More Expressive Feedforward Layers: Part I. Token-Adaptive Mixing of Activations

Mingze Wang[2,*,§,†] **Jinbo Wang**[2,§] **Yikuan Xia**[1] **Kai Shen**[1] **Shu Zhong**[1,†]

[1]ByteDance Seed [2]Peking University

mingzewang.math@gmail.com zhongshu@bytedance.com

Abstract

Feedforward network (FFN) layers account for a large fraction of parameters and nonlinear expressivity in Transformer-based large language models (LLMs). Despite the evolution from ReLU and GELU to gated variants such as SwiGLU, most FFN designs still use a single fixed activation function, applying the same nonlinear transformation to all tokens. In this work, we propose *Mixture of Activations* (MoA), a token-adaptive FFN design that mixes a dictionary of activation functions using lightweight input-dependent gates while sharing the same linear projections. As an input-independent counterpart, we also introduce learnable activations (LA), which form linear combinations of activation functions for both ReLU-type and SwiGLU-type FFNs. *Theoretically*, we establish strict finite-width expressive separations among fixed-activation FFNs, LA, and MoA: LA strictly contains fixed-activation FFNs, while MoA strictly contains LA, with the additional expressivity arising from input-dependent nonlinear hybridization. *Empirically*, we evaluate MoA through extensive pre-training experiments on dense and MoE language models ranging from 0.12B to 2B parameters under different token budgets, optimizers, and learning rate schedules. MoA consistently achieves lower terminal loss and exhibits more favorable scaling behavior than well-tuned baselines, with minimal parameter and computational overhead. These results suggest that token-adaptive activation mixing is a simple and effective mechanism for improving FFN expressivity in LLMs.

1 Introduction

Feedforward network (FFN) layers are a central component of Transformer-based large language models (LLMs) and account for a large fraction of model parameters (Vaswani et al., 2017). While self-attention enables token-to-token information exchange, FFN layers apply nonlinear transformations independently to each token and have been linked to knowledge and memory storage in LLMs (Dai et al., 2022; Wang and E, 2024; Zhong et al., 2025). Improving FFN layers is therefore an important direction for enhancing the expressivity of modern LLMs.

The expressive power of FFN layers is largely determined by their activation functions, whose design has evolved substantially with LLM architectures. The original Transformer used ReLU (Vaswani et al., 2017), while later language models widely adopted smooth nonlinearities such as GELU (Devlin et al., 2019; Radford et al., 2019). More recently, multiplicative gated activations, such as SwiGLU (Shazeer, 2020), have become standard in modern LLMs (Touvron et al., 2023).

Despite these advances, most FFN designs still rely on a single fixed activation function. This imposes the same nonlinear form across layers, tokens, and channels, which may restrict representation: (i)

* Work done at ByteDance Seed.
§ Equal contribution.
† Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

the desired nonlinearity may not be well captured by any single hand-designed activation function; (ii) different tokens may benefit from different nonlinearities. Prior work addresses the first issue by learning linear combinations of activation functions (Manessi and Rozza, 2018; Sütfeld et al., 2018). However, these combinations are typically input-independent, and therefore apply the same activation hybrid to all tokens.

In this work, we investigate how to improve FFN expressivity through token-adaptive nonlinear hybridization of activation functions. **Our contributions** are summarized as follows:

- We propose mixture-of-activations (**MoA**), a token-adaptive FFN design that uses lightweight input-dependent gates to mix activation functions for each token. Unlike mixture-of-experts (MoE), which routes tokens to different parameterized experts, MoA mixes activation functions while sharing the same linear projections. This design enables token-adaptive nonlinear transformations with minimal computation and parameter overhead. As an input-independent counterpart, we also introduce learnable activations (**LA**) for both ReLU-type and SwiGLU-type FFNs, which form linear combinations over a dictionary of modern activation functions.
- Theoretically, we establish **strict expressive separations** among standard FFNs, LA, and MoA. Specifically, (i) at the same width, LA contains all standard FFNs with a single fixed activation, and there exists a function representable by width-1 LA but not by any finite-width standard FFN with a single fixed activation. (ii) similarly, MoA contains LA at the same width, and there exists a function representable by width-1 MoA but not by any finite-width LA network. These results show that the expressive advantage of MoA arises from nonlinear, token-adaptive hybridization of activation functions.
- Empirically, we conduct **extensive language pre-training experiments** to evaluate MoA. We consider both dense and MoE LLMs, with model sizes ranging *from 0.12B to 2B* parameters, training on high-quality pre-training corpus under various token budgets. We evaluate multiple training configurations, including AdamW and Muon optimizers, as well as cosine decay and warmup-stable-decay learning rate (lr) schedules. MoA consistently achieves *lower terminal loss* and exhibits *more favorable scaling behavior* than well-tuned baselines, demonstrating its potential to scale to larger models. Experiments on dense models further show that MoA *tolerates larger lr* than the baseline. For MoE models, even without additional MoA-specific lr tuning, MoA improves over well-tuned Muon-trained baselines. Finally, we extend MoA to self-supervised vision pre-training, where it continues to improve the convergence.

2 Related Works

Evolution of activation functions. Activation functions are a central design choice in neural networks, as they determine nonlinear expressivity. Early deep networks commonly used sigmoidal or hyperbolic tangent nonlinearities, whereas rectified linear units (ReLU) improved optimization by alleviating saturation and inducing sparse activations (Glorot et al., 2011). Subsequent work introduced rectifier variants and smooth nonlinearities, such as GELU, Swish, and SiLU (He et al., 2015; Clevert et al., 2015; Klambauer et al., 2017; Hendrycks and Gimpel, 2016; Ramachandran et al., 2018; Elfwing et al., 2018; Misra, 2019). The original Transformer used ReLU in its FFN layers (Vaswani et al., 2017), while later language models widely adopted GELU (Devlin et al., 2019; Radford et al., 2019). More recently, multiplicative gated activations have become standard in LLMs. GLU was introduced as a gating mechanism for language modeling (Dauphin et al., 2017), and the variants such as GEGLU and SwiGLU were shown to improve FFNs (Shazeer, 2020). SwiGLU is widely adopted in modern LLMs. *Unlike these works*, which design or select a single activation function, our method constructs hybrid activations from a dictionary of candidate nonlinearities.

Learnable and combined activations. A related line of work makes activation functions trainable rather than fixed (Apicella et al., 2021). Parametric activations, such as PReLU (He et al., 2015), introduce a small number of learnable shape parameters. Adaptive piecewise linear units learn neuron-wise piecewise linear activations (Agostinelli et al., 2014), while Maxout units learn a convex piecewise linear activation by taking the maximum over affine functions (Goodfellow et al., 2013). Other methods explicitly learn linear combinations of activation functions (Manessi and Rozza, 2018; Sütfeld et al., 2018; Goyal et al., 2019; Zhuo et al., 2024). Our learnable activation (LA) variant is closely related to methods that learn input-independent activation combinations. However, LA is designed for both standard FFNs and SwiGLU-type gated FFNs, where activation combinations can

be applied to different branches. Recent work on KANs introduces learnable activation functions on network edges (Liu et al., 2024); however, their linear combinations of activation bases remain input-independent (Liu et al., 2024). *In contrast*, our MoA makes the mixing coefficients input-dependent, allowing different tokens to use different activation hybrids within the same FFN layer.

Mixture models and conditional computation. MoA is also related to mixture models and conditional computation. Classical mixture-of-experts models use a gating network to combine specialized expert networks according to the input (Jacobs et al., 1991; Jordan and Jacobs, 1994). Modern sparse MoE layers scale this idea by routing each token to a small subset of parameterized experts, increasing model capacity without a proportional increase in computation (Shazeer et al., 2017). *In comparison*, MoA applies input-dependent gating at the activation level: instead of routing tokens to different parameter experts, it mixes activation functions while sharing the same linear projections. Thus, MoA provides token-adaptive nonlinear transformations with substantially smaller architectural changes than standard MoE layers.

To the best of our knowledge, MoA is the first FFN design that performs input-dependent mixing over a heterogeneous activation dictionary within shared linear projections. It thereby combines the flexibility of learnable activations with the token adaptivity of conditional computation.

3 Method

In this section, we introduce FFN layers and their variants. We first review two standard FFN forms used in modern Transformers. We then introduce learnable activations (LA), which use input-independent linear combinations of activation functions. Finally, we propose mixture-of-activations (MoA), whose input-dependent mixing weights yield a nonlinear, adaptive hybrid of activations.

Notations. We denote the Hadamard product by $\odot$. For a finite set $\mathcal{K}$, $|\mathcal{K}|$ denotes its cardinality. For a positive integer m, let $[m] = \{1, \dots, m\}$. The Gaussian distribution with mean μ and variance σ^2 is denoted by $\mathcal{N}(\mu, \sigma^2)$. We use the activations $\mathrm{ReLU}(x) = \max\{0, x\}$, $\mathrm{ReLU}^2(x) = \max\{0, x\}^2$, $\mathrm{LeakyReLU}(x) = \max\{x, \eta x\}$, $\mathrm{sigmoid}(x) = 1/(1 + e^{-x})$, $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, $\mathrm{SiLU}(x) = x\mathrm{sigmoid}(x)$, $\mathrm{GELU}(x) = x\Phi(x)$, where $\eta \in (0, 1)$ is fixed and Φ is the cumulative distribution function of $\mathcal{N}(0, 1)$. The identity activation is denoted by $\mathrm{Id}(x) = x$. All activation functions are applied elementwise to vector inputs.

3.1 Standard FFN

Let $\boldsymbol{x} \in \mathbb{R}^d$ be the input to an FFN layer, and let D denote the hidden width. For simplicity, we take the input and output dimensions to be d. We consider two widely used standard FFN forms.

Type-I FFN. A Type-I FFN is defined as

$$f(\boldsymbol{x}) = \boldsymbol{W}_2\, \sigma(\boldsymbol{W}_1 \boldsymbol{x}), \tag{1}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{D\times d}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d\times D}$, and σ is the activation function. Transformer architectures typically use $D = 4d$, with common choices of σ including ReLU, ReLU^2, and GELU.

Type-II FFN. A Type-II FFN introduces multiplicative gating structure:

$$g(\boldsymbol{x}) = \boldsymbol{W}_3\big(\sigma(\boldsymbol{W}_1 \boldsymbol{x}) \odot (\boldsymbol{W}_2 \boldsymbol{x})\big), \tag{2}$$

where $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{D\times d}$ and $\boldsymbol{W}_3 \in \mathbb{R}^{d\times D}$. A representative example is SwiGLU, obtained by setting $\sigma = \mathrm{SiLU}$. To match the parameter budget of a Type-I FFN with width $4d$, many LLM architectures set $D = 8d/3$ for this form.

3.2 Linear Hybrid: Learnable Activations

We first consider a direct input-independent hybridization of activation functions. Let $\mathcal{K} = \{\sigma_1, \dots, \sigma_m\}$ be a small dictionary of candidate activations. For Type-I FFNs, we use

$$\mathcal{K}_{\mathcal{F}} \subseteq \{\mathrm{ReLU}^2, \mathrm{GELU}, \mathrm{SiLU}, \mathrm{LeakyReLU}, \mathrm{ReLU}, \tanh\}$$

whereas for Type-II FFNs, we use

$$\mathcal{K}_{\mathcal{G}} \subseteq \{\mathrm{Id}, \mathrm{GELU}, \mathrm{SiLU}, \mathrm{ReLU}, \mathrm{LeakyReLU}, \mathrm{ReLU}^2, \tanh\}.$$

We include the identity Id for Type-II FFNs because the second branch of SwiGLU (2) is identity.

Type-I LA. For Type-I FFNs, LA replaces the single nonlinearity with a linear combination:

$$f_{\mathrm{LA}}(\boldsymbol{x}) = \boldsymbol{W}_2\left(\sum_{k=1}^{|\mathcal{K}_{\mathcal{F}}|} \alpha_k\, \sigma_k(\boldsymbol{W}_1\boldsymbol{x})\right), \tag{3}$$

where $\alpha_k \in \mathbb{R}$ are trainable scalar coefficients. Since the coefficients are shared across inputs, LA is a linear, input-independent hybrid of activation functions. Moreover, all candidate activations share the same linear projection $\boldsymbol{W}_1\boldsymbol{x}$, so LA adds only $|\mathcal{K}_{\mathcal{F}}|$ scalar parameters.

Type-II LA. For Type-II FFNs, let $\boldsymbol{y} = \boldsymbol{W}_1\boldsymbol{x}, \boldsymbol{z} = \boldsymbol{W}_2\boldsymbol{x}$. Then SwiGLU takes the form $g(\boldsymbol{x}) = \boldsymbol{W}_3(\mathrm{SiLU}(\boldsymbol{y}) \odot \boldsymbol{z})$. Its multiplicative structure gives rise to the following LA variants.

- **One-sided LA.** This variant keeps the gating branch fixed and replaces the linear branch with a learnable activation mixture:

$$g_{\text{one-LA}}(\boldsymbol{x}) = \boldsymbol{W}_3\left(\mathrm{SiLU}(\boldsymbol{y}) \odot \sum_{k=1}^{|\mathcal{K}_{\mathcal{G}}|} \alpha_k\, \sigma_k(\boldsymbol{z})\right), \tag{4}$$

  where $\alpha_k \in \mathbb{R}$ are trainable scalar coefficients.
- **Bi-sided LA.** This variant applies learnable activation mixtures to both branches:

$$g_{\text{bi-LA}}(\boldsymbol{x}) = \boldsymbol{W}_3\left(\sum_{k=1}^{|\mathcal{K}_{\mathcal{G}}|} \beta_k\, \sigma_k(\boldsymbol{y}) \odot \sum_{\ell=1}^{|\mathcal{K}_{\mathcal{G}}|} \alpha_\ell\, \sigma_\ell(\boldsymbol{z})\right), \tag{5}$$

  where $\beta_k \in \mathbb{R}$ and $\alpha_\ell \in \mathbb{R}$ are trainable scalar coefficients.
- **Quadratic LA.** This variant directly mixes pairwise activation products across the two branches:

$$g_{\text{qd-LA}}(\boldsymbol{x}) = \boldsymbol{W}_3\left(\sum_{1\leqslant k\leqslant \ell\leqslant |\mathcal{K}_{\mathcal{G}}|} \alpha_{k\ell}\, \sigma_k(\boldsymbol{y}) \odot \sigma_\ell(\boldsymbol{z})\right), \tag{6}$$

  where $\alpha_{k\ell} \in \mathbb{R}$ are trainable scalar coefficients.

3.3 Nonlinear Hybrid: Mixture of Activations

LA uses fixed mixing coefficients and therefore defines a global activation hybrid *shared for all tokens*. We now introduce Mixture of Activations (MoA), whose mixing weights *depend on the input token*. This input dependence makes the activation hybrid nonlinear and adaptive.

MoA uses a lightweight gating function ϕ to generate input-dependent mixing weights. For training stability, we choose bounded gates, such as $\mathrm{sigmoid}$, $\tanh$, or a $\mathrm{softmax}$ over activation indices. We empirically compare these choices in Section 5.

Type-I MoA. For Type-I FFNs, MoA is defined as

$$f_{\mathrm{MoA}}(\boldsymbol{x}) = \boldsymbol{W}_2\left(\sum_{k=1}^{|\mathcal{K}_{\mathcal{F}}|} \pi_k(\boldsymbol{x})\, \sigma_k(\boldsymbol{W}_1\boldsymbol{x})\right), \quad \pi_k(\boldsymbol{x}) = \phi(\boldsymbol{u}_k^\top \boldsymbol{x}), \tag{7}$$

where $\boldsymbol{u}_k \in \mathbb{R}^d$ are trainable gating parameters. Unlike LA, the coefficients $\pi_k(\boldsymbol{x})$ vary across tokens, allowing different inputs to use different activation hybrids.

Type-II MoA. For Type-II FFNs, we again let $\boldsymbol{y} = \boldsymbol{W}_1\boldsymbol{x}$ and $\boldsymbol{z} = \boldsymbol{W}_2\boldsymbol{x}$. Analogously to Type-II LA, Type-II MoA admits three variants.

- **One-sided MoA.** This variant keeps the SwiGLU gating branch fixed and replaces the second branch with a token-adaptive activation mixture:

$$g_{\text{one-MoA}}(\boldsymbol{x}) = W_3\left(\mathrm{SiLU}(\boldsymbol{y}) \odot \sum_{k=1}^{|\mathcal{K}_{\mathcal{G}}|} \pi_k(\boldsymbol{x})\, \sigma_k(\boldsymbol{z})\right), \quad \pi_k(\boldsymbol{x}) = \phi(\boldsymbol{u}_k^\top \boldsymbol{x}), \tag{8}$$

  where $\boldsymbol{u}_k \in \mathbb{R}^d$ are trainable gating parameters.
- **Bi-sided MoA.** This variant uses token-dependent mixtures in both branches:

$$g_{\text{bi-MoA}}(\boldsymbol{x}) = \boldsymbol{W}_3\left(\sum_{k=1}^{|\mathcal{K}_{\mathcal{G}}|} \rho_k(\boldsymbol{x})\, \sigma_k(\boldsymbol{y}) \odot \sum_{\ell=1}^{|\mathcal{K}_{\mathcal{G}}|} \pi_\ell(\boldsymbol{x})\, \sigma_\ell(\boldsymbol{z})\right), \quad \begin{cases}\rho_k(\boldsymbol{x}) = \phi(\boldsymbol{v}_k^\top \boldsymbol{x}) \\ \pi_\ell(\boldsymbol{x}) = \phi(\boldsymbol{u}_\ell^\top \boldsymbol{x})\end{cases}, \tag{9}$$

  where $\boldsymbol{u}_k, \boldsymbol{v}_\ell \in \mathbb{R}^d$ are trainable gating parameters.

- **Quadratic MoA.** This variant makes the quadratic activation-pair coefficients input-dependent:

$$g_{\text{qd-MoA}}(\boldsymbol{x}) = \boldsymbol{W}_3 \left(\sum_{1 \le k \le \ell \le |\mathcal{K}_\mathcal{G}|} \pi_{k\ell}(\boldsymbol{x})\, \sigma_k(\boldsymbol{y}) \odot \sigma_\ell(\boldsymbol{z}) \right), \quad \pi_{k\ell}(\boldsymbol{x}) = \phi(\boldsymbol{u}_{k\ell}^\top \boldsymbol{x}), \tag{10}$$

where $\boldsymbol{u}_k \in \mathbb{R}^d$ are trainable gating parameters. This form allows the preferred activation pair to vary across tokens.

Soft versus hard gating. MoA uses soft gating by default, so all candidate activations contribute for each input. Alternatively, one could use MoE-style hard gating to select only a subset of activations for each input. We do not adopt hard gating in this work: unlike MoE, whose experts have distinct parameters, MoA shares the same linear mappings across activations. Therefore, soft gating introduces little additional overhead, and sparsity is not essential in our setting.

4 Theory

We establish strict expressive separations among FFNs with fixed activations, learnable activations, and mixture-of-activations. Although standard Type I FFNs are universal approximators, universality requires growing width, and the width needed to achieve a prescribed accuracy may be prohibitively large (Bach, 2017). We therefore compare expressivity at fixed finite width.

Following standard expressivity analyses (Barron, 1992; 1993), we consider scalar-valued outputs; vector-valued outputs follow componentwise. We also include bias terms, since two-layer neural networks without biases may lose universal approximation properties and lead to degenerate results. Accordingly, we use the augmented input $\bar{\boldsymbol{x}} = (\boldsymbol{x}, 1) \in \mathbb{R}^{d+1}$.

4.1 Expressive Separation for Type-I FFNs

Function classes. Let $\mathcal{K} = \{\sigma_1, \dots, \sigma_6\} = \{\mathrm{ReLU}, \mathrm{ReLU}^2, \mathrm{LeakyReLU}, \mathrm{GELU}, \mathrm{SiLU}, \tanh\}$ be the activation dictionary. For a fixed width m and activation $\sigma \in \mathcal{K}$, define the standard two-layer neural network class $\mathcal{F}_\sigma^{(m)} := \left\{ \boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sigma(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) : a_k \in \mathbb{R},\ \boldsymbol{w}_k \in \mathbb{R}^{d+1} \right\}$. The LA class is $\mathcal{F}_{\text{LA}}^{(m)} := \{\boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sum_{c=1}^6 \alpha_c \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) : a_k, \alpha_c \in \mathbb{R},\ \boldsymbol{w}_k \in \mathbb{R}^{d+1}\}$. The MoA class is $\mathcal{F}_{\text{MoA}}^{(m)} := \{\boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sum_{c=1}^6 \tanh(\boldsymbol{u}_c^\top \bar{\boldsymbol{x}}) \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) : a_k \in \mathbb{R},\ \boldsymbol{u}_c, \boldsymbol{w}_k \in \mathbb{R}^{d+1}\}$. For concreteness, we use $\tanh$ gates in the analysis; analogous arguments apply to other bounded gates.

We show that MoA is strictly more expressive than both LA and fixed-activation FFNs at the same finite width. For a domain $\Omega \subset \mathbb{R}^d$, define the $\mathcal{W}^{1,\infty}$-norm as $\|f\|_{\mathcal{W}^{1,\infty}} := \|f\|_{L^\infty(\Omega)} + \|\nabla f\|_{L^\infty(\Omega)}$, where ∇f denotes the weak gradient.

Theorem 4.1 (Strict expressive hierarchy for Type-I FFNs)**.** *Consider $d \ge 2$ and let $\Omega = [-1, 1]^d$. For every width $m \ge 1$, the following strict inclusions hold:*

$$\bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)} \subsetneq \mathcal{F}_{\text{LA}}^{(m)} \subsetneq \mathcal{F}_{\text{MoA}}^{(m)}.$$

Specifically, the strictness is witnessed by the following constructions:

- *There exists a width-1 target $T_{\text{LA}}(\boldsymbol{x}) = \mathrm{ReLU}(x_1) + \mathrm{ReLU}^2(x_1)$ such that $T_{\text{LA}} \in \mathcal{F}_{\text{LA}}^{(1)} \subset \mathcal{F}_{\text{MoA}}^{(1)}$, but, for every $m \ge 1$, $\inf_{f \in \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}} \|T_{\text{LA}} - f\|_{\mathcal{W}^{1,\infty}} \ge \frac{1}{m+1}$. Hence, $T_{\text{LA}} \notin \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}$.*
- *For any $\lambda > 0$, there exists a width-1 target $T_{\text{MoA}}(\boldsymbol{x}) = \tanh(\lambda x_1)\mathrm{ReLU}(x_2)$ such that $T_{\text{MoA}} \in \mathcal{F}_{\text{MoA}}^{(1)}$, but, for every $m \ge 1$, $\inf_{f \in \mathcal{F}_{\text{LA}}^{(m)}} \|T_{\text{MoA}} - f\|_{\mathcal{W}^{1,\infty}} \ge \frac{1}{2}\tanh(\lambda)$. Hence, $T_{\text{MoA}} \notin \mathcal{F}_{\text{LA}}^{(m)}$ and $T_{\text{MoA}} \notin \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}$.*

The full proof of Theorem 4.1 is provided in Appendix B.1. We summarize the main ideas below.

Main insight of the inclusions. The inclusion $\mathcal{F}_\sigma^{(m)} \subset \mathcal{F}_{\text{LA}}^{(m)}$ is immediate, since a fixed activation is a special case of LA: setting one activation coefficient to be one and the others to be zero recovers any fixed-activation FFN. The inclusion $\mathcal{F}_{\text{LA}}^{(m)} \subset \mathcal{F}_{\text{MoA}}^{(m)}$ follows because MoA gates can realize constants

mixing weights by using only the bias coordinate. Thus, MoA contains LA as the special case in which the gates are input-independent.

Main insight of the separation. The two strict inclusions arise from different sources of flexibility.

- LA allows each neuron to use a learned linear combination of several activation primitives. For example, $T_{\mathrm{LA}}(\boldsymbol{x}) = \mathrm{ReLU}(x_1) + \mathrm{ReLU}^2(x_1)$ can be represented by a single LA neuron by combining the ReLU and ReLU^2 branches. In contrast, a fixed-activation FFN cannot simultaneously reproduce both local behaviors at same finite width. ReLU-type networks are piecewise linear and therefore have piecewise-constant derivatives, which cannot represent the curvature induced by ReLU^2. Smooth activations such as GELU, SiLU, and $\tanh$ have continuous derivatives and therefore cannot reproduce the derivative discontinuity of ReLU at the origin. Thus, LA is strictly more expressive than any single fixed activation at the same width.
- LA uses global coefficients α_c, so the same activation hybrid is applied to every input. MoA replaces these constants with gates of the form $\tanh(\boldsymbol{u}_c^\top \bar{\boldsymbol{x}})$, making the activation mixture input-dependent. This enables multiplicative, input-adaptive interactions across coordinates. For example, $T_{\mathrm{MoA}}(\boldsymbol{x}) = \tanh(\lambda x_1)\mathrm{ReLU}(x_2)$ can be represented by a single MoA neuron: the $\mathrm{ReLU}(x_2)$ term creates a threshold feature in x_2, while the gate $\tanh(\lambda x_1)$ modulates its amplitude according to x_1. In contrast, LA cannot realize this behavior at fixed width because its activation weights are constants. This input-dependent modulation is the central expressive advantage of MoA.

4.2 Expressive Separation for Type-II FFNs

The Type-II setting is more subtle because even a fixed activation pair already introduces a multiplicative structure, $\sigma_p(\boldsymbol{w}^\top \bar{\boldsymbol{x}})\sigma_q(\boldsymbol{u}^\top \bar{\boldsymbol{x}})$.

Function classes. Let $\mathcal{K} = \{\sigma_1, \dots, \sigma_7\} = \{\mathrm{id}, \mathrm{ReLU}, \mathrm{ReLU}^2, \mathrm{LeakyReLU}, \mathrm{GELU}, \mathrm{SiLU}, \tanh\}$ and use the augmented input $\bar{\boldsymbol{x}} = (\boldsymbol{x}, 1)$. For fixed $p, q \in [7]$ and width m, define the fixed-activation Type-II FFN class $\mathcal{G}^{(m)}_{\sigma_p,\sigma_q} := \{\boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}})\sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) : a_k \in \mathbb{R},\ \boldsymbol{w}_k, \boldsymbol{u}_k \in \mathbb{R}^{d+1}\}$. The qd-LA in (6) class is $\mathcal{G}^{(m)}_{\mathrm{LA}} := \{\boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sum_{1 \leqslant p \leqslant q \leqslant 7} \alpha_{p,q} \sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}})\sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) : a_k, \alpha_{p,q} \in \mathbb{R},\ \boldsymbol{w}_k, \boldsymbol{u}_k \in \mathbb{R}^{d+1}\}$. The qd-MoA in (10) is $\mathcal{G}^{(m)}_{\mathrm{MoA}} := \{\boldsymbol{x} \mapsto \sum_{k=1}^m a_k \sum_{1 \leqslant p \leqslant q \leqslant 7} \tanh(\boldsymbol{v}_{p,q}^\top \bar{\boldsymbol{x}})\sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}})\sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) : a_k \in \mathbb{R},\ \boldsymbol{w}_k, \boldsymbol{u}_k, \boldsymbol{v}_{p,q} \in \mathbb{R}^{d+1}\}$. For clarity, our theory focuses on the qd-MoA with $\tanh$ gates. The same proof strategy applies to other MoA variants and other bounded gating functions such as sigmoid.

Theorem 4.2 (Strict expressive hierarchy for Type-II FFNs)**.** *Consider $d \geq 2$. For every width $m \geq 1$,*

$$\bigcup_{p,q \in [7]} \mathcal{G}^{(m)}_{\sigma_p,\sigma_q} \subsetneq \mathcal{G}^{(m)}_{\mathrm{LA}} \subsetneq \mathcal{G}^{(m)}_{\mathrm{MoA}}.$$

More precisely:

- *There exists a width-1 target $T_{\mathrm{LA}}(\boldsymbol{x}) = \mathrm{ReLU}(x_2)(\mathrm{ReLU}(x_1) + \tanh(x_1))$ such that $T_{\mathrm{LA}} \in \mathcal{G}^{(1)}_{\mathrm{LA}} \subset \mathcal{G}^{(1)}_{\mathrm{MoA}}$, but, for every $m \geq 1$, $T_{\mathrm{LA}} \notin \bigcup_{p,q \in [7]} \mathcal{G}^{(m)}_{\sigma_p,\sigma_q}$.*
- *There exists a width-1 target $T_{\mathrm{MoA}}(\boldsymbol{x}) = \mathrm{ReLU}(x_2)\mathrm{ReLU}(x_1)\tanh(x_1)$ such that $T_{\mathrm{MoA}} \in \mathcal{G}^{(1)}_{\mathrm{MoA}}$, but, for every $m \geq 1$, $T_{\mathrm{MoA}} \notin \mathcal{G}^{(m)}_{\mathrm{LA}}$ and $T_{\mathrm{MoA}} \notin \bigcup_{p,q \in [7]} \mathcal{G}^{(m)}_{\sigma_p,\sigma_q}$.*

The proof of Theorem 4.2 is deferred to Appendix B.2. The key idea parallels that of Theorem 4.1. Although standard Type-II FFNs already contain a two-factor multiplicative structure, they cannot represent simple input-adaptive three-factor interactions at fixed width. MoA overcomes this limitation by introducing an input-dependent activation mixture, yielding a strict expressive gain over both fixed-activation and learnable-activation Type-II FFNs.

5 Experiments

We mainly evaluate our methods on LLM pre-training across diverse architectures, model scales, optimizers, and learning rate (lr) schedules. The main configurations are summarized below; additional implementation details are deferred to Appendix A.

Models and Dataset. We conduct experiments on two widely used LLM architectures: **dense** models (Llama (Touvron et al., 2023)) and **MoE** models (LlamaMoE). Model sizes range **from 0.12B to 2B** parameters. All models are trained on a high-quality pre-training corpus.

Token Budget. Dense models are trained with approximately *20* tokens per model parameter (TPP) by default, following the Chinchilla-optimal regime (Hoffmann et al., 2022). We additionally evaluate the 0.12B dense model with *100* TPP. For MoE models, we use approximately *100* TPP, which is larger than the Chinchilla-optimal budget and is closer to industrial pre-training practice.

Optimization. We evaluate our method under multiple training configurations, including different optimizers and lr schedules.

- **Dense models.** We use AdamW as the baseline optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2017). Following standard Llama pre-training practice (Touvron et al., 2023), we set $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay $\lambda = 0.1$, and gradient clipping threshold 1.0. We use the `cos` lr schedule: a linear warm-up to the peak lr `lr_max`, followed by cosine decay to the terminal lr.
- **MoE models.** We use the `Muon` optimizer (Keller et al., 2024), which has recently shown strong efficiency and scalability in LLM pre-training (Liu et al., 2025). We adopt the implementation of (Liu et al., 2025) as the Muon baseline. For the lr schedule, we use warmup-stable-decay (wsd) schedule (Zhai et al., 2022; Hu et al., 2024), which includes a linear warm-up to peak `lr_max`, followed by a stable phase where lr remains at `lr_max`, and then a linear decay to zero.

Table 1: Activation-dictionary ablation on the 0.12B dense model. Relative loss is measured against its baseline; lower is better.

Type-I models	dictionary	rel. loss	`max_lr`
baseline ($\texttt{r}^2$)	-	0.000	`2e-3`
LA (3)	$\texttt{gsr}^2\texttt{lr}$	+0.003	`4e-3`
MoA (7)	$\texttt{gsr}^2\texttt{lr}$	**-0.015**	`4e-3`

Type-II models	dictionary	rel. loss	`max_lr`
baseline (SwiGLU)	-	0.000	`2e-3`
one-LA (4)	$\texttt{gsr}^2\texttt{ltr}$	-0.015	`3e-3`
bi-LA (5)	$\texttt{gsr}^2\texttt{ltr}$	**-0.016**	`4e-3`
qd-LA (6)	$\texttt{gsr}^2$	-0.004	`3e-3`
one-MoA (8)	$\texttt{gsr}^2\texttt{ltr}$	-0.011	`3e-3`
bi-MoA (9)	$\texttt{gsr}^2\texttt{ltr}$	**-0.016**	`3e-3`
qd-MoA (10)	$\texttt{gsr}^2$	-0.008	`3e-3`

Lr tuning. For the ablations in Section 5.1, we tune `lr_max` separately for each model to fairly assess each design variant. For the larger-scale experiments of MoE models in Section 5.2, we first tune `lr_max` for the baseline and then use the same value for both the baseline and its MoA variant.

Initialization of MoA and LA parameters. For LA models, following the strategy for linear combinations of values (Zhou et al., 2025), we initialize the linear-combination coefficients to 1. For MoA models, we initialize the additional gating parameters from $\mathcal{N}(0, 0.02^2)$ for training stability.

5.1 Design study on 0.12B dense models

Exhaustively evaluating all design combinations is computationally expensive. We therefore perform a step-by-step ablation on the 0.12B dense model: we first search activation dictionaries for LA and MoA, and then examine the MoA gating function.

Activation dictionary. We denote `g`=GELU, `s`=SiLU, $\texttt{r}^2$=ReLU^2, `l`=LeakyReLU, `t`=tanh, `r`=ReLU. We ablate the activation dictionary by progressively adding candidate activations from left to right and tune the peak lr for each model. For MoA variants, we use softmax gating in this ablation, following the common choice in MoE models. For the Type-I baseline, we use ReLU^2 following recent practice (Zhang et al., 2024).

Table 2: Gating-function ablation of MoA on the 0.12B dense model. Relative loss is measured against the corresponding baseline in Table 1; lower is better.

Type-I models	gating	rel. loss	`max_lr`
MoA (7)	softmax	-0.015	`4e-3`
MoA (7)	tanh	+0.127	`2e-3`
MoA (7)	sigmoid	**-0.016**	`3e-3`

Type-II models	gating	rel. loss	`max_lr`
bi-MoA (9)	softmax	-0.016	`3e-3`
bi-MoA (9)	tanh	-0.002	`3e-3`
bi-MoA (9)	sigmoid	**-0.029**	`4e-3`

Table 1 reports the best dictionary, terminal relative validation loss, and best peak lr for each model. The results show three main trends:

- **Type-I models.** Both LA and MoA select $\texttt{gsr}^2\texttt{lr}$ as the best dictionary. MoA improves over the ReLU^2 baseline, whereas LA slightly underperforms it. In our dictionary-growth ablation, most

activation functions contribute positively, while adding $\tanh$ degrades performance. We further standard FFN ablation shows that using $\tanh$ alone performs worse than ReLU^2 alone by 0.132 relative loss, suggesting that $\tanh$ is unsuitable for Type-I FFNs in LLMs.

- **Type-II models.** The one-sided and bi-sided variants favor the largest dictionary, whereas the quadratic variants favor the smaller dictionary $\texttt{gsr}^2$. We conjecture that quadratic variants already introduce rich pairwise nonlinear interactions, making a smaller dictionary sufficient. Notably, all Type-II LA and MoA variants improve over the SwiGLU baseline.
- **Lr tolerance.** For both Type-I and Type-II models, LA and MoA variants *tolerate larger peak lrs* than their baselines. This suggests that hybrid activations may improve training stability, although a detailed investigation is left for future work.

Gating function in MoA. We next ablate the gating function in MoA, comparing $\mathrm{softmax}$, $\mathrm{sigmoid}$, and $\tanh$. For Type-I MoA, we directly compare all three choices. For Type-II MoA, we focus on bi-MoA, which performs best in Table 1. The activation dictionaries are fixed to the best choices in Table 1. As shown in Table 2, $\mathrm{sigmoid}$ *gating performs best* for both Type-I and Type-II models.

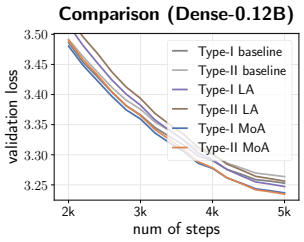


Figure 1: Validation loss after selecting the best activation dictionary and gate function for each 0.12B dense variant.

We clarify that this empirical result does not contradict our theoretical analysis with $\tanh$ gates. The theory establishes an expressivity separation, but it does not characterize optimization dynamics. In practice, $\tanh$ and $\mathrm{sigmoid}$ gates can induce different training behavior, especially near initialization: $\tanh(0) = 0$, whereas $\mathrm{sigmoid}(0) = 1/2$. Thus, $\mathrm{sigmoid}$ gates may provide stronger initial signal propagation and easier optimization, even though $\tanh$ gates are sufficient for the expressivity result.

Final comparison. Based on the above ablations, we select the best activation dictionary and gating function for each variant. We then compare six models: the Type-I baseline, Type-I LA, and Type-I MoA, Type-II baseline, Type-II LA (bi-LA), and Type-II MoA (bi-MoA). Figure 1 shows their validation loss curves. *MoA performs best for both Type-I and Type-II FFNs*, substantially outperforming the corresponding LA variants and baselines. Moreover, *Type-II MoA slightly outperforms Type-I MoA*, achieving the best overall performance.

5.2 Main Language Modeling Results

Because the design space contains many variants, we focus large-scale experiments on the most promising class identified in the 0.12B ablation study. Specifically, Type-II MoA achieves the best overall performance on the 0.12B dense model. We therefore compare Type-II MoA variants, including one-MoA, bi-MoA, and qd-MoA, against the standard SwiGLU-based Llama baseline.

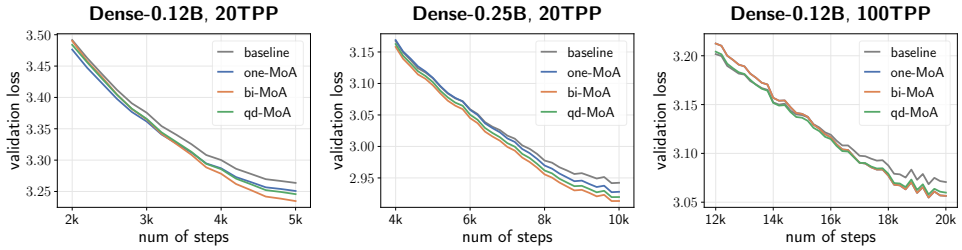


Figure 2: Comparison of MoA variants and the Llama baseline on dense models of 0.12B and 0.25B. MoA consistently achieves lower terminal loss across model scales.

Results for dense models. Figure 2 compares one-MoA, bi-MoA, and qd-MoA with the Llama baseline on 0.12B and 0.25B dense models trained with 20 tokens per parameter (TPP). We also include a 0.12B setting trained with 100 TPP. Across these settings, (i) MoA consistently achieves lower terminal loss than the well-tuned Llama baseline; (ii) Moreover, MoA variant can tolerate larger peak lr: for the 0.12B model trained with 20 TPP, the best peak lr is `2e-3` for the baseline and `3e-3` for all MoA variants; for the 0.12B model trained with 100 TPP, the best peak lr is `4e-3` for

the baseline, `5e-3` for one-MoA and qd-MoA, and `6e-3` for bi-MoA. Additionally, we present an analysis in Appendix A.5 showing that MoA changes the relative activation weights across tokens and layers, rather than using a static mixture.

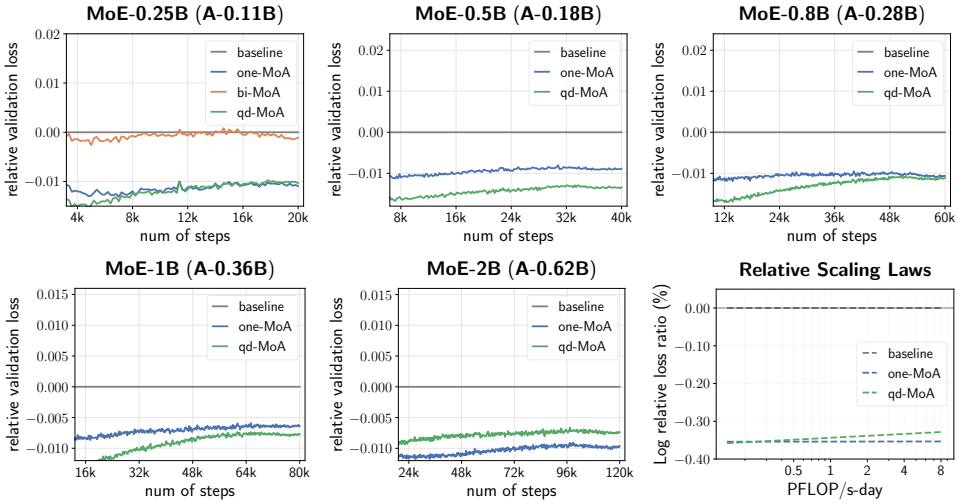


Figure 3: Comparison between MoA variants and the LlamaMoE baseline on MoE models of different sizes. MoA consistently achieves lower terminal loss and more favorable scaling law.

We next evaluate MoA on MoE models. Although the dense-model experiments show that MoA can tolerate larger lr, we avoid additional lr tuning for MoA for simplicity. For each MoE scale, we first tune the peak lr for the LlamaMoE baseline and then use the same value for the corresponding MoA variants. We use the Muon optimizer to obtain strong MoE baselines. MoA introduces a set of gating parameters, which can be concatenated into a parameter matrix and optimized with either Muon or AdamW. Controlled experiments show that these two choices yield nearly identical performance. We therefore use AdamW for these parameters.

Results for MoE models. Figure 3 compares MoA variants with the LlamaMoE baseline at MoE model sizes 0.25B (A0.11B), 0.5B (A0.18B), 0.8B (A0.28B), 1B (A0.38B), and 2B (A0.62B). In the 0.25B pilot experiment, one-MoA and qd-MoA clearly outperform the baseline, whereas bi-MoA performs similarly to the baseline. We therefore focus on one-MoA and qd-MoA in larger-scale experiments. Both variants consistently achieve **lower terminal loss** than the well-tuned Muon-trained baseline across all settings, with gains exceeding 0.01 in most experiments. Notably, these gains are obtained over strong Muon-trained baselines, indicating that the improvement is nontrivial. To examine scaling behavior, the last panel of Figure 3 reports **scaling laws** of one-MoA, qd-MoA, and the baseline. The performance gap remains stable across model sizes, and the scaling curves between one-MoA and the baseline are **nearly parallel**, suggesting that the gains from one-MoA may persist at larger scales.

Downstream evaluation. We further evaluate the zero-shot performance on common benchmarks, including ARC-C (Yadav et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), and WinoGrande (Sakaguchi et al., 2021). Table 3 reports the results. Under the same token budget, LlamaMoE-2B with one-MoA or qd-MoA outperforms the baseline on most tasks and improves the average score, demonstrating stronger downstream performance.

Table 3: Zero-shot evaluation results of pre-trained LlamaMoE-2B models. The best score in each row is bolded.

	LlamaMoE	one-MoA	qd-MoA
`ARC-C`	36.50	**36.86**	36.52
`HellaSwag`	43.31	44.54	**44.91**
`OpenBookQA`	**30.20**	29.80	**30.20**
`WinoGrande`	58.80	**60.22**	**60.22**
`Avg.`	42.20	42.86	**42.96**

5.3 Overhead Analysis

We evaluate the parameter and computational overhead of our method. Experimental details are provided in Appendix A.3.

Parameter count. Since the activation dictionary size $|\mathcal{K}|$ is small and independent of the Transformer hidden dimension d, the additional parameters introduced by MoA scale as $\mathcal{O}(d)$, which is *negligible* compared with the dominant Transformer parameter scale $\mathcal{O}(d^2)$. For rigor, we conduct a parameter-controlled ablation. Among our variants, the largest MoA variant, bi-MoA, has 0.11980B parameters, only 5.01×10^{-4} more than the Llama baseline (0.11974B). We therefore increase the baseline hidden size to approximately match 0.11980B parameters and denote the resulting model as Llama-large. As shown in Figure 4, MoA reduces terminal loss by 0.029, whereas Llama-large yields almost no gain.

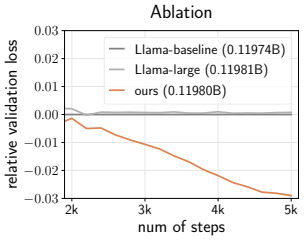


Figure 4: Ablation on parameter count.

FLOPs and memory. The additional computation of MoA mainly consists of elementwise activation evaluations and lightweight gating operations, both of which scale as $\mathcal{O}(d)$ per token because $|\mathcal{K}| = \mathcal{O}(1)$. This overhead is negligible compared with the $\mathcal{O}(d^2)$ cost of FFN linear projections. We measure practical overhead on Dense-0.5B model with `torch.compile` and Triton kernel (Tillet et al., 2019). As shown in Table 4, MoA requires only $1.02\times$ the wall-clock time of the baseline, while memory usage remains nearly unchanged. This runtime overhead could be further reduced through kernel optimization using high-performance DSLs such as TileLang (Wang et al., 2026) and CuTe-DSL (Thakkar et al., 2023).

Table 4: Runtime and memory overhead on Dense-0.5B model. Ratios are relative to the corresponding baseline.

model	wall-clock time	memory usage
Type-I baseline	190ms	26571MiB
Type-I MoA	194ms (1.02×)	26627MiB (1.00×)
Type-II baseline	196ms	28759MiB
Type-II MoA	200ms (1.02×)	28873MiB (1.00×)

5.4 Generalization to Vision Task

The preceding experiments focus on LLM pre-training. We further evaluate whether MoA generalizes to another pre-training setting, namely vision pre-training. The experimental details are provided in Appendix A.4.

Self-supervised vision pre-training. We evaluate FFN variants under the Masked Autoencoder (MAE) (He et al., 2022) framework on a large-scale image corpus using ViT-Base/16. The baseline employs a standard SwiGLU FFN, while our method replaces it with qd-MoA (10). All models are pre-trained for 800 epochs with a global batch size of 4096, a mask ratio of 0.75, and the standard MAE normalized reconstruction loss. Optimization and augmentation settings follow the MAE ViT-B protocol. The peak lr is tuned separately for the baseline and qd-MoA. In Figure 5, we report validation reconstruction losses to compare convergence. The results are consistent with those in LLM pre-training: (i) under the same lr, qd-MoA achieve lower validation loss than the baseline; (ii) the training of qd-MoA tolerates a larger lr, with the best peak lr increasing from `3e-4` for the baseline to `1.2e-3` for qd-MoA.

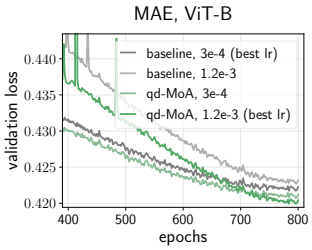


Figure 5: Validation reconstruction loss under MAE pre-training. MoA still achieves lower loss.

6 Conclusion

We introduced MoA, a simple token-adaptive FFN design that mixes a dictionary of activation functions through lightweight input-dependent gates while sharing the same linear projections. Theoretically, we prove strict finite-width expressive separations showing that MoA strictly contains learnable activations and fixed-activation FFNs. Empirically, MoA consistently improves pre-training loss and scaling behavior across dense and MoE language models. These results suggest that input-dependent activation mixing is an effective and parameter-efficient mechanism for improving FFN expressivity in modern LLMs. Future work includes applying distinct token-adaptive nonlinear mixing mechanisms across input dimension to further improve FFN adaptivity and expressivity.

Acknowledgments and Disclosure of Funding

We thank Guang Shi, Prof. Weinan E, and Guhao Feng for helpful discussions. The authors declare no external funding or competing interests beyond their disclosed institutional affiliations.

References

Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014. 2

Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021. 2

Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017. 5

Andrew R Barron. Neural net approximation. In *Proc. 7th Yale Workshop on Adaptive and Learning Systems*, volume 1, pages 69–72, 1992. 5

Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993. 5

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 2

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, 2022. 1

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 933–941, 2017. 2

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019. 1, 2

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. 2

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. 2

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1319–1327, 2013. 2

Mohit Goyal, Rajan Goyal, and Brejesh Lall. Learning activation functions: A new paradigm for understanding neural networks. *arXiv preprint arXiv:1906.09529*, 2019. 2

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*, pages 1026–1034, 2015. 2

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022. 10, 16

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 2

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 7, 15

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024. 7

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. 3

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994. 3

Andrej Karpathy. Nanogpt. https://github.com/karpathy/nanoGPT, 2022. 15

Jordan Keller et al. Muon optimizer. https://github.com/KellerJordan/Muon?tab=readme-ov-file, 2024. 7, 15

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7, 15

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 2017. 2

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025. 7, 15

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024. 3

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 7, 15

Franco Manessi and Alessandro Rozza. Learning combinations of activation functions. *arXiv preprint arXiv:1801.09403*, 2018. 2

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018. 9

Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019. 2

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1, 2

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *International Conference on Learning Representations Workshop*, 2018. 2

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. 9

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. 1, 2

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. 3

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 14

Leon René Sütfeld, Flemming Brieger, Holger Finger, Sonja Füllhase, and Gordon Pipa. Adaptive blending units: Trainable activation functions for deep neural networks. *arXiv preprint arXiv:1806.10064*, 2018. 2

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Aditya Atluri, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, January 2023. URL https://github.com/NVIDIA/cutlass. 10

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. 10

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1, 7, 14, 15

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. 1, 2

Lei Wang, Yu Cheng, Yining Shi, Zhiwen Mo, Zhengju Tang, Wenhao Xie, Tong Wu, Lingxiao Ma, Yuqing Xia, Jilong Xue, et al. Tilelang: Bridge programmability and performance in modern neural kernels. In *The Fourteenth International Conference on Learning Representations*, 2026. 10

Mingze Wang and Weinan E. Understanding the expressive power and mechanisms of transformer for sequence modeling. *Advances in Neural Information Processing Systems*, 2024. 1

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. *arXiv preprint arXiv:1911.07176*, 2019. 9

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. 14

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019. 9

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022. 7

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024. 7

Shu Zhong, Mingyu Xu, Tenglong Ao, and Guang Shi. Understanding transformer from the perspective of associative memory. *arXiv preprint arXiv:2505.19488*, 2025. 1

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28341–28356, 2025. 7

Zhijian Zhuo, Ya Wang, Yutao Zeng, Xiaoqing Li, Xun Zhou, and Jinwen Ma. Polynomial composition activations: Unleashing the dynamics of large language models. *arXiv preprint arXiv:2411.03884*, 2024. 2

Appendix

A Experimental Details **14**

A.1 Experimental Details for Section 5.2 14

A.2 Experimental Details for Section 5.1 15

A.3 Experimental Details for Section 5.3 15

A.4 Experimental Details for Section 5.4 16

A.5 Token-wise Gate Analysis 16

B Proofs **16**

B.1 Proofs in Section 4.1 16

B.2 Proofs in Section 4.2 22

A Experimental Details

A.1 Experimental Details for Section 5.2

Models. We utilize two popular classes of LLM models for our pre-training experiments:

- **Dense models.** Llama (Touvron et al., 2023) is a dense decoder-only Transformer architecture that uses Rotary Positional Encoding (RoPE) (Su et al., 2024), SwiGLU, RMSNorm, and a Pre-Norm design. We set $d_{\text{FFN}} = \text{int}(8d_{\text{model}}/3)$. We pre-train Llama models of 0.12B, 0.25B, and 0.5B parameters, with 20 TPP or 100 TPP. Detailed configurations are provided in Table 5.
- **MoE models.** LlamaMoE is a decoder-only mixture-of-experts architecture based on Llama. Each model uses 32 sparse experts, activates 4 sparse experts per token, and includes one shared expert. Following QwenMoE (Yang et al., 2024), the hidden dimension of the shared expert is $\text{int}(8d_{\text{model}}/3)$, whereas that of each sparse expert is $\text{int}(2d_{\text{model}}/3)$. We pre-train LlamaMoE models ranging from 0.25B to 2B parameters. Detailed configurations are provided in Table 6.

Table 5: Model configurations and optimally-tuned peak lr's for dense models.

Acronym	Size	d_{model}	n_head	n_layers	TPP	`lr_max` (AdamW, cos)
Dense-0.12B	0.12B	768	12	6	20	2e-3
Dense-0.12B	0.12B	768	12	6	100	4e-3
Dense-0.25B	0.25B	1024	16	12	20	2e-3
Dense-0.5B	0.48B	1280	20	18	20	2e-3

Table 6: Model configurations and optimally-tuned peak lr's for MoE models.

Acronym	Size	Activated Size	d_{model}	n_head	n_layers	`lr_max` (Muon, wsd)
MoE-0.25B	0.25B	0.11B	640	10	6	2e-3
MoE-0.5B	0.48B	0.18B	768	12	9	1e-3
MoE-0.8B	0.80B	0.28B	960	15	10	1e-3
MoE-1B	1.06B	0.38B	1024	16	12	1e-3
MoE-2B	2.00B	0.62B	1280	20	15	6e-4

Token Budget. Unless otherwise specified, dense models are trained with a token budget of approximately *20 times* (Chinchilla-optimal regime (Hoffmann et al., 2022)) or *100 times* the number of model parameters. For MoE models, we use a token budget of approximately *100 times* the number of activated parameters, which is larger than the Chinchilla-optimal budget and is closer to industrial pre-training practice. For each experiment, we use a sequence length of 1,024 and a batch size of 480, following (Karpathy, 2022).

Optimizers.

- **AdamW.** For dense models, We use AdamW as the default baseline optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2017). Following standard Llama pre-training practice (Touvron et al., 2023), we set $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay $\lambda = 0.1$, and the gradient clipping threshold to 1.0.
- **Muon.** For MoE models, we use Muon optimizer, following the setup (Keller et al., 2024): Muon is applied only to 2D matrix blocks in Transformer layers, while AdamW is used for all other parameters, including scale vectors, embedding layer, and the output layer. Additionally, following (Liu et al., 2025), we further use: (i) per-parameter update scaling, with learning-rate multiplier $c = 0.2\sqrt{\max\{m, n\}}$ for Muon blocks of shape $\mathbb{R}^{m\times n}$, so that the update RMS norm matches that of AdamW;. (ii) Nesterov momentum with coefficient $\theta_{\text{muon}} = 0.95$; (iii) weight decay $\lambda = 0.1$.

Lr schedules.

- **cos.** For dense models, we use the `cos` lr schedule: a linear warm-up to the peak learning rate `lr_max`, followed by cosine decay to the terminal `lr_min=lr_max/20`.
- **wsd.** For MoE models, we use the `wsd` schedule, we use a linear warmup to the peak learning rate `lr_max`, followed by a stable phase in which the learning rate remains at `lr_max` until 80% of the total training steps, and finally a linear decay to zero.

Lr tuning. For 0.12B and 0.25B dense models, we tune `lr_max` for all baselines and variants. For MoE models and 0.5B dense model, we first tune `lr_max` for the baseline model and then use the tuned value for both the baseline and the corresponding improved model. The grid search for `lr_max` is performed over {`6e-4`, `1e-3`, `2e-3`, `3e-3`, `4e-3`, `5e-3`, `6e-3`}. The selected peak lr's for baselines are reported in Tables 5 and 6.

For Figure 3, we report relative losses for MoE models at different parameter count. For the scaling law panel, we first fit the baseline loss $\mathcal{L}_{\text{baseline}}$ and our method's loss $\mathcal{L}_{\text{method}}$ as functions of training compute (measured in PFLOP/s-days), and then plot the log relative improvement, $\log(\mathcal{L}_{\text{method}}/\mathcal{L}_{\text{baseline}})$ for each method.

A.2 Experimental Details for Section 5.1

We conduct the ablation study based on the Llama-0.12B configuration described in Appendix A.1. For Type-II FFNs, we set the FFN hidden width to $\text{int}(8d/3)$, while for Type-I FFNs, we set it to $4d$; the resulting parameter counts are nearly identical. To fairly compare different designs, we tune `lr_max` separately for each FFN type and variant. The selected values of `lr_max` are reported in the tables in Section 5.1. All other training configurations follow those used for Llama-0.12B in Appendix A.1.

A.3 Experimental Details for Section 5.3

Parameter count. The baseline model uses the same training configuration as Llama-0.12B in Appendix A.1. For the model-size ablation, we construct a larger Llama variant based on Llama-0.12B. Directly increasing the hidden dimension would require the dimension to remain compatible with both RoPE and the attention heads, leading to a parameter count much larger than the target. Therefore, we instead increase the FFN hidden dimension from d_{FFN} to $d_{\text{FFN}} + 5$, which closely matches the target parameter count. We retune the learning rate for this larger model, and the selected value remains `2e-3`.

Computational overhead. We train Llama-0.5B model with standard `torch.compile`, under the setting in Appendix A.1. The results in Table 4 are averaged over 10 training steps after warmup.

A.4 Experimental Details for Section 5.4

The experiments are performed on a large-scale image corpus using a ViT-Base/16 encoder and the standard MAE decoder with hidden dimension 512 and depth 8. We follow the MAE pre-training recipe (He et al., 2022): the mask ratio is set to 0.75, and the training objective is the normalized patch reconstruction loss, implemented as normalized patch MSE. For data augmentation, we use random resized cropping with scale range $(0.2, 1.0)$ and random horizontal flipping.

The baseline replaces the standard MAE MLP with a Llama-style SwiGLU FFN, whose hidden width follows the Llama convention $8d/3$. For our method, we replace this SwiGLU FFN with the corresponding MoA variant while keeping the remaining MAE architecture unchanged. We pre-train all models for 800 epochs with a batch size of 4096. We use AdamW optimizer with $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and weight decay 0.05. Following standard practice, weight decay is not applied to bias parameters, normalization parameters, positional embeddings, the class token, or the mask token. The lr schedule uses linear warmup for 40 epochs followed by cosine decay to zero. The peak lr of each model is tuned over $\{$`1.5e-4`, `3e-4`, `6e-4`, `1.2e-3`, `2.4e-3`$\}$.

A.5 Token-wise Gate Analysis

We examine whether MoA adjusts the relative weights of different activation functions across tokens and layers, rather than using the same mixture for every token. We normalize each token's gate weights to focus on their relative proportions. Let $g_{\ell,t,k}$ denote the gate weight for activation k at token t in layer ℓ, and let K denote the activation dictionary size:

$$\widetilde{g}_{\ell,t,k} = \frac{g_{\ell,t,k}}{\sum_{j=1}^{K} g_{\ell,t,j}}, \qquad \widetilde{V}_\ell = \frac{1}{K}\sum_{k=1}^{K} \mathrm{Var}_t(\widetilde{g}_{\ell,t,k})\,.$$

We report $R_\ell = \widetilde{V}_\ell / V_{\mathrm{rand}}$, where V_{rand} is computed analogously from normalized token-wise i.i.d. $\mathcal{U}[0,1]$ gates. Because normalization removes any common multiplicative scale, $\widetilde{V}_\ell$ measures how the relative activation weights vary across tokens. For Dense-0.25B bi-MoA, the layerwise R_ℓ values for the gating branch are

$$\begin{aligned}(R_1, \ldots, R_{12}) = (&0.8790,\ 0.3261,\ 1.0792,\ 0.3953,\ 0.8269,\ 0.1844,\\ &0.3072,\ 0.3453,\ 0.3712,\ 0.2871,\ 0.5200,\ 0.3350).\end{aligned}$$

The mean ratio across layers is 0.4881. These results show that the relative activation weights vary across tokens in every layer, rather than remaining fixed.

B Proofs

B.1 Proofs in Section 4.1

Additional Notations. Let $\mathrm{ReLU}(t) = t_+$, $\mathrm{ReLU}^2(t) = t_+^2$ and $\mathrm{LeakyReLU}(t) = \max\{t, \eta t\}$ for some fixed $\eta \in (0,1)$. For a bounded function q on a set E, define its oscillation by $\mathrm{osc}_E(q) := \operatorname{ess\,sup}_{x\in E} q(x) - \operatorname{ess\,inf}_{x\in E} q(x)$. If q is continuous on compact E, this equal $\mathrm{osc}_E(q) = \sup_{x\in E} q(x) - \inf_{x\in E} q(x)$. Additionally, we define the jump notation: let $S = \{(x_1, x_2) \in [-1,1]^2 : x_2 = 0\}$. For a piecewise C^1 function f, define the jump of its x_2-derivative across S by $J_S(f)(x_1) := \partial_{x_2} f(x_1, 0^+) - \partial_{x_2} f(x_1, 0^-)$, whenever the two one-sided traces exist. For finite sums of ridge functions with ReLU-type activations, these traces exist for almost every $x_1 \in [-1,1]$, because there are only finitely many singular hyperplanes.

B.1.1 Proof of the Basic Inclusions

Proof of $\bigcup_{\sigma\in\mathcal{K}} \mathcal{F}_\sigma^{(m)} \subset \mathcal{F}_{\mathrm{LA}}^{(m)}$. Let $f \in \mathcal{F}_{\sigma_j}^{(m)}$ for some $\sigma_j \in \mathcal{K}$. Then

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma_j(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}).$$

In the definition of $\mathcal{F}_{\mathrm{LA}}^{(m)}$, choose

$$\alpha_j = 1, \qquad \alpha_c = 0 \quad (c \neq j).$$

Then

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sum_{c=1}^{6} \alpha_c \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}),$$

and hence $f \in \mathcal{F}_{\mathrm{LA}}^{(m)}$. Therefore,

$$\bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_{\sigma}^{(m)} \subset \mathcal{F}_{\mathrm{LA}}^{(m)}.$$

□

Proof of $\mathcal{F}_{\mathrm{LA}}^{(m)} \subset \mathcal{F}_{\mathrm{MoA}}^{(m)}$. Let

$$g(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sum_{c=1}^{6} \alpha_c \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \in \mathcal{F}_{\mathrm{LA}}^{(m)}.$$

If $\alpha_c = 0$ for all c, then $g \equiv 0 \in \mathcal{F}_{\mathrm{MoA}}^{(m)}$. Otherwise, choose $\rho > 0$ sufficiently small such that

$$|\rho \alpha_c| < 1, \qquad c = 1, \ldots, 6.$$

For each c, define

$$\boldsymbol{u}_c = (0, \ldots, 0, \operatorname{arctanh}(\rho \alpha_c)) \in \mathbb{R}^{d+1}.$$

Since $\bar{\boldsymbol{x}} = (\boldsymbol{x}, 1)$, we have

$$\tanh(\boldsymbol{u}_c^\top \bar{\boldsymbol{x}}) = \tanh(\operatorname{arctanh}(\rho \alpha_c)) = \rho \alpha_c.$$

Therefore,

$$g(\boldsymbol{x}) = \sum_{k=1}^{m} \frac{a_k}{\rho} \sum_{c=1}^{6} \tanh(\boldsymbol{u}_c^\top \bar{\boldsymbol{x}}) \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}).$$

Hence $g \in \mathcal{F}_{\mathrm{MoA}}^{(m)}$, and consequently

$$\mathcal{F}_{\mathrm{LA}}^{(m)} \subset \mathcal{F}_{\mathrm{MoA}}^{(m)}.$$

□

B.1.2 Separation between $\mathcal{F}_{\mathrm{LA}}^{(m)}$ and Fixed-Activation Classes

We first show that $\mathcal{F}_{\mathrm{LA}}^{(m)}$ strictly contains the union of fixed-activation classes. Consider the one-dimensional domain $[-1, 1]$ and define

$$T_{\mathrm{LA}}(t) = t_+ + t_+^2.$$

Clearly,

$$T_{\mathrm{LA}} \in \mathcal{F}_{\mathrm{LA}}^{(1)}.$$

Indeed, choose $a_1 = 1$, $\boldsymbol{w}_1 = (1, 0)$, and

$$\alpha_{\mathrm{ReLU}} = 1, \qquad \alpha_{\mathrm{ReLU}^2} = 1,$$

while setting all other α_c's to zero. Since

$$\mathcal{F}_{\mathrm{LA}}^{(1)} \subset \mathcal{F}_{\mathrm{MoA}}^{(1)},$$

we also have

$$T_{\mathrm{LA}} \in \mathcal{F}_{\mathrm{MoA}}^{(1)}.$$

We now prove that T_{LA} cannot be represented by any width-m fixed-activation network.

Lemma B.1 (Step-function approximation lower bound)**.** *Let s be any step function on $[0, 1]$ with at most m discontinuities. Then*

$$\|2t - s(t)\|_{L^\infty([0,1])} \geq \frac{1}{m+1}.$$

Consequently,

$$\inf_s \|2t - s(t)\|_{L^\infty([0,1])} \geq \frac{1}{m+1},$$

where the infimum is over all such step functions.

Proof. A step function with at most m discontinuities partitions $[0,1]$ into at most $m+1$ intervals on which it is constant. Hence at least one interval $I \subset [0,1]$ has length

$$|I| \geq \frac{1}{m+1}.$$

On this interval, the oscillation of $2t$ is at least

$$2|I| \geq \frac{2}{m+1}.$$

The best uniform approximation of a function by a constant on an interval has error at least one half of its oscillation. Therefore,

$$\|2t - s(t)\|_{L^\infty(I)} \geq \frac{1}{m+1}.$$

This proves the claim. □

Proposition B.2 (Quantitative separation of $\mathcal{F}_{\mathrm{LA}}^{(m)}$ from fixed activations)**.** *For every $m \geq 1$,*

$$\inf_{f \in \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}} \|T_{\mathrm{LA}} - f\|_{\mathcal{W}^{1,\infty}([-1,1])} \geq \frac{1}{m+1}.$$

Consequently,

$$T_{\mathrm{LA}} \notin \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}.$$

Proof. We split the proof according to the activation.

First consider

$$\sigma \in \{\mathrm{ReLU}, \mathrm{LeakyReLU}\}.$$

Then every $f \in \mathcal{F}_\sigma^{(m)}$ is a piecewise linear function on $[-1,1]$, and hence its weak derivative f' is a step function with at most m discontinuities. On $(0,1)$,

$$T'_{\mathrm{LA}}(t) = 1 + 2t.$$

Therefore, by Lemma B.1,

$$\|T'_{\mathrm{LA}} - f'\|_{L^\infty([0,1])} = \|1 + 2t - f'(t)\|_{L^\infty([0,1])} \geq \frac{1}{m+1}.$$

Thus,

$$\|T_{\mathrm{LA}} - f\|_{\mathcal{W}^{1,\infty}([-1,1])} \geq \frac{1}{m+1}.$$

Next consider

$$\sigma \in \{\mathrm{ReLU}^2, \mathrm{GELU}, \mathrm{SiLU}, \tanh\}.$$

For these activations, every $f \in \mathcal{F}_\sigma^{(m)}$ is C^1. Hence f' is continuous at $t = 0$. However,

$$T'_{\mathrm{LA}}(t) = \begin{cases} 0, & t < 0, \\ 1 + 2t, & t > 0. \end{cases}$$

Thus

$$T'_{\mathrm{LA}}(0^-) = 0, \qquad T'_{\mathrm{LA}}(0^+) = 1.$$

If

$$\|T'_{\mathrm{LA}} - f'\|_{L^\infty([-1,1])} \leq \varepsilon,$$

then by taking one-sided limits at $t = 0$ and using the continuity of f', we obtain

$$|f'(0)| \leq \varepsilon, \qquad |1 - f'(0)| \leq \varepsilon.$$

Therefore,

$$\varepsilon \geq \frac{1}{2}.$$

Since $m \geq 1$, we have

$$\frac{1}{2} \geq \frac{1}{m+1}.$$

Combining the two cases gives

$$\|T_{\mathrm{LA}} - f\|_{\mathcal{W}^{1,\infty}([-1,1])} \geq \frac{1}{m+1}$$

for every

$$f \in \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_{\sigma}^{(m)}.$$

Taking the infimum proves the proposition. □

This proves the strict inclusion

$$\bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_{\sigma}^{(m)} \subsetneq \mathcal{F}_{\mathrm{LA}}^{(m)}.$$

B.1.3 Separation between $\mathcal{F}_{\mathrm{MoA}}^{(m)}$ and $\mathcal{F}_{\mathrm{LA}}^{(m)}$

We now show that $\mathcal{F}_{\mathrm{MoA}}^{(m)}$ strictly contains $\mathcal{F}_{\mathrm{LA}}^{(m)}$. The key point is that $\mathcal{F}_{\mathrm{MoA}}^{(m)}$ can create ridge singularities whose amplitudes vary along the singular hyperplane, whereas $\mathcal{F}_{\mathrm{LA}}^{(m)}$ can only attach constant amplitudes to each ridge singularity.

Let $\Omega = [-1,1]^2$. Fix $\lambda > 0$, and define

$$T_{\mathrm{MoA}}(x_1, x_2) = \tanh(\lambda x_1)(x_2)_+.$$

Proposition B.3 (Exact representation by $\mathcal{F}_{\mathrm{MoA}}^{(1)}$)**.** *The target T_{MoA} belongs to $\mathcal{F}_{\mathrm{MoA}}^{(1)}$.*

Proof. Use the ReLU branch. Choose

$$a_1 = 1, \qquad \boldsymbol{w}_1 = (0,1,0) \in \mathbb{R}^3, \qquad \boldsymbol{u}_{\mathrm{ReLU}} = (\lambda, 0, 0) \in \mathbb{R}^3.$$

Then

$$\boldsymbol{w}_1^{\top} \bar{\boldsymbol{x}} = x_2, \qquad \boldsymbol{u}_{\mathrm{ReLU}}^{\top} \bar{\boldsymbol{x}} = \lambda x_1.$$

For all other activation branches, set $\boldsymbol{u}_c = \mathbf{0}$, so that

$$\tanh(\boldsymbol{u}_c^{\top} \bar{\boldsymbol{x}}) = 0.$$

Therefore,

$$h(\boldsymbol{x}) = \tanh(\lambda x_1)(x_2)_+ = T_{\mathrm{MoA}}(x_1, x_2).$$

Hence

$$T_{\mathrm{MoA}} \in \mathcal{F}_{\mathrm{MoA}}^{(1)}.$$

□

The weak derivative of T_{MoA} with respect to x_2 is

$$\partial_{x_2} T_{\mathrm{MoA}}(x_1, x_2) = \tanh(\lambda x_1) \mathbf{1}_{\{x_2 > 0\}}.$$

Hence the jump of the normal derivative across S is

$$J_S(T_{\mathrm{MoA}})(x_1) = \tanh(\lambda x_1).$$

For functions in $\mathcal{F}_{\mathrm{LA}}^{(m)}$, the corresponding jump amplitude across a fixed hyperplane is necessarily constant.

Lemma B.4 (Constant jump amplitude for $\mathcal{F}_{\mathrm{LA}}^{(m)}$)**.** *Let $f \in \mathcal{F}_{\mathrm{LA}}^{(m)}$. Then the jump of $\partial_{x_2} f$ across*

$$S = \{x_2 = 0\}$$

is almost everywhere a constant function of x_1. That is, there exists $C_f \in \mathbb{R}$ such that

$$J_S(f)(x_1) = C_f$$

for almost every $x_1 \in [-1,1]$.

Proof. Write

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sum_{c=1}^{6} \alpha_c \sigma_c(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}).$$

Among the activations in $\mathcal{K}$, only ReLU and $\mathrm{LeakyReLU}$ can generate jumps in first derivatives. The activations

$$\mathrm{ReLU}^2, \qquad \mathrm{GELU}, \qquad \mathrm{SiLU}, \qquad \tanh$$

have continuous first derivatives and therefore do not contribute to $J_S(f)$.

Consider a ReLU-type ridge term

$$\sigma(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}).$$

Its first derivative can jump only across the affine hyperplane

$$H_k = \{\boldsymbol{x} : \boldsymbol{w}_k^\top \bar{\boldsymbol{x}} = 0\}.$$

If $H_k \neq S$, then $H_k \cap S$ has lower dimension inside S, and hence this ridge term contributes no jump across S for almost every $x_1 \in [-1, 1]$.

If $H_k = S$, then $\boldsymbol{w}_k$ is proportional to the normal vector of S. Hence the jump of

$$\partial_{x_2} \sigma(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}})$$

across S is a constant independent of x_1. Summing over finitely many neurons gives

$$J_S(f)(x_1) = C_f$$

for almost every $x_1 \in [-1, 1]$. □

Proposition B.5 (Quantitative separation of $\mathcal{F}_{\mathrm{MoA}}^{(m)}$ from $\mathcal{F}_{\mathrm{LA}}^{(m)}$)**.** *For every $m \geq 1$,*

$$\inf_{f \in \mathcal{F}_{\mathrm{LA}}^{(m)}} \|T_{\mathrm{MoA}} - f\|_{\mathcal{W}^{1,\infty}(\Omega)} \geq \frac{1}{2} \tanh(\lambda).$$

Consequently,

$$T_{\mathrm{MoA}} \notin \mathcal{F}_{\mathrm{LA}}^{(m)}.$$

Proof. Let $f \in \mathcal{F}_{\mathrm{LA}}^{(m)}$, and suppose

$$\|T_{\mathrm{MoA}} - f\|_{\mathcal{W}^{1,\infty}(\Omega)} \leq \varepsilon.$$

Then, in particular,

$$\|\partial_{x_2} T_{\mathrm{MoA}} - \partial_{x_2} f\|_{L^\infty(\Omega)} \leq \varepsilon.$$

Taking one-sided traces from the two sides of S, we obtain

$$\|J_S(T_{\mathrm{MoA}}) - J_S(f)\|_{L^\infty([-1,1])} \leq 2\varepsilon.$$

This follows from applying the derivative bound on the two open half-domains

$$\Omega^+ = \Omega \cap \{x_2 > 0\}, \qquad \Omega^- = \Omega \cap \{x_2 < 0\},$$

and then taking one-sided limits at S, which exist for almost every x_1.

By Lemma B.4, there exists $C_f \in \mathbb{R}$ such that

$$J_S(f)(x_1) = C_f$$

for almost every $x_1 \in [-1, 1]$. Therefore,

$$2\varepsilon \geq \inf_{C \in \mathbb{R}} \|\tanh(\lambda x_1) - C\|_{L^\infty([-1,1])}.$$

Since $\tanh(\lambda x_1)$ ranges over

$$[-\tanh(\lambda), \tanh(\lambda)]$$

on $[-1, 1]$, the best uniform approximation by a constant has error

$$\inf_{C \in \mathbb{R}} \|\tanh(\lambda x_1) - C\|_{L^\infty([-1,1])} = \tanh(\lambda).$$

Thus,

$$2\varepsilon \geq \tanh(\lambda),$$

and hence

$$\varepsilon \geq \frac{1}{2} \tanh(\lambda).$$

Taking the infimum over $f \in \mathcal{F}_{\mathrm{LA}}^{(m)}$ proves the result. □

Since

$$\bigcup_{\sigma\in\mathcal{K}} \mathcal{F}_\sigma^{(m)} \subset \mathcal{F}_{\mathrm{LA}}^{(m)},$$

Proposition B.5 also implies

$$T_{\mathrm{MoA}} \notin \bigcup_{\sigma\in\mathcal{K}} \mathcal{F}_\sigma^{(m)}.$$

Therefore,

$$\mathcal{F}_{\mathrm{LA}}^{(m)} \subsetneq \mathcal{F}_{\mathrm{MoA}}^{(m)}.$$

B.1.4 A General Adaptive Target Class

The previous example is a special case of a broader family of adaptive ridge functions. Let

$$T_{\boldsymbol{u},\beta,\boldsymbol{w},b}(\boldsymbol{x}) = \tanh(\boldsymbol{u}^\top \boldsymbol{x} + \beta)(\boldsymbol{w}^\top \boldsymbol{x} + b)_+.$$

Then

$$T_{\boldsymbol{u},\beta,\boldsymbol{w},b} \in \mathcal{F}_{\mathrm{MoA}}^{(1)}.$$

Indeed, one can choose the ReLU ridge

$$\sigma(\boldsymbol{w}^\top \boldsymbol{x} + b) = (\boldsymbol{w}^\top \boldsymbol{x} + b)_+$$

and the adaptive gate

$$\tanh(\boldsymbol{u}^\top \boldsymbol{x} + \beta).$$

Let

$$S = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}.$$

If the function

$$\boldsymbol{x} \mapsto \tanh(\boldsymbol{u}^\top \boldsymbol{x} + \beta)$$

is nonconstant on $S \cap \Omega$, then $T_{\boldsymbol{u},\beta,\boldsymbol{w},b}$ cannot be represented by $\mathcal{F}_{\mathrm{LA}}^{(m)}$ at fixed width. More quantitatively, the same jump-amplitude argument yields

$$\inf_{f\in\mathcal{F}_{\mathrm{LA}}^{(m)}} \|T_{\boldsymbol{u},\beta,\boldsymbol{w},b} - f\|_{\mathcal{W}^{1,\infty}(\Omega)} \geq \frac{1}{4} \operatorname{osc}_{S\cap\Omega} \left(\tanh(\boldsymbol{u}^\top \boldsymbol{x} + \beta)\right).$$

Indeed, for any $f \in \mathcal{F}_{\mathrm{LA}}^{(m)}$, the jump of f across S has constant amplitude almost everywhere on S, whereas

$$J_S(T_{\boldsymbol{u},\beta,\boldsymbol{w},b})(\boldsymbol{x}) = \|\boldsymbol{w}\|_2 \tanh(\boldsymbol{u}^\top \boldsymbol{x} + \beta)$$

up to a fixed sign depending on the chosen normal orientation. Therefore, approximating this nonconstant jump profile by a constant incurs at least one half of its oscillation, and the trace argument loses another factor of 2. This gives the factor $1/4$.

This lower bound captures the essential adaptivity gap: $\mathcal{F}_{\mathrm{MoA}}^{(m)}$ can make the amplitude of a ridge singularity vary along the ridge hyperplane, while $\mathcal{F}_{\mathrm{LA}}^{(m)}$ can only assign constant amplitudes to such singularities.

B.1.5 Proof of Theorem 4.1

The inclusions

$$\bigcup_{\sigma\in\mathcal{K}} \mathcal{F}_\sigma^{(m)} \subset \mathcal{F}_{\mathrm{LA}}^{(m)} \subset \mathcal{F}_{\mathrm{MoA}}^{(m)}$$

were proved above.

The first inclusion is strict by Proposition B.2, which constructs

$$T_{\mathrm{LA}}(t) = t_+ + t_+^2$$

satisfying

$$T_{\mathrm{LA}} \in \mathcal{F}_{\mathrm{LA}}^{(1)} \subset \mathcal{F}_{\mathrm{MoA}}^{(1)}$$

but

$$\inf_{f \in \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}} \|T_{\mathrm{LA}} - f\|_{\mathcal{W}^{1,\infty}([-1,1])} \geq \frac{1}{m+1}.$$

Hence

$$T_{\mathrm{LA}} \notin \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}.$$

The second inclusion is strict by Proposition B.5, which constructs

$$T_{\mathrm{MoA}}(x_1, x_2) = \tanh(\lambda x_1)(x_2)_+$$

satisfying

$$T_{\mathrm{MoA}} \in \mathcal{F}_{\mathrm{MoA}}^{(1)}$$

but

$$\inf_{f \in \mathcal{F}_{\mathrm{LA}}^{(m)}} \|T_{\mathrm{MoA}} - f\|_{\mathcal{W}^{1,\infty}([-1,1]^2)} \geq \frac{1}{2}\tanh(\lambda).$$

Therefore,

$$T_{\mathrm{MoA}} \notin \mathcal{F}_{\mathrm{LA}}^{(m)}.$$

Since

$$\bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)} \subset \mathcal{F}_{\mathrm{LA}}^{(m)},$$

we also have

$$T_{\mathrm{MoA}} \notin \bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)}.$$

Combining these results proves

$$\bigcup_{\sigma \in \mathcal{K}} \mathcal{F}_\sigma^{(m)} \subsetneq \mathcal{F}_{\mathrm{LA}}^{(m)} \subsetneq \mathcal{F}_{\mathrm{MoA}}^{(m)}.$$

□

B.2 Proofs in Section 4.2

B.2.1 Basic Inclusions

Proof of $\bigcup_{p,q \in [7]} \mathcal{G}_{\sigma_{p,q}}^{(m)} \subset \mathcal{G}_{\mathrm{LA}}^{(m)}$. Let

$$f \in \mathcal{G}_{\sigma_{p_0,q_0}}^{(m)}$$

for some $(p_0, q_0) \in [7]^2$. Then

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma_{p_0}(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \sigma_{q_0}(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}).$$

If $p_0 \leq q_0$, choose

$$\alpha_{p_0,q_0} = 1, \qquad \alpha_{p,q} = 0 \quad \text{for } (p,q) \neq (p_0,q_0).$$

Then $f \in \mathcal{G}_{\mathrm{LA}}^{(m)}$.

If $p_0 > q_0$, use the commutativity of multiplication:

$$\sigma_{p_0}(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \sigma_{q_0}(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) = \sigma_{q_0}(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) \sigma_{p_0}(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}).$$

Thus the same function is represented in $\mathcal{G}_{\mathrm{LA}}^{(m)}$ by choosing

$$\alpha_{q_0,p_0} = 1,$$

and swapping the two weight vectors in each product branch.

Therefore,

$$\bigcup_{p,q \in [7]} \mathcal{G}_{\sigma_{p,q}}^{(m)} \subset \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

□

Proof of $\mathcal{G}_{\mathrm{LA}}^{(m)} \subset \mathcal{G}_{\mathrm{MoA}}^{(m)}$. Let

$$g(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sum_{1 \le p \le q \le 7} \alpha_{p,q} \sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}) \in \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

If all $\alpha_{p,q}$'s are zero, then $g \equiv 0 \in \mathcal{G}_{\mathrm{MoA}}^{(m)}$. Otherwise, choose $\rho > 0$ sufficiently small such that

$$|\rho \alpha_{p,q}| < 1, \qquad 1 \le p \le q \le 7.$$

For each pair $1 \le p \le q \le 7$, define

$$\boldsymbol{v}_{p,q} = (0, \dots, 0, \operatorname{arctanh}(\rho \alpha_{p,q})) \in \mathbb{R}^{d+1}.$$

Since $\bar{\boldsymbol{x}} = (\boldsymbol{x}, 1)$, we have

$$\tanh(\boldsymbol{v}_{p,q}^\top \bar{\boldsymbol{x}}) = \rho \alpha_{p,q}.$$

Therefore,

$$g(\boldsymbol{x}) = \sum_{k=1}^{m} \frac{a_k}{\rho} \sum_{1 \le p \le q \le 7} \tanh(\boldsymbol{v}_{p,q}^\top \bar{\boldsymbol{x}}) \sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}).$$

Thus $g \in \mathcal{G}_{\mathrm{MoA}}^{(m)}$, and hence

$$\mathcal{G}_{\mathrm{LA}}^{(m)} \subset \mathcal{G}_{\mathrm{MoA}}^{(m)}.$$

□

B.2.2 One-Dimensional Ridge Classes for Jump Profiles

For $\sigma \in \mathcal{K}$, define

$$\mathcal{R}_\sigma^{(M)} := \left\{ t \mapsto \sum_{\ell=1}^{M} b_\ell \sigma(r_\ell t + s_\ell) : b_\ell, r_\ell, s_\ell \in \mathbb{R} \right\}.$$

Define the finite dictionary ridge class

$$\mathcal{A}^{(M)} := \left\{ t \mapsto \sum_{\ell=1}^{M} \sum_{c=1}^{7} b_{\ell,c} \sigma_c(r_\ell t + s_\ell) : b_{\ell,c}, r_\ell, s_\ell \in \mathbb{R} \right\}.$$

This class is larger than the one-dimensional LA class because it allows activation coefficients to depend on ℓ.

Let

$$\mathcal{N}_1 := \{\mathrm{ReLU}, \mathrm{LeakyReLU}\}.$$

These are exactly the activations in $\mathcal{K}$ whose first derivatives have jump discontinuities. The activations

$$\mathrm{id}, \quad \mathrm{ReLU}^2, \quad \mathrm{GELU}, \quad \mathrm{SiLU}, \quad \tanh$$

are C^1 and therefore do not generate jumps in first derivatives.

B.2.3 Separation between $\mathcal{G}_{\mathrm{LA}}^{(m)}$ and Fixed Type-II Classes

Define

$$A(t) := t_+ + \tanh(t),$$

and consider

$$T_{\mathrm{LA}}(x_1, x_2) := (x_2)_+ A(x_1) = (x_2)_+ \big((x_1)_+ + \tanh(x_1)\big).$$

Proposition B.6 (Exact representation by $\mathcal{G}_{\mathrm{LA}}^{(1)}$)**.** *We have*

$$T_{\mathrm{LA}} \in \mathcal{G}_{\mathrm{LA}}^{(1)}.$$

Proof. Choose one neuron with

$$a_1 = 1, \qquad \boldsymbol{w}_1 = (0, 1, 0), \qquad \boldsymbol{u}_1 = (1, 0, 0).$$

Then

$$\boldsymbol{w}_1^\top \bar{\boldsymbol{x}} = x_2, \qquad \boldsymbol{u}_1^\top \bar{\boldsymbol{x}} = x_1.$$

Since

$$\mathrm{ReLU} \preceq \tanh$$

in the ordering of $\mathcal{K}$, the pair $(\mathrm{ReLU}, \tanh)$ satisfies $p \leq q$. Choose

$$\alpha_{\mathrm{ReLU},\mathrm{ReLU}} = 1, \qquad \alpha_{\mathrm{ReLU},\tanh} = 1,$$

and set all other $\alpha_{p,q}$'s to zero. Then

$$g_{\mathrm{LA}}(\boldsymbol{x}) = (x_2)_+(x_1)_+ + (x_2)_+ \tanh(x_1) = T_{\mathrm{LA}}(x_1, x_2).$$

Thus

$$T_{\mathrm{LA}} \in \mathcal{G}^{(1)}_{\mathrm{LA}}.$$

□

Lemma B.7 (Jump profiles of fixed Type-II classes)**.** *Let*

$$f \in \mathcal{G}^{(m)}_{\sigma_{p,q}}.$$

Then

$$J_S(f) \in \mathbf{1}_{\{\sigma_p \in \mathcal{N}_1\}} \mathcal{R}^{(m)}_{\sigma_q} + \mathbf{1}_{\{\sigma_q \in \mathcal{N}_1\}} \mathcal{R}^{(m)}_{\sigma_p}.$$

Here the corresponding term is omitted if the activation is not in $\mathcal{N}_1$.

Proof. Consider one summand

$$a_k \sigma_p(\boldsymbol{w}_k^\top \bar{\boldsymbol{x}}) \sigma_q(\boldsymbol{u}_k^\top \bar{\boldsymbol{x}}).$$

A jump in ∂_{x_2} across $S = \{x_2 = 0\}$ can only arise from a factor whose first derivative has a jump, hence only from a ReLU-type factor.

If $\sigma_p \in \mathcal{N}_1$ and

$$\{\boldsymbol{x} : \boldsymbol{w}_k^\top \bar{\boldsymbol{x}} = 0\} = S,$$

then the p-factor contributes a constant jump multiplier, while the q-factor is evaluated on S. Thus the contribution has the form

$$C_k \sigma_q(r_k x_1 + s_k).$$

If this hyperplane is not equal to S, it intersects S only in a lower-dimensional set and contributes no jump for almost every x_1.

Similarly, if $\sigma_q \in \mathcal{N}_1$ and

$$\{\boldsymbol{x} : \boldsymbol{u}_k^\top \bar{\boldsymbol{x}} = 0\} = S,$$

then the contribution has the form

$$D_k \sigma_p(\tilde{r}_k x_1 + \tilde{s}_k).$$

Summing over $k = 1, \ldots, m$ proves the claim. □

Lemma B.8. *For every $m \geq 1$ and every $p, q \in [7]$,*

$$A(t) = t_+ + \tanh(t)$$

does not belong to

$$\mathbf{1}_{\{\sigma_p \in \mathcal{N}_1\}} \mathcal{R}^{(m)}_{\sigma_q} + \mathbf{1}_{\{\sigma_q \in \mathcal{N}_1\}} \mathcal{R}^{(m)}_{\sigma_p}.$$

Proof. If neither σ_p nor σ_q belongs to $\mathcal{N}_1$, the above class is $\{0\}$, whereas $A \not\equiv 0$.

If exactly one of σ_p, σ_q belongs to $\mathcal{N}_1$, then the jump profile belongs to $\mathcal{R}^{(m)}_\sigma$ for some fixed $\sigma \in \mathcal{K}$. If

$$\sigma \in \{\mathrm{id}, \mathrm{ReLU}^2, \mathrm{GELU}, \mathrm{SiLU}, \tanh\},$$

then every element of $\mathcal{R}_\sigma^{(m)}$ is C^1, while A is not C^1 because

$$A'(0^-) = 1, \qquad A'(0^+) = 2.$$

Thus $A \notin \mathcal{R}_\sigma^{(m)}$.

If

$$\sigma \in \{\mathrm{ReLU}, \mathrm{LeakyReLU}\},$$

then every element of $\mathcal{R}_\sigma^{(m)}$ is piecewise affine. However, on $(0, 1)$,

$$A(t) = t + \tanh(t),$$

which is not affine because

$$\frac{d^2}{dt^2} \tanh(t) = -2(1 - \tanh^2(t)) \tanh(t)$$

is not identically zero on $(0, 1)$. Hence $A \notin \mathcal{R}_\sigma^{(m)}$.

Finally, if both $\sigma_p, \sigma_q \in \mathcal{N}_1$, the jump profile class is a sum of two finite piecewise affine classes and is therefore still piecewise affine. Again, A is not piecewise affine on $(0, 1)$. This proves the claim. □

Proposition B.9 (Strict separation of $\mathcal{G}_{\mathrm{LA}}^{(m)}$ from fixed Type-II classes)**.** *For every $m \geq 1$,*

$$T_{\mathrm{LA}} \notin \bigcup_{p,q\in[7]} \mathcal{G}_{\sigma_{p,q}}^{(m)}.$$

Proof. Suppose, for contradiction, that

$$T_{\mathrm{LA}} \in \mathcal{G}_{\sigma_{p,q}}^{(m)}$$

for some $p, q \in [7]$. Taking the jump of the x_2-derivative across S, we obtain

$$J_S(T_{\mathrm{LA}})(x_1) = A(x_1) = (x_1)_+ + \tanh(x_1).$$

By Lemma B.7,

$$J_S(T_{\mathrm{LA}}) \in \mathbf{1}_{\{\sigma_p\in\mathcal{N}_1\}}\mathcal{R}_{\sigma_q}^{(m)} + \mathbf{1}_{\{\sigma_q\in\mathcal{N}_1\}}\mathcal{R}_{\sigma_p}^{(m)}.$$

This contradicts Lemma B.8. Therefore,

$$T_{\mathrm{LA}} \notin \bigcup_{p,q\in[7]} \mathcal{G}_{\sigma_{p,q}}^{(m)}.$$

□

B.2.4 Separation between $\mathcal{G}_{\mathrm{MoA}}^{(m)}$ and $\mathcal{G}_{\mathrm{LA}}^{(m)}$

Fix $\lambda > 0$. Define

$$B_\lambda(t) := t_+ \tanh(\lambda t),$$

and consider

$$T_{\mathrm{MoA}}(x_1, x_2) := (x_2)_+ B_\lambda(x_1) = (x_2)_+(x_1)_+ \tanh(\lambda x_1).$$

Proposition B.10 (Exact representation by $\mathcal{G}_{\mathrm{MoA}}^{(1)}$)**.** *We have*

$$T_{\mathrm{MoA}} \in \mathcal{G}_{\mathrm{MoA}}^{(1)}.$$

Proof. Choose one neuron with

$$a_1 = 1, \qquad \boldsymbol{w}_1 = (0, 1, 0), \qquad \boldsymbol{u}_1 = (1, 0, 0).$$

Then

$$\boldsymbol{w}_1^\top \bar{\boldsymbol{x}} = x_2, \qquad \boldsymbol{u}_1^\top \bar{\boldsymbol{x}} = x_1.$$

Use the branch

$$(p, q) = (\mathrm{ReLU}, \mathrm{ReLU}),$$

which satisfies $p = q$, and set

$$\boldsymbol{v}_{\mathrm{ReLU,ReLU}} = (\lambda, 0, 0).$$

For all other branches, set $\boldsymbol{v}_{p,q} = \mathbf{0}$, so their gates vanish because $\tanh(0) = 0$. Then

$$\tanh(\boldsymbol{v}_{\mathrm{ReLU,ReLU}}^{\top}\bar{\boldsymbol{x}})\sigma_{\mathrm{ReLU}}(\boldsymbol{w}_1^{\top}\bar{\boldsymbol{x}})\sigma_{\mathrm{ReLU}}(\boldsymbol{u}_1^{\top}\bar{\boldsymbol{x}}) = \tanh(\lambda x_1)(x_2)_+(x_1)_+.$$

Thus

$$T_{\mathrm{MoA}} \in \mathcal{G}_{\mathrm{MoA}}^{(1)}.$$

□

Lemma B.11 (Jump profiles of Type-II LA networks)**.** *Let*

$$f \in \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

Then

$$J_S(f) \in \mathcal{A}^{(2m)}.$$

Proof. Write

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sum_{1 \le p \le q \le 7} \alpha_{p,q}\sigma_p(\boldsymbol{w}_k^{\top}\bar{\boldsymbol{x}})\sigma_q(\boldsymbol{u}_k^{\top}\bar{\boldsymbol{x}}).$$

A jump in $\partial_{x_2} f$ across S can only arise from factors whose activation is in

$$\mathcal{N}_1 = \{\mathrm{ReLU}, \mathrm{LeakyReLU}\}$$

and whose singular hyperplane coincides with S.

If the p-factor produces the jump for neuron k, then the remaining q-factor contributes

$$\sum_{q:\, p \le q \le 7} c_{k,q}\sigma_q(r_k x_1 + s_k),$$

for some coefficients $c_{k,q}$. This is contained in one dictionary ridge of the class $\mathcal{A}^{(M)}$, after setting the missing activation coefficients to zero.

If the q-factor produces the jump for neuron k, then the remaining p-factor contributes

$$\sum_{p:\, 1 \le p \le q} \tilde{c}_{k,p}\sigma_p(\tilde{r}_k x_1 + \tilde{s}_k).$$

This is again contained in one dictionary ridge of $\mathcal{A}^{(M)}$.

Thus each neuron contributes at most two dictionary ridges to $J_S(f)$. Therefore,

$$J_S(f) \in \mathcal{A}^{(2m)}.$$

□

Lemma B.12. *For every $M \ge 1$ and every $\lambda > 0$,*

$$B_\lambda(t) = t_+ \tanh(\lambda t) \notin \mathcal{A}^{(M)}.$$

Proof. Suppose, for contradiction, that

$$B_\lambda(t) = \sum_{\ell=1}^{M} \sum_{c=1}^{7} b_{\ell,c}\sigma_c(r_\ell t + s_\ell)$$

on $[-1, 1]$.

Let $\mathcal{Z}$ be the finite set of kink locations of all ReLU-type terms

$$\mathrm{ReLU}(r_\ell t + s_\ell), \qquad \mathrm{ReLU}^2(r_\ell t + s_\ell), \qquad \mathrm{LeakyReLU}(r_\ell t + s_\ell).$$

Choose $\varepsilon > 0$ such that

$$(-\varepsilon, \varepsilon) \cap \mathcal{Z} \subset \{0\}.$$

On $(-\varepsilon, 0)$, we have

$$B_\lambda(t) = 0.$$

On each of the intervals $(-\varepsilon, 0)$ and $(0, \varepsilon)$, every ReLU-type term is a polynomial of degree at most 2. Moreover, the only ReLU-type terms whose polynomial expression can differ between the two sides are those with kink at 0.

The smooth activations

$$\mathrm{id}, \quad \mathrm{GELU}, \quad \mathrm{SiLU}, \quad \tanh$$

are real analytic near 0. Therefore, the right-hand side can be written as

$$S(t) + P_-(t) \quad \text{on } (-\varepsilon, 0),$$

and

$$S(t) + P_+(t) \quad \text{on } (0, \varepsilon),$$

where S is real analytic near 0, and P_-, P_+ are polynomials of degree at most 2.

Since $B_\lambda(t) = 0$ on $(-\varepsilon, 0)$, we have

$$S(t) + P_-(t) = 0 \quad \text{on } (-\varepsilon, 0).$$

By analyticity,

$$S(t) = -P_-(t)$$

in a neighborhood of 0. Hence on $(0, \varepsilon)$,

$$B_\lambda(t) = S(t) + P_+(t) = P_+(t) - P_-(t),$$

which is a polynomial of degree at most 2.

However, for $t > 0$,

$$B_\lambda(t) = t \tanh(\lambda t) = \lambda t^2 - \frac{\lambda^3}{3} t^4 + O(t^6).$$

The t^4 coefficient is nonzero. Therefore B_λ is not a polynomial of degree at most 2 on any interval $(0, \varepsilon)$, a contradiction. Hence

$$B_\lambda \notin \mathcal{A}^{(M)}.$$

□

Proposition B.13 (Strict separation of $\mathcal{G}_{\mathrm{MoA}}^{(m)}$ from $\mathcal{G}_{\mathrm{LA}}^{(m)}$)**.** *For every $m \geq 1$ and every $\lambda > 0$,*

$$T_{\mathrm{MoA}} \notin \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

Proof. Suppose, for contradiction, that

$$T_{\mathrm{MoA}} \in \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

Taking the jump of the x_2-derivative across $S = \{x_2 = 0\}$, we obtain

$$J_S(T_{\mathrm{MoA}})(x_1) = B_\lambda(x_1) = (x_1)_+ \tanh(\lambda x_1).$$

By Lemma B.11,

$$J_S(T_{\mathrm{MoA}}) \in \mathcal{A}^{(2m)}.$$

This contradicts Lemma B.12. Therefore,

$$T_{\mathrm{MoA}} \notin \mathcal{G}_{\mathrm{LA}}^{(m)}.$$

□

B.2.5 Proof of Theorem 4.2

The two basic inclusions have already been proved:

$$\bigcup_{p,q\in[7]} \mathcal{G}^{(m)}_{\sigma_{p,q}} \subset \mathcal{G}^{(m)}_{\mathrm{LA}} \subset \mathcal{G}^{(m)}_{\mathrm{MoA}}.$$

The first inclusion is strict because

$$T_{\mathrm{LA}}(x_1, x_2) = (x_2)_+\big((x_1)_+ + \tanh(x_1)\big)$$

satisfies

$$T_{\mathrm{LA}} \in \mathcal{G}^{(1)}_{\mathrm{LA}} \subset \mathcal{G}^{(1)}_{\mathrm{MoA}},$$

but, by Proposition B.9,

$$T_{\mathrm{LA}} \notin \bigcup_{p,q\in[7]} \mathcal{G}^{(m)}_{\sigma_{p,q}}.$$

The second inclusion is strict because

$$T_{\mathrm{MoA}}(x_1, x_2) = (x_2)_+(x_1)_+ \tanh(\lambda x_1)$$

satisfies

$$T_{\mathrm{MoA}} \in \mathcal{G}^{(1)}_{\mathrm{MoA}},$$

but, by Proposition B.13,

$$T_{\mathrm{MoA}} \notin \mathcal{G}^{(m)}_{\mathrm{LA}}.$$

Since

$$\bigcup_{p,q\in[7]} \mathcal{G}^{(m)}_{\sigma_{p,q}} \subset \mathcal{G}^{(m)}_{\mathrm{LA}},$$

we also have

$$T_{\mathrm{MoA}} \notin \bigcup_{p,q\in[7]} \mathcal{G}^{(m)}_{\sigma_{p,q}}.$$

Combining these results gives

$$\bigcup_{p,q\in[7]} \mathcal{G}^{(m)}_{\sigma_{p,q}} \subsetneq \mathcal{G}^{(m)}_{\mathrm{LA}} \subsetneq \mathcal{G}^{(m)}_{\mathrm{MoA}}.$$

□

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We believe that the abstract and introduction reflect the contributions and scope of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: In Section 4 and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have explained our method and provided experimental details in Section 3, 5, and Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The dataset we used is public and can be accessed by everyone. Also, we will release our code after the review session.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: In Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not have error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: This paper did not provide information about the computer resources needed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have confirmed that the research is conducted with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of our work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited the original papers that produced the code package or dataset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Our paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: We do not use LLMs as an important, original, or non-standard component of its core methods. LLM is used only for writing, editing, or formatting purposes.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.