Exchangeability in Neural Network Architectures and its Application to Dynamic Pruning

Pu (Luke) Yi [1] Tianlang Chen [1] Yifan Yang [2] Sara Achour [1 3]

Abstract

Researchers have explored various ways to improve the efficiency of Neural networks (NNs) by identifying and reducing the redundancy, such as pruning or quantizing unimportant weights. Symmetry in the NN architectures has been identified by prior work as a possible type of redundancy, but exploiting it for efficient inference is not yet explored. In this work, we formalize the symmetry of parameters and intermediate values in NNs using the statistical property of exchangeablility. We identify that exchangeable values in NN computation may contain overlapping information, leading to redundancy. Exploiting the insight, we derive a principled general dynamic pruning algorithm ExPrune to remove symmetry-induced redundancy on a per-input basis. We also provide an instantiation of ExPrune that performs neuron-level dynamic pruning by predicting negative inputs to ReLU activations. We evaluate ExPrune on two computer vision models, one graph model and one language model. ExPrune provides 10.98–26.3% reduction in FLOPs with negligible accuracy drop and 21.01–39.05% reduction in FLOPs with at most 1% accuracy drop. We also demonstrate that ExPrune composes with static pruning. On models that have been aggressively pruned statically, ExPrune provides additional 10.24–11.11% reduction in FLOPs with negligible accuracy drop and 13.91–14.39% reduction in FLOPs with at most 1% accuracy drop.

[1]Department of Computer Science, Stanford University, CA, USA [2]Nvidia, Santa Clara, CA 95051 [3]Department of Electrical Engineering, Stanford University, CA, USA. Correspondence to: Pu (Luke) Yi <lukeyi@stanford.edu>.

Presented in 3^{rd} Efficient Systems for Foundation Models Workshop at the International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

1. Introduction

The demand for increasingly large and powerful deep neural networks (NNs) greatly increased the memory/compute footprint of NN inference tasks (Han et al., 2022; Deng et al., 2020). To tame resource usage, researchers have developed various NN optimizations that statically reduce model size before deployment, including static pruning (Cheng et al., 2024), quantization (Gholami et al., 2022), knowledge distillation (Gou et al., 2021), and neural architecture search (Elsken et al., 2019). To a lesser extent, researchers have also developed optimizations that dynamically prune the inference computation on a per-input basis (Shazeer et al., 2017; Teerapittayanon et al., 2016). A common thread linking many of these methods is that they focus on eliminating redundancies present in the trained model.

In this work, we present ExPrune, a general, dynamic pruning optimization that enables multi-granularity (e.g., neuron/kernel/layer) partial computation on a per-input basis. This optimization capitalizes on the presence of *exchangeable* model parameters and intermediate values, a property that we show is implied by certain forms of symmetry. Exchangeability is a statistical property that implies identically distributed and symmetric interdependence among random variables (Dean & Verducci, 1990). Intuitively, exchangeable model parameters and values contain overlapping information, which induces redundant computation. In this work, we show that certain groups of parameters are trained to be exchangeable with respect to random initializations and thus produce exchangeable intermediate values – these quantities are exchangeable by construction. We also establish a formal link between exchangeability and permutation invariance of parameters, a related model property that has not yet been exploited for efficiency. Because the ExPrune is grounded in these theoretical results, it can generalize across model architectures, and we identify exchangeable parameter/value patterns in a range of modern NNs, such as Convolution Neural Networks (CNNs), Graph Neural Networks (GNNs) and transformer-based language models (LMs).

To our knowledge, this work is both the first to model NN symmetry with exchangeability and to use symmetry-induced redundancies to perform dynamic pruning. Theoreticians have previously linked NN symmetries to redundancy (Lim et al., 2024b) and studied the impact of these symmetries on NN models' ability to generalize (Dinh et al., 2017), their interpretability (Godfrey et al., 2022a;b), and loss landscape (Zhao et al., 2023). These works did not link the symmetry to exchangeability, or mechanized it to perform static/dynamic pruning.

To demonstrate the efficacy of this method, we instantiate the ExPrune algorithm to prune computations of each neuron's activation by predicting negative inputs to the ReLU activation function. We evaluate this ExPrune instantiation on CNNs, GNNs, and LMs. We demonstrate that ExPrune provides 10.98–26.3% reduction in FLOPs with negligible accuracy drop and 21.01–39.05% reduction in FLOPs with at most 1% accuracy drop. We also empirically demonstrate that ExPrune composes with static pruning. On models that have been aggressively pruned statically, ExPrune provides additional 10.24–11.11% reduction in FLOPs with negligible accuracy drop and 13.91–14.39% reduction in FLOPs with at most 1% accuracy drop. Our evaluation code is released at `https://github.com/y553546436/Exchangeablility-NN-Dynamic-Pruning`.

2. Related Work

Symmetry in Deep Learning. Symmetry in NNs and its impact have been extensively studied by theoreticians. Symmetry is known to affect model generalization (Dinh et al., 2017), the loss landscape (Zhao et al., 2023; Lim et al., 2024b), interpretability (Godfrey et al., 2022a;b), Bayesian NN inference (Kurle et al., 2022), graph metanetwork processing (Lim et al., 2024a), and aligning symmetric substructures helps improving efficiency and performance of ensemble NN models (Singh & Jaggi, 2020; Imfeld et al., 2024; Ainsworth et al., 2023). Researchers have also explored deliberately breaking the symmetry in NN architectures (Lim et al., 2024b; Pourzanjani et al., 2017; Pittorino et al., 2022; Laurent et al., 2024; Ziyin et al., 2025). A related but different concept is equivariance (Zaheer et al., 2017; Satorras et al., 2021; Cohen & Welling, 2016), a feature of special NN architectures that the NNs produce consistent outputs under symmetry transformations of the inputs. To the best of our knowledge, we are the first to use dynamic pruning to reduce symmetry-induced redundancy.

Static NN Model Optimizations. Various techniques have been proposed to derive efficient NN models with smaller sizes and fewer parameters, including pruning (Han et al., 2015; Cheng et al., 2024), quantization (Saha et al., 2024; Gholami et al., 2022; Hubara et al., 2016; Qin et al., 2020), knowledge distillation (Hinton et al., 2015; Gou et al., 2021), and neural architectural search (Elsken et al., 2019). These methods are statically applied before model deployment. In contrast, ours is dynamic and applies on a per-input basis during inference. In our evaluation (Section 5.1), we show that our method composes with static pruning.

Dynamic Pruning at Inference. Researchers have explored coarse-grained dynamic pruning methods, which are applied on a per-input basis during NN inference (Cheng et al., 2024). They explored dynamically pruning layers (Teerapittayanon et al., 2016; Tambe et al., 2021; Han et al., 2022), tokens (Anagnostidis et al., 2023), channels (Lin et al., 2017; Elkerdawy et al., 2022), spatial domain (Liu et al., 2018), and branches (Shazeer et al., 2017). These methods are specialized to certain model architectures and coarse-grained, operating on structures larger than neurons. In contrast, our method can operate at very fine granularity (neuron level) and can be generalized across multiple architectures and granularities. SnaPEA (Akhlaghi et al., 2018) and ComPreEND (Kim et al., 2022) explored neuron-level dynamic pruning for convolution. These methods are narrow in scope, focusing on CNNs and do not natively support models with normalization layers (Ioffe & Szegedy, 2015; Ba et al., 2016) or skip connections (He et al., 2015) in modern CNNs. Some other works (Wakatsuki et al., 2021; Kong et al., 2023) exploited similar patches in input feature maps to derive upper bound of the weighted sum given the partial sum, and terminated early when the upper bound is below zero. These methods only take effect on similar input patches and are specialized to CNNs and video processing. In contrast, our method works many model architectures and does not require similar input patches.

3. Exchangeability in Neural Networks

We present our theories on exchangeability in NNs in Section 3.1, and then identify exchangeable parameters and values in transformers 3.2. We use a simple ReLU-activated multi-layer-perceptron (MLP) without bias as an example to provide intuitions, shown in Figure 1. In this example, we focus on two hidden layers (colored) with weights W' and W. Denote the neuron activations of sequential layers shown as a, b, c. We have $b = \mathrm{ReLU}(W'a), c = \mathrm{ReLU}(Wb) = \mathrm{ReLU}(W\mathrm{ReLU}(W'a))$. Note that the choice of ReLU is immaterial in this section, as our theorems are independent of specific activation functions.

An obvious symmetry in the model is that permuting the hidden (colored) neurons' positions (the connected edges move along) does not change the function mapping from a to c. We delineate the symmetry using a statistical property called exchangeability. There are two forms of exchangeability in Figure 1: *exchangeable parameters* are the colored edges $\zeta_i = (W'_{1i}, W'_{2i}, W'_{3i}, W_{i1}, W_{i2})$, and *exchangeable values* are colored intermediate values $\xi_i = b_i$ or $\xi_i = W_{1i}b_i$. Intuitively, exchangeability here means that if we view ζ_i's $(i = 1, 2, 3, 4)$ as random variables, with respect to random initializations of the NNs, they are from the same distribu-

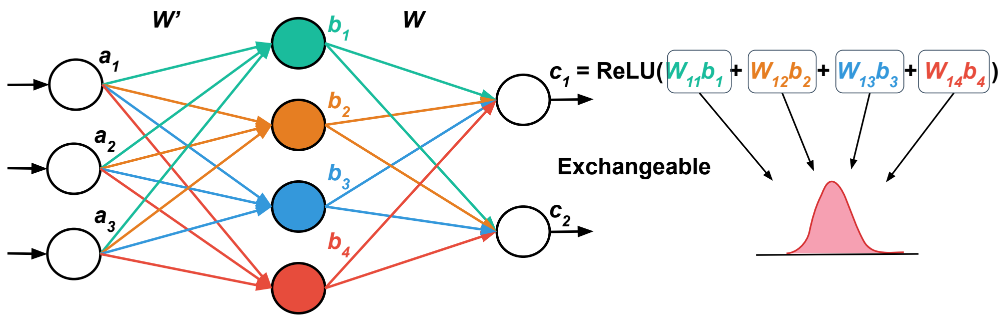

W'
W
a_1
a_2
a_3
b_1
b_2
b_3
b_4
c_1 = ReLU($W_{11}b_1$ + $W_{12}b_2$ + $W_{13}b_3$ + $W_{14}b_4$)
c_2
Exchangeable

Figure 1. Illustration of exchangeable parameters and values in ReLU-activated MLP without bias. a, b, c are neuron activations. W' and W are weight matrices. Symmetric parameters and values with different colors are exchangeable, thus identically distributed.

tion and have symmetric interdependence relationship. The same is true for ξ_i's, for any given input to the NN. We next introduce statistical exchangeability formally.

Let P be a $n \times n$ permutation matrix for some n. The following definition is taken from Kuchibhotla (Kuchibhotla, 2020) (assuming the probability density function exists).

Definition 1 (Exchangeablility). Suppose $\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{X}^n$ is a vector of random variables, ζ_i's are exchangeable iff their joint probability density function $p(\zeta)$ is invariant to input permutations, i.e., $\forall$ permutation matrix P and $\zeta \in \mathcal{X}^n$, $p(P\zeta) = p(\zeta)$.

Exchangeability is stronger than identically distributed, as it in addition implies the symmetry among ζ_i's, but it is weaker than iid since it does not exclude dependence among ζ_i's (Chow & Teicher, 2003).

3.1. Exchangeable Parameters and Values in NNs

We present the intuitive theorem statements about exchangeable parameters and intermediate values in NNs that can be derived from symmetry. Please refer to Appendix A.1 for the formal statements and proofs. Below is our key insight.

Key Insight. ζ_i's are initialized exchangeable, as most popular NN initialization schemes make the parameters iid. Some pose additional constraints on orthogonality or unit variances (Saxe et al., 2013; Mishkin & Matas, 2015), which introduce dependence among initialized parameters but still make them exchangeable. Therefore, we only need to prove that each training step *preserves the exchangeability*.

Theorem 1 (Exchangeable Parameters). Given an NN architecture, assume that the parameter groups of interest ζ_i's are initialized exchangeable. If permuting the order of ζ_i's (e.g., swap ζ_1 and ζ_2 in the NN) in any way does not change the NN function, then ζ_i's in the trained model are exchangeable with respect to random initializations.

Theorem 2 (Exchangeable Values). Under the conditions of Theorem 1, for any NN input x at inference, define ξ_i's as $\xi_i = g_{\theta',\zeta_i}(x)$, for some function g that can be parameterized by θ' (NN parameters other than ζ_i's) and one ζ_i. ξ_i's are exchangeable with respect to random NN initializations.

Note that it is possible that g_{θ',ζ_i} uses all or only a part of ζ_i. Given a group of exchangeable parameters ζ_i's, there might be multiple groups of exchangeable values ξ_i's in NN computation, each with a different g function.

3.2. Exchangeablility in Transformers

Using the insights from Theorems 1 and 2, we identify exchangeable parameters and values in transfomer architectures. Exchangeability in other popular NN architectures (MLP, CNNs, embeddings) is presented in Appendix A.2. The exchangeable parameters ζ_i's can often be found in a "*map-reduce*" pattern, where ζ_i's map the input into exchangeable values ξ_i's, which are then reduced with a permutation-invariant function.

We analyze a decoder-only transformer architecture due to its popularity in recent LLMs (Touvron et al., 2023; Radford et al., 2019; Bai et al., 2023). We formalize the simple case of single-head attention followed by a fully connected layer. Please refer to Appendix A.2 for how the basic analysis can be extended for normalization layers and skip connections (using similar techniques as in CNNs).

The key insight here is that the output dimensions of the attention layer are exchangeable. Denote the input sequence length as m. Let the vector dimensions for each token be d_1, d_2, d_3 before attention, after attention, and after fully-connected layer respectively. The input X has shape $d_1 \times m$, the key, query, value matrices K, Q, V all have shape $d_2 \times d_1$. The fully connected layer's weight W has shape $d_3 \times d_2$. The function of these two layers is $WVX\sigma((QX)^T(KX))$, where σ is column-wise softmax. We instantiate $n = d_2$ and $\zeta_i = V_{i\cdot} \oplus W_{\cdot i}$ (or $\zeta_i = K_{i\cdot} \oplus Q_{i\cdot} \oplus V_{i\cdot} \oplus W_{\cdot i}$). Permuting ζ_i's does not change the NN function because $(WP^T)(PV)X\sigma((QX)^T(KX)) = WVX\sigma((QX)^T(KX))$, for any permutation matrix P. The exchangeable values here are the attention layer's output $\xi_i = V_{i\cdot}X\sigma((QX)^T(KX))$, or the final output $\xi_i = W_{\cdot i}V_{i\cdot}X\sigma((QX)^T(KX))$. Interestingly, this indicates that the intermediate activations in every decoder layer are exchangeable.

4. ExPrune Algorithm

We next present ExPrune algorithm, a dynamic pruning algorithm for NNs that leverages the exchangeability property identified in Section 3. Exchangeable values are identically distributed. Intuitively, they may contain overlapping information, which is a form of redundancy. In this section, we present our algorithm ExPrune that removes this redundancy on a per-input basis at inference, by terminating computation of exchangeable values early, when certain confidence metric is satisfied.

4.1. General Dynamic Pruning Algorithm

Assume we want to compute a NN sub-module with function $\rho(\xi)$, where ξ_i's are exchangeable values. The input to the sub-module have been computed. ρ is invariant to permutations of ξ_i's, and can be approximated by n' ($n' \leq n$) different ξ_i's as input. Larger n' leads to more accurate approximation. Assume that we have a confidence metric $\mathrm{confident}$ that takes a set of ξ_i's as input, and outputs whether we are confident about the current result. We can also optionally have a heuristic function h, where h_i assigns a prioritization score for computation of ξ_i. For example, h can be based on the cost to compute ξ_i, or the estimated importance of ξ_i.

Algorithm 2a presents the ExPrune general algorithm. It sequentially computes ξ_i's in the order specified by h, and terminate early when the current result satisfies the confidence metric. Note that ExPrune can also optionally do confidence test less often to reduce overhead. This simple algorithm captures the idea behind many existing algorithms, e.g., finding the best arm in multi-armed bandit problem with smallest number of pulls possible (Bagaria et al., 2018), adaptively computing the top k softmax values without full evaluation (Baharav et al., 2024), and channel-level dynamic pruning in CNNs (Lin et al., 2017).

4.2. Early Negative Prediction for ReLU

We present an specific instantiation of ExPrune that dynamically prunes computation for each ReLU-activated neuron. The permutation-invariant function ρ is instantiated as summation. The algorithm exploits the property of ReLU that it outputs zero for negative input. We can therefore terminate computation of ξ_i's when we predict based on the computed ξ_i's that the final sum is likely negative. Note that this is only possible because ξ_i's are exchangeable and thus identically distributed. In some cases, the sum of ξ_i's is not directly processed by ReLU, but is first scaled and added biases, e.g., normalization layers, layer biases, shortcut connections. The negative prediction takes into account computed ξ_i's, the number of ξ_i's processed n', the scaling weight w, and the added bias b when determining confidence. We devise two negative prediction methods as follows (assuming $w = 1, b = 0$ for simplicity).

- **Threshold.** We set a predetermined threshold T and terminate when the current mean is below threshold $\sum_{i=1}^{n'} \xi'_i < n'T$. This simple method only requires one more operation per invocation, as the running sum is computed by NN already.
- **StatsTest.** We perform a Wald's test (Wald, 1992) with confidence level α, checking if $\frac{(\sum_{i=1}^{n'} \xi'_i)^2}{n' \sum_{i=1}^{n'} \xi'^2_i - (\sum_{i=1}^{n'} \xi'_i)^2} < (\Phi^{-1}(\alpha))^2$, where Φ is the cumulative density function of the standard normal distribution, and the right hand side of the simplified inequality can be stored as constant. This method introduces overhead that scales linearly the number of ξ_i's, as it requires computing running sum of the squared term. It is more costly but potentially more accurate. Note that the assumptions of Wald's test are not met by all exchangeable sequences, but we find that it works well in practice (Section 5).

Note that one parameter T or α can be shared across many neurons, e.g., all neurons in a NN layer. We discuss the limitations of the algorithm in Appendix A.5.

5. Evaluation

We evaluate ExPrune for early negative prediction for ReLU (Section 4.2) on various models with ReLU activation functions. Table 1 summarizes our benchmarks, models, and fidelity metrics. We do early negative prediction for computation of exchangeable partial activations followed by ReLU, specifically, in all convolution layers in GCN, all convolution layers except for the first one in CNNs, and the first linear layer after each attention layer in OPT. We configure ExPrune to perform one negative prediction when $n' = 32$ to reduce overhead, i.e., ExPrune invokes $\mathrm{negative\_predict}$ once when $n' = 32$ and terminate if confident, otherwise computes all other ξ_i's. We choose $n' = 32$ as many statistics methods target sample sizes of at least 30 (VanVoorhis et al., 2007). We use the floating-point operation performed (FLOPs) as our performance metric, as it is a good proxy for inference efficiency (Mirzadeh et al., 2024). We take into account the overhead of ExPrune when calculating FLOPs. We report FLOPs of the whole model for CNNs and GCNs as ExPrune is applied to most of the computation, and the FLOPs of all the linear layers in between an attention layer and a ReLU activation for OPT, as ExPrune is only applied to these layers. The FLOPs

```
Require: confident, $h$, $n$
  $\xi' = <>$
  for $i$ in argsort($h$) do
    Compute $\xi_i$, $\xi' = \xi' \oplus \xi_i$
    if confident($\xi'$) then
      return $\rho(\xi)$
    end if
  end for
  return $\rho(\xi)$
```

(a) ExPrune general algorithm.

```
Require: negative_predict, h, $w$, $b$, $n$
  $\xi' = <>$, $n' = 0$
  for $i$ in argsort($h$) do
    Compute $\xi_i$, $\xi' = \xi' \oplus \xi_i$, $n'$ += 1
    if negative_predict($\xi'$, $n'$, $w$, $b$) then
      return 0
    end if
  end for
  return w$\sum_{i=1}^{n} \xi_i$+b
```

(b) ExPrune algorithm instantiated for neuron-level early negative prediction of ReLU.

Figure 2. Pseudocode for ExPrune algorithm. $\xi_1, \ldots, \xi_n$ are exchangeable values to be computed. h is heuristic prioritization scores, and by default $h_i = i$. ρ is invariant to input permutations and can be approximately evaluated with fewer than n input ξ_i's.

Task	Dataset	Models	Fidelity Metric
Image Classification	CIFAR10	VGG11-BN, ResNet18-BN	Accuracy
Graph Property Prediction	ogbg-molhiv	Graph Convolutional NN (GCN)	ROC-AUC
Question Answering	PIQA	OPT-1.3B (pretrained model)	Accuracy

Table 1. Datasets and models. BN means the model is enhanced with BatchNorm (Ioffe & Szegedy, 2015).

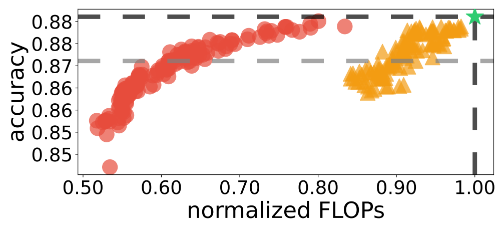
of these linear layers account for approximately $1/3$ of the total FLOPs in the unoptimized inference (Ding et al., 2024; Zhang et al., 2022; Wolf et al., 2020). We use Optuna (Akiba et al., 2019) to find optimal parameter combinations for ExPrune on validation set, and evaluate the best candidates on test set. Specifically, each layer has one parameter T (Threshold) or α (StatsTest). Please refer to Appendix A.4 for details of our experimental setup.

5.1. Results and Analysis

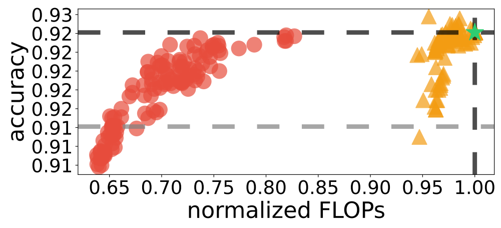
***Performance of* ExPrune.** Figure 3 shows the performance of ExPrune. We use the default evaluation order ($h_i = i$). Across four models and compared to the unoptimized baseline, ExPrune is able to deliver 10.98–26.3% reduction in FLOPs with negligible (<0.1%) fidelity drop, and 21.01–39.05% reduction in FLOPs with at most 1% fidelity drop. We find that StatsTest performs much better than Threshold for CNNs. Though StatsTest has higher overhead, it offers more accurate negative prediction for CNNs. In GCN and OPT, StatsTest and Threshold perform similarly. This is because it takes fewer FLOPs to compute each exchangeable value ξ_i in the 1D convolution of GCN and the linear layer of OPT, compared to 2D convolution in CNNs, and thus the overhead of StatsTest acounts for a larger portion in the total FLOPs.

ExPrune *and Static Pruning.* We demonstrate that ExPrune can be applied to statically pruned VGG11-BN models and offer additional reduction in FLOPs. We statically prune the VGG11-BN model by iteratively setting the 5% parameters in all convolution kernels with smallest magnitude to zero, and finetune the model on the training set to recover accuracy, following the practice of Song et al. (Han et al., 2015). Note that the static pruning also preserves the exchangeability similarly as training does (Section 3.1). We take the models at 78th, 79th, and 80th iterations, where only 1.93%, 1.83%, and 1.74% weights in convolution layers are left. We choose aggressively pruned models because we want to study how ExPrune works with already extremely compressed models. To work with these models, we configure ExPrune to let h_i equals the number of FLOPs required to compute ξ_i (number of none-zero weights in ζ_i), computing the cheap ξ_i's first. This corresponds to sorting channels of convolution kernels, which can be done statically and introduces no overhead during inference.

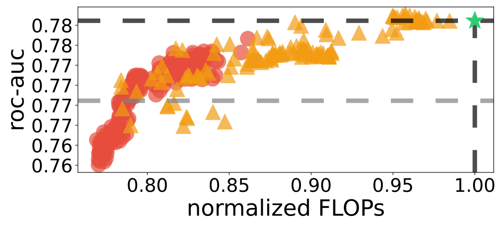
The results are shown in Figure 4. Across three pruned models, ExPrune still provides 10.24–11.11% reduction in FLOPs with negligible (<0.1%) accuracy drop, and 13.91–14.39% reduction in FLOPs with at most 1% accuracy drop, compared to the unoptimized inference on statically pruned models. Threshold achieves better results than StatsTest in pruned models, because in these models the exchangeable values ξ_i's are cheaper to compute, making the overhead of StatsTest offset the FLOPs reduction of computing fewer ξ_i's. Figure 4d shows all the points in three models compared together. We find that ExPrune combined with static pruning achieve better accuracy-performance trade-off than only static pruning. This indicates ExPrune composes with static pruning, because ExPrune is able to remove symmetry-induced redundancy, which cannot be removed by static pruning.



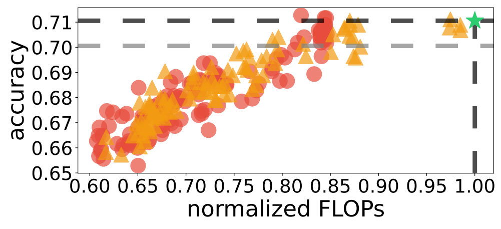
(a) VGG11-BN



(b) ResNet18-BN

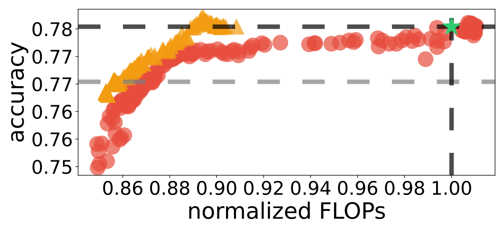


(c) GCN

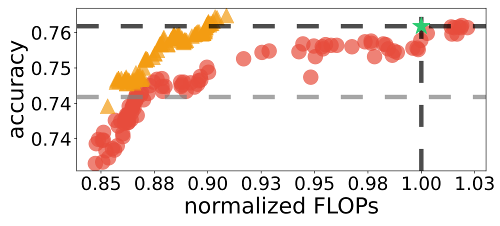


(d) OPT (FLOPs number of linear layers.)

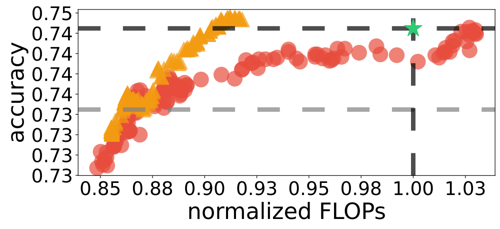
Figure 3. ● is StatsTest, ▲ is Threshold, ★ is the unoptimized baseline. — show fidelty and normalized FLOPs for unoptimized baseline. — shows baseline fidelity minus 1%.



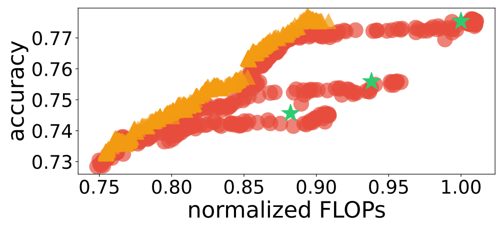
(a) 78th Iteration (1.93% dense convolution layers)



(b) 79th Iteration (1.83% dense convolution layers)



(c) 80th Iteration (1.74% dense convolution layers)



(d) 78th, 79th, and 80th iterations

Figure 4. Fidelity-FLOPs scatter plots for statically pruned VGG11-BN models. FLOPs are normalized to largest model's unoptimized baseline in (d). Colors and lines have the same meaning as in Figure 3.

6. Conclusion

We present a novel theory that formalizes a type of symmetry in NN architectures as statistical exchangeability. We identify exchangeable parameters and intermediate values in popular NNs. Exploiting this insight, we devise a general dynamic pruning algorithm ExPrune that removes the redundancy induced by exchangeable intermediate values. We instantiate ExPrune for neuron-level dynamic pruning, by terminating the computation early when a negative input to ReLU is predicted. We demonstrate that ExPrune is able to provide large FLOPs reduction in image CNNs, GCNs, and LMs. In addition, we show that ExPrune is able to compose with static pruning, provding additional FLOPs reduction on CNNs that are heavily pruned statically.

Impact Statement

ExPrune enables deploying AI with fewer resources, and makes AI more energy efficient and environment-friendly. It also facilitates AI inference on edge devices with limited resources, where data is collected and processed locally without sending it to centralized servers, enhancing user data security and privacy.

References

Ainsworth, S., Hayase, J., and Srinivasa, S. Git Re-Basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Represen-*

tations, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

Akhlaghi, V., Yazdanbakhsh, A., Samadi, K., Gupta, R. K., and Esmaeilzadeh, H. SnaPEA: Predictive early activation for reducing computation in deep convolutional neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 662–673, 2018. doi: 10.1109/ISCA.2018.00061.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Anagnostidis, S., Pavllo, D., Biggio, L., Noci, L., Lucchi, A., and Hofmann, T. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 36:65202–65223, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Bagaria, V., Kamath, G., Ntranos, V., Zhang, M., and Tse, D. Medoids in almost-linear time via multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics*, pp. 500–509. PMLR, 2018.

Baharav, T., Kang, R., Sullivan, C., Tiwari, M., Luxenberg, E., Tse, D., and Pilanci, M. Adaptive sampling for efficient softmax approximation. *Advances in Neural Information Processing Systems*, 37:117580–117613, 2024.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., Ma, J., Men, R., Ren, X., Ren, X., Tan, C., Tan, S., Tu, J., Wang, P., Wang, S., Wang, W., Wu, S., Xu, B., Xu, J., Yang, A., Yang, H., Yang, J., Yang, S., Yao, Y., Yu, B., Yuan, H., Yuan, Z., Zhang, J., Zhang, X., Zhang, Y., Zhang, Z., Zhou, C., Zhou, J., Zhou, X., and Zhu, T. Qwen technical report, 2023. URL https://arxiv.org/abs/2309.16609.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Cheng, H., Zhang, M., and Shi, J. Q. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578, 2024. doi: 10.1109/TPAMI.2024.3447085.

Chow, Y. S. and Teicher, H. *Probability theory: independence, interchangeability, martingales*. Springer Science & Business Media, 2003.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/cohenc16.html.

Dean, A. M. and Verducci, J. S. Linear transformations that preserve majorization, schur concavity, and exchangeability. *Linear algebra and its applications*, 127:121–138, 1990.

Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108 (4):485–532, 2020.

Ding, N., Tang, Y., Qin, H., Zhou, Z., Xu, C., Li, L., Han, K., Heng, L., and Wang, Y. MemoryFormer: Minimize transformer computation by removing fully-connected layers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=04EC4ZnZJj.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1019–1028. JMLR.org, 2017.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017. URL https://arxiv.org/abs/1702.03118.

Elkerdawy, S., Elhoushi, M., Zhang, H., and Ray, N. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: a survey. *J. Mach. Learn. Res.*, 20(1):1997–2017, January 2019. ISSN 1532-4435.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Godfrey, C., Brown, D., Emerson, T., and Kvinge, H. On the symmetries of deep learning models and their internal representations. In *Proceedings of the 36th International*

Conference on Neural Information Processing Systems, NIPS '22, Red Hook, NY, USA, 2022a. Curran Associates Inc. ISBN 9781713871088.

Godfrey, C., Brown, D., Emerson, T., and Kvinge, H. On the symmetries of deep learning models and their internal representations. *Advances in Neural Information Processing Systems*, 35:11893–11905, 2022b.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *Int. J. Comput. Vision*, 129(6): 1789–1819, June 2021. ISSN 0920-5691. doi: 10.1007/s11263-021-01453-z. URL https://doi.org/10.1007/s11263-021-01453-z.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic Neural Networks: A Survey . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(11):7436–7456, November 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3117837. URL https://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3117837.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1503.html#HintonVD15.

Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 4114–4122, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Imfeld, M., Graldi, J., Giordano, M., Hofmann, T., Anagnostidis, S., and Singh, S. P. Transformer fusion with optimal transport. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LjeqMvQpen.

Ioffe, S. and Szegedy, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015.

Kim, N., Park, H., Lee, D., Kang, S., Lee, J., and Choi, K. ComPreEND: Computation pruning through predictive early negative detection for ReLU in a deep neural network accelerator. *IEEE Transactions on Computers*, 71 (7):1537–1550, 2022. doi: 10.1109/TC.2021.3092205.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Kong, R., Li, Y., Yuan, Y., and Kong, L. ConvReLU++: Reference-based lossless acceleration of Conv-ReLU operations on mobile cpu. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, pp. 503–515, 2023.

Krizhevsky, A. Learning multiple layers of features from tiny images. 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Kuchibhotla, A. K. Exchangeability, conformal prediction, and rank tests. *arXiv preprint arXiv:2005.06095*, 2020.

Kurle, R., Herbrich, R., Januschowski, T., Wang, B., and Gasthaus, J. On the detrimental effect of invariances in the likelihood for variational inference. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=ft4xGJ8tIZH.

Laurent, O., Aldea, E., and Franchi, G. A symmetry-aware exploration of bayesian neural network posteriors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=FOSBQuXgAq.

Lim, D., Maron, H., Law, M. T., Lorraine, J., and Lucas, J. Graph metanetworks for processing diverse neural architectures. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=ijK5hyxs0n.

Lim, D., Putterman, T., Walters, R., Maron, H., and Jegelka, S. The empirical impact of neural parameter symmetries, or lack thereof. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=pCVxYw6FKg.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/a51fb975227d6640e4fe47854476d133-Paper.pdf.

Liu, Z., Xu, J., Peng, X., and Xiong, R. Frequency-Domain dynamic pruning for convolutional neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/a9a6653e48976138166de32772b1bf40-Paper.pdf.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations (ICLR 2013)*, 2013.

Mirzadeh, S. I., Alizadeh-Vahid, K., Mehta, S., del Mundo, C. C., Tuzel, O., Samei, G., Rastegari, M., and Farajtabar, M. ReLU strikes back: Exploiting activation sparsity in large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=osoWxY8q2E.

Mishkin, D. and Matas, J. All you need is a good init. In *4th International Conference on Learning Representations (ICLR 2016)*, 2015.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Pittorino, F., Ferraro, A., Perugini, G., Feinauer, C., Baldassi, C., and Zecchina, R. Deep networks on toroids: removing symmetries reveals the structure of flat regions in the landscape geometry. In *International Conference on Machine Learning*, pp. 17759–17781. PMLR, 2022.

Pourzanjani, A. A., Jiang, R. M., and Petzold, L. R. Improving the identifiability of neural networks for bayesian inference. In *NIPS workshop on bayesian deep learning*, volume 4, pp. 31, 2017.

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Saha, R., Sagan, N., Srivastava, V., Goldsmith, A., and Pilanci, M. Compressing large language models using low rank and low precision decomposition. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=lkx3OpcqSZ.

Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) equivariant graph neural networks. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9323–9332. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/satorras21a.html.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*, pp. 1–14. Computational and Biological Learning Society, 2015.

Singh, S. P. and Jaggi, M. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33, 2020.

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A. M., Brooks, D., and Wei, G.-Y. EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pp. 830–844, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385572. doi: 10.1145/3466752.3480095. URL https://doi.org/10.1145/3466752.3480095.

Teerapittayanon, S., McDanel, B., and Kung, H. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016. doi: 10.1109/ICPR.2016.7900006.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

VanVoorhis, C. W., Morgan, B. L., et al. Understanding power and rules of thumb for determining sample sizes. *Tutorials in quantitative methods for psychology*, 3(2): 43–50, 2007.

Wakatsuki, T., Kanai, S., and Fujiwara, Y. Accelerate inference of cnns for video analysis while preserving exactness exploiting activation sparsity. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems*, volume 3, pp. 860–872, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/file/b9799a12d683d136cc817f94b73a8938-Paper.pdf.

Wald, A. Sequential tests of statistical hypotheses. In *Breakthroughs in statistics: Foundations and basic theory*, pp. 256–298. Springer, 1992.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. MoleculeNet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 3394–3404, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. Opt: Open pre-trained transformer language models, 2022.

Zhao, B., Ganev, I., Walters, R., Yu, R., and Dehmamy, N. Symmetries, flat minima, and the conserved quantities of gradient flow. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9ZpciCOunFb.

Ziyin, L., Xu, Y., and Chuang, I. L. Remove symmetries to control model expressivity. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Gv0TOAigIY.

A. Technical Appendices and Supplementary Material

A.1. Formalism

The following theorem states a condition under which a transformation on exchangeable random variables preserves their exchangeability (Kuchibhotla, 2020; Dean & Verducci, 1990) (stronger condition).

Theorem 3 (Exchangeability Preservation). Suppose $\zeta = (\zeta_1, \dots, \zeta_n) \in \mathcal{X}^n$ is a vector of exchangeable random variables. Fix a transformation $G : \mathcal{X}^n \to \mathcal{X}^n$. If G is permutation equivariant, i.e., $\forall$ permutation matrix P and $\zeta \in \mathcal{X}^n$, $PG(\zeta) = G(P\zeta)$, then $G(\cdot)$ preserves exchangeability of ζ.

Parameter space symmetry is a relevant concept to describe the symmetry in parameter space. Given a NN architecture with N real-valued parameters, we denote the NN function parameterized by $\theta \in \mathcal{R}^N$ as $f_\theta : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}, \mathcal{Y}$ are input and output spaces respectively. A parameter space symmetry is defined as follows (Lim et al., 2024b).

Definition 2 (Parameter Space Symmetry). A function $\omega : \mathcal{R}^N \to \mathcal{R}^N$ is a parameter space symmetry if $f_{\omega(\theta)}(x) = f_\theta(x), \forall x \in \mathcal{X}, \theta \in \mathcal{R}^N$, i.e., $f_{\omega(\theta)}$ and f_θ are the same function for any parameters $\theta \in \mathcal{R}^N$.

In the example (Figure 1), each permutation on $\zeta_i = (W'_{1i}, W'_{2i}, W'_{3i}, W_{i1}, W_{i2})$ (the associated weights of each hidden neuron) is a parameter space symmetry.

Denote the parameters of interest as $\zeta = (\zeta_1, \dots, \zeta_n)$, where $\forall i, \zeta_i \in \mathcal{R}^m, \zeta \in (\mathcal{R}^m)^n$. For simple notations, when clear from the context, we use the same variable name x to denote a mn-long vector in $\mathcal{R}^{mn}$ or a n-long vector of m-long vectors $(\mathcal{R}^m)^n$ with the same elements. This means x can be automatically flattened ($(\mathcal{R}^m)^n \to \mathcal{R}^{mn}$) as $x = \oplus_{i=1}^n x_i$, where $\oplus$ is vector concatenation, or reshaped ($\mathcal{R}^{mn} \to (\mathcal{R}^m)^n$) as $x_{ij} = x_{(i-1)m+j}$. Assume that $\theta = \theta' \oplus \zeta$, where θ' is a vector of other parameters. Define a function $\omega_P : \mathcal{R}^N \to \mathcal{R}^N$ as $\omega_P(\theta) = \theta' \oplus P\zeta$, where $P\zeta = \oplus_{i=1}^n (P\zeta)_i \in \mathcal{R}^{mn}$, for some permutation matrix P. In other words, ω_P permutes ζ_i's in θ with the permutation matrix P.

Theorem 4 (Exchangeable Parameters). Given an NN architecture with function f_θ, assume that ζ_i's are initialized exchangeable. If for any permutation matrix P, ω_P defined above is a parameter space symmetry, then ζ_i's are exchangeable after training, with respect to random initializations.

Proof. Here we assume using simple gradient descent for supervised learning on classification tasks for simplicity, but the the proof easily generalizes to other optimization algorithms. As we are only interested in ζ_i's, we formalize a training step as a transformation on only the ζ_i's.[1] Denote the NN loss function as $L_\zeta : \mathcal{X} \times \mathcal{Y} \times \mathcal{R}^{N-mn} \to \mathcal{R}$, parameterized by ζ. $L_\zeta(x, y, \theta') = \psi(f_\theta(x), y)$ for some metric function ψ such as cross entropy. A training step $G_S : (\mathcal{R}^m)^n \to (\mathcal{R}^m)^n$ takes a training batch of B samples $S \in (\mathcal{X} \times \mathcal{Y})^B$, and does a gradient descent step $G_S(\zeta) = \zeta - \gamma \sum_{(x,y)\in S} \nabla_\zeta L_\zeta(x, y, \theta')$, where γ is the learning rate, and $\nabla_\zeta L_\zeta(x, y, \theta')$ is the gradient of L_ζ with respective to ζ, evaluated at (x, y, θ').

According to Theorem 3, we only need to prove that each training step G_S is equivariant with respect to any permutation P of ζ_i's, i.e., $PG_S(\zeta) = G_S(P\zeta)$, for all ζ, θ', S. We have

$$PG_S(\zeta) = P(\zeta - \gamma \sum_{(x,y)\in S} \nabla_\zeta L_\zeta(x, y, \theta')) = P\zeta - \gamma \sum_{(x,y)\in S} P\nabla_\zeta L_\zeta(x, y, \theta')$$

$$G_S(P\zeta) = P\zeta - \gamma \sum_{(x,y)\in S} \nabla_{P\zeta} L_{P\zeta}(x, y, \theta').$$

Note that $\nabla_{P\zeta} L_{P\zeta}$ is not the same as $\nabla_\zeta L_{P\zeta}$, as the derivatives in the gradient vector of the former case shall match the parameter order in $P\zeta$ rather than ζ. Since ω_P is a parameter space symmetry, by definition 2, $\forall \zeta \in (\mathcal{R}^m)^n$, f_θ and $f_{\omega_P(\theta)}$ are the same function, and thus L_ζ and $L_{P\zeta}$ are the same function. Therefore, it is straightforward that $\nabla_{P\zeta} L_{P\zeta}(x, y, \theta') = P\nabla_\zeta L_\zeta(x, y, \theta')$ for all x, y, ζ, θ'. This implies that $PG_S(\zeta) = G_S(P\zeta)$. $\square$

Theorem 5 (Exchangeable Values). Under the conditions of Theorem 4, after training, for any NN input $x \in \mathcal{X}$, define ξ_i's as $\xi_i = g_{\theta', \zeta_i}(x)$, for some function g that can be parameterized by θ' and one ζ_i. ξ_i's are exchangeable with respect to random NN initializations.

Proof. Since θ' is shared among ξ_i's, the only variable in ξ_i is ζ_i. $(\zeta_1, \dots, \zeta_n) \to (\xi_1, \dots, \xi_n)$ is an exchangeability preserving transformation by Theorem 3. Therefore, ξ_i's are exchangeable. $\square$

Note that it is possible that g_{θ', ζ_i} uses all or only a part of ζ_i in its computation. Given a group of exchangeable parameters ζ_i's, there might be multiple groups of exchangeable values ξ_i's in NN computation, each with a different g function.

[1] In fact, one can prove that permuting ζ_i's does not affect the update to other parameters in θ'.

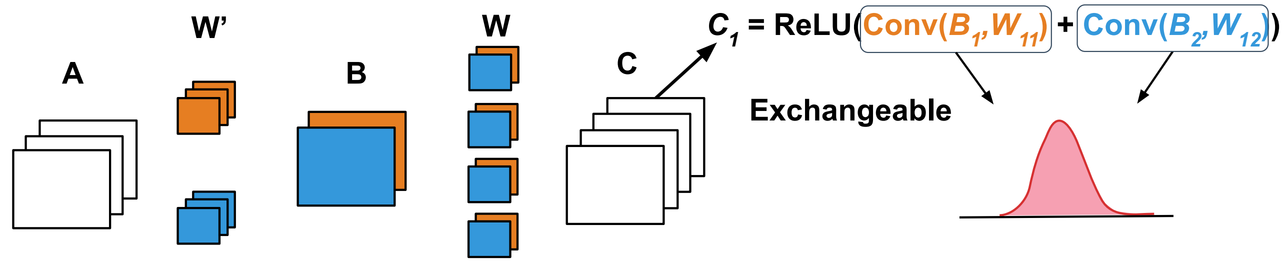


Figure 5. Illustration of exchangeable parameters and values in a ReLU-activated CNN without bias. A, B, C are neuron activations. W' and W are convolution kernels. Symmetric parameters and values with different colors are exchangeable, thus identically distributed.

A.2. Exchangeable Parameters and Values in Popular Neural Networks

We omit the proof of parameter space symmetry when it is straightforward.

Multi-Layer Perceptron: A small MLP example without bias is shown in Figure 1. In general, for two sequential fully-connected layers with weight matrices and biases (W', b') and (W, b), we can instantiate n as the number of neurons in the middle, $\zeta_i = W'_{i\cdot} \oplus b'_i \oplus W_{\cdot i}$ (exchangeable parameters) and $\xi_i = \sigma(W'_{i\cdot}a + b'_i)W_{\cdot i}$, or $\xi_i = \sigma(W'_{i\cdot}a + b'_i)$ (exchangeable values), where σ is any activation function. Since biases in other NN structures can be similarly handled, for simplicity, we assume that there is no bias in the following cases.

Convolutions: We analyze 2D convolution in CNNs with 1 group as an example. The analysis easily extends to 1D or multi-dimensional convolutions and grouped convolutions, covering models such as GCNs. An illustration of the exchangeable structures in CNNs is shown in Figure 5. Consider two sequential convolution layers where the number of channels of input A, first output B are C_1, C_2 respectively. The convolution weights W' and W have shapes (C_2, C_1, KW_1, KH_1) and (C_3, C_2, KW_2, KH_2) (output channels, input channels, kernel width, kernel height) respectively. Define the 2D convolution function conv, such that $B = \mathrm{conv}(A, W')$. Now we consider multiple common structures in CNNs. When we instantiate ζ, we assume that the weight tensors are flattened when concatenated together.

Two consecutive convolution layers. We instantiate $n = C_2$, $\zeta_i = W'_{i\cdot} \oplus W_{\cdot i}$, and $\xi_i = \mathrm{conv}(\sigma(\mathrm{conv}(A, W'_{i\cdot})), W_{\cdot i})$, or $\xi_i = \sigma(\mathrm{conv}(A, W'_{i\cdot}))$, for any activation function σ.

Normalization Layers. Normalization layers (NL) such as batch normalization (Ioffe & Szegedy, 2015) and layer normalization (Ba et al., 2016) are popular for stabilizing training. It applies a channel-wise scaling and bias. In CNNs, one normalization layer is usually inserted right before or after each activation function. We present an analysis for the latter case but the analysis also applies to the former case. Denote the NL functions as NL with parameters nl, and NL_i denotes the NL applied to the i-th channel, parameterized by $\mathrm{nl_i}$. We instantiate $n = C_2$, $\zeta_i = W'_{i\cdot} \oplus \mathrm{nl}_i \oplus W_{\cdot i}$, and $\xi_i = \mathrm{conv}(\mathrm{NL}_i(\sigma(\mathrm{conv}(A, W'_{i\cdot}))), W_{\cdot i})$.

One convolution layer followed by a fully connected layer. For simplicity, we assume B goes through channel-wise average pooling pool ($\mathrm{pool}(B)$ is of shape C_2) before the fully connected layer. The analysis easily generalizes to other/no pooling as well. The second layer then has function $W\mathrm{pool}(B)$. We instantiate $n = C_2$, $\zeta_i = W'_{i\cdot} \oplus W_{\cdot i}$, and $\xi_i = \mathrm{pool}(\mathrm{conv}(A, W'_{i\cdot}))W_{\cdot i}$.

Skip Connections. Skip connections from A to C do not affect the rest of the analysis. Skip connections from a layer before A to B, and from B to a layer after C are similar, and we present an analysis for the former case. We can simply include in ζ_i the parameters that produce, and also the parameters that consume, the i-th channel of the shortcut values. The rest of the analysis is unaffected.

Embeddings Embeddings are used for many tasks such as NLP (Mikolov et al., 2013; Pennington et al., 2014) and graphs (Kipf & Welling, 2017). Embeddings are "distributed" representations as the relevant information is represented in many dimensions. Our theory characterizes the symmetry often present in the embedding dimensions as statistically exchangeable. For example, in Word2vec (Mikolov et al., 2013), the NN architecture is simply an embedding layer (look-up table) and a fully connected layer, followed by a softmax operation σ. It takes a word as input, and output likelihoods for each word to be in its context. Let m be the number of words, n be the embedding dimension, A be the embedding matrix for all the words, M be the fully connected layer weight matrix (both of size $m \times n$). The NN takes into input a word index k, and outputs $\sigma(MA_k^T)$. We instantiate $\zeta_i = A_{\cdot i} \oplus M_{\cdot i}$ and $\xi_i = A_{ki}M_{\cdot i}$. Alternatively, $\xi_i = A_{ki}$, which indicates that

```
Require: winner_confident, $h$, $n$
  $\xi' = <>$, class_scores $= < 0, 0, \ldots, 0 >$, $n' = 0$
  for $i$ in argsort$(h)$ do
    Compute $\xi_i$, $\xi' = \xi' \oplus \xi_i$, class_scores += $\xi_i$, $n'$ += 1
    cur_winner = argmax(class_scores)
    if winner_confident($\xi'$, class_scores, cur_winner, $n'$) then
      return cur_winner
    end if
  end for
  return argmax(class_scores)
```

Figure 6. Pseudocode for ExPrune algorithm instantiated for prediction head in classification. $\xi_1, \ldots, \xi_n$ are exchangeable values to be computed. h is heuristic prioritization scores, and by default $h_i = i$. ξ_i's and class_scores are in $\mathcal{R}^k$, where k is the number of classes.

learned embedding dimensions are exchangeable.

A.3. ExPrune Instantiated for Prediction Head in Classification

We present another possible instantiation of ExPrune for prediction head in NNs, i.e., the last fully-connected layer preceding the argmax, which is widely used in modern NNs such as MLP, CNNs, GNNs, transformers for classification tasks. Note that this instantiation is not evaluated in this work as the prediction head often comprises only a small portion of the total computation. However, one can potentially adapt it to replace argmax with softmax, and this sub-structure accounts for a large portion of computation in the attention computation in transformers. Let k be the number of classes, each $\xi_i \in \mathcal{R}^k$ provides a partial score for each class. Note that the granularity of this instantiation is larger than the instantiation in Section 4.2, because each ξ_i is a vector instead of a single value, i.e., it is the computation from one neuron to all k output neurons, not to just 1 output neuron. We provide two possible winner_confident metrics as follows.

Threshold. We set a predetermined threshold and terminate when the margin of mean between the largest and second largest scores is below threshold, i.e., terminate when $c_1 - c_2 < nT$, where the largest and second largest values in class_scores as c_1, c_2, respective.

StatsTest. We can conduct a Wald's test (Wald, 1992) for cur_winner against each of other classes, and terminate when all $k - 1$ tests pass. Each test compares the mean of the largest class score to another class score. Since multiple tests are involved, we can use the Holm-Bonferroni method (Holm, 1979) to assign adjusted confidence levels for each test, given the overall type-I error bound α.

A.4. Evaluation Details

All the details can be found in our codebase at `https://github.com/y553546436/Exchangeability-NN-Dynamic-Pruning`.

Datasets and Models. We have 3 datasets, covering 3 different tasks and input types. The test set labels of PIQA (Bisk et al., 2020) are not published, so we use the validation set as the test set, and 10% of the training set as the validation set. This is reasonable because the pretrained OPT model was not trained on PIQA training set. For CIFAR10 (Krizhevsky, 2009), we use a fixed split of the training set, with 90% samples used for training, 10% used as validation set, and use the default test set. We use the default dataset splits for ogbg-molhiv dataset (Wu et al., 2018). We use popular CNNs for image inputs, GCN for graph inputs, and pretrained transformer language model for text inputs. We choose OPT (Zhang et al., 2022) because it is an off-the-shelf ReLU activated language model. We use the default OPT architecture from HuggingFace (Wolf et al., 2020). The GCN architectures follow Kipf and Welling (Kipf & Welling, 2017). We use the adapted CNNs (Simonyan & Zisserman, 2015; He et al., 2015) for CIFAR10 from `https://github.com/kuangliu/pytorch-cifar`. Specifically, the first layer and the pooling layer before the prediction head are adapted in size for the image size and class number in CIFAR10. We train image CNNs and GCNs locally using the training set, and used the pretrained weights for OPT (Zhang et al., 2022) from HuggingFace (Wolf et al., 2020). For local training of CNNs, we use AdamW optimizer, 5×10^{-3} learning rate, 16 batch size, 10^{-4} weight decay, 1cycle learning rate scheduler, and 30 epochs. When statically pruning VGG11-BN, we use the same training scheme in every pruning iteration to finetune the model after 5% parameters in all convolution kernels are set to zero. For local training of GCN, we use AdamW optimizer, 10^{-3} learning rate, 32 batch size,

10^{-4} weight decay, 1cycle learning rate scheduler, and 80 epochs.

Baselines and Metrics. As noted in Section 2, there is no neuron-level dynamic pruning algorithm for modern NN architectures to the best of our knowledge. Therefore, we configure ExPrune with threshold and StatsTest, described in Section 4.2, and compare with the unoptimized inference. Threshold uses one FLOP as the threshold $n'T$ is stored as a constant. StatsTest uses $2n' + 6$ FLOPs as it uses $2n'$ FLOPs for computing the sum of $\xi_i'^2$, and 6 more to compute the Wald's statistic and compare it to the stored threshold.

Hyperparameter optimization. As different layers in NN may have different error sensitivities, we use Optuna (Akiba et al., 2019) to find optimal parameter combinations for ExPrune. Specifically, each layer has one parameter T (Threshold) or α (StatsTest). We tune the hyperparameters for 2000 trials on the validation set. Using the data we have on validation set, we select a subset of promising parameter configurations to run on the test set. This emulates the process of selecting the hyperparameter combination for deployment. We iteratively select all the combinations on the fidelity-FLOPs Parato Frontier (dubbed one Parato slice), remove them from the set, and choose all on the next Parato slice. We include all points on the first 5 Parato slices.

Across all models, we set the range of α as $[0, 0.5]$ in StatsTest for all layers. The range of T in Threshold is $[-30, 0]$ for CNNs, $[-1, 0]$ for GCN, $[-0.005, 0]$ for OPT. For statically pruned models, we additionally add a parameter r in range $[0.1, 0.5]$ to tune for each layer, which controls when ExPrune is disabled. When the ratio of the total FLOPs that can be potentially pruned for a channel's computation to the overhead of ExPrune is below r, ExPrune is disabled. For each model, we provide a set of initial points to warm up Optuna's surrogate model. Please see our code for details.

A.5. Limitations and Future Work

Exchangeability and Confidence Test. Exchangeability of a finite sequence of random variables could indicate either that they are conditionally iid, or that they form a case of sampling without replacement. The latter case does not satisfy the assumptions made by Wald's test (Wald, 1992). We do not know which case that exchangeable parameters and values in NNs fall into exactly. It is interesting and valuable to investigate stronger statistical properties to describe them. It is of great practical value to devise better confidence test for ExPrune that is more accurate and more efficient.

Other* ExPrune *Instantiations. We provided one ExPrune instantiation that targets ReLU activation, but ExPrune can work with other activation functions (e.g., GELU (Hendrycks & Gimpel, 2023) and SiLU (Elfwing et al., 2017)) with adjusted confidence metric. It should be noted that ReLU can effectively replace GELU or SiLU for efficient inference (Mirzadeh et al., 2024). ExPrune can also be instantiated to other granularities (e.g., kernel, layer, branches) and target other forms of partial evaluation.

ExPrune *and Hardware Acceleration.* ExPrune reduces FLOPs but also breaks certain structures of computation which are exploited in hardware accelerators. We note that SnaPEA (Akhlaghi et al., 2018) shared similar compute patterns to ExPrune, and it is demonstrated that the FLOP reduction of SnaPEA could translate to real speed up and energy saving with special architecture support. It is of great practical value to design special architectures and hardware support to accelerate ExPrune.