisson's equation, the Heat equation, the Navier-Stokes (NS) equation, and inverse problems of varying dimensionality and complexity. We use the vanilla PINN with five hidden layers of width 100 as the baseline architecture, and replace its default Tanh AF with

Table 3: Top-1 accuracy (%) on CIFAR-100. Results are averaged over 3 runs. The numbers in parentheses show the performance differences between each AF and the baseline model, with green denoting outperformance and red denoting underperformance. If a model diverges or runs into numerical issues during training, we denote its performance with "–".

AF	ShuffleNetV1 70.50	VGG-8 70.36	ResNet-56 72.34	VGG-13 74.64	ViT-T 71.84	EfficientFormer-L1 80.26
ReLU	70.50 (+0.00)	70.36 (+0.00)	72.34 (+0.00)	74.64 (+0.00)	72.06 (+0.22)	80.67 (+0.41)
GELU	71.28 (+0.78)	69.94 (-0.42)	72.45 (+0.11)	74.53 (-0.11)	71.84 (+0.00)	80.86 (+0.60)
LReLU	71.51 (+1.01)	70.48 (+0.12)	72.71 (+0.37)	75.00 (+0.36)	71.43 (-0.41)	80.52 (+0.26)
PReLU	70.38 (-0.12)	68.30 (-2.06)	71.76 (-0.58)	73.33 (-1.31)	72.06 (+0.22)	79.76 (-0.50)
SiLU	72.08 (+1.58)	70.28 (-0.08)	72.16 (-0.18)	74.68 (+0.04)	71.68 (-0.16)	80.33 (+0.07)
Swish	72.25 (+1.75)	70.50 (+0.14)	73.25 (+0.91)	74.86 (+0.22)	71.44 (-0.40)	80.55 (+0.29)
CRReLU	68.18 (-2.32)	68.22 (-2.14)	71.43 (-0.91)	72.78 (-1.86)	71.17 (-0.67)	79.19 (-1.07)
PolyReLU	–	–	–	–	70.66 (-1.18)	–
PolyNorm	65.58 (-4.92)	66.65 (-3.71)	65.52 (-6.82)	68.53 (-6.11)	71.44 (-0.40)	79.53 (-0.73)
DiTAC	71.29 (+0.79)	71.03 (+0.67)	73.22 (+0.88)	74.52 (-0.12)	4.35 (-67.49)	34.41 (-45.85)
SQUAF	**72.74** (+2.24)	**72.57** (+2.21)	**73.93** (+1.59)	**76.05** (+1.41)	**73.15** (+1.31)	**81.48** (+1.22)

Table 4: L2RE of PINNs with different AFs. ↑ indicates worse performance than the baseline Tanh, while green numbers show the L2RE reduction over Tanh in percentage.

		ReLU	GELU	LReLU	PReLU	SiLU	Swish	CRReLU	PolyReLU	PolyNorm	Tanh	sine	LAAF	GAAF	SQUAF
Burgers	1d-C	6.33E-1 (↑)	1.44E-2 (0.69%)	6.30E-1 (↑)	6.29E-1 (↑)	1.38E-2 (4.83%)	1.50E-2 (↑)	3.90E-1 (↑)	4.11E-1 (↑)	1.40E-2 (3.45%)	1.45E-2	2.12E-2 (↑)	1.43E-2 (1.38%)	5.20E-2 (↑)	**1.34E-2** (7.59%)
	2d-C	5.33E-1 (↑)	4.82E-1 (6.77%)	5.18E-1 (↑)	5.10E-1 (1.35%)	4.85E-1 (6.19%)	4.86E-1 (6.00%)	5.05E-1 (2.32%)	–	4.67E-1 (9.67%)	5.17E-1	4.52E-1 (12.57%)	4.74E-1 (8.32%)	4.85E-1 (6.19%)	**4.42E-1** (14.51%)
Poisson	2d-C	5.24E-1 (24.50%)	6.78E-1 (2.31%)	4.94E-1 (28.82%)	4.90E-1 (29.39%)	7.01E-1 (↑)	7.05E-1 (↑)	4.89E-1 (29.54%)	4.82E-1 (30.55%)	7.11E-1 (↑)	6.94E-1	6.65E-1 (4.18%)	7.68E-1 (↑)	6.04E-1 (12.97%)	**4.06E-1** (41.50%)
	3d-CG	1.38E+0 (↑)	5.87E-1 (↑)	1.42E+0 (↑)	1.41E+0 (↑)	6.02E-1 (↑)	6.38E-1 (↑)	1.32E+0 (↑)	7.54E-1 (↑)	5.87E-1 (↑)	5.60E-1	5.85E-1 (↑)	5.79E-1 (↑)	5.02E-1 (10.36%)	**3.33E-1** (40.54%)
Heat	2d-VC	1.56E+0 (↑)	8.95E-1 (11.39%)	2.95E+0 (↑)	3.89E+0 (↑)	1.02E+0 (↑)	9.58E-1 (5.15%)	2.32E+0 (↑)	–	8.24E-1 (18.42%)	1.01E+0	8.58E-1 (15.05%)	6.42E-1 (36.44%)	8.49E-1 (15.94%)	**4.64E-1** (54.06%)
	2d-MS	2.68E+0 (↑)	4.09E-2 (34.14%)	2.70E+0 (↑)	2.66E+0 (↑)	5.24E-2 (15.62%)	4.25E-2 (31.56%)	1.72E+0 (↑)	–	6.00E-2 (3.38%)	6.21E-2	3.94E-2 (36.55%)	7.40E-2 (↑)	9.85E-1 (↑)	**3.57E-2** (42.51%)
NS	2d-C	8.85E-1 (↑)	2.78E-2 (40.85%)	8.92E-1 (↑)	8.82E-1 (↑)	3.65E-2 (22.34%)	3.16E-2 (32.77%)	8.43E-1 (↑)	3.48E-1 (↑)	2.61E-2 (44.47%)	4.70E-2	4.00E-2 (14.89%)	3.60E-2 (23.40%)	3.79E-2 (19.36%)	**2.51E-2** (46.60%)
	2d-CG	2.93E-1 (↑)	6.96E-2 (41.51%)	2.89E-1 (↑)	2.90E-1 (↑)	7.44E-2 (37.48%)	7.27E-2 (38.91%)	2.88E-1 (↑)	–	1.72E-1 (↑)	1.19E-1	8.36E-2 (29.75%)	8.24E-2 (30.76%)	1.74E-1 (↑)	**6.69E-2** (43.78%)
Inverse	PInv	5.25E-1 (↑)	1.03E-1 (↑)	4.67E-1 (↑)	1.68E+0 (↑)	9.26E-2 (1.70%)	9.14E-2 (2.97%)	9.73E-2 (↑)	4.75E-2 (49.58%)	1.36E-1 (↑)	9.42E-2	9.94E-2 (↑)	1.30E-1 (↑)	2.54E-1 (↑)	**3.41E-2** (63.80%)
	HInv	9.43E-1 (39.94%)	1.13E-1 (92.80%)	7.93E-1 (49.49%)	1.61E+0 (↑)	**9.84E-2** (93.73%)	1.80E-1 (88.54%)	5.28E-1 (66.37%)	3.78E-1 (75.92%)	9.69E+0 (↑)	1.57E+0	1.84E-1 (88.28%)	5.59E-1 (64.39%)	2.12E-1 (86.50%)	1.78E-1 (88.66%)

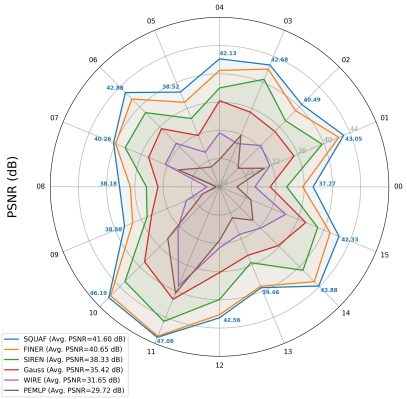


Figure 2: PSNR results for SQUAF and other INR methods with various AFs.

Table 5: Top-1 accuracy (%) on CIFAR-100 and the number of trainable parameters in a model. Results are averaged over 3 runs.

Teacher	VGG-13 74.64	9462180	WRN-40-2 75.61	2255156
Student	VGG-8		WRN-16-2	
ReLU	72.98	3965028	74.92	703284
GELU	73.51 (+0.53)	3965028	75.30 (+0.38)	703284
LReLU	73.37 (+0.39)	3965028	74.90 (-0.02)	703284
PReLU	72.27 (-0.71)	3965033	74.93 (+0.01)	703297
SiLU	73.23 (+0.25)	3965028	75.28 (+0.36)	703284
Swish	73.45 (+0.47)	3965033	75.36 (+0.44)	703297
CRReLU	72.88 (-0.10)	3965033	74.98 (+0.06)	703297
PolyReLU	–	3965048	–	703336
PolyNorm	70.44 (-2.54)	3965048	72.86 (-2.06)	703336
DiTAC	73.49 (+0.51)	3965073	75.45 (+0.53)	703401
SQUAF	**74.95** (+1.97)	3965063	**75.93** (+1.01)	703319

Table 6: Quantitative comparison for 3D shape INRs.

		Armadillo	Asian Dragon	Lucy	Thai Statue	Avg.
Chamfer ↓	PEMLP	3.453E-6	2.082E-6	2.313E-6	3.758E-6	2.901E-6
	Gauss	2.015E-5	1.019E-5	4.899E-6	2.765E-5	1.572E-5
	SIREN	3.458E-6	2.610E-6	2.851E-6	4.759E-6	3.419E-6
	WIRE	3.312E-6	2.123E-6	2.112E-6	3.532E-6	2.770E-6
	FINER	3.447E-6	2.258E-6	2.267E-6	4.215E-6	3.047E-6
	SQUAF	**3.246E-6**	**1.915E-6**	**1.982E-6**	**3.026E-6**	**2.542E-6**
IoU ↑	PEMLP	9.883E-1	9.815E-1	9.767E-1	9.599E-1	9.766E-1
	Gauss	9.719E-1	9.684E-1	9.645E-1	9.139E-1	9.547E-1
	SIREN	9.894E-1	9.703E-1	9.668E-1	9.482E-1	9.687E-1
	WIRE	9.895E-1	9.760E-1	9.760E-1	9.624E-1	9.760E-1
	FINER	9.889E-1	9.732E-1	9.745E-1	9.547E-1	9.728E-1
	SQUAF	**9.902E-1**	**9.823E-1**	**9.786E-1**	**9.677E-1**	**9.797E-1**

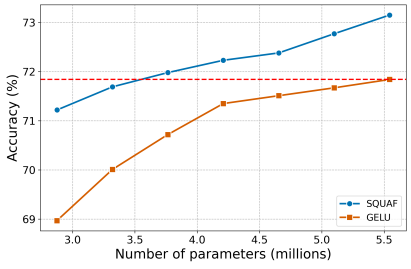


Figure 3: Reducing model depth for ViT-T.

SQUAF or another AF for comparison. Besides the AFs compared in previous experiments, we further include 3 popular AFs used in PINNs for comparison, namely sine (Raissi et al., 2019b), LAAF (Jagtap et al., 2020b) and GAAF (Jagtap et al., 2020a). Note that DiTAC doesn't support second- or higher-order differentiation, thus being unsuitable for PINNs; therefore, we exclude it from the comparison. We follow the training recipe given by Hao et al. (2024), and the results are averaged over three runs. More experimental details are provided in Appendix A.4. As shown in Tab. 4, SQUAF is the only AF that consistently improves upon the baseline Tanh, reducing the ℓ_2 relative error (L2RE) by 7.59%$\sim$88.66%, and achieves the lowest L2RE in 9 out of 10 cases.

4.4 2D Image Representation

In this task, the 2D pixel coordinates of an image are treated as inputs to an MLP, which is trained to output the corresponding RGB values. The learned $\mathbb{R}^2 \to \mathbb{R}^3$ model is called an implicit neural representation (INR) of the 2D image. For INR evaluation, we adopt 16 natural images with 512×512 resolution from the DIV2K dataset (Timofte et al., 2017; Liu et al., 2024). Since conventional AFs are known to perform poorly in INR tasks (Tancik et al., 2020; Sitzmann et al., 2020), we directly compare an MLP equipped with SQUAF against five specialized INR methods with varying AFs: PEMLP (Tancik et al., 2020), SIREN (Sitzmann et al., 2020), Gauss (Ramasinghe & Lucey, 2022), WIRE (Saragadam et al., 2023), and FINER (Liu et al., 2024). Following Liu et al. (2024), all models have four hidden layers and comparable parameter counts, and are trained to minimize the MSE between the model output and the ground truth (GT). More experimental details are provided in Appendix A.4. As shown in Fig. 2, SQUAF obtains the highest PSNR on all 16 images, achieving an average PSNR of 41.60 dB. SQUAF also achieves the highest average SSIM as shown in Appendix A.6. The visual comparison in Fig. 4 further shows that SQUAF better preserves fine details in the fitted image. Among compared INRs, only SQUAF faithfully reconstructs the region in the green box, showing a smooth red strip in the middle and a sharp complex pattern on the right. In contrast, other INRs introduce unwelcome noise to the red strip and/or overly smooth the complex pattern.

4.5 3D Shape Representation

In this task, an INR is a $\mathbb{R}^3 \to \mathbb{R}$ mapping, which computes the signed distance field (SDF) value for each 3D coordinate, given a 3D shape. We conduct experiments using four shapes from the Stanford 3D Scanning Repository (Stanford Computer Graphics Laboratory, 2014), and compare against the same five INR methods as in Sec. 4.4. Following Liu et al. (2024), all models have three hidden layers and comparable parameter counts. Here, we report SQUAF's performance by embedding it into MLPs with positional encoding (PEMLPs) instead of plain MLPs, replacing the original ReLU AF. More experimental details are provided in Appendix A.4. As shown in Tab. 6, SQUAF-equipped PEMLP not only significantly outperforms the original PEMLP, but also achieves the lowest Chamfer distance and the highest intersection over union (IoU) on all four shapes. This shows that SQUAF is compatible with the popular positional encoding technique, as the two can be used together to get improved results. Fig. 5 further shows that SQUAF visually enhances fine structures, e.g., the scales on the base of the statue, while avoiding excessive noise or over-smoothing.

4.6 Extensions

Distilling knowledge into SQUAF-DNNs. KD (Hinton et al., 2015; Zheng & YANG, 2024) is one of the most popular ways towards model compression. However, in most cases, a student model can only get close to its teacher's performance but cannot catch up with or surpass that. This is verified in Tab. 5, where student models with baseline AFs never outperform their respective teacher models. However, student models with SQUAFs are not only smaller than their respective teachers but also obtain higher classification accuracy, achieving up to 3.2$\times$ compression ratio with non-trivial accuracy improvement. Experiments are carried out with VGGs and Wide ResNets (WRNs) (Zagoruyko & Komodakis, 2016) using the distillation recipe in Tian et al. (2019). More KD results on ImageNet-1K are provided in Appendix A.7.

Model depth reduction. To further examine SQUAF's potential for model compression without KD, we configure ViT-T with 6 to 12 transformer blocks and train each model from scratch on CIFAR-100, using either SQUAF or the default GELU as the AF. The results, averaged over 3 runs, are shown in Fig. 3, where the points from right to left correspond to model depths from 12

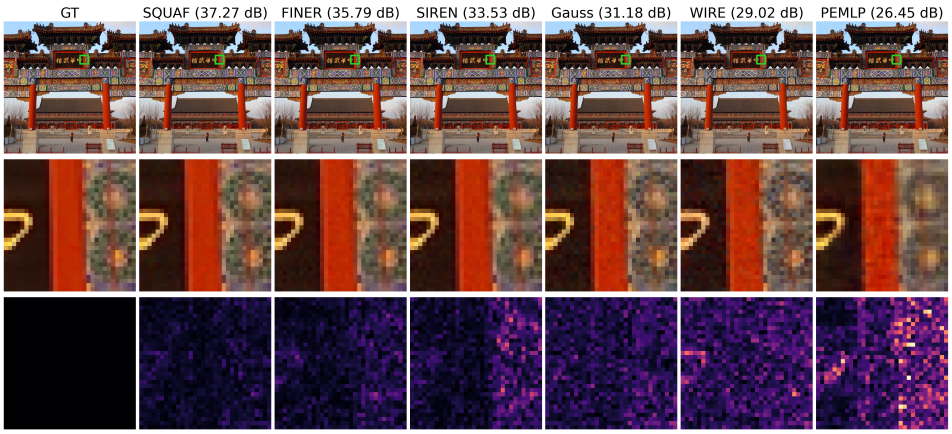


Figure 4: Visual comparison among INRs. The first row shows the whole image, the second row zooms into the green box region, and the third row shows the residual map of the zoomed-in region.

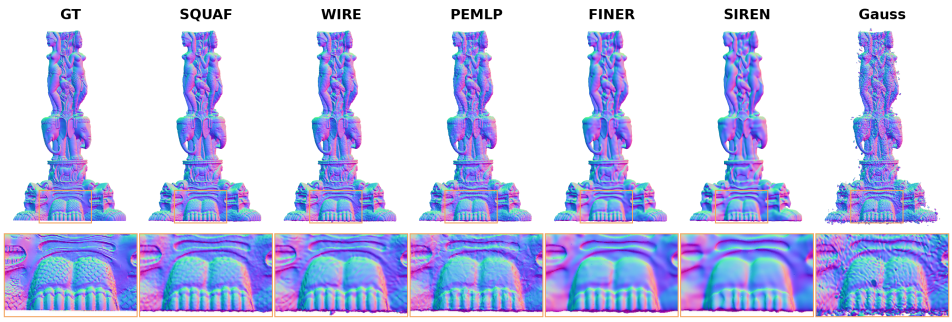


Figure 5: Visual comparison on the Thai Statue. The bottom row zooms into the orange box region.

blocks to 6 blocks, respectively. It's shown that SQUAF consistently outperforms GELU at every model depth. Notably, the 8-block ViT-T with SQUAF achieves 71.98% accuracy, higher than the 71.84% accuracy of the standard 12-block ViT-T with GELU, despite having 1/3 fewer transformer blocks and 32.1% fewer parameters. This suggests that the increased AF expressiveness provided by SQUAF can compensate for the reduced network size and thus facilitate model compression. More model depth reduction results on ImageNet-1K are provided in Appendix A.8.

Further discussions on the **initialization of SQUAF parameters**, **computational efficiency**, and **convergence stability** are provided in Appendices A.4.2, A.10, and A.11, respectively. Additionally, the **ablation study** conducted in Appendix A.5 shows that randomly initialized SQUAFs perform decently, while fitting the model's default AF provides additional gain, and jointly training all SQUAF parameters works better than training $\boldsymbol{z}$ alone.

5 Conclusion

In this paper, we propose SQUAF, a family of trainable, highly expressive AFs based on learnable soft quantization, and prove that SQUAFs can approximate any continuous 1D function defined over a closed interval with arbitrary precision. Extensive experiments spanning function fitting, image classification, PDE solving, and INRs show consistent gains over existing AFs, demonstrating SQUAF's effectiveness on both standard learning tasks and complex signal modeling tasks. The KD and model depth reduction results further show SQUAF's potential for model compression. A natural extension of our work is to evaluate SQUAF's effectiveness in large language models (LLMs). Due to limited computational resources, we provide some preliminary results in Appendix A.12 and leave a more comprehensive evaluation as future work. Beyond their application in conventional DNNs, SQUAFs can also be used as the edge functions in Kolmogorov-Arnold Networks (KANs) (Liu et al., 2025). Preliminary results in Appendix A.13 show that KANs equipped with SQUAF outperform the standard KANs based on B-splines on PDE solving tasks. A broader evaluation of this extension and the exploration of novel SQUAF-based KAN variants are left for future work.

AI USE STATEMENT

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

REPRODUCIBILITY STATEMENT

Please refer to Section 4 and Appendix A.4 to reproduce our experimental results.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

Irit Chelly, Shahaf E Finder, Shira Ifergane, and Oren Freifeld. Trainable highly-expressive activation functions. In *European Conference on Computer Vision*, pp. 200–217. Springer, 2024.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 702–703, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. Understanding dataset difficulty with $\mathcal{V}$-usable information. In *International Conference on Machine Learning*, pp. 5988–6008. PMLR, 2022.

Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *Advances in neural information processing systems*, 37:76721–76774, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020a.

Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings. Mathematical, physical, and engineering sciences*, 476(2239):20200334, 2020b.

Serhat Kiliçarslan and Mete Celik. Kaf+ rsigelu: a nonlinear and kernel-based activation function for deep neural networks. *Neural Computing and Applications*, 34(16):13909–13923, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yanyu Li, Geng Yuan, Yang Wen, Eric Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*, 2022.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

Zhen Liu, Hao Zhu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer: Flexible spectral-bias tuning in implicit neural representation by variableperiodic activation functions. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2713–2722. IEEE, 2024.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljacic, Thomas Hou, and Max Tegmark. Kan: Kolmogorov–arnold networks. In *International conference on learning representations*, volume 2025, pp. 70367–70413, 2025.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Atlanta, GA, 2013.

TorchVision maintainers and contributors. Image classification reference training scripts. https://github.com/pytorch/vision/tree/main/references/classification, 2016. Accessed: 2025-09-22.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019a.

Maziar Raissi, Zhicheng Wang, Michael S Triantafyllou, and George Em Karniadakis. Deep learning of vortex-induced vibrations. *Journal of fluid mechanics*, 861:119–137, 2019b.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Sameera Ramasinghe and Simon Lucey. Beyond periodicity: Towards a unifying framework for activations in coordinate-mlps. In *European Conference on Computer Vision*, pp. 142–158. Springer, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Ahmed H Salamah, Kaixiang Zheng, Yiwen Liu, and EN-HUI YANG. Jpeg inspired deep learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *2023 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 18507–18516. IEEE, 2023.

Simone Scardapane, Steven Van Vaerenbergh, Simone Totaro, and Aurelio Uncini. Kafnets: Kernel-based non-parametric activation functions for neural networks. *Neural Networks*, 110:19–32, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

Stanford Computer Graphics Laboratory. The stanford 3d scanning repository. http://graphics.stanford.edu/data/3Dscanrep/, 2014.

Haoyuan Sun, Zihao Wu, Bo Xia, Pu Chang, Zibin Dong, Yifu Yuan, Yongzhe Chang, and Xueqian Wang. Entropy-based activation function optimization: A method on searching better activation functions. In *The Thirteenth International Conference on Learning Representations*, 2025.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. In *International Conference on Learning Representations*, 2019.

Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee, et al. Ntire 2017 challenge on single image super-resolution: Methods and results. In *2017 IEEE conference on computer vision and pattern recognition workshops (CVPRW)*, pp. 1110–1121. IEEE, 2017.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021a.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 32–42. IEEE, 2021b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zhiqiu Xu, Yanjie Chen, Kirill Vishniakov, Yida Yin, Zhiqiang Shen, Trevor Darrell, Lingjie Liu, and Zhuang Liu. Initializing models with larger ones. In *The Twelfth International Conference on Learning Representations*, 2023.

En-hui Yang and Shayan Mohajer Hamidi. Coded deep learning: Framework and algorithm. *arXiv preprint arXiv:2501.09849*, 2025.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

Kaixiang Zheng and EN-HUI YANG. Knowledge distillation based on transformed teacher matching. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 13001–13008, 2020.

Zhijian Zhuo, Ya Wang, Yutao Zeng, Xiaoqing Li, Xun Zhou, and Jinwen Ma. Polynomial composition activations: Unleashing the dynamics of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

A Appendix

A.1 Related Work

General-purpose AFs. Early AFs adopted in DNNs are sigmoidal functions, e.g., logistic and Tanh. These functions, however, suffer from vanishing gradients. To alleviate this issue, $\mathrm{ReLU}(x) = \max(0, x)$ was proposed. ReLU is cheap to compute, and more crucially, does not face the vanishing-gradient issue. Due to these advantages, ReLU has been used predominantly in DNNs for more than a decade. However, ReLU is a simple piecewise-linear function and offers limited nonlinearity, constraining the expressiveness achievable by a DNN of fixed architecture and size.

To increase the degree of nonlinearity in AFs, and consequently the overall expressiveness of DNNs, Leaky ReLU (LReLU), SiLU, and GELU were then proposed. Specifically, LReLU improves upon ReLU by allowing a small, nonzero slope (a "leak") for negative inputs, rather than outputting zero, thus maintaining a small gradient and preventing neurons from becoming inactive during training. SiLU, defined as $\mathrm{SiLU}(x) = x\sigma(x)$, where $\sigma(x)$ is the logistic function, and GELU, defined as $\mathrm{GELU}(x) = x\Phi(x)$, where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, are smooth variants of ReLU, removing its nondifferentiability at the origin and further increasing nonlinearity around the origin. Since GPT-1 (Radford et al., 2018), GELU has been dominant in modern transformer-based models, e.g., GPT-2 (Radford et al., 2019), BERT (Devlin et al., 2019) and ViT (Dosovitskiy et al., 2020). On the other hand, SiLU has become dominant in UNet-based modern diffusion models (Ho et al., 2020; Rombach et al., 2022).

The AFs mentioned above are all fixed and non-adaptive. To provide task-adaptive nonlinearity, trainable AFs have also been explored in the literature. For instance, PReLU is a trainable variant of LReLU where the slope of the negative part is optimized during training. Likewise, SiLU has been generalized to Swish, where a trainable temperature is inserted into the logistic function. More recently, CRReLU adds a regularization term to ReLU with a trainable weighting factor guided by an entropy-based AF optimization methodology. In general, trainable AFs like PReLU, Swish, and CRReLU only moderately extend their non-trainable baselines due to simple parametrization. Therefore, to achieve even higher degrees of nonlinearity, recent studies on trainable AFs have focused on more complex functions with increased numbers of parameters and diverse functional classes. Zhuo et al. (2025) proposed polynomial composition activation functions (PolyCom), including PolyReLU and PolyNorm, which embed ReLU or normalization operations into weighted sums of polynomial functions. Chelly et al. (2024) introduced DiTAC, a diffeomorphism-based trainable AF using continuous piecewise-affine-based transformations. These trainable AFs take much more flexible shapes compared to earlier AFs. Nevertheless, they (1) remain tied to predefined functional families, limiting their expressiveness to some fixed patterns, and (2) have issues in optimization; e.g., PolyReLU, as commented in the original paper, may be unstable in non-transformer architectures, and DiTAC doesn't support second- or higher-order differentiation according to their official implementation. Kernel-based activation functions (KAFs) considered in Scardapane et al. (2019), as opposed to aforementioned trainable AFs, can approximate any continuous function over a subset of the real line. However, in practice, KAFs face many issues such as bias shift, vanishing gradients, and neuron death during training (Kiliçarslan & Celik, 2022). These studies motivate AFs that combine high expressiveness with well-behaved optimization behaviors. SQUAF follows this direction by leveraging learnable soft quantization and, unlike methods only suitable for a particular architecture or task, can be used as a general-purpose AF.

AFs for PINNs. The differential operators in PDE residuals impose additional requirements on the smoothness and differentiability of AFs used in PINNs. In the original PINN paper (Raissi et al., 2019a), Tanh was chosen to be the default AF. Another early work on PINNs (Raissi et al., 2019b) adopts sine as the AF as it seems to be numerically more stable than Tanh. The global adaptive activation function (GAAF) method introduces a trainable scaling factor into the base AF, shared across all layers and neurons, and optimizes it jointly with the network parameters, accelerating convergence and improving solution accuracy (Jagtap et al., 2020a). As a generalization of GAAF, locally adaptive activation functions (LAAFs) further assign distinct trainable scaling factors to individual layers, i.e., layer-wise locally adaptive activation functions (L-LAAF), or even neurons, i.e., neuron-wise locally adaptive activation functions (N-LAAF), to further boost the performance (Jagtap et al., 2020b)[1]. While GAAF and LAAF indeed increase the flexibility of PINNs, they actually

[1]The LAAF compared in Sec. 4.3 is the stronger variant N-LAAF.

don't introduce additional nonlinearity to the base AF. In fact, the adaptive slopes can be absorbed into the linear layers of MLPs, so these two methods are equivalently adding more trainable parameters to linear modules. In contrast, SQUAF aims to adapt the nonlinear response of the AF itself to the target PDE problems, in order to address their complex geometries, multi-scale phenomena, and nonlinear behaviors.

AFs for INRs. Coordinate-based MLPs with conventional AFs exhibit spectral bias and have difficulty learning high-frequency details. To address this, a popular method is to use MLPs with positional encoding (PEMLPs), where the input coordinates are encoded into random Fourier features (RFFs) before being fed into the MLP (Tancik et al., 2020). Another line of work focuses on proposing specialized AFs for coordinate-based MLPs towards better INR performance. SIREN proposes to use periodic sine AFs in MLPs with a tailored weight initialization scheme (Sitzmann et al., 2020). Ramasinghe & Lucey (2022) show that periodicity is not necessary and propose Gaussian AFs. WIRE (Saragadam et al., 2023) combines the advantages of both periodic and Gaussian AFs by using a complex Gabor wavelet AF, which includes both sine and Gaussian AFs as special cases. More recently, FINER introduces a variable-periodic AF whose oscillation rate changes across its domain, and adjusts the MLP's bias initialization scheme to leverage a wider range of supported frequencies (Liu et al., 2024). These AFs, despite their flexibility and effectiveness, are fixed and non-trainable. SQUAF, as a universal 1D function approximator, can therefore be initialized with their shapes and then be further optimized during training to get improved results[2]; this includes initializing SQUAF with ReLU and using it within PEMLPs. Moreover, little evidence shows that these specialized AFs could be effective in standard learning tasks, while SQUAF is a general-purpose AF that can be used in a wide range of tasks, including INRs.

A.2 Partial Derivatives of SQUAF

We now compute the partial derivatives of $\phi(x)$ w.r.t. each component of $(x, \boldsymbol{y}, \boldsymbol{z}, \alpha)$ or $(x, q, \boldsymbol{z}, \alpha)$ when $P(\cdot|x)$ is the discrete conditional Gaussian distribution in Example 1.

For x, we have

$$\frac{\partial}{\partial x}P(y_i|x) = P(y_i|x)\left[-2\alpha(x-y_i)\right] - P(y_i|x)\sum_{j=1}^{L} P(y_j|x)\left[-2\alpha(x-y_j)\right] \tag{10}$$

$$= -2\alpha P(y_i|x)\left[(x-y_i) - \sum_{j=1}^{L} P(y_j|x)(x-y_j)\right] \tag{11}$$

$$= 2\alpha P(y_i|x)\left[y_i - \sum_{j=1}^{L} y_j P(y_j|x)\right], \tag{12}$$

$$\frac{\partial}{\partial x}\phi(x) = 2\alpha \sum_{i=1}^{L} z_i P(y_i|x)\left[y_i - \sum_{j=1}^{L} y_j P(y_j|x)\right] \tag{13}$$

$$= 2\alpha\left[\sum_{i=1}^{L} z_i y_i P(y_i|x) - \left(\sum_{i=1}^{L} z_i P(y_i|x)\right)\left(\sum_{j=1}^{L} y_j P(y_j|x)\right)\right] \tag{14}$$

$$= 2\alpha \cdot \mathrm{Cov}(\mathrm{z}, \mathrm{y} \mid x). \tag{15}$$

Note that equation 15 implies that the partial derivative of $\phi(x)$ with respect to x is equal to the (scaled) covariance between $\boldsymbol{z}$ and $\boldsymbol{y}$ calculated according to $P(\cdot|x)$, which is nonzero unless $\boldsymbol{z}$ and $\boldsymbol{y}$ are uncorrelated. In practice, $\boldsymbol{z}$ generally depends on $\boldsymbol{y}$, and since $\boldsymbol{z}$ and $\boldsymbol{y}$ are trainable parameters, even if they are uncorrelated at some point of training, future updates could make them correlated; therefore, it's unlikely for them to be persistently uncorrelated through frequent training steps. Thus, $\phi(x)$ generally won't hinder the learning process with persistent vanishing gradients.

[2]Even the complex Gabor wavelet AF can be decomposed into three real 1D AFs, which can then be approximated by SQUAFs.

The convergence curves in Fig. 7 also empirically show that SQUAF-DNNs have well-behaved optimization behaviors.

In the special case when $\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}$, equation 15 implies

$$\frac{\partial}{\partial x}\phi(x) = 2\alpha q \left[\sum_{i=-k}^{k} i z_i P(iq|x) - \left(\sum_{i=-k}^{k} z_i P(iq|x) \right) \left(\sum_{j=-k}^{k} j P(jq|x) \right) \right]. \tag{16}$$

For $\boldsymbol{y}$, we have

$$\frac{\partial}{\partial y_i} P(y_i|x) = P(y_i|x) \left[2\alpha(x - y_i) \right] - P^2(y_i|x) \left[2\alpha(x - y_i) \right], \tag{17}$$

$$\frac{\partial}{\partial y_j} P(y_i|x) = -P(y_i|x) P(y_j|x) \left[2\alpha(x - y_j) \right] \quad \forall j \neq i, \tag{18}$$

$$\frac{\partial}{\partial y_j}\phi(x) = \sum_{i=1}^{L} z_i \frac{\partial}{\partial y_j} P(y_i|x) \tag{19}$$

$$= z_j \cdot P(y_j|x) \left[2\alpha(x - y_j) \right] - 2\alpha(x - y_j) P(y_j|x) \sum_{i=1}^{L} z_i P(y_i|x) \tag{20}$$

$$= 2\alpha(x - y_j)\ P(y_j|x)\ \left[z_j - \sum_{i=1}^{L} z_i\ P(y_i|x) \right]. \tag{21}$$

In view of equation 21 and the chain rule, we have

$$\frac{\partial}{\partial q}\phi(x) = \sum_{j=-k}^{k} 2\alpha(x - jq) P(jq|x) \left[z_j - \sum_{i=-k}^{k} z_i P(iq|x) \right] \cdot j \tag{22}$$

$$= 2\alpha \sum_{j=-k}^{k} j(x - jq) P(jq|x) \left[z_j - \sum_{i=-k}^{k} z_i P(iq|x) \right] \tag{23}$$

when $\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}$.

For $\boldsymbol{z}$, we have

$$\frac{\partial \phi(x)}{\partial z_i} = P(y_i|x) \tag{24}$$

in the general case and

$$\frac{\partial \phi(x)}{\partial z_i} = P(iq|x) \tag{25}$$

in the uniform case.

For α, we have

$$\frac{\partial}{\partial \alpha} P(y_i|x) = P(y_i|x) \left[-(x - y_i)^2 \right] - P(y_i|x) \sum_{j=1}^{L} P(y_j|x) \left[-(x - y_j)^2 \right] \tag{26}$$

$$= -P(y_i|x) \left[(x - y_i)^2 - \sum_{j=1}^{L} (x - y_j)^2 P(y_j|x) \right], \tag{27}$$

$$\frac{\partial}{\partial \alpha}\phi(x) = \sum_{i=1}^{L} z_i \frac{\partial}{\partial \alpha} P(y_i|x) \tag{28}$$

$$= -\left[\sum_{i=1}^{L} z_i (x - y_i)^2 P(y_i|x) - \left(\sum_{i=1}^{L} z_i P(y_i|x) \right) \left(\sum_{j=1}^{L} (x - y_j)^2 P(y_j|x) \right) \right] \tag{29}$$

$$= -\mathrm{Cov}\left(\mathrm{z}, (x - \mathrm{y})^2 | x \right) \tag{30}$$

in the general case, and

$$\frac{\partial}{\partial\alpha}\phi(x) = -\left[\sum_{i=-k}^{k} z_i(x-iq)^2 P(iq|x) - \left(\sum_{i=-k}^{k} z_i P(iq|x)\right)\left(\sum_{j=-k}^{k}(x-jq)^2 P(jq|x)\right)\right] \tag{31}$$

when $\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}$.

A.3 Proof of Theorems

We begin with the proof of Theorem 1.

Proof of Theorem 1: Without loss of generality, we assume that $[a,b] = [-1,1]$. Otherwise, translation and scaling can be applied to convert $[a,b]$ to $[-1,1]$ so that $g(x)$ is defined over $[-1,1]$. Since g is continuous on the closed interval $[-1,1]$, it is uniformly continuous. Let $\omega(\delta) = \sup\{|g(x)-g(y)| : |x-y| \le \delta,\ x,y \in [a,b]\}$ be the modulus of continuity. Then $\omega(\delta) \to 0$ as $\delta \downarrow 0$.

Choose $q > 0$ small enough so that

$$\omega(q) < \frac{\varepsilon}{3}.$$

Let $k = \lfloor 1/q \rfloor$. Consider

$$\hat{\mathcal{A}} = \{0, \pm q, \pm 2q, \ldots, \pm kq\}.$$

For each $-k \le i \le k$, let $z_i = g(iq)$, and

$$C_i = \begin{cases} [-1, (-k+\frac{1}{2})q] & \text{if } i = -k \\ ((i-\frac{1}{2})q, (i+\frac{1}{2})q] & \text{if } -k < i < k \\ (k-\frac{1}{2})q, 1] & \text{if } i = k \end{cases}. \tag{32}$$

With $\hat{\mathcal{A}}$ (hence $\boldsymbol{y}$) and $\boldsymbol{z}$ specified, we write the corresponding $\phi(x)$ as $\phi(x|q,\boldsymbol{z},\alpha)$. Define a step function $f(x)$ as follows

$$f(x) = \sum_{i=-k}^{k} z_i \mathcal{X}_{C_i}(x). \tag{33}$$

Then it follows that for any x, and any $0 \le \lambda \le 1$,

$$\begin{aligned} &|\phi(x|q,\boldsymbol{z},\alpha) - f(x)| \\ &= |\phi(x|\boldsymbol{y},\boldsymbol{z},\alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)} + (\lambda z_{i(x)} + (1-\lambda) z_{j(x)} - f(x))| \\ &= |\phi(x|\boldsymbol{y},\boldsymbol{z},\alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)} + \lambda(z_{i(x)} - f(x)) + (1-\lambda)(z_{j(x)} - f(x))| \\ &\le |\phi(x|\boldsymbol{y},\boldsymbol{z},\alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)}| + \omega(q). \end{aligned} \tag{34}$$

In the above, equation 34 is due to the fact that in view of the definitions of $f(x), i(x), j(x)$, and z_i, $f(x)$ is equal to either $z_{i(x)}$ or $z_{j(x)}$, and $|z_{i(x)} - z_{j(x)}| \le \omega(q)$. Since equation 34 is valid for any x and any $0 \le \lambda \le 1$, we have

$$|\phi(x|q,\boldsymbol{z},\alpha) - f(x)| \le \min_{0\le\lambda\le 1} |\phi(x|\boldsymbol{y},\boldsymbol{z},\alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)}| + \omega(q) \tag{35}$$

for any x, which further implies that

$$\begin{aligned} \sup_x |\phi(x|q,\boldsymbol{z},\alpha) - f(x)| &\le \sup_x \min_{0\le\lambda\le 1} |\phi(x|\boldsymbol{y},\boldsymbol{z},\alpha) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)}| + \omega(q) \\ &\le \frac{\varepsilon}{3} + \omega(q) \end{aligned} \tag{36}$$

for large α, where equation 36 follows from Property P2.

Finally,

$$\begin{aligned} \sup_x |\phi(x|q,\boldsymbol{z},\alpha) - g(x)| &= \sup_x |\phi(x|q,\boldsymbol{z},\alpha) - f(x) + f(x) - g(x)| \\ &\le \sup_x |\phi(x|q,\boldsymbol{z},\alpha) - f(x)| + \sup_x |f(x) - g(x)| \\ &\le \frac{\varepsilon}{3} + 2\omega(q) \\ &\le \varepsilon. \end{aligned} \tag{37}$$

This completes the proof of Theorem 1.

We next prove Theorem 2.

Proof of Theorem 2: It suffices to prove Theorem 2 when $P_\alpha(\cdot|x)$ is the discrete conditional Gaussian distribution in Example 1. A similar argument applies to the discrete conditional Laplacian distribution.

Fix $\boldsymbol{y}$ and $\boldsymbol{z}$. Let

$$Z = \max\{|z_i| : 1 \le i \le L\} \text{ and } s = \min\{|y_i - y_j| : 1 \le i, j \le L, i \ne j\}.$$

For each x, let $y_{i^*(x)}$ be the nearest point of x in $\hat{\mathcal{A}}$. Note that $i^*(x)$ is equal to either $i(x)$ or $j(x)$, and for any $i \notin \{i(x), j(x)\}$,

$$|x - y_i| - |x - y_{i^*(x)}|$$

is at least a positive integer multiplier of s. Then we have

$$\begin{aligned}
&|\phi(x) - z_{i(x)}P_\alpha(y_{i(x)}|x) - z_{j(x)}P_\alpha(y_{j(x)}|x)| \\
&= |\sum_{i \ne i(x), j(x)} z_i P_\alpha(y_i|x)| \\
&\le Z \sum_{i \ne i(x), j(x)} P_\alpha(y_i|x) \\
&\le Z \sum_{i \ne i(x), j(x)} \frac{e^{-\alpha(x-y_i)^2}}{e^{-\alpha(x-y_{i^*(x)})^2}} \\
&\le 2Z \sum_{j=1}^{\infty} e^{-\alpha s^2 j^2} \\
&\le 2Z \sum_{j=1}^{\infty} e^{-\alpha s^2 j} \\
&= 2Z \frac{e^{-\alpha s^2}}{1 - e^{-\alpha s^2}}
\end{aligned} \tag{38}$$

Let

$$r = \sum_{i \ne i(x), j(x)} P_\alpha(y_i|x).$$

In view of equation 38 and its derivation, it follows that

$$\begin{aligned}
&\min_{0 \le \lambda \le 1} |\phi(x) - \lambda z_{i(x)} - (1-\lambda) z_{j(x)}| \\
&\le |\phi(x) - z_{i(x)}(P_\alpha(y_{i(x)}|x) + r/2) - z_{j(x)}(P_\alpha(y_{j(x)}|x) + r/2)| \\
&\le |\phi(x) - z_{i(x)}P_\alpha(y_{i(x)}|x) - z_{j(x)}P_\alpha(y_{j(x)}|x)| + rZ \\
&\le 2Z \frac{e^{-\alpha s^2}}{1 - e^{-\alpha s^2}} + rZ \\
&\le 4Z \frac{e^{-\alpha s^2}}{1 - e^{-\alpha s^2}}.
\end{aligned} \tag{39}$$

Note that the upper bound in equation 39 is independent of x. Letting α go to ∞, Theorem 2 follows. This completes the proof of Theorem 2.

A.4 Implementation Details

A.4.1 General Settings

Function fitting. We train a model for 40K iterations, with a batch size of 98, using Adam optimizer without weight decay, a learning rate of 0.001 for the 1D function and 0.01 for the 2D function. The

Table 7: The training recipe for transformer-based models on CIFAR-100.

Setting	Configuration
training epochs	300
batch size	512
optimizer	AdamW
base learning rate	2e-3
weight decay	0.05
optimizer momentum	β_1, β_2=0.9, 0.999
learning rate decay	cosine
warmup epochs	50
warmup schedule	linear
layer-wise lr decay (Clark et al., 2020; Bao et al., 2021)	None
randaugment (Cubuk et al., 2020)	(9, 0.5)
mixup (Zhang et al., 2017)	0.8
cutmix (Yun et al., 2019)	1.0
random erasing (Zhong et al., 2020)	0.25
label smoothing (Szegedy et al., 2016)	0.1
layer scale (Touvron et al., 2021b)	1e-6
head init scale (Touvron et al., 2021b)	None
gradient clip	None

loss function is MSE. More details can be found in Chelly et al. (2024) and their released source code.

CNNs on CIFAR-100. We train for 240 epochs with an initial learning rate of 0.05 by default, which decays by 0.1 at the 150th, 180th, and 210th epochs. For ShuffleNetV1, a smaller initial learning rate of 0.01 is used. Batch size is set to 64, and the optimizer is SGD. More details, including KD settings, can be found in the source code released by Tian et al. (2019). Note that the distillation method used in our experiments is just the default KD as in Hinton et al. (2015).

Transformer-based models on CIFAR-100. The settings are summarized in Tab. 7.

ResNets and Swin-T on ImageNet-1K. We follow the recipe given on the PyTorch official website (maintainers & contributors, 2016).

ViT-T on ImageNet-1K. We follow the configuration specified in Tab. 7, except that batch size is now 4096.

PDE solving. Following the setup in PINNacle (Hao et al., 2024), all models are trained for 20K iterations using Adam with momentum $(0.9, 0.999)$, and a learning rate of 0.001. More details can be found in Hao et al. (2024) and their released source code.

2D image representation. Following Liu et al. (2024), we train all models for 5K epochs using Adam and the MSE loss function. For SQUAF, SIREN, Gauss, and FINER, we use an MLP with architecture $[2, 256, 256, 256, 256, 3]$. PEMLP uses architecture $[42, 256, 256, 256, 256, 3]$, where its positional encoding expands the MLP's input dimension from 2 to 42. Since WIRE uses complex-valued weights, its hidden width is reduced by a factor of $\sqrt{2}$ to maintain a comparable number of real-valued parameters. More details can be found in Liu et al. (2024) and their released source code.

3D shape representation. Following Liu et al. (2024), all models are trained for 200K iterations using the same coarse-to-fine loss function, with 10K points randomly sampled in each iteration. For SIREN, Gauss, and FINER, we use an MLP with architecture $[3, 256, 256, 256, 1]$. PEMLP and SQUAF use architecture $[63, 256, 256, 256, 1]$, where positional encoding expands the input dimension from 3 to 63. Since WIRE uses complex-valued weights, its hidden width is reduced by a factor of $\sqrt{2}$ to maintain a comparable number of real-valued parameters. We extract a 512^3 grid for quantitative evaluation and a 1024^3 grid for visualization. More details can be found in Liu et al. (2024) and their released source code.

A.4.2 SQUAF-RELATED SETTINGS

Initializing SQUAF parameters. Across all experiments, we use one of the following two general strategies to initialize SQUAFs: (1) direct random initialization, and (2) initialization through fitting

model default AFs. The former makes no assumption about a suitable AF shape, while the latter preserves the nonlinear response of an established model at initialization.

Random initialization. Without leveraging any prior, we directly initialize the SQUAF parameters from scratch. This setting is used for both function fitting with MLPs and all PDE experiments in Sec. 4.3, where we set $k = 2$, initialize q with 0.5 and α with 5, and independently sample each z_i from a uniform distribution over $[-1, 1]$. To examine the sensitivity of this strategy w.r.t the choice of the initial q and α, we perform a grid search over them on the 2D function fitting task. The pre-activations from the first training mini-batch are roughly zero-mean with a standard deviation of 0.51, based on which we search q within $[0.1, 1]$. We vary α by searching over the gradient scaling constant $\hbar = \alpha q^2$ (Salamah et al., 2025), which controls the smoothness of SQUAFs. As shown in Tab. 8, SQUAF outperforms the best compared AF in Tab. 1 under all 20 tested $(q, \hbar)$ pairs, demonstrating its robustness across a wide range of random initialization settings.

Table 8: MSE results for fitting the 2D function with different initialization of q and α. $\downarrow$ indicates the performance is better than all compared AFs in Tab. 1 — the best baseline AF achieves 4.0E-3 MSE.

MSE	$\alpha = 0.5/q^2$	$\alpha = 1/q^2$	$\alpha = 2/q^2$	$\alpha = 5/q^2$	$\alpha = 10/q^2$
$q = 0.1$	1.5E-4 ($\downarrow$)	2.4E-4 ($\downarrow$)	5.7E-4 ($\downarrow$)	1.6E-3 ($\downarrow$)	3.7E-3 ($\downarrow$)
$q = 0.2$	3.9E-4 ($\downarrow$)	6.0E-4 ($\downarrow$)	2.2E-4 ($\downarrow$)	2.1E-4 ($\downarrow$)	5.0E-4 ($\downarrow$)
$q = 0.5$	4.1E-4 ($\downarrow$)	5.3E-4 ($\downarrow$)	6.3E-4 ($\downarrow$)	6.6E-4 ($\downarrow$)	3.7E-4 ($\downarrow$)
$q = 1.0$	1.7E-3 ($\downarrow$)	9.3E-4 ($\downarrow$)	5.4E-4 ($\downarrow$)	1.4E-3 ($\downarrow$)	6.0E-4 ($\downarrow$)

For the SQUAF-only function fitting experiments in Tab. 1, we set $k = 95$ or $k = 13$, initialize q with $1/k$ so that $\hat{\mathcal{A}}$ covers the input domain $[-1, 1]$, and freeze it during training, thus excluding it from the trainable parameter count. We initialize $\alpha = 1.5/q^2$, i.e., $\hbar = 1.5$, to obtain a relatively smooth initial SQUAF and independently sample each z_i from a uniform distribution over $[-1, 1]$.

Initialization through fitting model default AFs. When SQUAF replaces an AF in an established model architecture, we initialize it by fitting the default AF of the model architecture so that the SQUAF-equipped model starts with a similar nonlinear response. This initialization follows a similar procedure across tasks. Given k, we probe the randomly initialized original DNN with its default AF using one training mini-batch and choose q such that $[-kq, kq]$ covers the observed pre-activation range. Then, given k and q, we fit SQUAF to the default AF within $[-kq, kq]$ using the SQUAF-only 1D function fitting procedure described above. The task specific settings are detailed below.

For image classification, SQUAF is initialized by fitting ReLU in CNNs and by fitting GELU in transformer-based models, which are the default AFs in respective architectures. Here, we set $k = 16$, as prior work on quantization-aware training (Esser et al., 2019; Yang & Hamidi, 2025) shows that 4-bit activation quantization can approximately preserve the accuracy of full-precision ReLU networks. Since SQUAF needs to cover both negative and positive pre-activations, unlike the non-negative ReLU activations considered in prior work, we use 4 bits for each side and set $k = 2^4 = 16$. We use $q = 1$ for CNNs on CIFAR-100 and all models on ImageNet-1K, while $q = 0.25$ is used for transformer-based models on CIFAR-100 because their initial pre-activations have a smaller variance.

For 2D image representation, SQUAF is initialized by fitting the FINER AF $\sin(30(|x|+1)x)$. Here, we increase k to 32, i.e., increasing soft quantization by one bit, to capture the highly oscillatory shape of FINER, and we set $q = 0.025$ to cover the observed pre-activation range.

For 3D shape representation, SQUAF replaces ReLU in PEMLP and is therefore initialized by fitting ReLU. Here, we set $k = 16$ to be consistent with the ReLU case in image classification, and $q = 0.05$ to cover the observed pre-activation range.

In all cases, the fitting can be completed within seconds. The fitted SQUAFs are then embedded into the DNNs as initialization. To justify and provide understanding about our initialization strategy, we further conduct an ablation study; please refer to Appendix A.5 for details.

Training SQUAF parameters. SQUAF parameters are exempted from weight decay in all experiments, with α and q clipped to be above 0 before any forward pass. For a fair comparison, the

parameters of all other trainable AFs are also exempted from weight decay. On CIFAR-100, we only train the first SQUAF with parameters $(q_1, \boldsymbol{z}_1, \alpha_1)$ and freeze the deeper SQUAFs to prevent overfitting. On ImageNet-1K, for DNNs with stages, i.e., ResNets and Swin-T, SQUAFs in the last stage are frozen to prevent overfitting, while all SQUAFs in ViT-T are trained. For other experiments not mentioned above, all SQUAFs are trainable by default.

A.5 Ablation Study on SQUAF Initialization and Training

To examine the effects of SQUAF initialization and parameter learning, we conduct an ablation study on the four 3D shapes in Tab. 6, using the same PEMLP architecture and training protocol described in Appendix A.4. For SQUAFs initialized by fitting ReLU (ReLU init.), we consider three settings: freezing all SQUAF parameters, training only $\boldsymbol{z}$ while freezing (q, α), and jointly training all parameters $(q, \boldsymbol{z}, \alpha)$. The last setting corresponds to our main setting in Tab. 6. We also evaluate randomly initialized SQUAFs (Random init.) with all parameters trainable. The original PEMLP with ReLU serves as the baseline.

All SQUAF variants use $k = 16$ and an initial quantization step size $q = 0.05$. For ReLU initialization, $(\boldsymbol{z}, \alpha)$ are obtained by fitting ReLU as described in Appendix A.4.2. For random initialization, we set $\alpha = 5$ and independently sample each z_i from a uniform distribution over $[-1, 1]$. The DNN weights are fully trained in all five settings.

As shown in Tab. 9, freezing the ReLU-initialized SQUAF degrades both Chamfer distance and IoU compared to the baseline on all four shapes. A possible reason is that a fixed SQUAF fitted over a finite interval may no longer approximate ReLU well as the pre-activation distribution changes during training. Training only $\boldsymbol{z}$ improves both metrics over the frozen variant on every shape, but still underperforms the baseline on average. Jointly training $(q, \boldsymbol{z}, \alpha)$, however, improves both metrics over the baseline on all four shapes. Thus, under the same ReLU initialization, adapting the quantization grid and smoothness together with the output amplitudes is more effective than adapting the output amplitudes alone.

Randomly initialized and fully trained SQUAF also outperforms the baseline on average in both metrics, showing that fitting an existing AF is not essential for effective learning in these experiments. Nevertheless, ReLU initialization followed by full SQUAF training achieves the best average Chamfer distance and IoU among the five compared settings. These findings support fitting the model's native AF as a practical initialization strategy when reusing an established training protocol, followed by adapting SQUAF to the task.

Table 9: Ablation study on 3D shape representation.

		Armadillo	Asian Dragon	Lucy	Thai Statue	Avg.
Chamfer ↓	PEMLP	3.453E-6	2.082E-6	2.313E-6	3.758E-6	2.901E-6
	SQUAF (ReLU init., frozen)	3.990E-6	2.516E-6	3.515E-6	3.971E-6	3.498E-6
	SQUAF (ReLU init., only $\boldsymbol{z}$ trained)	3.890E-6	2.114E-6	3.220E-6	3.683E-6	3.227E-6
	SQUAF (Random init., fully trained)	3.247E-6	2.019E-6	2.357E-6	3.142E-6	2.691E-6
	SQUAF (ReLU init., fully trained)	**3.246E-6**	**1.915E-6**	**1.982E-6**	**3.026E-6**	**2.542E-6**
IoU ↑	PEMLP	9.883E-1	9.815E-1	9.767E-1	9.599E-1	9.766E-1
	SQUAF (ReLU init., frozen)	9.868E-1	9.735E-1	9.575E-1	9.351E-1	9.632E-1
	SQUAF (ReLU init., only $\boldsymbol{z}$ trained)	9.872E-1	9.778E-1	9.585E-1	9.580E-1	9.704E-1
	SQUAF (Random init., fully trained)	9.899E-1	9.800E-1	9.708E-1	**9.680E-1**	9.772E-1
	SQUAF (ReLU init., fully trained)	**9.902E-1**	**9.823E-1**	**9.786E-1**	9.677E-1	**9.797E-1**

A.6 SSIM

Fig. 6 shows the SSIM results for SQUAF and other INR methods with various AFs. SQUAF achieves the highest average SSIM of 0.9833.

A.7 KD on ImageNet-1K

To further demonstrate SQUAF's effectiveness in KD, we conduct additional experiments on ImageNet-1K by distilling a ResNet-18 teacher into a MobileNet student (Howard et al., 2017), following the standard distillation setup in Tian et al. (2019). Table 10 reports the Top-1 accuracy

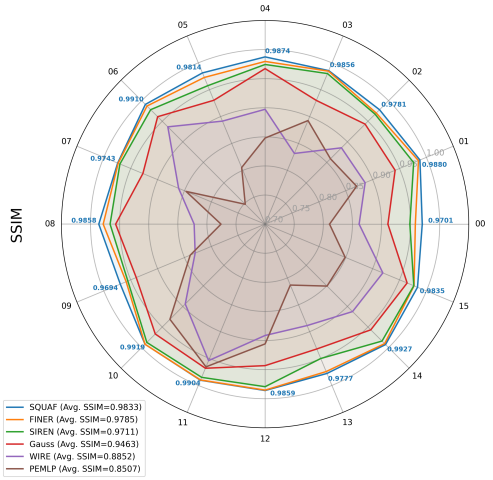


Figure 6: SSIM results for SQUAF and other INR methods with various AFs.

and number of trainable parameters for the teacher, the ReLU-based student, and the SQUAF-based student. The results show that a MobileNet equipped with SQUAF not only achieves a substantial improvement over the ReLU counterpart, but also outperforms the ResNet-18 teacher, despite having only 36.2% of the teacher's parameter count. In contrast, the MobileNet student using ReLU does not surpass the teacher. These findings provide additional evidence that SQUAF enhances the effectiveness of KD on a large-scale classification benchmark.

Table 10: Top-1 accuracy (%) on ImageNet-1K and the number of trainable parameters in a model.

Teacher	ResNet-18	
	69.76	11689512
Student	MobileNet	
ReLU	69.71	4231976
SQUAF	**70.87** (+1.16)	4232781

A.8 Model Depth Reduction on ImageNet-1K

To further demonstrate SQUAF's effectiveness in model depth reduction, we extend the experiments to ImageNet-1K using ViT-T. We configure SQUAF-equipped models with 11 and 10 transformer blocks and train them from scratch following the recipe given by Xu et al. (2023). Table 11 compares their Top-1 accuracies and numbers of trainable parameters with those of the standard 12-block ViT-T using GELU and SQUAF.

The 11-block ViT-T with SQUAF achieves 74.01% Top-1 accuracy, surpassing 73.90% achieved by the 12-block GELU baseline, while reducing the parameter count from 5.72M to 5.27M, i.e., by 7.8%. Thus, the smaller model preserves the baseline accuracy with a slightly higher reported result. Further reducing the depth to 10 blocks lowers the accuracy to 72.34%, below the baseline.

These results extend the evidence for model depth reduction from CIFAR-100 to ImageNet-1K. Under the tested training recipe, SQUAF enables a reduction of one transformer block while maintaining comparable accuracy. Together with the KD results in Appendix A.7, these findings provide additional evidence that SQUAF can facilitate model compression on a large-scale classification benchmark.

Table 11: Top-1 accuracy (%) on ImageNet-1K and the number of trainable parameters in a ViT-T.

GELU	73.90	5717416
SQUAF	75.00 (+1.10)	5717836
SQUAF (11-block)	74.01 (+0.11)	5272937
SQUAF (10-block)	72.34 (-1.56)	4828038

A.9 Number of Trainable Parameters in SQUAFs

In Tab.12, we list the number of trainable parameters for each DNN used in image classification, PDE solving, and INRs before incorporating SQUAFs, and the number of trainable parameters in SQUAFs corresponding to that DNN. Clearly, the number of additional trainable parameters introduced by SQUAFs is negligible compared to the original DNN sizes, ranging from 0.0003% to 0.13%.

Table 12: The number of trainable parameters in original DNNs compared to the number of additional trainable parameters introduced by SQUAFs.

CIFAR-100	Model	ShuffleNetV1	VGG-8	ResNet-56	VGG-13	ViT-T	EfficientFormer-L1
	Original Trainable Param.	949258	3965028	856420	9462180	5543716	11481728
	SQUAF Trainable Param.	35	35	35	35	35	35
	Increase Rate	0.0037%	0.00088%	0.0041%	0.00037%	0.00063%	0.00030%
ImageNet-1K	Model	ResNet-18	ResNet-34	ViT-T	Swin-T		
	Original Trainable Param.	11689512	21797672	5717416	28288354		
	SQUAF Trainable Param.	455	945	420	350		
	Increase Rate	0.0039%	0.0043%	0.0073%	0.0012%		
PDE	Model	MLP					
	Original Trainable Param.	40801					
	SQUAF Trainable Param.	35					
	Increase Rate	0.086%					
2D Image	Model	MLP					
	Original Trainable Param.	198915					
	SQUAF Trainable Param.	268					
	Increase Rate	0.13%					
3D Shape	Model	PEMLP					
	Original Trainable Param.	148225					
	SQUAF Trainable Param.	105					
	Increase Rate	0.071%					

A.10 Computational Efficiency

Tab. 13 shows the inference latency and training time of SQUAF compared to other AFs for ViT-T on CIFAR-100. It's shown that SQUAF is faster than other complex AFs such as PolyNorm and DiTAC. In addition, we also show the results for the 8-block ViT-T variant with SQUAF, which is faster than the 12-block ViT-T with the default GELU, while achieving better accuracy performance as shown in Fig. 3. Moreover, Tab. 14 compares the inference latency and training time for 3D INRs fitted to the Thai Statue. Although embedding SQUAF into the PEMLP leads to a moderate slowdown, it's still faster than other complex INRs such as WIRE and FINER.

A.11 Convergence Stability

In Fig. 7, we plot the convergence curves of validation accuracy for ResNet-34 and ViT-T trained on ImageNet-1K, including those for SQUAF and compared AFs. It's shown that the convergence of SQUAF is as stable as other well-established AFs, while achieving better final accuracy performance.

A.12 Embedding SQUAFs into LLM

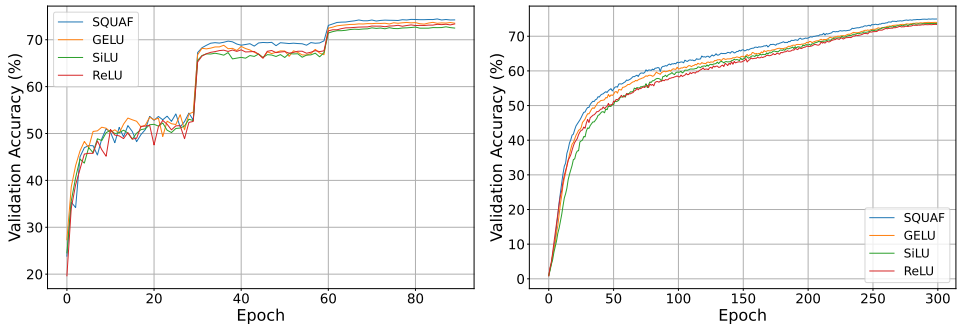
Using direct preference optimization (DPO) (Rafailov et al., 2023), we fine-tune GPT-2 (124M parameters) (Radford et al., 2019) on a combination of Stanford Human Preferences dataset (Etha-

Table 13: Comparison of computational efficiency. The inference latency is evaluated on an NVIDIA RTX A6000 GPU, and the training time is measured for distributed training on 8 NVIDIA RTX A6000 GPUs.

AF	Inference Latency (ms/img)	Training Time (s/epoch)
ReLU	5.727	9
GELU	5.896	9
LReLU	5.792	9
PReLU	5.834	9
SiLU	5.882	9
Swish	6.185	9
CRReLU	6.959	10
PolyReLU	7.241	11
PolyNorm	8.694	12
DiTAC	9.642	19
SQUAF	8.399	11
SQUAF (8-block)	5.262	8

Table 14: Comparison of computational efficiency. Both the inference latency and the training time are measured on an NVIDIA RTX A6000 GPU. The inference latency is measured by the wall time of each model's forward pass on a 512^3 grid with a batch size of 10K.

	Inference Latency (s)	Training Time (s/iteration)
PEMLP	2.822	0.038
Gauss	3.871	0.041
SIREN	3.300	0.039
WIRE	7.744	0.063
FINER	6.081	0.062
SQUAF	4.660	0.046



(a) ResNet-34 (b) ViT-T

Figure 7: Convergence of validation accuracy on ImageNet-1K.

yarajh et al., 2022) and Anthropic HH dataset (Bai et al., 2022), following the recipe given by Sun et al. (2025). The pretrained GPT-2 first undergoes supervised fine-tuning (SFT). The resulting model serves as the fixed reference policy, while another copy is further trained using DPO. SQUAFs are initialized with SQUAF-fitted-GELU with $q = 1$ and $k = 16$, embedded into GPT-2, and are trained in both the SFT and DPO stages. As suggested in Rafailov et al. (2023), we select the hyperparameter β in the DPO loss to be within $[0.1, 0.5]$, and compare SQUAF to the baseline GELU in GPT-2 at $\beta = 0.1, 0.2, 0.4$. Following Sun et al. (2025), we report evaluation reward accuracy (ERA), evaluation reward margin (ERM) and evaluation loss (EL) as performance metrics in Tab. 15, where SQUAF is shown to consistently outperform GELU. These preliminary results demonstrate the potential of using SQUAFs in LLMs.

Table 15: Comparison of performance metrics between SQUAF and GELU for GPT-2 fine-tuning based on DPO.

		ERA (%) ↑	ERM ↑	EL ↓
$\beta = 0.1$	GELU	62.11	0.0977	0.6646
	SQUAF	**62.50** (+0.39)	**0.1352**	**0.6510**
$\beta = 0.2$	GELU	59.76	0.1639	0.6733
	SQUAF	**62.89** (+3.13)	**0.1838**	**0.6544**
$\beta = 0.4$	GELU	58.59	0.3033	0.7184
	SQUAF	**61.33** (+2.74)	**0.4201**	**0.6547**

A.13 Embedding SQUAFs into KAN

KANs adopt fully-connected architectures but, unlike MLPs, place trainable 1D AFs on edges, while each neuron simply performs summation over its incoming signals. In the original KAN, each edge AF is a weighted sum of SiLU and a trainable spline. We construct SQUAF-KAN by replacing the spline with the non-uniform SQUAF in Eq. 6, while retaining the SiLU component for a fair comparison. Similar to the MLP notation used in Section 4, we denote a KAN architecture by an array of neuron counts across its layers. For example, a $[2, 10, 1]$ KAN has 2 input neurons, 10 hidden neurons, and 1 output neuron, containing 30 edge AFs in total.

Following the settings in Appendices B.1 and B.3 of the original KAN paper (Liu et al., 2025), we conduct PDE solving experiments on the Poisson equation with frequencies $n = 1, 2, 4$ and the Darcy flow equation, respectively. We adopt three architectures for evaluation, including a simple one $[2, 10, 1]$, a wide one $[2, 100, 1]$, and a deep one $[2, 10, 10, 10, 1]$. For the standard KAN, we use third-order splines with a grid size $G = 20$. For SQUAF-KAN, we set $L = 11$ to exactly match the standard KAN's parameter count, initialize parameters y_i uniformly within $[-1, 1]$, initialize each z_i independently by sampling from a uniform distribution over $[-1, 1]$, and initialize α with 5. Other aspects of experimental details are the same as in Liu et al. (2025); please see their released source code.

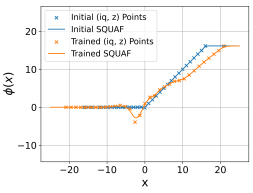
As shown in Tab. 16, SQUAF-KAN consistently outperforms KAN across all three architectures and four PDE problems, reducing the MSE of its KAN counterpart by up to 223×. These preliminary results demonstrate the potential of using SQUAFs as edge functions in KAN-style architectures.

Table 16: MSE results and numbers of trainable parameters of KAN and SQUAF-KAN on PDE solving tasks. Results are averaged over 3 runs.

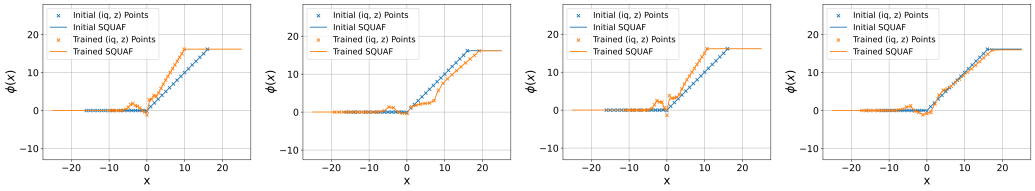
	Poisson ($n = 1$)	Poisson ($n = 2$)	Poisson ($n = 4$)	Darcy Flow	Param.
KAN [2,10,1] $G = 20$	2.90E-8	1.70E-5	2.66E-2	4.65E-4	750
SQUAF-KAN [2,10,1] $L = 11$	**1.48E-9**	**2.51E-7**	**1.60E-3**	**2.99E-5**	750
KAN [2,100,1] $G = 20$	2.92E-9	3.73E-7	5.95E-5	4.15E-6	7500
SQUAF-KAN [2,100,1] $L = 11$	**6.13E-10**	**1.81E-9**	**6.03E-7**	**1.86E-8**	7500
KAN [2,10,10,10,1] $G = 20$	2.86E-8	2.92E-6	6.54E-5	6.49E-6	5750
SQUAF-KAN [2,10,10,10,1] $L = 11$	**2.94E-9**	**6.42E-8**	**8.24E-6**	**1.97E-7**	5750

A.14 Visualization of SQUAFs

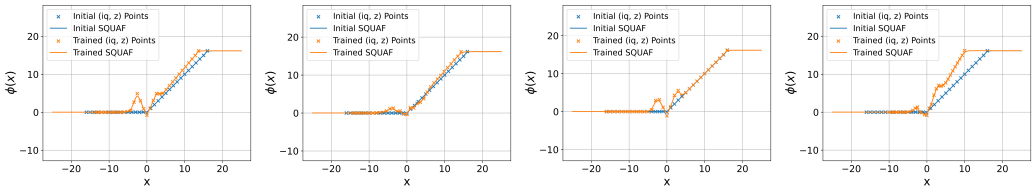
We visualize the initial and learned SQUAFs for ResNet-18, ResNet-34, ViT-T, and Swin-T on ImageNet-1K, shown in Figures 8, 9, 10, and 11, respectively. Interestingly, a dent near the origin is a very common pattern in the learned SQUAFs, which echoes the designs of SiLU and GELU where that dent is incorporated into the canonical ReLU.



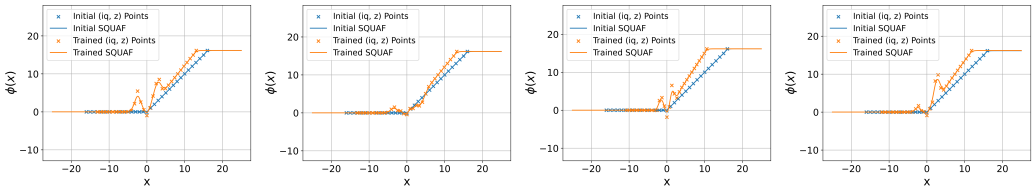
(a) Stem.



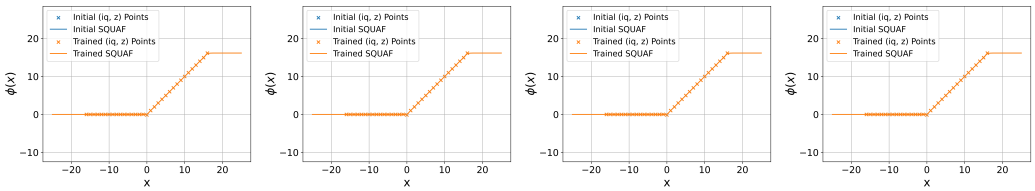
(b) Stage 1.



(c) Stage 2.

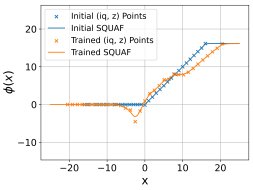


(d) Stage 3.

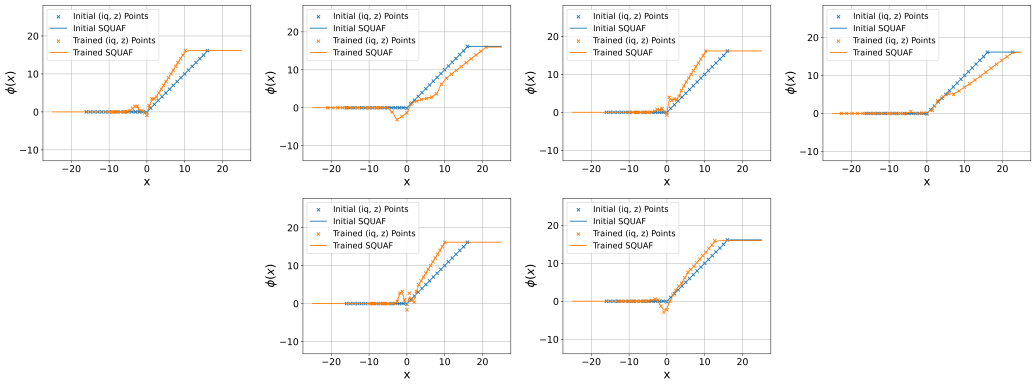


(e) Stage 4.

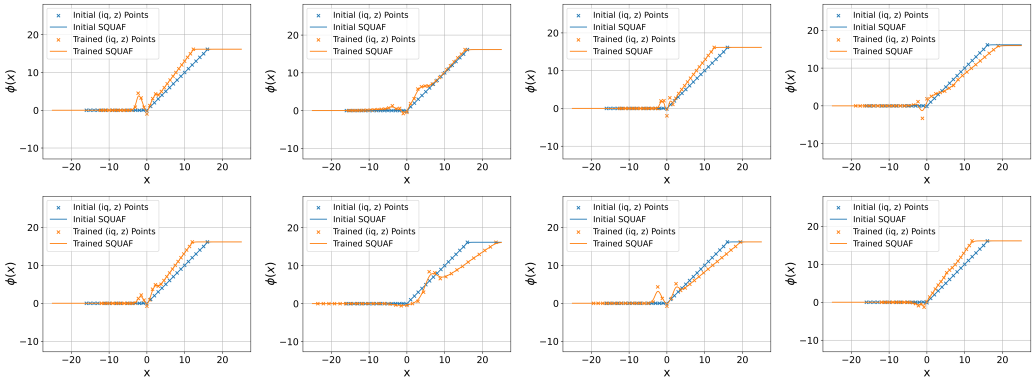
Figure 8: Visualization of SQUAFs in ResNet-18. The plots are arranged in raster order (top-to-bottom, left-to-right) corresponding to increasing depth in the network.



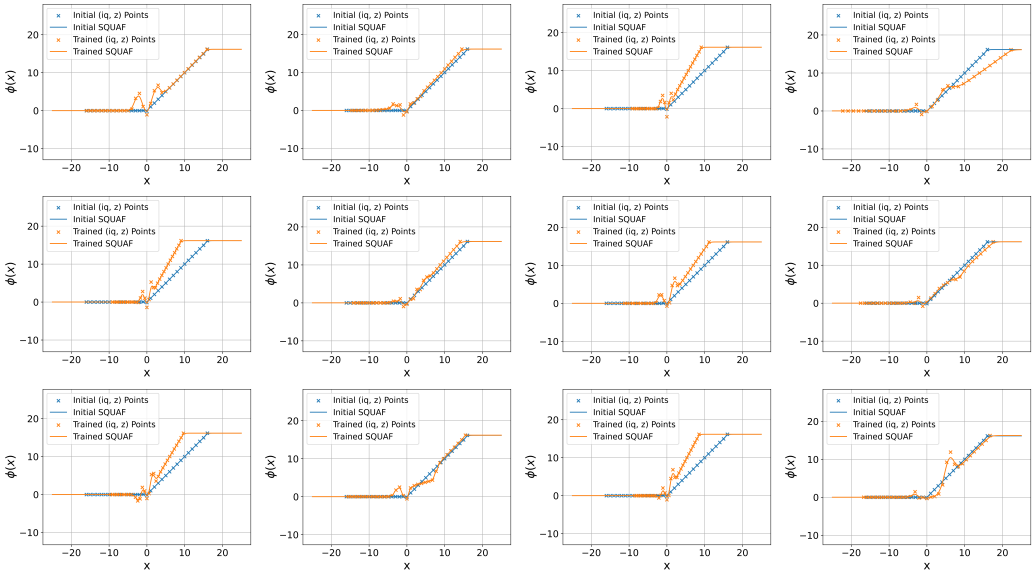
(a) Stem.



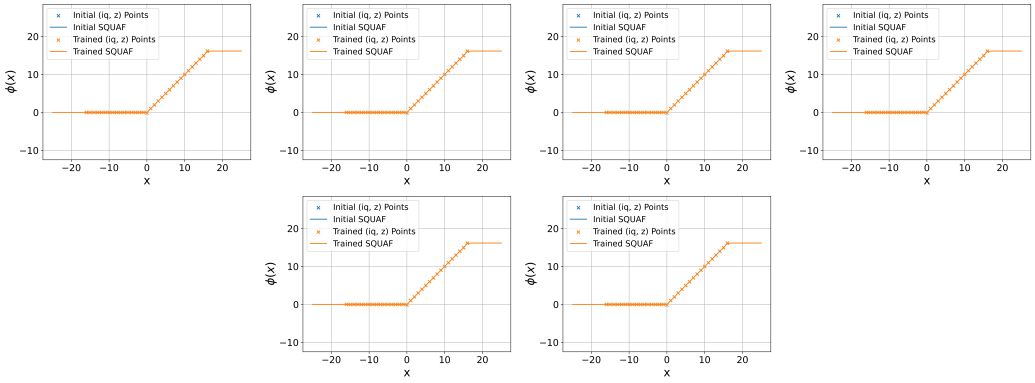
(b) Stage 1.



(c) Stage 2.



(d) Stage 3.



(e) Stage 4.

Figure 9: Visualization of SQUAFs in ResNet-34.

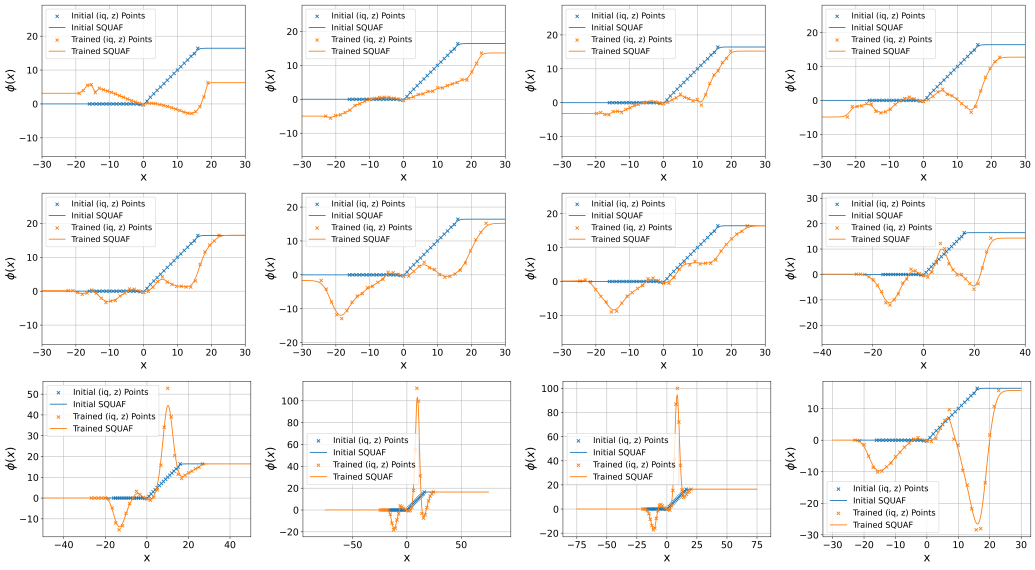
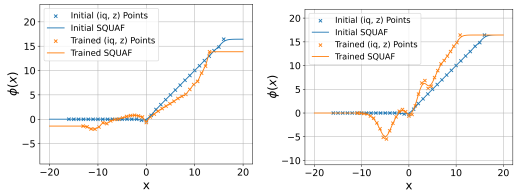
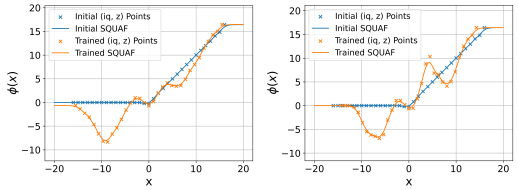


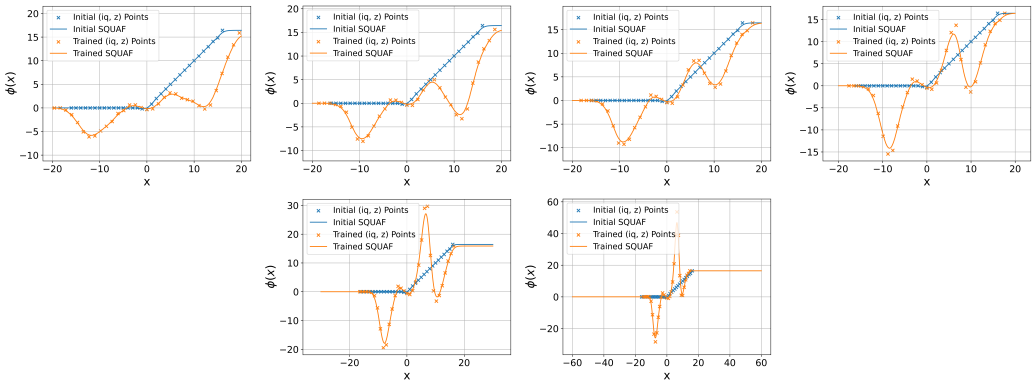
Figure 10: Visualization of SQUAFs in ViT-T.



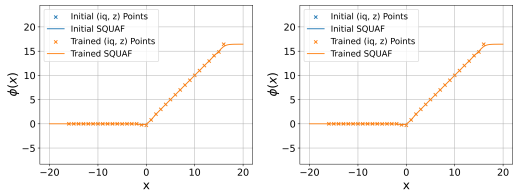
(a) Stage 1.



(b) Stage 2.



(c) Stage 3.



(d) Stage 4.

Figure 11: Visualization of SQUAFs in Swin-T.