SPARSEGPT: MASSIVE LANGUAGE MODELS CAN BE ACCURATELY PRUNED IN ONE-SHOT

PREPRINT. VERSION JANUARY 3, 2023.

Elias Frantar
IST Austria
elias.frantar@ist.ac.at

Dan Alistarh
IST Austria & Neural Magic
dan.alistarh@ist.ac.at

arXiv:2301.00774v1 [cs.LG] 2 Jan 2023

ABSTRACT

We show for the first time that large-scale generative pretrained transformer (GPT) family models can be pruned to at least 50% sparsity in *one-shot, without any retraining*, at minimal loss of accuracy. This is achieved via a new pruning method called `SparseGPT`, specifically designed to work efficiently and accurately on massive GPT-family models. When executing `SparseGPT` on the largest available open-source models, OPT-175B and BLOOM-176B, we can reach 60% sparsity with negligible increase in perplexity: remarkably, more than 100 billion weights from these models can be ignored at inference time. `SparseGPT` generalizes to semi-structured (2:4 and 4:8) patterns, and is compatible with weight quantization approaches.

1 Introduction

Large Language Models (LLMs) from the Generative Pretrained Transformer (GPT) family have shown remarkable performance on a wide range of tasks, but are difficult to deploy because of their massive size and computational costs. For instance, the top-performing GPT-175B model has 175 billion parameters, which total at least 320GB (counting multiples of 1024) of storage in half-precision (FP16) format, leading it to require at least five A100 GPUs with 80GB of memory each for inference. It is therefore natural that there has been significant interest in reducing these costs via *model compression*. To date, virtually all existing GPT compression approaches have focused on *quantization* [3, 47, 46, 9], that is, reducing the precision of the numerical representation of individual weights.

A complementary approach for model compression is *pruning*, which removes network elements, from individual weights (unstructured pruning) to higher-granularity components such as entire rows/columns of the weight matrices (structured pruning). This approach has a long history [28, 19], and has been applied successfully in the case of vision and smaller-scale language models and tasks [21]. Yet, the best-performing pruning methods require *extensive retraining* of the model to recover from the accuracy loss due to removed elements, which is extremely expensive in the case of GPT-scale models. While some *one-shot* pruning methods also exist [22, 11], which compress the model without retraining, they are unfortunately too computationally-expensive to be applied to models with billions of parameters. Thus, to date, there is virtually no work on accurate pruning of GPT3-scale models.

Overview. In this paper, we propose `SparseGPT`, the first accurate one-shot pruning method which works efficiently at the scale of models with 10-100 billion parameters. `SparseGPT` works by reducing the pruning problem to an extremely large-scale instance of *sparse regression*. It is based on a new approximate sparse regression solver, used to solve a layer-wise compression problem, which is efficient enough to execute in a few hours on the largest openly-available GPT models (175B parameters), using a single GPU. At the same time, `SparseGPT` is accurate enough to drop negligible accuracy post-pruning, without any fine-tuning. For example, when executed on the largest publicly-available generative language models (OPT-175B and BLOOM-176B), `SparseGPT` induces 50-60% sparsity in one-shot, with minor accuracy loss, measured either in terms of perplexity or zero-shot accuracy.

Our experimental results, for which we provide a snapshot in Figure 1, illustrate the following two key points. First, as shown in Figure 1 (left), `SparseGPT` can induce uniform layer-wise sparsity of up to 60% in e.g. the 175-billion-parameter variant of the OPT family [48], with minor accuracy loss. By contrast, the only known one-shot baseline which works at this scale, Magnitude Pruning [16, 18], preserves accuracy only until 10% sparsity, and completely collapses beyond 30% sparsity. Second, as shown in Figure 1 (right), SparseGPT can also accurately impose sparsity in the more stringent, but hardware-friendly, 2:4 and 4:8 semi-structured sparsity patterns [32]. Although these patterns tend to lose additional accuracy relative to the dense baseline, especially for the smaller models, these sparsity patterns can be directly exploited to obtain computational speedups. Moreover, as we show later in the paper, the sparsity induced by our technique compounds well with additional compression obtained through quantization.

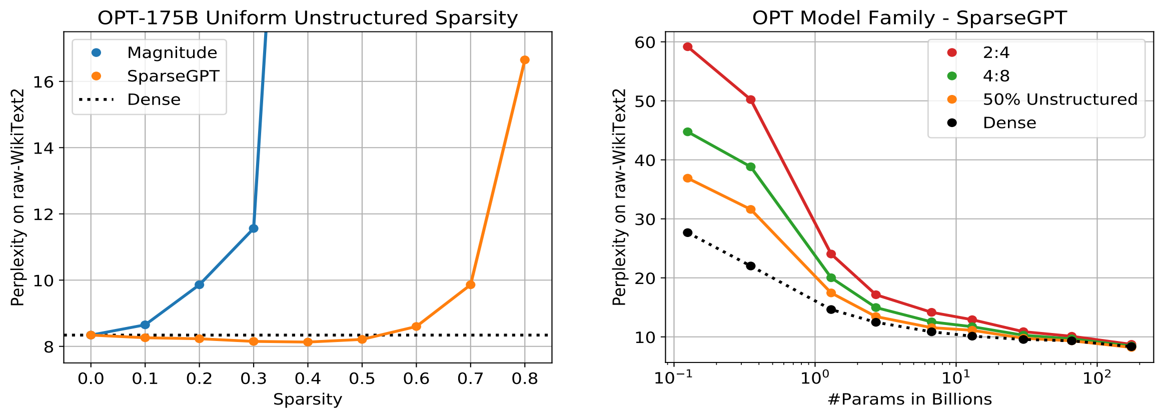

Figure 1: Left: Comparison of SparseGPT against magnitude pruning on OPT-175B. Right: Compressing the entire OPT model family (135M, 350M, . . . , 66B, 175B) to different sparsity patterns using SparseGPT.

One interesting fact is that our method is *entirely local*, in the sense that it relies solely on weight updates designed to preserve the input-output relationship for each layer, which are computed without any global gradient information. As such, it is remarkable that one can directly identify such sparse models in the "neighborhood" of dense pretrained models, whose output correlates extremely closely with that of the dense model. Another general finding, illustrated in Figure 1 (right), is that *larger models are easier to sparsify*: specifically, we found that, for a fixed sparsity level, the relative accuracy gap between the dense and sparse model variant narrows as we increase the model size, to the point where inducing 50% sparsity results in practically no accuracy decrease on the largest models. This observation, illustrated in full detail in the experimental section, should be seen as very encouraging for future work on compressing such massive models.

2 Background

Post-Training Pruning is a practical scenario where we are given a well-optimized model $\theta^\star$, together with some calibration data, and must obtain a compressed version of $\theta^\star$ which satisfies some compression predicate $\mathcal{C}$, specifying a set of weight quantization or sparsity constraints. Post-training compression has traditionally been investigated in the context of quantization [23, 34, 29] to reduce the computational cost of quantization-aware training. More recently, it has been shown that it is possible to also perform accurate post-training *pruning* [22, 11, 27], although existing work focuses on classic CNN and Transformer models, which have less than 100 million parameters.

Layer-Wise Pruning. Post-training compression usually works by splitting the full-model compression problem into *layer-wise* subproblems, whose solution quality is measured in terms of the ℓ_2-error between the output of the uncompressed layer, and that of the compressed one. Specifically, for each layer $\mathbf{W}_\ell$ with calibration input $\mathbf{X}_\ell$, the objective is to solve the following constrained optimization problem:

$$\text{argmin}_{\widehat{\mathbf{W}}_\ell} \, ||\mathbf{W}_\ell \mathbf{X}_\ell - \widehat{\mathbf{W}}_\ell \mathbf{X}_\ell||_2^2, \tag{1}$$

where $\widehat{\mathbf{W}}_\ell$ is a set of weights satisfying the compression constraint $\mathcal{C}$. Specifically for pruning, Hubara et al. [22] posed this problem as that of finding, for each layer ℓ, a sparsity mask[1] $\mathbf{M}$ satisfying the constraint, and weights $\widehat{\mathbf{W}}_\ell$ such that

$$\text{argmin}_{\text{mask } \mathbf{M}, \widehat{\mathbf{W}}_\ell} \, ||\mathbf{W}_\ell \mathbf{X}_\ell - (\mathbf{M} \odot \widehat{\mathbf{W}}_\ell) \mathbf{X}_\ell||_2^2, \tag{2}$$

where $\widehat{\mathbf{W}}_\ell$ is a possibly-updated version of the original dense weights $\mathbf{W}_\ell$. Once each one of these layer-wise subproblems is solved, the model can then be "stitched back together" by re-composing the compressed layers.

[1] Throughout the paper, by *sparsity mask* for a given tensor we mean a binary tensor of the same dimensions, with 0 at the indices of the sparsified entries, and 1 at the other indices.

Intuitively, if the layer-wise errors are small enough, the resulting model should preserve the accuracy of the original dense model.

Mask Selection & Weight Reconstruction. A key aspect of the the layer-wise pruning problem in (2) is that both the mask $\mathbf{M}$ as well as the remaining weights $\widehat{\mathbf{W}}_\ell$ are optimized *jointly*, which makes this problem NP-hard [1]. Thus, exactly solving it for larger layers is unrealistic, leading all existing approaches to resort to approximations.

A particularly popular approach is to separate the problem into *mask selection* and *weight reconstruction* [20, 27, 22]. Concretely, this means to first choose a pruning mask $\mathbf{M}$ according to some saliency criterion, like the weight magnitude [50], and then optimize the remaining unpruned weights while keeping the mask unchanged. Importantly, once the mask is fixed, (2) turns into a linear squared error problem, which is convex and thus easily optimized. It can even be solved in closed form by applying the standard linear regression formula to each matrix row $\mathbf{w^i}$:

$$\mathbf{w^i}_{\mathbf{M_i}} = (\mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^\top)^{-1}\mathbf{X}_{\mathbf{M_i}}(\mathbf{w}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}})^\top, \tag{3}$$

where $\mathbf{X}_{\mathbf{M_i}}$ denotes only the subset of input features whose corresponding weights have not been pruned in row i, and $\mathbf{w}_{\mathbf{M_i}}$ represents the respective weights. It is also worth noting that $\mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^\top$ is the problem's Hessian matrix, which needs to be inverted.

The AdaPrune approach [22] has shown good results for this problem in the context of post-training pruning via magnitude-based weight selection, followed by applying SGD steps to reconstruct the remaining weights. Follow-up works demonstrate that pruning accuracy can be further improved by removing the strict separation between mask selection and weight reconstruction. Iterative AdaPrune [8] performs pruning in gradual steps with reoptimization in between and OBC [11] introduces a greedy solver which removes weights one-at-a-time, fully reconstructing the remaining weights after each iteration, via efficient closed-form equations. Yet, these improvements also come with increased runtime, which, as we will discuss next, is particularly problematic in the context of extremely large models.

Difficulty of Scaling to 100+ Billion Parameters. Prior post-training techniques have been successfully applied to models up to a few hundred million parameters [22, 11, 27], on which they are able to produce good results within a few hours of computation. However, our goal in this paper is to accurately sparsify models up to $1000\times$ larger.

The currently most accurate post-training method, OBC [11], exhibits runtime scaling to the 4th power of a transformer's hidden dimension, while taking > 1 hour to compress a 100M parameter model. Thus, this is unsuitable for 100B-parameter models. Even the fastest known accurate post-training method, AdaPrune [22], requires a few minutes to prune a 100M model. Assuming best-case linear runtime scaling, this extrapolates to several hundreds of hours (a few weeks) of computation for a GPT3-sized Transformer.

Despite extremely large models being a highly active research area for the past several years, to the best of our knowledge, so far no model with 10+ billion parameters has been accurately pruned to nontrivial amounts of sparsity. This suggests that scaling up existing methods might actually lead to significantly higher runtime costs than optimistically estimated in the previous paragraph[2] and/or bring other major unexpected challenges like overfitting to calibration data. In this work, we introduce the first method that is fast enough to run on 100+ billion parameter models in a few hours on a single GPU and accurate enough to prune them to sparsity levels of 50% to 60% without significant performance drop.

3 The SparseGPT Algorithm

3.1 Fast Approximate Reconstruction

Motivation. As outlined in Section 2, for a fixed pruning mask $\mathbf{M}$, the optimal values of all weights in the mask can be calculated by solving the sparse reconstruction problem corresponding to each row. However, doing so exactly requires inverting the Hessian matrix corresponding to the values preserved by the pruning mask $\mathbf{M_i}$ for row i, i.e. computing $(\mathbf{H}_{\mathbf{M_i}})^{-1}$, for all rows $1 \leq i \leq d_{\text{row}}$. One such inversion takes $O(d_{\text{col}}^3)$ time, for a total computational complexity of $O(d_{\text{row}} \cdot d_{\text{col}}^3)$ over d_{row} rows. In practical terms, for a Transformer model, this means that the overall runtime scales with the 4th power of the hidden dimension d_{hidden} and will thus clearly be infeasible to run on the largest GPT variants. To arrive at a practical algorithm, we need to improve the overall runtime by at least a full factor of d_{hidden}, corresponding to a $> 10000\times$ compute reduction for models with more than 100 billion parameters. We will achieve this via a series of careful approximations.

[2] In the context of quantization, there is evidence [47] that optimization steps also have to be scaled with increasing model size for AdaPrune-like methods, which would lead to quadratic rather than linear runtime scaling.

Different Row-Hessian Challenge. The high computational complexity of optimally reconstrucing the unpruned weights following Equation 3 mainly stems from the fact that solving *each row* requires the *individual* inversion of a $O(d_{\text{col}} \times d_{\text{col}})$ matrix. This is because the row masks $\mathbf{M_i}$ are generally different and $(\mathbf{H}_{\mathbf{M_i}})^{-1} \neq (\mathbf{H}^{-1})_{\mathbf{M_i}}$, i.e., the inverse of a masked Hessian does *not* equal the masked version of the full inverse. This is illustrated also in Figure 2. If all row-masks were the same, then we would only need to compute a single shared inverse, as $\mathbf{H} = 2\mathbf{X}\mathbf{X}^\top$ depends just on the layer inputs which are the same for all rows.

Such a constraint could be enforced in the mask selection, but this would have a major impact on the final model accuracy, as sparsifying weights in big structures, like entire columns, is known to be much more difficult than pruning them individually[3]. The key towards designing an approximation algorithm that is both accurate and efficient lies in enabling the reuse of Hessians between rows with distinct pruning masks. We now propose an algorithm that achieves this in a principled manner.

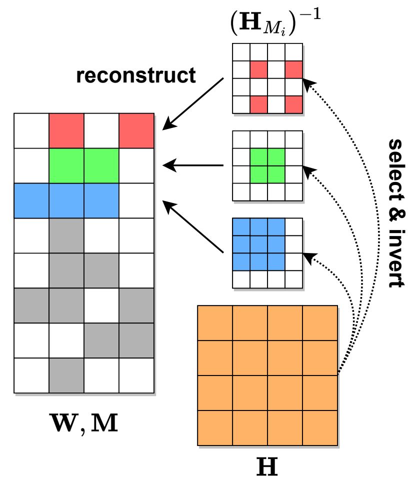

Figure 2: Illustration of the Row-Hessian challenge. Rows are pruned independently, pruned weights are in white. Hessian information is used for weight reconstruction. Since masks may be different per row, the inverse computation must be performed independently for each row.

Equivalent Iterative Perspective. To motivate our algorithm, we first have to look at the row-wise weight reconstruction from a different *iterative* perspective, using the classic OBS update [19, 43, 10]. Assuming a quadratic approximation of the loss, for which the current weights $\mathbf{w}$ are optimal, the OBS update $\boldsymbol{\delta}_m$ provides the optimal adjustment of the remaining weights to compensate for the removal of the weight at index m:

$$\boldsymbol{\delta_m} = -\frac{w_m}{[\mathbf{H}^{-1}]_{mm}} \cdot \mathbf{H}^{-1}_{:,m} \text{ incurring error } \varepsilon_m = \frac{w_m^2}{2[\mathbf{H}^{-1}]_{mm}}. \tag{4}$$

Since the loss function corresponding to the layer-wise pruning of one row of $\mathbf{W}$ is a quadratic, the OBS formula is exact in this case. Hence, $\mathbf{w} + \boldsymbol{\delta}_m$ is the optimal weight reconstruction corresponding to mask $\{m\}^C$. Further, given an optimal sparse reconstruction $\mathbf{w}^{(\mathbf{M})}$ corresponding to mask $\mathbf{M}$, we can apply OBS again to find the optimal reconstruction for mask $\mathbf{M}' = \mathbf{M} - \{m\}$. Consequently, this means that instead of solving for a full mask $\mathbf{M} = \{m_1, \ldots, m_p\}^C$ directly, we could iteratively apply OBS to individually prune the weights m_1 up until m_p in order, one-at-a-time, reducing an initially complete mask to $\mathbf{M}$, and will ultimately arrive at *the same* optimal solution as applying the standard closed-form linear regression reconstruction with the full $\mathbf{M}$ directly.

Optimal Partial Updates. Applying the OBS update $\boldsymbol{\delta}_m$ potentially adjusts the values of all available parameters (in the current mask $\mathbf{M}$) in order to compensate for the removal of w_m as much as possible. However, what if we wanted to update only the weights in a subset $\mathbf{U} \subseteq \mathbf{M}$, of all remaining unpruned weights? Thus, we could still benefit from significant error compensation using only weights in $\mathbf{U}$ while reducing the cost of applying OBS.

Such a partial update can indeed be accomplished by simply computing the OBS update using $\mathbf{H}_\mathbf{U}$, the Hessian corresponding to $\mathbf{U}$, rather than $\mathbf{H}_\mathbf{M}$, and updating only $\mathbf{w}_\mathbf{U}$. Importantly, the loss of our particular layer-wise problem remains quadratic also for $\mathbf{U}$ and the OBS updates are still optimal: the restriction to $\mathbf{U}$ does not incur any extra approximation error by itself, only the error compensation might not be as effective, as less weights are available for adjustment. At the same time, if $|\mathbf{U}| < |\mathbf{M}|$, then inverting $\mathbf{H}_\mathbf{U}$ will be a lot faster than inverting $\mathbf{H}_\mathbf{M}$. We will now utilize this mechanism to accomplish our goal of synchronizing the masked Hessians across all rows of $\mathbf{W}$.

Hessian Synchronization. In the following, assume a fixed ordering of the input features $j = 1, \ldots, d_{\text{col}}$. Since those are typically arranged randomly, we will just preserve the given order for simplicity, but any permutation could in principle be chosen. Next, we define a sequence of d_{col} index subsets U_j recursively as

$$U_{j+1} = U_j - \{j\} \text{ with } U_1 = \{1, \ldots, d_{\text{col}}\}. \tag{5}$$

In words, starting with U_1 being the set of all indices, each subset U_{j+1} is created by removing the smallest index from the previous subset U_j. These subsets also impose a sequence of inverse Hessians $(\mathbf{H}_{U_j})^{-1} = ((2\mathbf{X}\mathbf{X}^\top)_{U_j})^{-1}$ which we are going to share across all rows of $\mathbf{W}$. Crucially, following [11], the updated inverse $(\mathbf{H}_{U_{j+1}})^{-1}$ can be calculated efficiently by removing the first row and column, corresponding to j in the original $\mathbf{H}$, from the inverse of $(\mathbf{H}_{U_j})^{-1}$ in

[3]For example, structured (column-wise) pruning ResNet50 to 50% structured sparsity without accuracy loss is challenging, even with extensive retraining [30], while unstructured pruning to 90% sparsity is easily achievable with state-of-the-art methods [6, 39].

$O(d_{\text{col}}^2)$ time via one step of Gaussian elimination:

$$(\mathbf{H}_{U_{j+1}})^{-1} = \left((\mathbf{H}_{U_j})^{-1} - \frac{1}{[(\mathbf{H}_{U_j})^{-1}]_{11}} \cdot (\mathbf{H}_{U_j})^{-1}_{:,1} (\mathbf{H}_{U_j})^{-1}_{1,:} \right)_{1:,1:} \text{ with } (\mathbf{H}_{U_1})^{-1} = \mathbf{H}^{-1}. \tag{6}$$

Hence, the entire sequence of d_{col} inverse Hessians can be calculated recursively in $O(d_{\text{col}}^3)$ time, i.e. at similar cost to a single extra matrix inversion on top of the initial one for $\mathbf{H}^{-1}$.

Once some weight w_k has been pruned, it should not be updated anymore. Further, when we prune w_k, we want to update as many unpruned weights as possible for maximum error compensation. This leads to the following strategy: iterate through the U_j and their corresponding inverse Hessians $(\mathbf{H}_{U_j})^{-1}$ in order and prune w_j if $j \in M_i$, for all rows i. Importantly, each inverse Hessian $(\mathbf{H}_{U_j})^{-1}$ is computed only once and reused to remove weight j in all rows where it is part of the pruning mask. A visualization of the algorithm can be found in Figure 3.

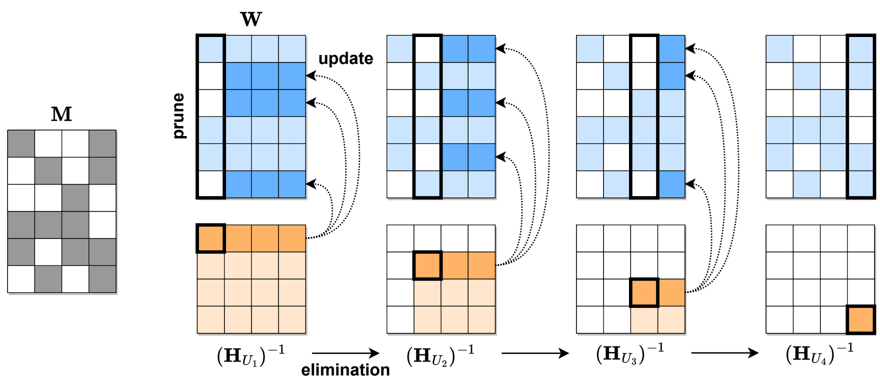

Figure 3: Visualization of the SparseGPT reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights in those rows, located to the "right" of the column being processed. Specifically, the weights to the "right" of a pruned weight (dark blue) will be updated to compensate for the pruning error, whereas the unpruned weights do not generate updates (light blue).

Computational Complexity. The overall cost of the approximate reconstruction process thus consists of three parts: (a) the computation of the initial Hessian, which takes time $\Theta(n \cdot d_{\text{col}}^2)$ where n is the number of input samples used[4], (b) iterating through the inverse Hessian sequence in time $O(d_{\text{col}}^3)$ and (c) the reconstruction/pruning itself. The latter can be upper bounded by the time it takes to apply (4) to all d_{row} rows of $\mathbf{W}$ for all d_{col} columns in turn, which is $O(d_{\text{col}} \cdot d_{\text{row}} \cdot d_{\text{col}})$. In total, this sums up to $O(d_{\text{col}}^3 + d_{\text{row}} \cdot d_{\text{col}}^2)$. For Transformer models, this is simply $O(d_{\text{hidden}}^3)$, and is thus a full d_{hidden}-factor more efficient than exact reconstruction. This means that we have reached our initial goal, as this complexity will be sufficient to make our scheme practical, even for extremely large models.

Weight Freezing Interpretation. While we have motivated the SparseGPT algorithm as an approximation to the exact reconstruction using optimal partial updates, there is also another interesting view of this scheme. Specifically, consider an exact greedy framework which compresses a weight matrix column by column, always optimally updating all not yet compressed weights in each step [11, 9]. At first glance, SparseGPT does not seem to fit into this framework as we only compress some of the weights in each column and also only update a subset of the uncompressed weights. Yet, mechanically, "compressing" a weight ultimately means fixing it to some specific value and ensuring that it is never "decompressed" again via some future update, i.e. that it is *frozen*. Hence, by defining column-wise compression as:

$$\text{compress}(\mathbf{w^j})_i = 0 \text{ if } i \notin \mathbf{M_i} \text{ and } w_i^j \text{ otherwise}, \tag{7}$$

i.e. zeroing weights not in the mask and fixing the rest to their current value, our algorithm can be interpreted as an exact column-wise greedy scheme. As we will show later, this perspective allows us to cleanly merge sparsification and quantization into a single compression pass, as well as inherit some other algorithmic enhancements from post-training quantization [9].

3.2 Adaptive Mask Selection

So far, we have only focused on the reconstruction aspect, i.e. assuming a fixed pruning mask $\mathbf{M}$. How should this mask be decided? One simple option would be to follow AdaPrune [22] and choose the mask for the whole layer in

[4]Taking the number of samples n to be a small multiple of d_{col} is sufficient for good results, even on very large models.

advance using e.g. the standard magnitude criterion [50] or including also second-order information [10]. However, recent work [8, 11] has shown that the updates applied during the pruning process change weights significantly due to correlations, and that taking this into account for the mask selection yields significantly more accurate results. This insight can be integrated into `SparseGPT` by *adaptively* choosing the mask while running the reconstruction pass.

One obvious way of doing so would be to pick the $p\%$ easiest weights to prune in each column i just when it is compressed, which will lead to $p\%$ overall sparsity. At the same time, this approach has one big disadvantage: the sparsity cannot be distributed non-uniformly across columns. This is a significant restriction, which will generally make unstructured pruning more difficult. This is particularly problematic for massive language models, as they are known to have a small number of highly sensitive outlier features [3, 46]. Further, [11] observe that some OPT models appear to have a large number of dead RELUs in the earlier layers, leading to many columns that are trivial to prune.

We propose to alleviate this disadvantage while still exploiting the significant accuracy gains of adaptive weight selection during pruning via *iterative blocking*. More precisely, we always select the pruning mask for $B_s = 128$ columns at-a-time based on the OBS reconstruction error ε from Equation (4), using the diagonal values in our Hessian sequence. We then perform the next B_s updates as discussed in the previous section, before selecting the mask for the next block, and so on. This procedure allows *non-uniform selection* per column, in particular also using the corresponding Hessian information[5], while at the same time considering also previous weight updates in the selection process.

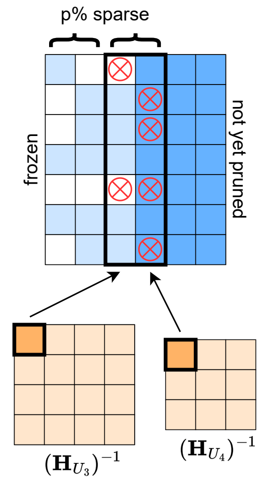

Figure 4: Mask selection.

3.3 Extension to Semi-Structured Sparsity

While so far we have only discussed *unstructured* pruning, `SparseGPT` is also easily adapted to *semi-structured* patterns such as the popular n:m sparsity format [49, 22] which delivers speedups in its 2:4 implementation on Ampere NVIDIA GPUs. Specifically, every consecutive m weights should contain exactly n zeros. Hence, we can simply choose blocksize $B_s = m$ and then enforce the zeros-constraint in the mask selection for each row by picking the n weights which incur the lowest error as per Equation (4). A similar strategy could also be applied for other semi-structured pruning patterns. Finally, we note that a larger B_s would not be useful in this semi-structured scenario since zeros cannot be distributed non-uniformly between different column-sets of size m.

3.4 Full Algorithm Pseudocode

Algorithm 1 The `SparseGPT` algorithm. We prune the layer matrix $\mathbf{W}$ to $p\%$ unstructured sparsity given inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$, lazy batch-update blocksize B and adaptive mask selection blocksize B_s; each B_s consecutive columns will be $p\%$ sparse.

$\mathbf{M} \leftarrow \mathbf{1}_{d_{\text{row}} \times d_{\text{col}}}$ *// 0/1 pruning mask*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})$ *// Hessian inverse information*
for $i = 0, B, 2B, \dots$ **do**
  for $j = i, \dots, i + B - 1$ **do**
    if $j \bmod B_s = 0$ **then**
      $\mathbf{M}_{:,j:(j+B_s)} \leftarrow$ mask of $(1-p)\%$ weights $w_c \in \mathbf{W}_{:,j:(j+B_s)}$ with largest $w_c^2 / [\mathbf{H}^{-1}]_{cc}$
    end if
    $\mathbf{E}_{:,j-i} \leftarrow \mathbf{W}_{:,j}^2 \,/\, [\mathbf{H}^{-1}]_{jj}$ *// pruning error*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{1} - \mathbf{M}_{:,j}) \cdot \mathbf{E}_{:,j-i}$ *// freeze weights that are not pruned*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update weights in block*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update all remaining weights*
end for
$\mathbf{W} \leftarrow \mathbf{W} \cdot \mathbf{M}$ *// set pruned weights to 0*

With the weight freezing interpretation discussed at the end of Section 3.1, the `SparseGPT` reconstruction can be cast in the column-wise greedy framework of the recent quantization algorithm GPTQ [9]. This means we can also inherit

[5] For a single column j, the OBS selection criterion would degrade to just the magnitude, as $[\mathbf{H}^{-1}]_{jj}$ is constant across rows.

several algorithmic enhancements from GPTQ, specifically: precomputing all the relevant inverse Hessian sequence information via a Cholesky decomposition to achieve numerical robustness and applying lazy batched weight matrix updates to improve the compute-to-memory ratio of the algorithm. Our adaptive mask selection, as well as its extensions to semi-structured pruning, are compatible with all of those extra techniques as well.

The pseudocode in Algorithm 1 presents the the unstructured sparsity version of the `SparseGPT` algorithm in its fully developed form, integrating also all relevant techniques from GPTQ discussed above.

3.5 Joint Sparsification & Quantization

Algorithm 1 operates in the column-wise greedy framework of GPTQ, thus sharing the computationally heavy steps of computing the Cholesky decomposition of $\mathbf{H}^{-1}$ and continuously updating $\mathbf{W}$. This makes it possible to merge both algorithms into a single joint procedure. Specifically, all weights that are frozen by `SparseGPT` are additionally quantized, leading to the following generalized errors to be compensated in the following update step:

$$\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{M}_{:,j} \cdot \text{quant}(\mathbf{W}_{:,j}))^2 \,/\, [\mathbf{H}^{-1}]_{jj}, \tag{8}$$

where $\text{quant}(\mathbf{w})$ rounds each weight in $\mathbf{w}$ to the nearest value on the quantization grid. Crucially, in this scheme, sparsification and pruning are performed *jointly* in *a single pass* at essentially no extra cost over just running `SparseGPT`.

We emphasize that doing quantization and pruning jointly means that later pruning decisions are influenced by earlier quantization rounding, and vice-versa. This is in contrast to prior techniques, such as OBC [11], which first sparsify a layer and then quantize the remaining weights, where quantization consequently has no influence on pruning outcomes.

4 Experiments

Setup. We implement `SparseGPT` in PyTorch [38] and use the HuggingFace Transformers library [45] for handling models and datasets. All experiments are conducted on *a single* NVIDIA A100 GPU with 80GB of memory. In this setup, `SparseGPT` can fully sparsify the 175-billion-parameter models in approximately 4 hours. Similar to [47, 9], we sparsify Transformer layers sequentially in order. This significantly reduces memory requirements, and also noticeably improves accuracy over handling all layers in parallel. All our compression experiments are performed in one-shot, without any finetuning, following a similar setup to that of recent work on post-training quantization of GPT-scale models [9, 47, 3].

For calibration data, following [9], we use 128 2048-token segments, randomly chosen from the C4 [40] dataset. This represents generic text data crawled from internet and makes sure that our experiments remain actually zero-shot since *no task-specific data is seen during pruning*.

Models, Datasets & Evaluation. We primarily work with the OPT model family as it provides a suite of models ranging from 125 million to 175 billion parameters, allowing us to study the scaling behavior of pruning relative to model size. Additionally, we also consider the 176 billion parameter variant of BLOOM [42]. In general, *our focus lies on the very largest variants* but we also show some results on smaller models to provide a broader picture, and in particular ablations with respect to model size.

In terms of datasets and evaluation, we mainly focus on perplexity on the raw WikiText2 test set [31], a popular benchmark in LLM compression literature [47, 36, 9, 46]. In the Appendix, we also show results on a set of text segments sampled from the C4 validation set. In general, perplexity is known be a challenging and stable metric that is well suited for evaluating the accuracy of compression methods [47, 11, 4]. Yet, for additional interpretability, we also provide some ZeroShot accuracy results for LAMBADA [35], ARC (Easy and Challenge) [2], PIQA [44] and StoryCloze [33].

We emphasize that the main focus of our evaluation lies on *accuracy of the sparse models, relative to the dense baseline* rather than on absolute numbers. We calculate perplexity in easily-reproducible fashion following the procedure described by HuggingFace[6]: we concatenate all samples of the raw dataset with "\n\n" separators, encode and split the entire sequence into non-overlapping segments of 2048 tokens (the maximum window size of both OPT and BLOOM), on which the standard average causal language modelling loss is computed. The final perplexity is the exponentiated version of this result. Different preprocessing may influence absolute numbers, but does not affect our claims, as we mainly focus on performance relative to the dense model. Our ZeroShot evaluations are performed using GPTQ's [9] implementation, which is in turn based on the popular [46, 4] EleutherAI-eval-harness[7]. We emphasize that all dense and sparse results were computed with exactly the same code to ensure a fair comparison.

[6] `https://huggingface.co/docs/transformers/perplexity`

[7] `https://github.com/EleutherAI/lm-evaluation-harness`

Baselines. We believe to be the first academic work to perform post-training pruning of massive models. (As discussed previously, prior post-training pruning techniques have only been applied to models 1000x smaller in size, and do not scale to GPT model sizes.) As such, there are no standard benchmarks, and we therefore primarily evaluate what levels of sparsity are achievable with `SparseGPT` while maintaining close to dense model accuracy.

Nevertheless, we also compare with the highly popular magnitude pruning criterion [50] which drops the weights of smallest absolute value, applied layerwise. This technique easily scales to extremely large models in terms of computational efficiency, but, as we show, does not perform well in terms of accuracy.

4.1 Results

Pruning Difficulty Scaling with Model Size. In our first set of experiments, we study how the difficulty of sparsifying LLMs changes with their size. For this, we consider the entire OPT model family and uniformly prune all linear layers, excluding the embeddings and the head as is standard [41, 25], to 50% unstructured sparsity, full 4:8 or full 2:4 semi-structured sparsity. (All three correspond to 50% overall sparsity, but the 2:4 pattern is the most stringent, followed by 4:8 and unstructured sparsity.) The raw-WikiText2 performance numbers are given in Table 1 and visualized in Figure 1 (right).

OPT	Sparsity	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
dense	0%	27.66	22.01	14.63	12.46	10.86	10.12	9.56	9.33	8.34
Magnitude	50%	193.	97.80	1.7e4	265.	969.	1.2e5	168.	4.2e4	4.3e4
`SparseGPT`	50%	36.89	31.60	17.45	13.44	11.56	11.12	9.77	9.33	8.21
`SparseGPT`	4:8	44.75	38.82	20.02	14.97	12.54	11.72	10.28	9.66	8.45
`SparseGPT`	2:4	59.17	50.21	24.04	17.14	14.16	12.91	10.88	10.10	8.73

Table 1: OPT perplexity results on raw-WikiText2.

One immediate finding is that the accuracy of magnitude-pruned models collapses across all scales, with larger variants generally dropping worse, relative to smaller ones. This is in stark contrast to smaller vision models which can usually be pruned via simple magnitude to 50% or more at very little loss of accuracy [43, 11]. This highlights the importance of more accurate pruners in the context of extremely large generative language models, but also the fact that perplexity is a very sensitive metric.

For `SparseGPT`, the trend is very different: already at 2.7B parameters, the perplexity loss is < 1 point, at 66B, there is zero loss and at the very largest scale there is even a slight accuracy improvement over the dense baseline. (As can be seen in Figure 1, sparse OPT models of $\leq$ 50% sparsity all have lower perplexity than the dense baseline, although the differences are small.)

In general, there is a clear trend of larger models being significantly easier to sparsify, which we speculate may be due to them being more overparametrized and also more noise resistant in general. We think a more detailed investigation of this phenomenon would be a great topic for future work. For 4:8 and 2:4 sparsity, the behavior is very similar, but accuracy drops are typically a bit higher due to the sparsity pattern being significantly more constrained [22]. Nevertheless, at the largest scale, the perlexity increases are only 0.11 and 0.39 for 4:8 and 2:4 sparsity, respectively. We emphasize that these sparsity pattern can actually achieve $2\times$ speedup in practice [22, 32], with commercially available NVIDIA Ampere GPUs already including support for 2:4 sparsity.

Sparsity Scaling for 100+ Billion Parameter Models. Next, we take a closer look at the largest publicly-available dense models, OPT-175B and BLOOM-176B, and investigate how their performance scales with the degree of sparsity induced by either `SparseGPT` or magnitude pruning. The results are visualized in Figures 1 and 5 (both left panels).

For the OPT-175B model, for which the results are presented in Figure 1 (left), magnitude pruning can achieve at most 10% sparsity before significant accuracy loss occurs; meanwhile, `SparseGPT` enables up to 60% sparsity at a comparable perplexity increase. BLOOM-176B, for which the results are provided in Figure 5 (left), appears to be more favorable for magnitude pruning, admitting up 30% sparsity without major loss; still, `SparseGPT` can deliver 50% sparsity, a $1.66\times$ improvement, at a similar level of perplexity degradation. Even at 80% sparsity, models compressed by `SparseGPT` still score reasonable perplexities, while magnitude pruning leads to a complete collapse (> 100 perplexity) already at 40% sparsity for OPT and 60% sparsity for BLOOM, respectively. Remarkably, `SparseGPT` is able to remove around *100 billion weights* from these models, with limited impact on model accuracy.

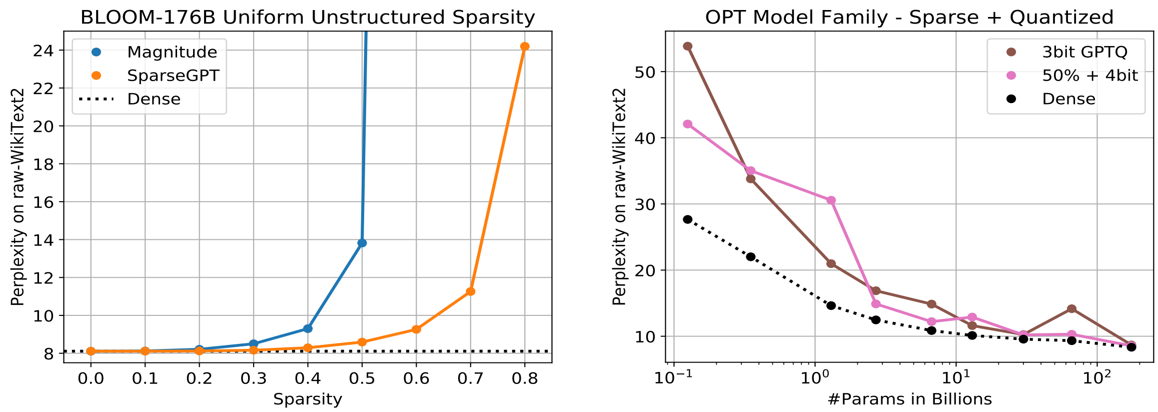

Figure 5: Left: Uniformly compressing BLOOM-176B to various sparsity levels with `SparseGPT` and magnitude pruning, respectively. Right: 50% sparsity + 4-bit quantization joint compression vs. 3-bit on the OPT family.

ZeroShot Experiments. To complement our perplexity evaluations, we now also provide results for various sparsified variants of OPT-175B on several ZeroShot tasks. ZeroShot evaluations are known to be relatively noisy [3] but at the same time more interpretable. All numbers are summarized in Table 2.

Method	Sparsity	Lambada	PIQA	ARC-easy	ARC-ch	StoryCloze	**Average**
Dense	0%	75.59	81.07	71.04	43.94	79.82	**70.29**
Magnitude	50%	00.02	54.73	28.03	25.60	47.10	**31.10**
`SparseGPT`	50%	76.51	80.03	69.65	41.30	78.87	**69.27**
`SparseGPT`	4:8	78.77	79.16	68.35	39.85	77.02	**68.63**
`SparseGPT`	2:4	79.47	79.16	67.08	38.99	76.19	**68.18**

Table 2: ZeroShot results on several datasets for sparsified variants of OPT-175B.

Overall, a similar trend to the perplexity results seems to hold with magnitude pruned models collapsing to close to random performance on several datasets while `SparseGPT` models stay close to the original accuracy. However, as expected, these numbers are significantly more noisy: for instance, 2:4 pruning appears to achieve noticeably higher accuracy than the dense model on Lambada despite being the most constrained sparsity pattern investigated here. Notice also that these effects ultimately average out when considering many different tasks, which is consistent to the literature [47, 3, 4].

Joint Sparsification & Quantization. Finally, another interesting research direction is the combination of sparsity and quantization, which would allow combining computational speedups from sparsity [26, 5] with memory savings from quantization [9, 3, 4]. Specifically, if we compress a model to 50% sparse + 4-bit weights, store only the non-zero weights and use a bitmask to indicate their positions, then this has the same overall memory consumption as 3-bit quantization. Hence, in Figure 5 (right) we compare `SparseGPT` 50% + 4-bit with state-of-the-art GPTQ [9] 3-bit numbers. While there seem to be a few outliers, 50% + 4-bit models are more accurate than their respective 3-bit versions for several models sizes, including 175B with 8.55 vs. 8.68 3-bit. We also tested 2:4 and 4:8 in combination with 4-bit on OPT-175B yielding encouraging 9.20 and 8.86 perplexities, which can likely be improved further using additional quantization tricks such as blocking [9, 4].

5 Related Work

Model compression aims to produce more efficient models, using approaches such as *pruning*, *quantization*, and *knowledge distillation*—we refer the reader to the respective surveys [21, 14, 15] for an in-depth discussion. Since the focus of our work is on compressing massive models, with 10-100s of billions of parameters, we will mainly focus on work in this specific area.

Pruning Methods. To our knowledge, we are the first academic work to investigate pruning of massive GPT-scale models, e.g. with more than 10 billion parameters. This gap in the literature may seem surprising, given both the widespread popularity of these models, and the significant amount of existing work on pruning, e.g. [17, 7, 12, 6, 43, 41, 39, 10, 13, 25]. One justification for this gap is the fact that most existing pruning methods, such as *gradual magnitude pruning* [16, 17, 12, 24], require *extensive retraining* following the pruning step in order to recover accuracy, while GPT-scale models usually require massive amounts of computation and parameter tuning both for training or finetuning [48], which renders retraining-based approaches difficult to apply. Thus, we are not aware of any work applying such gradual pruning methods at GPT scale.

`SparseGPT` is a *post-training* method for GPT-scale models, as it does not perform any finetuning. So far, post-training pruning methods have only been investigated at the scale of classic CNN or BERT-type models [22, 11, 27], which have 100-1000x fewer weights than our models of interest. We discussed the challenges of scaling existing post-training methods to GPT models, and the technical relationship between `SparseGPT` and these methods, in Section 2.

Post-Training Quantization. By contrast, there has been a significant amount of emerging work on post-training methods for *quantizing* GPT-scale models, closely following the first open releases of such models [48, 42]. Specifically, the ZeroQuant [47], LLM.int8() [3] and nuQmm [36] methods investigated the feasibility of round-to-nearest quantization for billion-parameter models, showing that 8-bit quantization for weights is feasible via this approach, but that activation quantization can be difficult due to the existence of outlier features. GPTQ [9] leverages approximate second-order information for accurate quantization of weights down to 2–4 bits, for the very largest models, and shows that this can bring inference speedups of 2-5x when coupled with efficient GPU kernels. Follow-up work by Xiao et al. [46] investigated joint activation and weight quantization down to 8 bits per component, proposing a smoothing-based scheme which reduces the difficulty of activation quantization and is complemented by efficient GPU kernels for fast inference. Concurrent work by Park et al. [37] tackles the hardness of quantizing activation outliers via *quadapters*, a set of learnable parameters whose goal is to scale activations channel-wise, while keeping the other model parameters unchanged. Very recent work by Dettmers and Zettlemoyer [4] investigate scaling relationships between model size, quantization bits, and different notions of accuracy for massive LLMs, observing a high degree of correlation between perplexity scores and aggregated zero-shot accuracy across tasks, as well as saturation behavior.

Since it focuses on sparsification rather than quantization, `SparseGPT` is complementary to quantization approaches. Specifically, as we have shown in Section 3.5, the `SparseGPT` algorithm can be applied in conjunction with GPTQ, the current state-of-the-art algorithm for weight quantization, and should be compatible with activation quantization approaches [3, 46, 37]. Thus, it would be very interesting to investigate in depth how compression errors compound when quantization and pruning are applied in conjunction.

6 Discussion

We have provided a new post-training pruning method called `SparseGPT`, specifically tailored to massive language models from the GPT family. Our results show for the first time that large-scale generative pretrained Transformer-family models can be compressed to high sparsity via weight pruning in *one-shot, without any retraining*, at low loss of accuracy, when measured both in terms of perplexity and zero-shot performance. Specifically, we have shown the largest open-source GPT-family models (e.g. OPT-175B and BLOOM-176B) can reach 50-60% sparsity with low accuracy fluctuations. Surprisingly, this means that more than 100 billion weights from these models can be ignored at inference time. Central to our approach is a new large-scale approximate sparse regression algorithm, which generalizes to semi-structured (2:4 and 4:8) patterns, and is also compatible with existing weight quantization approaches.

Interestingly, our method is *local*: after each pruning step, it performs weight updates, designed to preserve the input-output relationship for each layer. These updates are computed without any global gradient information. Thus, it appears that the high degree of parametrization of massive GPT models allows our method to directly identify sparse accurate models in the "close neighborhood" of the dense pretrained model. Remarkably, since our main accuracy measure (perplexity) is extremely sensitive, it appears that the output of the generated sparse model correlates extremely closely with that of the dense model. Our second main finding is that *larger models are easier to sparsify*: at a fixed sparsity level, the relative accuracy drop for the sparse model, relative to the dense one, narrows as we increase the model size, to the point where inducing 50% sparsity results in practically no accuracy decrease on the largest models. This finding should be seen as very encouraging for future work on compressing such massive models.

One natural avenue for future work would be to investigate *fine-tuning mechanisms* for such large-scale models, which would allow further accuracy recovery. We conjecture that this should be possible, and that probably at least 80-90% sparsity can be achieved with progressive pruning and fine-tuning. Another extension which we plan to investigate is the applicability of our approaches *during training*, to reduce the computational cost of pre-training these massive models.

Acknowledgments

The authors gratefully acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 programme (grant agreement No. 805223 ScaleML), as well as experimental support from Eldar Kurtic, and from the IST Austria IT department, in particular Stefano Elefante, Andrei Hornoiu, and Alois Schloegl.

References

[1] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14(5-6):629–654, 2008.

[2] Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, et al. A systematic classification of knowledge, reasoning, and context within the ARC dataset. *arXiv preprint arXiv:1806.00358*, 2018.

[3] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[4] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*, 2022.

[5] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[6] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, 2020.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[8] Elias Frantar and Dan Alistarh. SPDY: Accurate pruning with speedup guarantees. *arXiv preprint arXiv:2201.13096*, 2022.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[10] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-FAC: Efficient matrix-free approximations of second-order information. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[11] Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal Brain Compression: A framework for accurate post-training quantization and pruning. *arXiv preprint arXiv:2208.11580*, 2022. Accepted to NeurIPS 2022, to appear.

[12] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

[13] Zhe Gan, Yen-Chun Chen, Linjie Li, Tianlong Chen, Yu Cheng, Shuohang Wang, Jingjing Liu, Lijuan Wang, and Zicheng Liu. Playing lottery tickets with vision and language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 652–660, 2022.

[14] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[15] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[16] Masafumi Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207 – 218, 1994. Backpropagation, Part IV.

[17] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[19] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

[20] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 2018.

[21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[22] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Seffi Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find N:M transposable masks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[23] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Accurate post training quantization with small calibration sets. In *International Conference on Machine Learning (ICML)*, 2021.

[24] Eldar Kurtic and Dan Alistarh. Gmp*: Well-tuned global magnitude pruning can outperform most bert-pruning methods. *arXiv preprint arXiv:2210.06384*, 2022.

[25] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The Optimal BERT Surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

[26] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Bill Nell, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

[27] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. *arXiv preprint arXiv:2204.09656*, 2022.

[28] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Conference on Neural Information Processing Systems (NeurIPS)*, 1990.

[29] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations (ICLR)*, 2021.

[30] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning (ICML)*, 2021.

[31] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[32] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[33] Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James Allen. Lsdsem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 46–51, 2017.

[34] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, 2020.

[35] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[36] Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. nuQmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

[37] Minseop Park, Jaeseong You, Markus Nagel, and Simyung Chang. Quadapter: Adapter for gpt-2 quantization. *arXiv preprint arXiv:2211.16912*, 2022.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[39] Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating compressed/decompressed training of deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[40] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[41] Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*, 2020.

[42] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[43] Sidak Pal Singh and Dan Alistarh. WoodFisher: Efficient second-order approximation for neural network compression. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[44] Sandeep Tata and Jignesh M Patel. PiQA: An algebra for querying protein data sets. In *International Conference on Scientific and Statistical Database Management*, 2003.

[45] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[46] Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.

[47] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*, 2022.

[48] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[49] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning N:M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021.

[50] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

7 Appendix

7.1 Additional Results

In addition to our raw-WikiText2 numbers in the main paper, we now also present some OPT-175B and BLOOM-175B results on a subsample from the C4 dataset consisting of randomly crawled website text. Specifically, we take the first validation shard and randomly sample 256 2048-token segments (each contained in a single large enough document). This is exactly the same sampling procedure that is also used for the calibration dataset, the latter is just sampled from the first *training shard*. The results are shown in Figure 6.

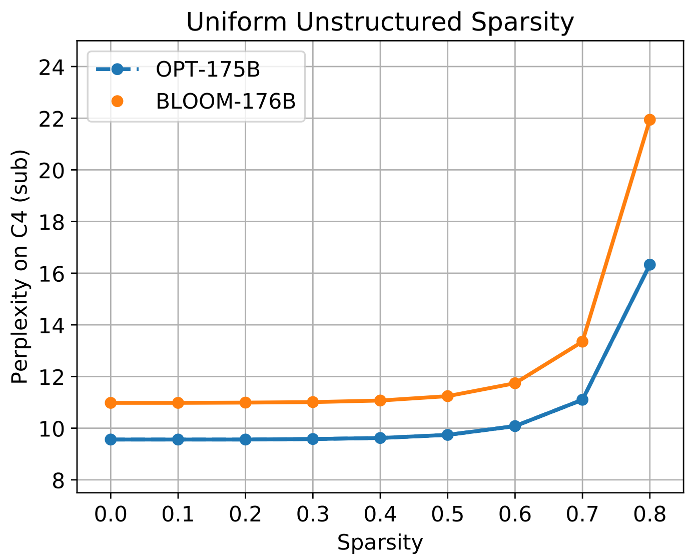

Figure 6: C4 perplexity for OPT-175B and BLOOM-176B at various `SparseGPT` sparsity levels.

In summary, these results confirm our raw-WikiText2 findings from the main paper that `SparseGPT` is able to achieve 50-60% sparsity with an only very minor perplexity increase. In this direct comparison between models one can also see how BLOOM's accuracy starts to degrade slightly more quickly, especially at the higher sparsity levels.