# SUPER SEEDS: extreme model compression by trading off storage with compute

**Nayoung Lee**[*] [w], **Shashank Rajput**[*] [w], **Jy-yong Sohn**[w] **Hongyi Wang**[c] **Alliot Nagle**[w]
**Eric P. Xing**[mpc], **Kangwook Lee**[w] **Dimitris Papailiopoulos**[w]

[c] Carnegie Mellon University [m] MBZUAI [p] Petuum, Inc.
[w] University of Wisconsin-Madison

## Abstract

We discover the existence of SUPER SEEDS, extremely compressible models, that can be decompressed to their full accuracy within a few epochs of dense training. SUPER SEEDS achieve compression rates of $250 - 1000\times$ and are orders of magnitude smaller than state-of-the-art model compression baselines, which typically require minimal accuracy loss. We discover that one can trade off some accuracy of the model with significant gains in storage cost, at a relatively small decompression cost. The main application of SUPER SEEDS is archival storage, and settings where communicating a fully trained model carries significant cost. Our Code is available at `https://anonymous.4open.science/r/SuperSeed-4834/`

## 1 Introduction

The recent success of extremely large-scale, overparameterized neural networks (Brown et al., 2020; Ramesh et al., 2021; Jumper et al., 2021) in various domains has kept motivating the community to build models with an ever increasing number of parameters. These large networks consequently require high memory footprint, storage space, and communication costs for model compression.

Typically, model compression techniques aim to maintain the accuracy of the original uncompressed model, while minimizing the numbers of bits required to store the model parameters. The state-of-the-art model compression techniques are able to reach moderate to high levels of compression across many modern tasks (Han et al., 2015a,b; Wen et al., 2016; Frankle & Carbin, 2018; Gale et al., 2019; Hubara et al., 2017; Wiedemann et al., 2020; Khodak et al., 2021; Wang et al., 2021). This is typically achieved by compressing through sparsification and low-precision, and the use of optimized data structures. For instance, for standard benchmarks such as CIFAR and ImageNet classification, it has been shown that neural networks can be compressed by $10 - 50\times$, without loss of accuracy (Han et al., 2015a; Wiedemann et al., 2020; Isik et al., 2021).

We note that the requirement for no loss of accuracy stems from the hidden assumption that compressed models will be deployed without any additional processing. In this work, we challenge this very assumption made in the literature. That is, we consider a more general model compression problem where compressed models can be 'retrained' before being deployed, assuming that the same training data is available. Under this new setting, compressed models are deemed good as long as they can 'recover' high accuracies after appropriate retraining.

Clearly, this new setting will allow us to enjoy a much higher level of compression rate than before. Indeed, not storing anything is a valid scheme as one can always recover the full accuracy by training the model from scratch. However, this would require a large compute cost. On the other hand, the classic model compression algorithms can be viewed as another extreme case where models are

[*] Authors contributed equally to this paper.

UpML, 2022

compressed only up to the point where no retraining is required. What has not been explored is the trade-off between these two extreme cases – can we achieve a higher model compression rate while requiring minimal retraining cost? Inspired by this, we attempt at answering the following question on the fundamental trade-off between compression rate and retraining cost:

*For a given level of model compression, what is the minimum computation required to recover a high accuracy model?*

Here, we assume that during "decompression" we have access to the full training dataset, and hence the storage cost is measured only in terms of the size of the compressed model. This assumption is reasonable in many practical scenarios where one must compress a large number of models trained on a common dataset. The computation cost is measured as the number of retraining epochs required to attain the full accuracy.

Surprisingly, we find that indeed we can achieve very high levels of compression, by only requiring a relatively small amount of retraining to reach high accuracy. For various benchmark image classification tasks, we empirically show the existence of $1000\times$ compressed models that can be retrained to full accuracy within just 20% – 66% of the epochs needed for training a randomly initialized model. We dub this quickly-recoverable & highly compressed model as SUPER SEEDS. These SUPER SEEDS are computationally easier to find than other compressed models which maintain accuracy and lottery tickets (Frankle & Carbin, 2018). Based on empirical observations, we further conjecture that there is a *lower bound* on the number of bits SUPER SEEDS can be compressed into, to speed up the training of neural network. Given less information than this threshold, it is infeasible to outperform training from scratch.

We further confirm that SUPER SEEDS passes sanity checks: (1) when we shuffle or reinitialize the weights of SUPER SEEDS, it has worse convergence speed compared with SUPER SEEDS, and (2) when we sample a random network having the same layer-wise sparsity pattern with SUPER SEEDS, it also converges slower than SUPER SEEDS. This shows that SUPER SEEDS indeed achieve a non-trivial storage-computation tradeoff.

## 2 Related Work

**Reducing storage/computation burden.** To reduce the required resources for training/inference, model compression has been considered for several decades. This paper focuses on two main techniques for model compressing: pruning and quantization. Various pruning/quantization techniques (Gao et al., 2019; Park et al., 2020; Lee et al., 2020; Isik et al., 2021; Han et al., 2015a; Fan et al., 2020) have been suggested to compress the model while maintaining the full accuracy of dense model. The present paper has a different goal with these existing works: we allow losing accuracy after *extreme* compression and aim at reaching the full accuracy after retraining for only few epochs. Recently, several methods (Hu et al., 2021; Houlsby et al., 2019) have been suggested to reduce the storage/computation burden of fine-tuning language models on downstream tasks.

**Training a sparse network.** Given a highly sparse network which does not preserve full accuracy of dense model, one can consider two options for training this network: (1) sparse training which only updates the unpruned weights, and (2) reviving the pruned weights and moves towards the dense regime. Most of the existing works lie in the first category, e.g., lottery tickets (Frankle & Carbin, 2018; Frankle et al., 2020a; Wang et al., 2019; Sreenivasan et al., 2022; Su et al., 2020; Frankle et al., 2020b). Unlike these existing works, the current paper is in the second category: we allow moving from sparse regime to dense regime, and expects quick recovery of full accuracy.

**Speeding up convergence.** Recall that our work aims at reducing the number of epochs required to reach full accuracy, so that we can enjoy a good tradeoff between computation and storage. Some works have focused on the ability to train or fine-tune a model within only few epochs. The authors of (Lai et al., 2021) empirically showed the existence of sparse subnetwork that is quickly fine-tunable to downstream automatic speech recognition tasks. However, this observation is on the non-extreme (greater than 10% sparsity) compression regime, while our work explores the quick fine-tunability of extremely sparse models (around 0.1% sparsity). An interesting work (Smith & Topin, 2019) showed that learning rate control can speed up the convergence of a dense model significantly. Unlike (Smith & Topin, 2019), our work focuses on compression-retraining framework and explores the convergence speed of a sparse model transformed to a dense model with full accuracy.

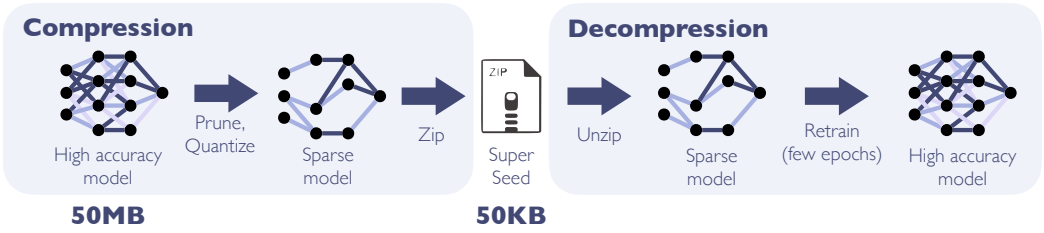


Figure 1: Conceptual visualization of a use case of SUPER SEEDS, a highly compressed model that can be retrained quickly. The size of a trained dense small VGG-16 model for CIFAR-10 is 50MB. Our goal is to archive an extremely compressed version (50KB) of this model such that it can be quickly recovered to full accuracy within few epochs of retraining. We first generate and store such compressed model (SEED$_{PQ}$) using pruning, quantization, and a customized method of storing sparse networks. Then, when we want to deploy a model with high predictive performance, we load the stored seed and retrain it. The suggested method trades off (1) storage for archiving the model and (2) computation for retraining the model to reach full accuracy.

## 3 SUPER SEEDS

In this section, we formally describe the proposed framework for creating and using SUPER SEEDS.Fig. 1 depicts the use case for SUPER SEEDS, where we want to compress a good model into a SUPER SEED for storage and archival purposes, and then quickly decompress it for usage and deployment. In the following, we describe each of the two phases (compression and decompression).

**Framework** Our framework consists of two phases: (1) compression and (2) decompression. Note that unlike existing model compression techniques that aim for compressing models without compromising their performance, we take post-processing into account during decompression. In other words, we consider a lossy compression, with a expectation that the compressed model can quickly recover full performance with just a little retraining.

We emphasize that the novelty of this paper is in proposing the extreme compression-decompression framework for better storage-computation tradeoff, not in the algorithm for generating SUPER SEEDS. One can use any off-the-shelf pruning/quantization schemes in our framework.
While SUPER SEEDS are not restricted to specific compression and decompression methods, the particular scheme that we have used to generate our SUPER SEEDS and decompress it, is based on pruning (**P**) followed by quantization (**Q**). We use the subscript to indicate the algorithm used to create our SUPER SEEDS, for example, SEED$_{PQ}$ using pruning and quantization. We use this technique as an example to explain our compression - decompression framework.

**Phase 1, Compression:** We use a modified iterative magnitude pruning (IMP) with learning rate rewinding Renda et al. (2020) to prune our model and achieve extreme sparsity, *e.g.,* 0.3% sparsity. That is, we go through multiple rounds of training and then pruning 20% or 25% of the smallest weights in the model (global pruning). After pruning in each round, the learning rate is rewinded back to the initial setting.

To further compress the model, we quantize its weights using adaptive quantization techniques based on k-means clustering. We then serialize the weights of the compressed model and store it into a memory-efficient way. As shown in Fig. 1, SUPER SEEDS created in this way achieve a 1000$\times$ reduction in storage compared to original high accuracy model. Note that at this compression level, the model loses significant accuracy. However, as we show in Section 4, the model retains enough structure and information to be easily trained back to full accuracy in the decompression phase.

**Phase 2, Decompression:** Given SEED$_{PQ}$, we first deserialize it into a neural network, and then retrain the model to reach high accuracy. We show that SEED$_{PQ}$ require 1.6$\times$ to 5$\times$ fewer epochs to reach the full accuracy, compared with training from scratch. The small size of SEED$_{PQ}$, and the fact that they can be retrained to full accuracy quickly, allow them to achieve a good tradeoff between computation and storage (or communication).

We also highlight that creating SEED$_{PQ}$ is significantly cheaper than the original method Renda et al. (2020). While iterative pruning methods require high computational cost since the model aims to

retrain to full accuracy after each pruning round, we show in Section 4 that using only 5 epochs ($\times \frac{1}{18}$) from Renda et al. (2020)) for training at each round, we can obtain SUPER SEED.

# 4 Experiments

Here we show experimental results on the proposed SUPER SEEDS and provide evidences that our framework achieves significant storage-computation trade-off in various classification tasks. We also report experimental results that helps better understand the behavior of SUPER SEEDS.

**Tasks.** We test our algorithm on **(Task 1)** CIFAR-10 (Krizhevsky et al., 2009) classification on ResNet-18 (He et al., 2016) and VGG-16 (Simonyan & Zisserman, 2014), **(Task 2)** CIFAR-100 classification on ResNet-34 and VGG-16, **(Task 3)** ImageNet (Deng et al., 2009) classification on ResNet-50, and **(Task 4)** next-word prediction task on LSTM model (Hochreiter & Schmidhuber, 1997) using WikiText-2 dataset (Merity et al., 2016). **(Task 5)** transfer learning task of classifying Caltech-101 dataset (Fei-Fei et al., 2004) using ImageNet pretrained ResNet-18.

**Suggested schemes.** We generate $\text{SEED}_{PQ}$ in three steps, (1) pruning, (2) quantization and (3) serialization. First, we use iterative magnitude pruning (IMP) with learning rate rewinding (Renda et al., 2020) to get a sparse model[2]. The sparsity of the model is set to 0.317% or 0.38%, depending on different tasks. Second, we quantize the weights of pruned network into $n = 2$ bits, by using k-means clustering on the weights at each layer, following (Han et al., 2015a). The quantized weights can be represented as $\{w_1, \cdots, w_{2^n}\}$. Third, we store the quantized weights in a key-value pair, where the key is $w_i$ and the value is the set of positions within a weight tensor having $w_i$ as the weight. Detailed settings and hyper-parameter choices are provided in Appendix.

**Comparisons.** We compare $\text{SEED}_{PQ}$ with five baselines. The first two baselines are existing methods: (i) loading SoTA compression method, i.e., the best of (Isik et al., 2021; Renda et al., 2020; Frankle & Carbin, 2018), denoted by *(SoTA compression)* and (ii) training a dense model starting from random weights, denoted by *(Scratch)*. We also test on three other baselines for the purpose of sanity-checking $\text{SEED}_{PQ}$: (iii) training a sparse model obtained by shuffling the weights of $\text{SEED}_{PQ}$ at each layer, denoted by *(Weight shuffle)*, (iv) training a sparse model obtained by re-initializing only the non-zero weights of $\text{SEED}_{PQ}$, denoted by *(Weight reinit)*, and (v) training a sparse model having random weights and masks, where the sparsity ratio of each layer matches that of $\text{SEED}_{PQ}$, denoted by *(Matching ratio)*. Note that the baselines (iii), (iv) and (v) are inspired by the existing works Frankle et al. (2020b); Su et al. (2020) for sanity-checking the pruning methods. In generating the baseline (v), we re-initialized the random mask until the generated model architecture is connected.

**Fine-tuning options.** We consider various options for fine-tuning SUPER SEEDS and baselines. (i) training all layers, (ii) training only the last linear layer, (iii) training only batchnorm layers, (iv) LP-FT (Kumar et al., 2022) which first trains the last linear layer for few epochs and then train all layers, (v) sparse training which only trains the weights that are non-zero before training. We found that the option (i) is outperforming others, thus mainly focus on the result of (i) in the main body. Please check Appendix for full results on different fine-tuning options.

**Performance metrics.** We use two metrics to evaluate the trade-off between storage and computation. First, we define the **normalized minimum epoch** as the number of epochs required for achieving the full test performance (accuracy or perplexity), normalized by the number of epochs required for vanilla training to reach the SoTA performance of given data/network. Here, 'full test accuracy' is defined as 1% below the highest accuracy of training from scratch, as in Frankle et al. (2020b). The 'full test perplexity' is defined as (the lowest perplexity achievable by training from scratch)+10. Note that it is desired to have *small* normalized minimum epoch, for reducing the computation cost. Second, we define the **compression rate** of a model as the storage gain of the compressed model compared with fully dense model. Note that having *large* compression rate is better in order to reduce the storage cost.

[2]For the results of different pruning methods we tested, please see Appendix. In the main body, we focus on the method that provides the best performance in terms of the convergence speed.

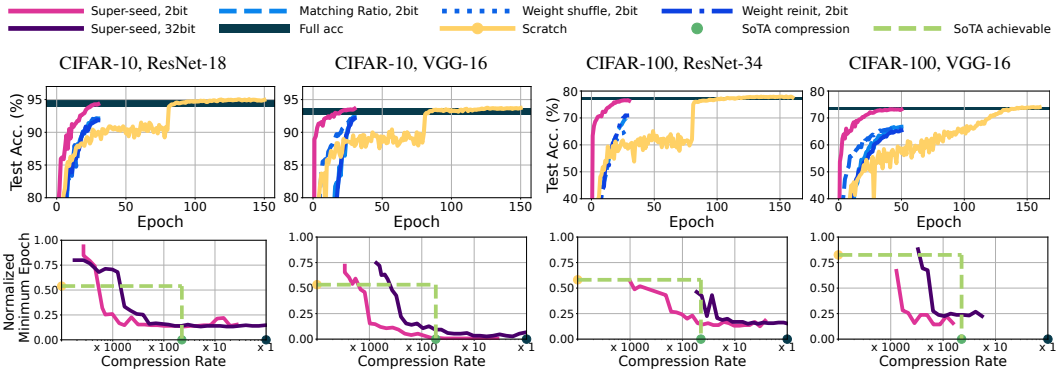


Figure 2: Performance of SEED$_{PQ}$ and baselines on CIFAR-10, CIFAR-100 classification tasks. **(Top)** Convergence curves for SUPER SEEDS (0.3% sparsity, 2-bit quantization) and baselines. Compared with training from scratch (yellow line), SEED$_{PQ}$ (pink line) enjoys 1.5× to 5× faster convergence. One can also confirm that SEED$_{PQ}$ is having faster convergence than matching ratio, weight shuffle and weight reinit curves, which implies that SEED$_{PQ}$ passes the sanity checks. **(Bottom)** Tradeoff between the model compression rate (*storage*) and the normalized minimum epochs (*computation*) required to reach the full accuracy. Here we compare the tradeoff curve of SEED$_{PQ}$ with three baselines: training from scratch (yellow dot), loading SoTA compression model (green dot) and loading SoTA dense model (black dot). We also added the boundary of achievable regime of conventional methods as light green line. Note that the left bottom corner surrounded by light green line is not achievable by existing methods. One can confirm that SEED$_{PQ}$ is crossing this boundary and opens up the possibility of trading off storage and computation.

### 4.1 Performance of SUPER SEEDS

**CIFAR-10, CIFAR-100 classification** Fig. 2 shows the performance of SEED$_{PQ}$ on CIFAR-10, CIFAR-100 classification task. The top row compares the convergence speed of SEED$_{PQ}$ and training from scratch, showing that our method gains 1.6× to 5.3× speed-up depending on the task. We also perform sanity check on our method by comparing with three baseline models: shuffle, reinit and matching ratio. SEED$_{PQ}$ outperforms the three baseline models, implying that our SUPER SEEDS has a meaningful property in the structure and weights for quickly reaching the full accuracy.

The bottom row shows the tradeoff between storage and computation achievable by SEED$_{PQ}$, and compare it with SoTA compression[3], training from scratch, and loading fully dense pretrained model. The light green box is the region left unexplored from previous works since this level of compression does not maintain the full accuracy. We observe that SEED$_{PQ}$ crosses this boundary, providing an evidence on the existence of SUPER SEEDS. In addition, comparing the purple (32 bit, i.e., no quantization) with pink (2 bit quantization), we show that quantization allows a better tradeoff between storage and computation. Compared with SoTA compression (Isik et al., 2021; Renda et al., 2020; Frankle & Carbin, 2018), SEED$_{PQ}$ trades 18.5× compression (storage) gain with 20 to 30 epochs of training (computing). Compared with training from scratch, SUPER SEEDS trades 16× to 5.3× smaller epoch with storing $\frac{1}{500}$ of the size of the model.

**ImageNet classification and language tasks** Observing a promising result on CIFAR-10 and CIFAR-100 classification tasks, we extend our experiments to ImageNet classification and language tasks. Previous results on sparsifying/compressing ImageNet models showed 10× to 50× (Han et al., 2015a; Wiedemann et al., 2020; Isik et al., 2021) compression rates without losing more than 1% accuracy. Iterative magnitude pruning based methods generally requires multiple rounds of training and pruning, where each round consists of multiple iterations (90 epochs (Renda et al., 2020)), making the sparsifying process computationally expensive.

[3]Note that while Isik et al. (2021) provides compression rate identical to our performance metric, Renda et al. (2020); Frankle & Carbin (2018) focus on sparsity levels. For the latter works, we use 1/(fraction of remaining parameters) as the compression rate, which is generally larger than the actual compression rate obtained from size of stored models.

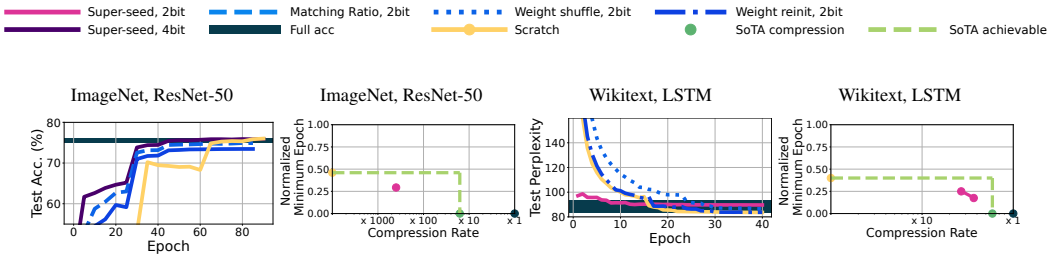


Figure 3: Performance of SEED$_{PQ}$ and baselines on ResNet-50 for ImageNet classification and LSTM for word language tasks. The behavior of SEED$_{PQ}$ is similar to what has been reported for CIFAR classification tasks in Fig. 2.

Table 1: The size of the model obtained by our SUPER SEEDS and existing model compression methods: SuRP (Isik et al., 2021) and IMP (Renda et al., 2020; Frankle & Carbin, 2018). For each scheme, we show (compression rate, normalized minimum epoch), which corresponds to (storage, computation) cost for getting full accuracy. Note that the compression rate of SUPER SEEDS is at least $5-10\times$ higher than that of existing methods, at the cost of a small fraction of computation.

| Model, Data | Original size | SUPER SEEDS | SuRP | IMP |
|---|---|---|---|---|
| ResNet-18, CIFAR-10 | 44.77MB | 140KB ($\times$**319**, $\frac{1}{3.38}$) | 3.1MB ($\times 15$, 0) | 2.25% sparse ($\times 44$, 0) |
| Small-VGG-16, CIFAR-10 | 58.98MB | 217KB ($\times$ **272**, $\frac{1}{5.33}$) | 1.1MB ($\times 54$, 0) | 2.25% sparse ($\times 44$, 0) |
| ResNet-34, CIFAR-100 | 83.45MB | 324KB ($\times$ **257**, $\frac{1}{3.32}$) | - | - |
| Small-VGG-16, CIFAR-100 | 57.78MB | 219KB ($\times$ **264**, $\frac{1}{3.67}$) | - | - |
| ResNet-50, ImageNet | 102.55MB | 368KB ($\times$ **279**, $\frac{1}{1.57}$) | 6.1MB ($\times 16$, 0) | 10.7% sparse ($\times 9.31$, 0) |

Instead, we reduce the number of epochs per round from 90 epochs to 5 epochs to speed up the process of making our SEED$_{PQ}$. The SEED$_{PQ}$ created with a short version of IMP is $272\times$ more compressed than a dense pretrained ResNet-50 model while requiring $66\%$ of epochs to reach full accuracy compared with training from scratch. We also note that other sanity check baselines (weight shuffle, weight reinit) are unable to reach full accuracy. We also ran experiments on next-word prediction task for LSTM model trained on WikiText-2 dataset. Results show that compressed SEED$_{PQ}$ can enjoy quick convergence to the full perplexity.

**Transfer Learning Task** We provide a preliminary results on applying SEED$_{PQ}$ to transfer learning tasks. We loaded ResNet-18 model pretrained on ImageNet, generated SUPER SEEDS for downstream task (Caltech101 classification), and then retrain the compressed model to reach high accuracy. The results are provided in Fig. 10 in the Appendix.

**Compression rate of SUPER SEEDS and SoTA baselines** Table 1 shows the size of SUPER SEEDS created at each image classification tasks, and compare it with original model size as well as SoTA compression methods (Isik et al., 2021; Frankle & Carbin, 2018). One can confirm that SUPER SEEDS is orders of magnuitude smaller than SoTA methods, and can be retrained to full accuracy by using dsmall fraction of computation.

### 4.2 Analysis on the fast recovery of SEED$_{PQ}$

Here we analyze the fast recoverability of SEED$_{PQ}$ in various perspectives.

**Weight distribution** Fig. 4 shows the weight distribution of VGG-16 networks before and after training on CIFAR-10 dataset. We compare our sparse SEED$_{PQ}$ with a random network with matching ratio (MR) in layer-wise sparsity level. We observe that the SEED$_{PQ}$ have a smaller change in the weight distribution before and after training then MR. Measuring the $L2$-norm of the distance between parameters of models before and after training, This provides one explanation to fast recoverability of SEED$_{PQ}$: it requires smaller change of weights to reach full accuracy, compared with MR.

**Distance of global minima** We conjecture that SEED$_{PQ}$ we created is close to the global minima, thus, making the retraining easier. We validate this conjecture by observing the difference in the

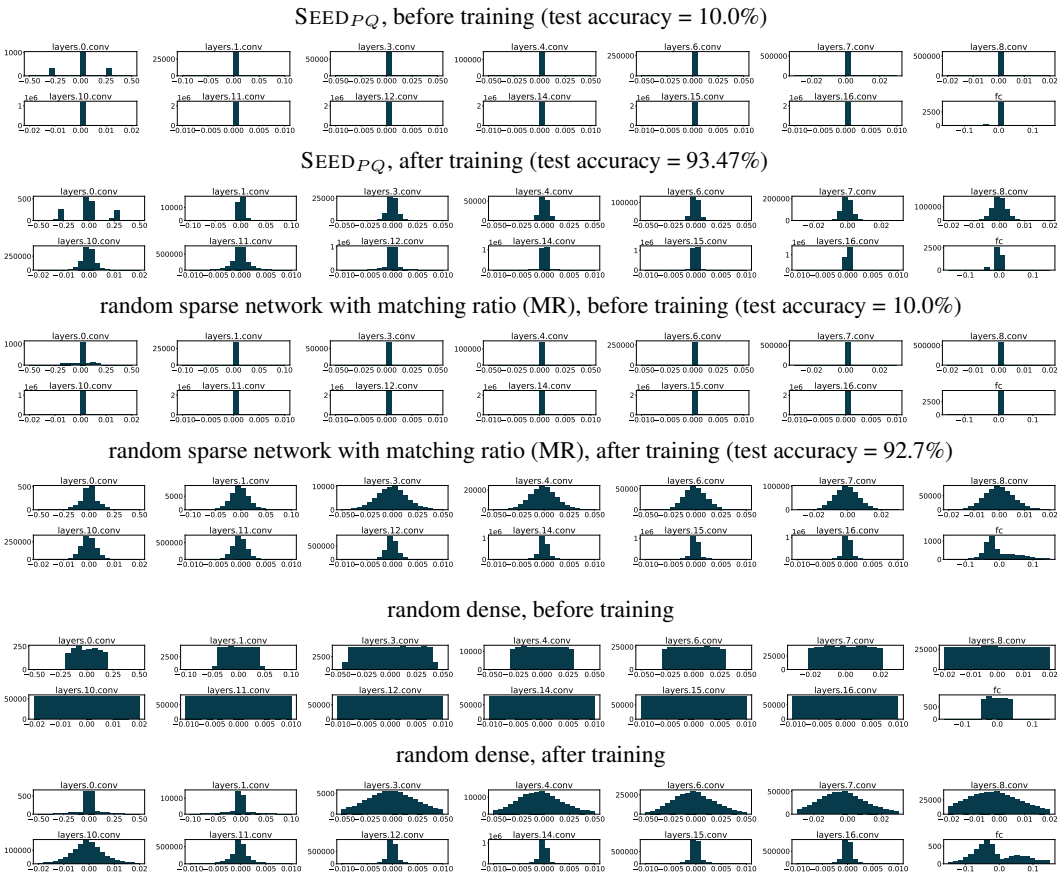


Figure 4: Weight distribution of VGG-16 networks before/after training on CIFAR-10. (a), (b): SEED$_{PQ}$ with 0.3% sparsity, (c),(d): 0.3% sparse random network with matching ratio (MR). As in (a) and (c), both models are sparse before training, due to the design of SEED$_{PQ}$. As shown in (b) and (d), the weights of SEED$_{PQ}$ have a smaller change in the distribution during training, compared with the weights of MR. Recall that in Fig. 2b, SEED$_{PQ}$ quickly reaches the full accuracy within 30 epochs, while MR slowly reaches to high accuracy. This weight distribution comparison explains why SEED$_{PQ}$ can enjoy quick recoverability: SEED$_{PQ}$ needs less effort in adjusting the weights to get full accuracy.

global minima reached by following different retraining trajectories from three initial points: (a) SEED$_{PQ}$, (b) random dense model, and (c) Matching Ratio (MR). From each of the three each initial points, we shuffle the sequence of training data batches to create five distinct training trajectories and train the models to full convergence. Cosine similarity is used to measure the difference in the parameters of the converged models. Results in Fig. 5 demonstrate that the global optima achieved by SEED$_{PQ}$ are close in distance, while training from other initial points (random dense, MR) lead to different global optima, distant from each other. This aligns with our conjecture that SEED$_{PQ}$ is close to the optimum that is highly compressed and can achieve fast recovery.

**Linear mode connectivity** Extreme quantization, in the process of making the SEED$_{PQ}$, perturb the sparsified model in a way that the model is only as good as random guessing. We analyze the effect of quantization on SEED$_{PQ}$ in the loss landscape by observing the linear mode connectivity (Frankle et al., 2020a) of the models trained from SEED$_{PQ}$ (1) before and (2) after quantization. We linearly interpolate the two models by sampling 30 evenly spaced points between the two and measure the test error on each of these points. Fig. 6 demonstrates that the two models are linearly connected. This suggests that while quantization initially perturbs the starting point of the model before training, it does not move the model too much.

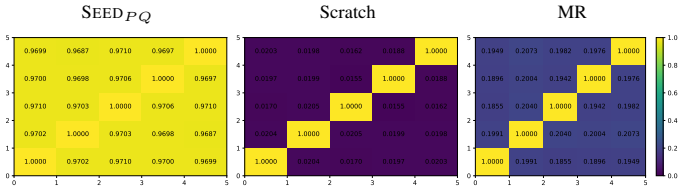


Figure 5: Cosine similarity of five different models converged from the three initial points: (a) SEED$_{PQ}$, (b) random dense model, and (c) Matching Ratio (MR). We train VGG-16 on CIFAR-10 from each of the three initial points, with five different learning trajectories and measure the cosine similarity of the parameters of the converged models. Each models are trained to full convergence for 150 epochs. Results show that while training from scratch or MR leads to local minima that is distant from each other, SEED$_{PQ}$ converges to close local minima. This agrees with our conjecture that SEED$_{PQ}$ provides a compressible initialization point close to the optimum, where training is fast.

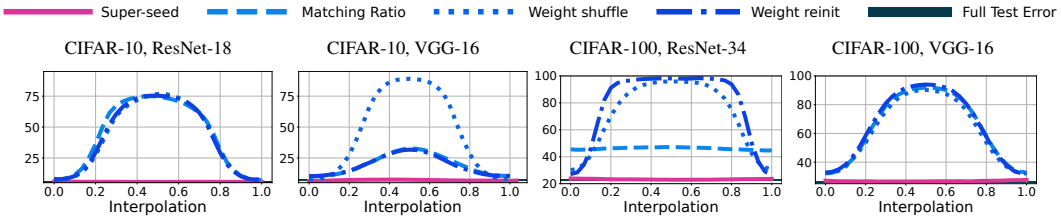


Figure 6: Linear mode connectivity between (1) model trained from SEED$_{PQ}$ with quantization and (2) model trained from SEED$_{PQ}$ without quantization. The proportion of remaining weights of SEED$_{PQ}$ on ResNet-18, VGG-16s and ResNet-34 are 0.32%, 0.32%, 0.38%, respectively, while all four models are quantized to 2bits. Following the methodology from (Frankle et al., 2020a), we linearly interpolate the two models with ratio from 30 evenly spaced points between 0 and 1, and plot the test error of the interpolated model. Results show that model optimized from SEED$_{PQ}$ with quantization are linearly connected with models optimized from SEED$_{PQ}$ without quantization. This implies that the quantization process does not move the SEED$_{PQ}$ too much.

## 5 Discussion

As is evident from our experiments (cf. Figure 2), SUPER SEEDS can be retrained to full accuracy in a few number of epochs, as compared to training a random model from scratch. Further the distance between the retrained SEED$_{PQ}$ model and the SEED$_{PQ}$ model is much smaller than the distance between model trained from scratch and its original, random initialization (cf. Figure 4). Thus the phenomenon of SEED$_{PQ}$ opens up interesting research questions regarding the loss landscape of neural networks, which we discuss in the subsections below.

**Existence of SUPER SEEDS** Our experimental results show the existence of SUPER SEEDS for different models on various tasks. However, we see that the amount of compression we can achieve varies, depending on the model and task at hand. For example, we see a much better computation-compression trade-off for VGG-16 on CIFAR-10, than MobileNet-V2 We think that for very overparameterized models, SUPER SEEDS can achieve a good computation-compression trade-off, whereas difficult tasks like ImageNet might not have very compressible SUPER SEEDS.

**Generalization error** We think that models trained from SUPER SEEDS should enjoy good theoretical bounds on their generalization error. This is because SUPER SEEDS enjoy two favourable properties, both of which have been used in the literature to prove generalization bounds. The first is the small bit complexity of SUPER SEEDS. Thus SUPER SEEDS themselves represent a very small hypothesis class, which indicatesthat VC-dimension style arguments can be used to prove generalization bounds. Further, retraining them only needs a few epochs, and we also know that the model reached after retraining is not too far from the SUPER SEEDS (cf. Figure 4). Hence, 'train faster, generalize better' style techniques (Hardt et al., 2016) that use algorithmic-stability (Bousquet & Elisseeff, 2000) to bound generalization error are relevant. Although, these two techniques cannot

directly be applied, but we believe a combination of the two might give better generalization bounds than the ones we have for models trained from scratch.

**Computational complexity of generating SUPER SEEDS versus Lottery Tickets** Vanilla iterative magnitude pruning (IMP) methods Frankle & Carbin (2018); Renda et al. (2020) used for creating Lottery Tickets typically require large number epochs to generate sparse networks. On the other hand, our ImageNet experiments use a shorter version of IMP, requiring only 5 epochs per pruning round to generate SUPER SEEDS for ImageNet classification task. These SUPER SEEDS still perform well in the sense that they can be retrained to full accuracy quickly. We believe that this relative ease of generating SUPER SEEDS as compared to Lottery Tickets is due to the fact that a) Lottery Tickets are constrained to be a sparse subnetwork of the network at initialization, whereas SUPER SEEDS do not have any such restrains, and b) Lottery Tickets are designed for sparse retraining whereas SUPER SEEDS are designed for dense retraining.

**Density of global minima.** We also observe that the global minima found by retraining SUPER SEEDS are close to the SUPER SEEDS themselves (cf. Appendix). Hence, there exist global mimima which have highly compressible models within a small neighbourhood of them. Since given a serialization / deserialization scheme, there can only be a few models in the parameter space with small description length (that is, a few SUPER SEEDS), and yet we are able to show that at least one global minima lies close to one such SUPER SEEDS. Since the serialization / deserialization scheme itself is independent of the data and model, and in particular, independent of the distribution of global minima, we get that the number of global minima has to be significantly large for one such global minima to lie close to one of the SUPER SEEDS. This can give some insight into the number of global minima in the loss landscape of deep neural networks.

## 6 Conclusion and Future Work

In this paper, we focused on the extreme regime of model compression and explored the achievable storage-computation tradeoff in the proposed compression-decompression framework. We empirically showed the existence of SUPER SEEDS, an extremely compressed model that can be quickly recovered to full accuracy, for various benchmark image classification tasks.

### 6.1 Limitations and Future Work

While we mainly focus on the fast recoverability of SUPER SEEDS in terms of test accuracy, it is unclear whether other aspects such as fairness and robustness are preserved with extreme compression.

Currently, we mostly tested the efficacy of SUPER SEEDS generated by iterative pruning methods. Exploring the computation-storage tradeoff achievable by pruning at initialization methods (Lee et al., 2018; Wang et al., 2020; Tanaka et al., 2020; Sreenivasan et al., 2022) and post-training pruning methods (Lee et al., 2020; Park et al., 2020; Isik et al., 2021) remain as interesting future directions.

## References


Bousquet, O. and Elisseeff, A. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems*, 13, 2000.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., and Joulin, A. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

Fei-Fei, L., Fergus, R., and Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178. IEEE, 2004.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020a.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020b.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gao, W., Liu, Y.-H., Wang, C., and Oh, S. Rate distortion for model compression: From theory to practice. In *International Conference on Machine Learning*, pp. 2102–2111. PMLR, 2019.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

Hardt, M., Recht, B., and Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, pp. 1225–1234. PMLR, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Isik, B., No, A., and Weissman, T. Rate-distortion theoretic model compression: Successive refinement for pruning. *arXiv preprint arXiv:2102.08329*, 2021.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Khodak, M., Tenenholtz, N. A., Mackey, L., and Fusi, N. Initialization and regularization of factorized neural layers. In *International Conference on Learning Representations*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kumar, A., Raghunathan, A., Jones, R., Ma, T., and Liang, P. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*, 2022.

Lai, C.-I. J., Zhang, Y., Liu, A. H., Chang, S., Liao, Y.-L., Chuang, Y.-S., Qian, K., Khurana, S., Cox, D., and Glass, J. Parp: Prune, adjust and re-prune for self-supervised speech recognition. *Advances in Neural Information Processing Systems*, 34, 2021.

Lee, J., Park, S., Mo, S., Ahn, S., and Shin, J. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Park, S., Lee, J., Mo, S., and Shin, J. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

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

Smith, L. N. and Topin, N. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pp. 1100612. International Society for Optics and Photonics, 2019.

Sreenivasan, K., Sohn, J.-y., Yang, L., Grinde, M., Nagle, A., Wang, H., Lee, K., and Papailiopoulos, D. Rare gems: Finding lottery tickets at initialization. *arXiv preprint arXiv:2202.12002*, 2022.

Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. Sanity-checking pruning methods: Random tickets can win the jackpot. *arXiv preprint arXiv:2009.11094*, 2020.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Wang, H., Agarwal, S., and Papailiopoulos, D. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Wiedemann, S., Kirchhoffer, H., Matlage, S., Haase, P., Marban, A., Marinč, T., Neumann, D., Nguyen, T., Schwarz, H., Wiegand, T., et al. Deepcabac: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):700–714, 2020.

# A Experimental Setup

In this section, we summarize the datasets and models used in our experiments as well as hyper-parameter choices of creating and retraining the $\text{SEED}_{PQ}$. All of our experiments are run using PyTorch 1.9.1 on Nvidia 2080 TIs, Nvidia 3090s, and Nvidia V100s. Detailed dependencies are provided on our anonymized github repository[4].

## A.1 Dataset

In this section we provide details on the dataset used in our experiments.

**CIFAR-10.** CIFAR-10 dataset consists of 60,000 images of 10 classes, with size $32 \times 32$. 60,000 images are split into 50,000 images for train dataset and 10,000 images for test dataset. We further randomly split the training images by size 45,000 for train dataset ad 5,000 for validation dataset. We follow the standard augmentation of channel-wise normalization, horizontal flipping and random cropping.

**CIFAR-100.** CIFAR-100 dataset consists of 60,000 images of 100 classes (600 images per class), which are split into 50,000 images for train dataset and 10,000 images for test dataset. Identical to CIFAR-10, we randomly split the train images to 45,000 and 5,000 for train and validation dataset and use follow the standard augmentation: channel-wise normalization, horizontal flipping and random cropping.

**Caltech-101.** Caltech-101 dataset contains images of 101 different categories, each with around 40 800 images and size 200 300 pixels. We resize each images to $224 \times 224$ and use channel-wise normalization. Train, validation, test dataset are split by 60%, 20%, 20%.

**WikiText-2.** WikiText-2 consists of 33,278 vocabulary and 2.6% out of vocabulary (OoV) tokens. Train, validation, test dataset are split by 600, 60, 60 articles with 2,088,628, 217,646, and 245,569 tokens respectively.

## A.2 Model

Implementation of models for image classification are based on the Github repository[5] for CIFAR-10 and CIFAR-100 dataset models, and Github repository[6] for ImageNet dataset models. LSTM model for language tasks are based on the Github repository[7]. Note for ResNet models, we do not use any biases for the convolutional layers, however, VGG-16 models do have biases for convolutional layers. Output layer (fully connected layer) for all models have biases. As in (Frankle & Carbin, 2018), pruning operations are only performed on weights on linear and convolutional layers and are not performed on batchnorm layers or bias terms. We use PyTorch default initialization for all layers.

**ResNet.** Detailed ResNet architectures used in our experiments are given in Table 2. For CIFAR-10 and CIFAR-100 image classification tasks, we have the first convolution layer to be a kernel of size $3 \times 3$, while for ImageNet classification, we follow the standard ResNet architecture.

**VGG-16.** Detailed architecture is given in Table 3. Note that the original VGG-16 (Simonyan & Zisserman, 2014) has 13 convolution layers and 3 FC layers. Instead of this original version, we follow the architecture used in Frankle et al. (2020b,a), removing the first two FC layers while keeping the last linear classification layer. This finally leads to a 14-layer architecture, but we still call it VGG-16 as it is modified from the original VGG-16 architectural design. We base our implementation on the GitHub repository[8].

[4] https://anonymous.4open.science/r/SuperSeed-4834

[5] https://github.com/kuangliu/pytorch-cifar

[6] https://github.com/facebookresearch/open_lth

[7] https://github.com/pytorch/examples/tree/main/word_language_model

[8] `https://github.com/kuangliu/pytorch-cifar/blob/master/models/vgg.py`

Table 2: ResNet models used in our experiments. ResNet-18, ResNet-34 model, that are respectively used for CIFAR-10, CIFAR-100 classification has the first convolutional layer with kernel size 3. The shape of the output layer (fully connected layer) is changed according to the number of classes.

| **Layer** | **ResNet-18** | **ResNet-34** | **ResNet-50** |
|---|---|---|---|
| Conv 1 | 3×3, 64<br>padding 1<br>stride 1 | 3×3, 64<br>padding 1<br>stride 1 | 7×7, 64<br>padding 3<br>stride 2 |
| Layer stack 1 | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| Layer stack 2 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ |
| Layer stack 3 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ |
| Layer stack 4 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| FC | Avg Pool, kernel 4<br>512 × NUM_CLASSES | Avg Pool, kernel 4<br>512 × NUM_CLASSES | Avg Pool, kernel 2<br>2048 × NUM_CLASSES |

**LSTM.** Detailed architecture is given in Table 4. The network has encoder/decoder at the first/last layer, and two LSTM blocks in the intermediate layers. The embedding size is set to 1500, and the number of hidden neurons at each LSTM block is set to 1500. We base our implementation on the GitHub repository [9].

Table 4: Detailed information on LSTM architecture in our experiment.

| **Parameter** | Shape | Hyper-param. |
|---|---|---|
| **encoder.weight** | $33278 \times 1500$ | N/A |
| **dropout** | N/A | $p = 0.65$ |
| **lstm0.weight.ii/f/g/o** | $1500 \times 1500$ | N/A |
| **lstm0.weight.hi/f/g/o** | $1500 \times 1500$ | N/A |
| **dropout** | N/A | $p = 0.65$ |
| **lstm1.weight.ii/f/g/o** | $1500 \times 1500$ | N/A |
| **lstm1.weight.hi/f/g/o** | $1500 \times 1500$ | N/A |
| **decoder.weight**(shared) | $1500 \times 33278$ | N/A |

### A.3 Hyper-Parameter Choices

In this section, we provide the hyper-parameter choices for creating/retraining the SEED$_{PQ}$.

#### A.3.1 Hyperparameters for creating SEED$_{PQ}$

We mostly follow the iterative magnitude pruning Renda et al. (2020) schedule for creating the SEED$_{PQ}$, except for ImageNet experiments. For ImageNet experiments, we use a shorter pruning period (5 epochs instead of 160 epochs) for each pruning rounds. For all cases, we use multi-step

[9] `https://github.com/pytorch/examples/tree/main/word_language_model`

Table 3: VGG-16 models used in our experiments. We use Small-VGG-16, a modified version of VGG-16 used in where the 13 convolutional layers are followed by 1 fully connected layer instead of 3. Affine batchnormalization is followed by a ReLU activation after each convolutional layer. Shape for convolution layers follows $(c_{in}, c_{out}, k, k)$.

| **Parameter** | Shape | Layer hyper-parameter |
|---|---|---|
| **layer1.conv1.weight** | $3 \times 64 \times 3 \times 3$ | stride:1;padding:1 |
| **layer2.conv2.weight** | $64 \times 64 \times 3 \times 3$ | stride:1;padding:1 |
| **pooling.max** | N/A | kernel size:2;stride:2 |
| **layer3.conv3.weight** | $64 \times 128 \times 3 \times 3$ | stride:1;padding:1 |
| **layer4.conv4.weight** | $128 \times 128 \times 3 \times 3$ | stride:1;padding:1 |
| **pooling.max** | N/A | kernel size:2;stride:2 |
| **layer5.conv5.weight** | $128 \times 256 \times 3 \times 3$ | stride:1;padding:1 |
| **layer6.conv6.weight** | $256 \times 256 \times 3 \times 3$ | stride:1;padding:1 |
| **layer7.conv7.weight** | $256 \times 256 \times 3 \times 3$ | stride:1;padding:1 |
| **pooling.max** | N/A | kernel size:2;stride:2 |
| **layer8.conv9.weight** | $256 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **layer9.conv10.weight** | $512 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **layer10.conv11.weight** | $512 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **pooling.max** | N/A | kernel size:2;stride:2 |
| **layer11.conv11.weight** | $512 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **layer12.conv12.weight** | $512 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **layer13.conv13.weight** | $512 \times 512 \times 3 \times 3$ | stride:1;padding:1 |
| **pooling.max** | N/A | kernel size:2;stride:2 |
| **pooling.avg** | N/A | kernel size:1;stride:1 |
| **fc.weight** | $512 \times 10$ | |

decay with decay rate of size 0.1 on fixed milestone epochs. Detailed hyper-parameter configurations are given in Table 5. The overall training curve for each model and data pair is shown in Fig. 7.

Table 5: Hyper-parameters used for creating the SEED$_{PQ}$. We use the Github repository of OpenLTH to perform the iterative magnitude pruning process.

| Model | Dataset | Pruning Rounds | Epochs | Batch Size | LR | Multi-Step Milestone | Gamma | Pruning Rate |
|---|---|---|---|---|---|---|---|---|
| ResNet-18 | CIFAR-10 | 20 | 160 | 512 | 0.1 | [80, 120] | 10 | 0.25 |
| VGG-16 | CIFAR-10 | 20 | 20 | 256 | 0.1 | [80, 120] | 10 | 0.25 |
| ResNet34 | CIFAR-100 | 25 | 200 | 256 | 0.1 | [60, 120, 160] | 10 | 0.2 |
| VGG-16 | CIFAR-100 | 20 | 160 | 256 | 0.1 | [80, 120] | 10 | 0.25 |
| ResNet-50 | ImageNet | 20 | 5 | 256 | 0.1 | [2,3,4] | 10 | 0.25 |
| LSTM | WikiText-2 | 10 | 40 | 20 | 20 | when validation loss is increasing | 4 | 0.1 |

### A.3.2 Hyperparameters for retraining from SEED$_{PQ}$ and baseline methods

For each method considered, we used three learning rates (0.1, 0.01, 0.001) and two different learning rate schedule (multi-step decay, cosine annealing) and measured the validation accuracy on each. We chose the combination that gave the highest final validation accuracy. Detailed hyper-paramter configurations are given in Table 6.

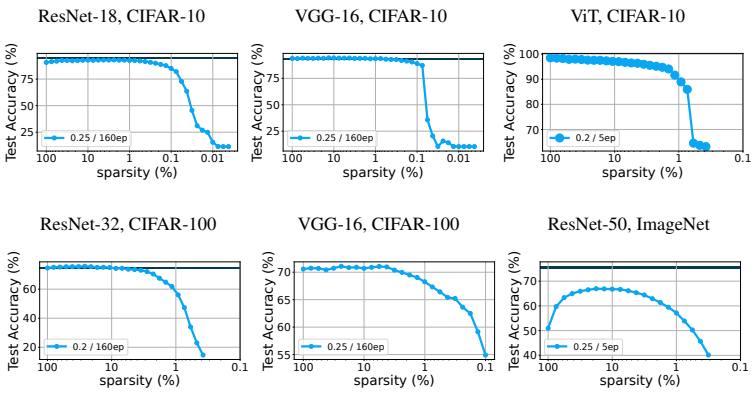


Figure 7: Generation of SEED$_{PQ}$ using iterative pruning algorithm. Each legend represents (pruning rate / number of epochs). We measure the test accuracy after retraining from after each pruning round. Note that for ResNet-50, ImageNet, each pruning round is 5 epochs (instead of 90 epochs used for full training), thus the curve is not reaching the full accuracy even in the 100% sparsity regime.

Table 6: Hyper-parameters used for retraining the SEED$_{PQ}$. We use cosine annealing learning rate schedule for all model, dataset pairs as it resulted in largest highest accuracy for all cases.

| Model | Dataset | Sparsity | Epochs | Batch Size | LR |
|---|---|---|---|---|---|
| ResNet-18 | CIFAR-10 | 0.32% | 30 | 64 | 0.01 |
| VGG-16 | CIFAR-10 | 0.32% | 30 | 64 | 0.001 |
| ResNet34 | CIFAR-100 | 0.38% | 30 | 256 | 0.01 |
| VGG-16 | CIFAR-100 | 0.32% | 30 | 256 | 0.01 |
| ResNet-50 | ImageNet | 0.32% | 90 | 256 | 0.1 |
| LSTM | WikiText-2 | 38% | 40 | 20 | 20 |

### A.3.3 Hyperparameters for reinitializing zero weights during retraining

We note that extremely sparse networks often have disconnected parts of the network where there are no gradient information to update the weights during the retraining phase. We overcome this by randomly reinitializing a small fraction of zero components of the network. To be more specific, for every 5 epochs, we reinitialize 20% of the pruned elements to a value randomly sampled from Kaiming normal distribution scaled by 0.01.

## B Additional Experimental Results

### B.1 Weight distribution comparison

We observe the weight distribution of our sparse and quantized SUPER SEEDS and a randomly initialized network with layer-wise matching ratio (MR) before and after training. The histograms shown in Fig. 4 shows that change in the weight distribution of SUPER SEEDS is smaller than MR. This can also be seen in Table 7 which shows the distance between the initial and final model after training process.

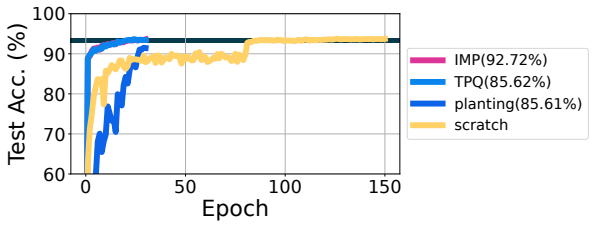


Figure 8: Performance of SUPER SEEDS, generated with different sparsification methods on CIFAR-10, VGG-16. We consider (1) iterative magnitude pruning (IMP) with learning rate rewinding (Renda et al., 2020), iterative train-prune-quantization (TPQ) method, and (3) planting small, fully dense model to a large sparse network. We use IMP with learning rate reweinding method to create all SEED$_{PQ}$ since it outperforms other methods.

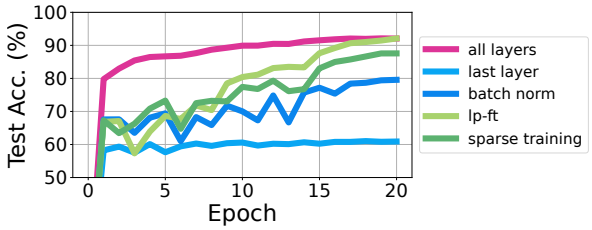


Figure 9: Performance of different retrain algorithms. We compare different fine-tuning options described in Section 4. All layers perform dense training in the entire network while last layer and batch norm trains only the linear classifier and batch norm layer, respectively. LP-FT (Kumar et al., 2022) first trains the linear classifier for few epochs and then trains the entire model. Sparse training method only trains the non-zero elements and does not update the pruned weights.

| | RANDOM DENSE | MATCHING RATIO | SUPER SEEDS |
|---|---|---|---|
| **Epoch 0**, $\|\boldsymbol{w}_0\|$ | 75.0742 | 65.1691 | 78.5013 |
| **Epoch 150**, $\|\boldsymbol{w}_{150}\|$ | 29.6695 | 50.5017 | 74.3063 |
| **Distance**, $\|\boldsymbol{w}_{150} - \boldsymbol{w}_0\|$ | 74.8466 | 21.2913 | **8.1473** |

Table 7: We see that the model does not move very far for SUPER SEEDS, as compared to the baselines of training a dense network from scratch and training a network initialized with Matching Ratio.

### B.2 Comparison with different sparsification method

While we emphasize again that the novelty of our paper is in the exploration of extremely compressed regime, we show the performance of our SUPER SEEDS created using different sparsification methods. These include (1) iterative magnitude pruning (IMP) with learning rate rewinding (Renda et al., 2020) where the learning rate is reset to default learning rate schedule after each pruning round, (2) iterative train-prune-quantization (TPQ) method where quantization is performed after pruning in each pruning round, and (3) planting a small, fully dense, network planted inside a larger, but sparser network. Based on the results shown in Fig. 8, we focus on (1) IMP with learning rate rewinding to create SUPER SEEDS in our experiments.

### B.3 Comparison with different retraining methods

We provide experimental results in Fig. 9 on different methods to retrain our SUPER SEEDS mentioned in Section 4. Based on the result that training all layers outperforms all the other methods, we decided to restrict the scope of our experiments to training all layers.

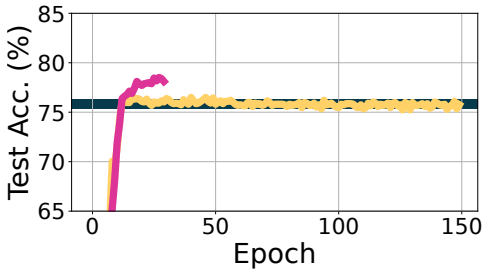


Figure 10: Downstream task-specific transfer learning with SEED$_{PQ}$ on Caltech101 dataset with ResNet-18. The yellow curve corresponds to training a dense model from scratch and the pink curve corresponds to retraining Super Seed.

### B.4 Transfer learning task

We performed Transfer Learning for a SUPER SEEDS created for Caltech101 dataset. For this, we took a ResNet18 model pretrained on Imagenet and created SUPER SEEDS for it using the Caltech101 dataset. We see in Figure 10 that this SUPER SEEDS trains to a higher test accuracy than simply training a model on Caltech101 from scratch.