SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot

Elias Frantar [1] **Dan Alistarh** [1 2]

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. We can execute `SparseGPT` on the largest available open-source models, OPT-175B and BLOOM-176B, in under 4.5 hours, and can reach 60% unstructured 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. The code is available at: https://github.com/IST-DASLab/sparsegpt.

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 illustration, the top-performing GPT-175B models have 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* (Dettmers et al., 2022; Yao et al., 2022; Xiao et al., 2022; Frantar et al., 2022a), that is, reducing the precision of the model's numerical representation.

[1]Institute of Science and Technology Austria (ISTA) [2]Neural Magic Inc. Correspondence to: Elias Frantar <elias.frantar@ist.ac.at>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

A complementary approach for compression is *pruning*, which removes network elements, from individual weights (unstructured pruning) to higher-granularity structures such as rows/columns of the weight matrices (structured pruning). Pruning has a long history (LeCun et al., 1989; Hassibi et al., 1993), and has been applied successfully in the case of vision and smaller-scale language models (Hoefler et al., 2021). Yet, the best-performing pruning methods require *extensive retraining* of the model to recover accuracy. In turn, this is extremely expensive for GPT-scale models. While some accurate *one-shot* pruning methods exist (Hubara et al., 2021a; Frantar et al., 2022b), compressing the model without retraining, unfortunately even they become very expensive when applied to models with billions of parameters. Thus, to date, there is essentially no work on accurate pruning of billion-parameter 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 a set of extremely large-scale instances of *sparse regression*. It then solves these instances via a new approximate sparse regression solver, which is efficient enough to execute in a few hours on the largest openly-available GPT models (175B parameters), on 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 experiments, from which we provide a snapshot in Figures 1 and 2, lead to the following observations. First, as shown in Figure 1, `SparseGPT` can induce uniform layer-wise sparsity of up to 60% in e.g. the 175-billion-parameter variant of the OPT family (Zhang et al., 2022), with minor accuracy loss. By contrast, the only known one-shot baseline which easily extends to this scale, Magnitude Pruning (Hagiwara, 1994; Han et al., 2015), preserves accuracy only until 10% sparsity, and completely collapses beyond 30% sparsity. Second, as shown in Figure 2, `SparseGPT` can also accurately impose sparsity in the more stringent, but hardware-friendly, 2:4 and 4:8 semi-structured sparsity patterns (Mishra et al., 2021), although this comes at an accuracy loss relative to the dense baseline for smaller models.

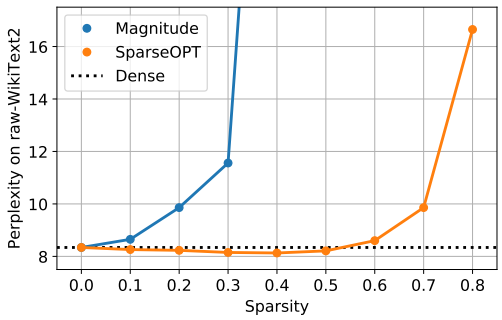

Figure 1. Sparsity-vs-perplexity comparison of `SparseGPT` against magnitude pruning on OPT-175B, when pruning to different *uniform* per-layer sparsities.

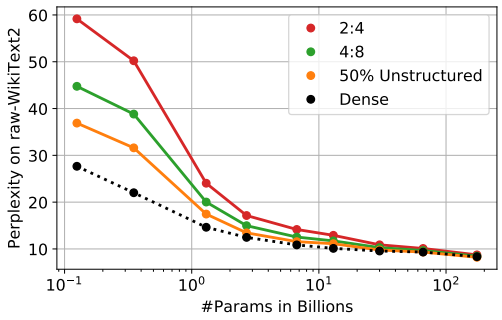

Figure 2. Perplexity vs. model and sparsity type when compressing the entire OPT model family (135M, 350M, ..., 66B, 175B) to different sparsity patterns using `SparseGPT`.

One key positive finding, illustrated in Figure 2, is that *larger models are more compressible*: they drop significantly less accuracy at a fixed sparsity, relative to their smaller counterparts. (For example, the largest models from the OPT and BLOOM families can be sparsified to 50% with almost no increase in perplexity.) In addition, our method allows sparsity to be *compounded* with weight quantization techniques (Frantar et al., 2022a): for instance, we can induce 50% weight sparsity jointly with 4-bit weight quantization with negligible perplexity increase on OPT-175B.

One notable property of `SparseGPT` is that it 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, we find it 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.

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 (e.g., sparse and/or quantized) version of $\theta^\star$. Originally popularized in the context of quantization (Hubara et al., 2021b; Nagel et al., 2020; Li et al., 2021), this setting has also recently been successfully extended to pruning (Hubara et al., 2021a; Frantar et al., 2022b; Kwon et al., 2022).

Layer-Wise Pruning. Post-training compression is usually done 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, for given inputs $\mathbf{X}_\ell$, of the uncompressed layer with weights $\mathbf{W}_\ell$ and that of the compressed one. Specifically, for pruning, (Hubara et al., 2021a) posed this problem as that of finding, for each layer ℓ, a sparsity mask[1] $\mathbf{M}_\ell$ with a certain target density, and possibly updated weights $\widehat{\mathbf{W}}_\ell$ such that

$$\text{argmin}_{\text{mask } \mathbf{M}_\ell, \widehat{\mathbf{W}}_\ell} ||\mathbf{W}_\ell \mathbf{X}_\ell - (\mathbf{M}_\ell \odot \widehat{\mathbf{W}}_\ell)\mathbf{X}_\ell||_2^2. \quad (1)$$

The overall compressed model is then obtained by "stitching together" the individually compressed layers.

Mask Selection & Weight Reconstruction. A key aspect of the layer-wise pruning problem in (1) is that both the mask $\mathbf{M}_\ell$ as well as the remaining weights $\widehat{\mathbf{W}}_\ell$ are optimized *jointly*, which makes this problem NP-hard (Blumensath & Davies, 2008). Thus, exactly solving it for larger layers is unrealistic, leading all existing methods to resort to approximations.

A particularly popular approach is to separate the problem into *mask selection* and *weight reconstruction* (He et al., 2018; Kwon et al., 2022; Hubara et al., 2021a). Concretely, this means to first choose a pruning mask $\mathbf{M}$ according to some saliency criterion, like the weight magnitude (Zhu & Gupta, 2017), and then optimize the remaining unpruned weights while keeping the mask unchanged. Importantly, once the mask is fixed, (1) turns into a *linear squared error problem* that is easily optimized.

Existing Solvers. Early work (Kingdon, 1997) applied iterated linear regression to small networks. More recently, the AdaPrune approach (Hubara et al., 2021a) has shown good results for this problem on modern models 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 (Frantar & Alistarh, 2022) performs pruning in gradual steps with re-optimization in between and OBC (Frantar et al., 2022b) introduces a greedy solver which removes weights one-at-a-time, fully reconstructing the remaining weights after each iteration, via efficient closed-form equations.

[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.

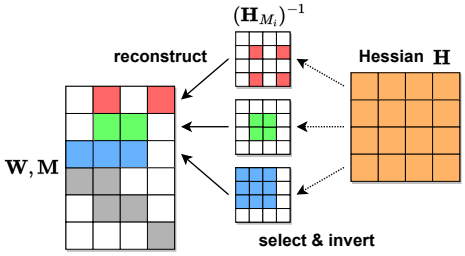

Figure 3. Illustration of the row-Hessian challenge: rows are sparsified independently, pruned weights are in white.

Difficulty of Scaling to 100+ Billion Parameters. Prior post-training techniques have all been designed to accurately compress models up to a few hundred million parameters with several minutes to a few hours of compute. However, our goal here is to sparsify models up to $1000\times$ larger.

Even AdaPrune, the method optimized for an ideal speed/accuracy trade-off, takes a few hours to sparsify models with just 1.3 billion parameters (see also Section 4), scaling linearly to several hundred hours (a few weeks) for 175B Transformers. More accurate approaches are at least several times more expensive (Frantar & Alistarh, 2022) than AdaPrune or even exhibit worse than linear scaling (Frantar et al., 2022b). This suggests that scaling up existing accurate post-training techniques to extremely large models is a challenging endeavor. Hence, we propose a new layer-wise solver `SparseGPT`, based on careful approximations to closed form equations, which easily scales to giant models, both in terms of runtime as well as accuracy.

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 exactly by solving the sparse reconstruction problem corresponding to each matrix row $\mathbf{w^i}$ via:

$$\mathbf{w^i}_{\mathbf{M_i}} = (\mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^\top)^{-1}\mathbf{X}_{\mathbf{M_i}}(\mathbf{w}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}})^\top, \qquad (2)$$

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 their respective weights. However, this requires inverting the Hessian matrix $\mathbf{H}_{\mathbf{M_i}} = \mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^\top$ corresponding to the values preserved by the pruning mask $\mathbf{M_i}$ for row i, i.e. computing $(\mathbf{H}_{\mathbf{M_i}})^{-1}$, separately 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. For a Transformer model, this means that the overall runtime scales with the 4th power of the hidden dimension d_{hidden}; we need a speedup by at least a full factor of d_{hidden} to arrive at a practical algorithm.

Different Row-Hessian Challenge. The high computational complexity of optimally reconstructing the unpruned weights following Equation 2 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 3. If all row-masks were the same, then we would only need to compute a single shared inverse, as $\mathbf{H} = \mathbf{X}\mathbf{X}^\top$ 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[2]. 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.

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 (Hassibi et al., 1993; Singh & Alistarh, 2020; Frantar et al., 2021). 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, incurring error ε_m:

$$\boldsymbol{\delta_m} = -\frac{w_m}{[\mathbf{H}^{-1}]_{mm}} \cdot \mathbf{H}^{-1}_{:,m}, \quad \varepsilon_m = \frac{w_m^2}{[\mathbf{H}^{-1}]_{mm}}. \qquad (3)$$

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 closed-form 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. However, what if we only update the weights in a subset $\mathbf{U} \subseteq \mathbf{M}$ among remaining unpruned weights? Thus,

[2]For example, structured (column-wise) pruning ResNet50 to $> 50\%$ structured sparsity without accuracy loss is challenging, even with extensive retraining (Liu et al., 2021), while unstructured pruning to 90% sparsity is easily achievable with state-of-the-art methods (Evci et al., 2020; Peste et al., 2021).

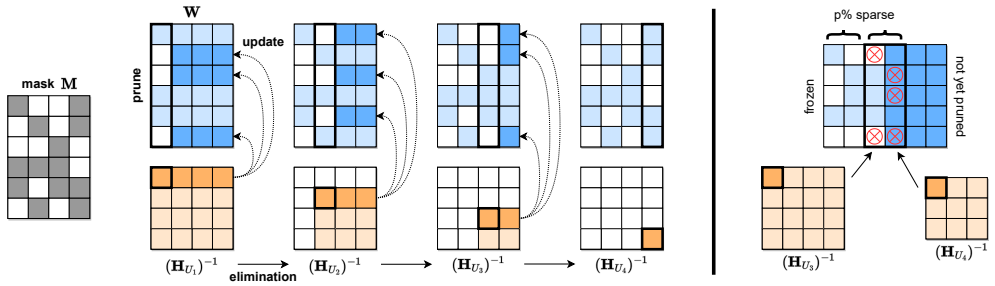

Figure 4. [Left] Visualization of the SparseGPT reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights 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). [Right] Illustration of the adaptive mask selection via iterative blocking.

we could still benefit from 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:

$$U_{j+1} = U_j - \{j\} \text{ with } U_1 = \{1, \ldots, d_{\text{col}}\}. \tag{4}$$

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 $(\mathbf{H}_{U_j})^{-1} = ((\mathbf{X}\mathbf{X}^\top)_{U_j})^{-1}$ which we are going to share across all rows of $\mathbf{W}$. Crucially, following (Frantar et al., 2022b), 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 $\mathbf{B} = (\mathbf{H}_{U_j})^{-1}$ in $O(d_{\text{col}}^2)$ time via one step of Gaussian elimination:

$$(\mathbf{H}_{U_{j+1}})^{-1} = \left(\mathbf{B} - \frac{1}{[\mathbf{B}]_{11}} \cdot \mathbf{B}_{:,1}\mathbf{B}_{1,:}\right)_{2:,2:}, \tag{5}$$

with $(\mathbf{H}_{U_1})^{-1} = \mathbf{H}^{-1}$. 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 $j \notin M_i$, 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 4.

Computational Complexity. The overall cost 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—we found that taking the number of samples n to be a small multiple of d_{col} is sufficient for good and stable results, even on very large models (see Appendix A); (b) iterating through the inverse Hessian sequence in time $O(d_{\text{col}}^3)$ and (c) the reconstruction/pruning itself. The latter cost can be upper bounded by the time it takes to apply (3) to all d_{row} rows of $\mathbf{W}$ for all d_{col} columns in turn, which is $O(d_{\text{col}} d_{\text{row}} d_{\text{col}})$. In total, this sums up to $O(d_{\text{col}}^3 + d_{\text{row}} d_{\text{col}}^2)$. 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 (Frantar et al., 2022b;a). 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:

$$\text{compress}(\mathbf{w^j})_i = 0 \text{ if } j \notin M_i \text{ and } w_i^j \text{ otherwise,} \quad (6)$$

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. This perspective will allow us to cleanly merge sparsification and quantization into a single compression pass.

3.2. Adaptive Mask Selection

So far, we have focused only on weight reconstruction, i.e. assuming a fixed pruning mask $\mathbf{M}$. One simple option for deciding the mask, following AdaPrune (Hubara et al., 2021a), would be via magnitude pruning (Zhu & Gupta, 2017). However, recent work (Frantar et al., 2022b) shows that updates during pruning change weights significantly due to correlations, and that taking this into account in the mask selection yields better results. This insight can be integrated into `SparseGPT` by *adaptively* choosing the mask while running the reconstruction.

One obvious way of doing so would be picking the $p\%$ easiest weights to prune in each column i when it is compressed, leading to $p\%$ overall sparsity. The big disadvantage of this approach is that sparsity cannot be distributed non-uniformly across columns, imposing additional unnecessary structure. This is particularly problematic for massive language models, which have a small number of highly-sensitive outlier features (Dettmers et al., 2022; Xiao et al., 2022).

We remove this disadvantage via *iterative blocking*. More precisely, we always select the pruning mask for $B_s = 128$ columns at a time (see Appendix A), based on the OBS reconstruction error ε from Equation (3), using the diagonal values in our Hessian sequence. We then perform the next B_s weight updates, 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, while at the same time considering also previous weight updates for selection. (For a single column j, the selection criterion becomes the magnitude, as $[\mathbf{H}^{-1}]_{jj}$ is constant across rows.)

3.3. Extension to Semi-Structured Sparsity

`SparseGPT` is also easily adapted to *semi-structured* patterns such as the popular n:m sparsity format (Zhou et al., 2021; Hubara et al., 2021a) 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 (3). 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 $\mathbf{H}^{-1} = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$, lazy batch-update blocksize B and adaptive mask selection blocksize B_s; each B_s consecutive columns will be $p\%$ sparse.

$\mathbf{M} \leftarrow \mathbf{1}_{d_{\text{row}} \times d_{\text{col}}}$ *// binary pruning mask*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^\top$ *// Hessian inverse information*
for $i = 0, B, 2B, \ldots$ **do**
  for $j = i, \ldots, i + B - 1$ **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 $w_c^2/[\mathbf{H}^{-1}]_{cc}^2$
    end if
    $\mathbf{E}_{:,j-i} \leftarrow \mathbf{W}_{:,j} \,/\, [\mathbf{H}^{-1}]_{jj}$ *// pruning error*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{1} - \mathbf{M}_{:,j}) \cdot \mathbf{E}_{:,j-i}$ *// freeze weights*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update*
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 (Frantar et al., 2022a). This means we can also inherit 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.

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

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

Table 1. OPT perplexity results on raw-WikiText2.

OPT - 50%	125M	350M	1.3B
Dense	27.66	22.00	14.62
Magnitude	193.	97.80	1.7e4
AdaPrune	58.66	48.46	32.52
SparseGPT	**36.85**	**31.58**	**17.46**

OPT	Sparsity	2.7B	6.7B	13B	30B	66B	175B
Dense	0%	12.47	10.86	10.13	9.56	9.34	8.35
Magnitude	50%	265.	969.	1.2e4	168.	4.2e3	4.3e4
SparseGPT	50%	**13.48**	**11.55**	**11.17**	**9.79**	**9.32**	**8.21**
SparseGPT	4:8	14.98	12.56	11.77	10.30	9.65	8.45
SparseGPT	2:4	17.18	14.20	12.96	10.90	10.09	8.74

quantized, leading to the following generalized errors to be compensated in the subsequent update step:

$$\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{M}_{:,j} \cdot \text{quant}(\mathbf{W}_{:,j})) \,/\, [\mathbf{H}^{-1}]_{jj}, \quad (7)$$

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 SparseGPT. Moreover, 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 joint techniques (Frantar et al., 2022b), which first sparsify a layer and then simply quantize the remaining weights.

4. Experiments

Setup. We implement SparseGPT in PyTorch (Paszke et al., 2019) and use the HuggingFace Transformers library (Wolf et al., 2019) for handling models and datasets. All pruning 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 Yao et al. (2022); Frantar et al. (2022a), we sparsify Transformer layers sequentially in order, which significantly reduces memory requirements. All our experiments are performed in one-shot, without finetuning, in a similar setup to recent work on post-training quantization of GPT-scale models (Frantar et al., 2022a; Yao et al., 2022; Dettmers et al., 2022). Additionally, in Appendix E we investigate the real-world acceleration of our sparse models with existing tools.

For calibration data, we follow Frantar et al. (2022a) and use 128 2048-token segments, randomly chosen from the first shard of the C4 (Raffel et al., 2020) dataset. This represents generic text data crawled from the 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 (Zhang et al., 2022), to study scaling behavior, but also consider the 176 billion parameter version of BLOOM (Scao et al., 2022). While *our focus lies on the very largest variants*, we also show some results on smaller models to provide a broader picture.

In terms of metrics, we mainly focus on *perplexity*, which is known to be a challenging and stable metric that is well suited for evaluating the accuracy of compression methods (Yao et al., 2022; Frantar et al., 2022b; Dettmers & Zettlemoyer, 2022). We consider the test sets of raw-WikiText2 (Merity et al., 2016) and PTB (Marcus et al., 1994) as well as a subset of the C4 validation data, all popular benchmarks in LLM compression literature (Yao et al., 2022; Park et al., 2022a; Frantar et al., 2022a; Xiao et al., 2022). For additional interpretability, we also provide ZeroShot accuracy results for Lambada (Paperno et al., 2016), ARC (Easy and Challenge) (Boratko et al., 2018), PIQA (Tata & Patel, 2003) and StoryCloze (Mostafazadeh et al., 2017).

We note that the main focus of our evaluation lies on the *accuracy of the sparse models, relative to the dense baseline* rather than on absolute numbers. Different preprocessing may influence absolute accuracy, but has little impact on our relative claims. The perplexity is calculated following precisely the procedure described by HuggingFace (HuggingFace, 2022), using full stride. Our ZeroShot evaluations are performed with GPTQ's (Frantar et al., 2022a) implementation, which is in turn based on the popular EleutherAI-eval-harness (EleutherAI, 2022). Additional evaluation details can be found in Appendix B. All dense and sparse results were computed with exactly the same code, available as supplementary material, to ensure a fair comparison.

Baselines. We compare against the standard magnitude pruning baseline (Zhu & Gupta, 2017), applied layer-wise, which scales to the very largest models. On models up to 1B parameters, we compare also against AdaPrune (Hubara et al., 2021a), the most efficient among existing accurate post-training pruning methods. For this, we use the memory-optimized reimplementation of Frantar & Alistarh (2022) and further tune the hyper-parameters provided by the AdaPrune authors. We thus achieve a $\approx 3\times$ speedup without impact on solution quality, for our models of interest.

4.1. Results

Pruning vs. Model Size. We first study how the difficulty of pruning LLMs changes with their size. We consider the entire OPT model family and uniformly prune all linear layers, excluding the embeddings and the head, as standard (Sanh et al., 2020; Kurtic et al., 2022), to 50% unstructured sparsity, full 4:8 or full 2:4 semi-structured sparsity (the 2:4 pattern is the most stringent). The raw-WikiText2 per-

formance numbers are given in Table 1 and visualized in Figure 2. The corresponding results for PTB and C4 can be found in Appendix C and show very similar trends overall.

One immediate finding is that the accuracy of magnitude-pruned models collapses across all scales, with larger variants generally dropping faster than smaller ones. This is in stark contrast to smaller vision models which can usually be pruned via simple magnitude selection to 50% sparsity or more at very little loss of accuracy (Singh & Alistarh, 2020; Frantar et al., 2022b). It highlights the importance of accurate pruners for massive 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 essentially zero loss and at the very largest scale there is even a slight accuracy improvement over the dense baseline, which however seems to be dataset specific (see also Appendix C). AdaPrune, as expected, also yields a big improvement over magnitude pruning, but is significantly less accurate than `SparseGPT`. Despite the efficiency of AdaPrune, running it takes approximately $\approx$ 1.3h on a 350M model and $\approx$ 4.3h on a 1.3B one, while `SparseGPT` can fully sparsify 66B and 175B models in roughly the same time, executing on the same A100 GPU.

In general, there is a clear trend of larger models being easier to sparsify, which we speculate is due to overparametrization. A detailed investigation of this phenomenon would be a good direction for future work. For 4:8 and 2:4 sparsity, the behavior is similar, but accuracy drops are typically higher due to the sparsity patterns being more constrained (Hubara et al., 2021a). Nevertheless, at the largest scale, the perplexity increases are only of 0.11 and 0.39 for 4:8 and 2:4 sparsity, respectively.

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.

For the OPT-175B model (Figure 1) 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 (Figure 5) 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 40/60% sparsity for OPT and BLOOM, respectively. Remarkably, `SparseGPT` removes around *100 billion weights* from these models, with low impact on accuracy.

ZeroShot Experiments. To complement the perplexity evaluations, we provide results on several ZeroShot tasks. These evaluations are known to be relatively noisy (Dettmers et al., 2022), but more interpretable. Please see Table 2.

Overall, a similar trend holds, with magnitude-pruned models collapsing to close to random performance, while `SparseGPT` models stay close to the original accuracy. However, as expected, these numbers are more noisy: 2:4 pruning appears to achieve noticeably higher accuracy than the dense model on Lambada, despite being the most constrained sparsity pattern. These effects ultimately average out when considering many different tasks, which is consistent to the literature (Yao et al., 2022; Dettmers et al., 2022; Dettmers & Zettlemoyer, 2022).

Joint Sparsification & Quantization. Another interesting research direction is the combination of sparsity and quantization, which would allow combining computational speedups from sparsity (Kurtz et al., 2020; Elsen et al., 2020) with memory savings from quantization (Frantar et al., 2022a; Dettmers et al., 2022; Dettmers & Zettlemoyer, 2022). 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 6 (right) we compare `SparseGPT` 50% + 4-bit with state-of-the-art GPTQ (Frantar et al., 2022a) 3-bit numbers. It can be seen that 50% + 4-bit models are more accurate than their respective 3-bit versions for 2.7B+ parameter models, including 175B with 8.29 vs. 8.68 3-bit. We also tested 2:4 and 4:8 in combination with 4-bit on OPT-175B yielding 8.55 and 8.85 perplexities, suggesting that 4bit weight quantization only brings an ≈ 0.1 perplexity increase on top semi-structured sparsity.

Sensitivity & Partial N:M Sparsity. One important practical question concerning n:m pruning is what to do when the fully sparsified model is not accurate enough? The overall sparsity level cannot simply be lowered uniformly, instead one must choose a subset of layers to n:m-sparsify completely. We now investigate what a good selection is in the context of extremely large language models: we assume that 2/3 of the layers of OPT-175B/BLOOM-176B should be pruned to 2:4 sparsity and consider skipping either all layers of one type (attention, fully-connected-1, fully-connected-2) or skipping one third of consecutive layers (front, middle, back). The results are shown in Figure 7.

While the sensitivity of layer-types differs noticeably between models, there appears to be a clear trend when it comes to model parts: *later layers are more sensitive than earlier ones*; skipping the last third of the model gives the best accuracy. This has a very practical consequence in that, due to the sequential nature of `SparseGPT`, we can generate a sequence of increasingly 2:4 sparsified models

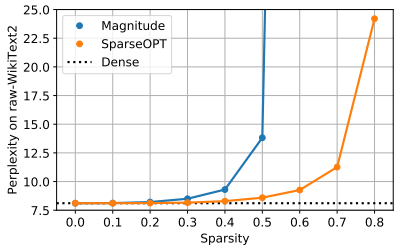

Figure 5. Uniform pruning BLOOM-176B.

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

Method	Spars.	Lamb.	PIQA	ARC-e	ARC-c	Story.	**Avg.**
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%	78.47	80.63	70.45	43.94	79.12	**70.52**
SparseGPT	4:8	80.30	79.54	68.85	41.30	78.10	**69.62**
SparseGPT	2:4	80.92	79.54	68.77	39.25	77.08	**69.11**

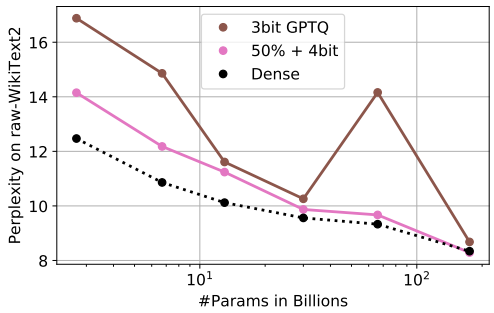

Figure 6. Comparing joint 50% sparsity + 4-bit quantization with size-equivalent 3-bit on the OPT family for $\geq$ 2.7B params.

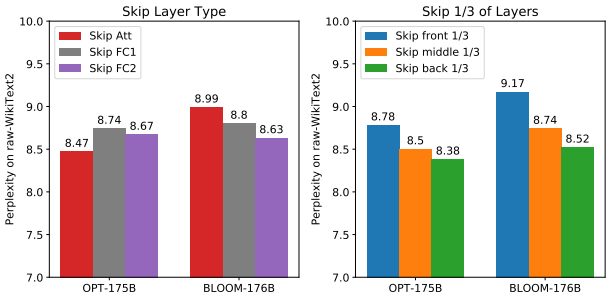

Figure 7. Sensitivity results for partial 2:4 pruning.

(e.g. 1/2, 2/3, 3/4, ...) in *a single pruning pass* by combining the first x layers from a SparseGPT run with the last $n_{\text{layers}} - x$ of the original model. The accuracy of such model sequences are shown in Appendix D.

5. Related Work

Pruning Methods. To our knowledge, we are the first to investigate pruning of massive GPT-scale models, e.g. with more than 10 billion parameters. One justification for this surprising gap is the fact that most existing pruning methods, e.g. (Han et al., 2016; Gale et al., 2019; Kurtic & Alistarh, 2022), 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 (Zhang et al., 2022). 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 (Hubara et al., 2021a; Frantar et al., 2022b; Kwon et al., 2022), which have 100-1000x fewer weights than our models of interest. We discussed the challenges of scaling these methods, and their relationship to SparseGPT, in Section 2.

Post-Training Quantization. By contrast, there has been significant work on post-training methods for *quantizing* open GPT-scale models (Zhang et al., 2022; Scao et al., 2022). Specifically, the ZeroQuant (Yao et al., 2022), LLM.int8() (Dettmers et al., 2022) and nuQmm (Park et al., 2022a) 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. Frantar et al. (2022a) leverage approximate second-order information for accurate quantization of weights down to 2–4 bits, for the very largest models, and show generative batch-size 1 inference speedups of 2-5x when coupled with efficient GPU kernels. Follow-up work (Xiao et al., 2022) investigated joint activation and weight quantization to 8 bits, proposing a smoothing-based scheme which reduces the difficulty of activation quantization and is complemented by efficient GPU kernels. Park et al. (2022b) tackle the hardness of quantizing activation outliers via *quadapters*, learnable parameters whose goal is to scale activations channel-wise, while keeping the other model parameters unchanged. Dettmers & Zettlemoyer (2022) investigate scaling relationships between model size, quantization bits, and different notions of accuracy for massive LLMs, observing high correlations between perplexity scores and aggregated zero-shot accuracy across tasks. 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 (Xiao et al., 2022; Park et al., 2022b).

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 that the largest open-source GPT-family models (e.g. OPT-175B and BLOOM-176B) can reach 50-60% sparsity, dropping more than 100B weights, with low accuracy fluctuations.

In terms of limitations, we focus primarily on uniform per-layer sparsity but non-uniform distributions are a promising topic for future work. Further, `SparseGPT` is currently not quite as accurate on smaller and medium sized variants as on the very largest ones. We think this may be addressable through careful partial or full finetuning, which is starting to become feasible at the scale of such models up to a few billion parameters. Finally, while we study the sparsification of pretrained foundation models in this work, we think investigating how additional post-pretraining techniques like instruction tuning or reinforcement learning with human feedback interact with compressibility will also be an important future research area.

Overall, our work shows that the high degree of parametrization of massive GPT models allows pruning to directly identify sparse accurate models in the "close neighborhood" of the dense model, without gradient information. Remarkably, the output of such sparse models correlates extremely closely with that of the dense model. We also show that *larger models are easier to sparsify*: at a fixed sparsity level, the relative accuracy drop for the larger sparse models narrows as we increase model size, to the point where inducing 50% sparsity results in practically no accuracy decrease on the largest models, which should be seen as very encouraging for future work on compressing such massive models.

7. Acknowledgements

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

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

Boratko, M., Padigela, H., Mikkilineni, D., Yuvraj, P., Das, R., McCallum, A., Chang, M., Fokoue-Nkoutche, A., Kapanipathi, P., Mattei, N., et al. A systematic classification of knowledge, reasoning, and context within the ARC dataset. *arXiv preprint arXiv:1806.00358*, 2018.

Dettmers, T. and Zettlemoyer, L. The case for 4-bit precision: k-bit inference scaling laws. *arXiv preprint arXiv:2212.09720*, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

EleutherAI. EleutherAI LM Evaluation Harness, 2022. URL https://github.com/EleutherAI/lm-evaluation-harness.

Elsen, E., Dukhan, M., Gale, T., and Simonyan, K. Fast sparse convnets. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning (ICML)*, 2020.

Frantar, E. and Alistarh, D. SPDY: Accurate pruning with speedup guarantees. *arXiv preprint arXiv:2201.13096*, 2022.

Frantar, E., Kurtic, E., and Alistarh, D. M-FAC: Efficient matrix-free approximations of second-order information. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022a.

Frantar, E., Singh, S. P., and Alistarh, D. Optimal Brain Compression: A framework for accurate post-training quantization and pruning. *arXiv preprint arXiv:2208.11580*, 2022b. Accepted to NeurIPS 2022, to appear.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

Hagiwara, M. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207 – 218, 1994. ISSN 0925-2312. Backpropagation, Part IV.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*, 2018.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

Hubara, I., Chmiel, B., Island, M., Banner, R., Naor, S., and Soudry, D. Accelerated sparse neural training: A provable and efficient method to find N:M transposable masks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021a.

Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., and Soudry, D. Accurate post training quantization with small calibration sets. In *International Conference on Machine Learning (ICML)*, 2021b.

HuggingFace. HuggingFace Perplexity Calculation, 2022. URL https://huggingface.co/docs/transformers/perplexity.

Kingdon, J. *Hypothesising Neural Nets*, pp. 81–106. Springer London, London, 1997. ISBN 978-1-4471-0949-5. doi: 10.1007/978-1-4471-0949-5_5.

Kurtic, E. and Alistarh, D. Gmp*: Well-tuned global magnitude pruning can outperform most bert-pruning methods. *arXiv preprint arXiv:2210.06384*, 2022.

Kurtic, E., Campos, D., Nguyen, T., Frantar, E., Kurtz, M., Fineran, B., Goin, M., and Alistarh, D. The Optimal BERT Surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Nell, B., Shavit, N., and Alistarh, D. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning (ICML)*, 2020.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. *arXiv preprint arXiv:2204.09656*, 2022.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Conference on Neural Information Processing Systems (NeurIPS)*, 1989.

Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., and Gu, S. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations (ICLR)*, 2021.

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *International Conference on Machine Learning (ICML)*, 2021.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

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

Mishra, A., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Mostafazadeh, N., Roth, M., Louis, A., Chambers, N., and Allen, J. Lsdsem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pp. 46–51, 2017.

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, 2020.

NeuralMagic. DeepSparse, 2022. URL https://github.com/neuralmagic/deepsparse.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Park, G., Park, B., Kwon, S. J., Kim, B., Lee, Y., and Lee, D. nuQmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022a.

Park, M., You, J., Nagel, M., and Chang, S. Quadapter: Adapter for gpt-2 quantization. *arXiv preprint arXiv:2211.16912*, 2022b.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Peste, A., Iofinova, E., Vladu, A., and Alistarh, D. AC/DC: Alternating compressed/decompressed training of deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Sanh, V., Wolf, T., and Rush, A. M. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*, 2020.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Singh, S. P. and Alistarh, D. WoodFisher: Efficient second-order approximation for neural network compression. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Tata, S. and Patel, J. M. PiQA: An algebra for querying protein data sets. In *International Conference on Scientific and Statistical Database Management*, 2003.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Xiao, G., Lin, J., Seznec, M., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*, 2022.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. Learning N:M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

A. Ablation Studies

In this section, we conduct ablations studies with respect to several of the main parameters of `SparseGPT`. For a fast iteration time and making it possible to also explore more compute and memory intensive settings, we focus on the OPT-2.7B model here. Unless stated otherwise, we always prune uniformly to the default 50% sparsity. For brevity we only show raw-WikiText2 results here, but would like to note that the behavior on other datasets is very similar.

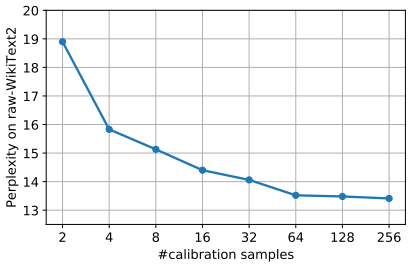

Figure 8. Calibration samples ablation.

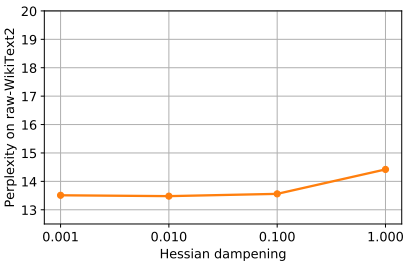

Figure 9. Hessian dampening ablation.

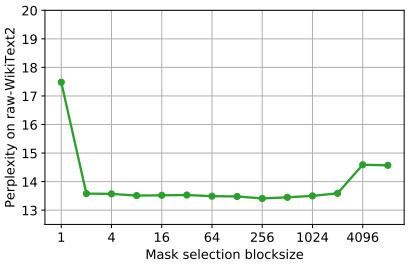

Figure 10. Mask select. blocksize ablation.

Amount of Calibration Data. First, we investigate how the accuracy of `SparseGPT` scales with the number calibration data samples, which we vary in powers of two. The results are shown in Figure 8. Curiously, `SparseGPT` is already able to achieve decent results even with just a few 2048-token segments; using more samples however yields significant further improvements, but only up to a certain point as the curve flattens quite quickly. Thus, since using more samples also increases compute and memory costs, we stick to 128 samples in all our experiments.

Hessian Dampening. Next, we study the impact of Hessian dampening by testing values varying as powers of ten (see Figure 9) which are multiplied by the average diagonal value, following (Frantar et al., 2022a). Overall, this parameter does not seem to be too sensitive, 0.001 to 0.1 appear to perform quite similar; only when the dampening is very high, the solution quality decreases significantly. We choose 1% (i.e. 0.01) dampening to be on the safe side with respect to inverse calculations also for the very largest models.

Mask Selection Blocksize. Another important component of our method is the adaptive mask selection as shown in Figure 10 where we vary the corresponding blocksize parameter with powers of two. Both column-wise (blocksize 1) as well as near full blocksize (4096 and 8192) perform significantly worse than reasonable blocking. Interestingly, a wide range of block-sizes appear to work well, with ones around a few hundred being very slightly more accurate. We thus choose blocksize 128 which lies in that range while also slightly simplifying the algorithm implementation as it matches the default lazy weight update batchsize.

Sensitivity to Random Seeds. Finally, we determine how sensitive the results of our algorithm are with respect to randomness; specifically, relative to the random sampling of the calibration data. We repeat a standard 50% pruning run 5 times with different random seeds for data sampling and get 13.52 ± 0.075 (mean/std) suggesting that `SparseGPT` is quite robust to the precise calibration data being used, which is in line with the observations in other post-training works (Nagel et al., 2020; Hubara et al., 2021b; Frantar et al., 2022b).

A.1. Approximation Quality

In this section we investigate how much is lost by the partial-update approximation employed by `SparseGPT`, relative to (much more expensive) exact reconstruction. We again consider the OPT-2.7B model at 50% sparsity and plot the layer-wise squared error of `SparseGPT` relative to the error of exact reconstruction (with the same mask and Hessian) for the first half of the model in Figure 11. Apart from some outliers in form of the early attention out-projection layers, the final reconstruction errors of `SparseGPT` seem to be on average only around 20% worse than exact reconstruction; on the later fully-connected-2 layers, the approximation error even gets close to only 10%, presumably because these layers have a very large number of total inputs and thus losses by considering only correlations within subsets are less severe than on smaller layers. Overall, these results suggest that, despite its dramatic speedup, `SparseGPT` also remains quite accurate.

B. Evaluation Details

Perplexity. As mentioned in the main text, our perplexity calculation is carried out in standard fashion, following exactly the description of (HuggingFace, 2022). Concretely, that means we concatenate all samples in the test/validation dataset,

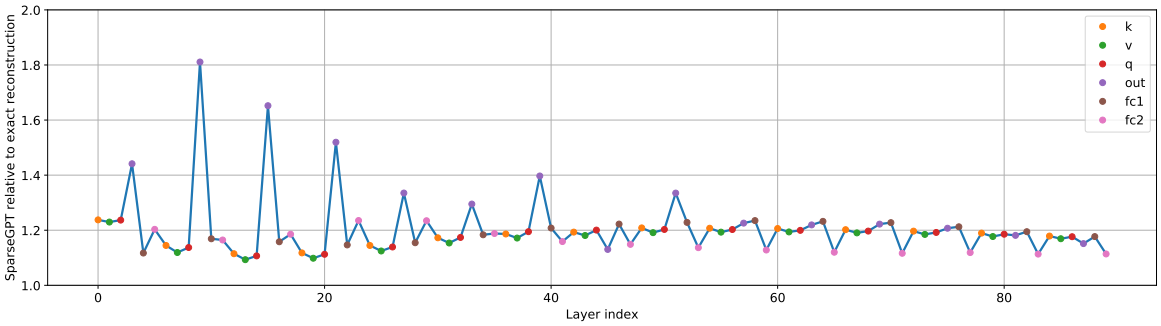

Figure 11. Error of `SparseGPT` reconstruction relative to exact reconstruction for the first half of OPT-2.7B at 50% sparsity.

encode the result with the model's matching tokenizer and then split it into non-overlapping segments of 2048 tokens (the maximum history of the models we study). Those are run through the model to calculate the corresponding average language modelling loss. The exponentiated number is the perplexity we report.

Datasets. In terms of datasets, we use the raw version of the WikiText2 test-set and concatenate samples, as recommended by the HuggingFace description referenced above, with "\n\n" to produce properly formatted markdown. For PTB, we use the test-set of HuggingFace's "ptb_text_only" version and concatenate samples directly, without separators, as PTB is not supposed to contain any punctuation. Our C4 subset consists of the starting (the dataset comes in random order) 256 times 2048 encoded tokens in the first shard of the directly concatenated validation set; this choice is made to keep evaluation costs manageable.

C. Additional Results

Pruning Difficulty Scaling on PTB & C4. Tables 3 and 4 present the equivalent results to Table 1 in the main text, but on PTB and our C4 subset, respectively. Overall, they follow very similar trends to those discussed in Section 4.1. The main notable difference is that no slight perplexity decrease relative to the dense baseline is observed at 50% sparsity for the largest models, hence we have labelled this as a dataset specific phenomenon.

Table 3. OPT perplexity results on PTB.

OPT - 50%	125M	350M	1.3B
Dense	38.99	31.07	20.29
Magnitude	276.	126.	3.1e3
AdaPrune	92.14	64.64	41.60
`SparseGPT`	**55.06**	**43.80**	**25.80**

OPT	Sparsity	2.7B	6.7B	13B	30B	66B	175B
Dense	0%	17.97	15.77	14.52	14.04	13.36	12.01
Magnitude	50%	262.	613.	1.8e4	221.	4.0e3	2.3e3
`SparseGPT`	50%	**20.45**	**17.44**	**15.97**	**14.98**	**14.15**	**12.37**
`SparseGPT`	4:8	23.02	18.84	17.23	15.68	14.68	12.78
`SparseGPT`	2:4	26.88	21.57	18.71	16.62	15.41	13.24

Table 4. OPT perplexity results on a C4 subset.

OPT - 50%	125M	350M	1.3B
Dense	26.56	22.59	16.07
Magnitude	141.	77.04	403.
AdaPrune	48.84	39.15	28.56
`SparseGPT`	**33.42**	**29.18**	**19.36**

OPT	Sparsity	2.7B	6.7B	13B	30B	66B	175B
Dense	0%	14.32	12.71	12.06	11.45	10.99	10.13
Magnitude	50%	63.43	334.	1.1e4	98.49	2.9e3	1.7e3
`SparseGPT`	50%	**15.78**	**13.73**	**12.97**	**11.97**	**11.41**	**10.36**
`SparseGPT`	4:8	17.21	14.77	13.76	12.48	11.77	10.61
`SparseGPT`	2:4	19.36	16.40	14.85	13.17	12.25	10.92

50% Sparse + 3-bit. The main paper only presents near loss-less results for 50% + 4-bit joint sparsification and quantization, corresponding to 3-bit quantization in terms of storage. For 50% + 3-bit (corresponding to 2.5-bit), OPT-175B achieves 8.60 PPL on raw-WikiText2, which is also more accurate than GPTQ's (Frantar et al., 2022a) 8.94 state-of-the-art 2.5-bit result. `SparseGPT` scores the same 8.93 for 4:8 + 3-bit. Based on these initial investigations, we believe that combining sparsity + quantization is a promising direction towards even more extreme compression of very large language models.

D. Partial 2:4 Results

Tables 5 and 6 show the performance of a sequence of partially 2:4 sparse models on three different language modelling datasets. The first fraction of layers is fully sparsified while the remainder is kept dense. In this way, speedup and accuracy can be traded off also from binary compression choices, such as n:m-pruning.

Table 5. Pruning different fractions (as consecutive segments from the beginning) of OPT-175B layers to the 2:4 pattern.

OPT-175B – 2:4	dense	1/2	2/3	3/4	4/5	full
raw-WikiText2	8.34	8.22	8.38	8.49	8.52	8.74
PTB	12.01	12.15	12.80	13.02	13.12	13.25
C4-subset	10.13	10.22	10.41	10.52	10.59	10.92

Table 6. Pruning different fractions (as consecutive segments from the beginning) of BLOOM-176B layers to the 2:4 pattern.

BLOOM-176B – 2:4	dense	1/2	2/3	3/4	4/5	full
raw-WikiText2	8.11	8.20	8.50	8.67	8.74	9.20
PTB	14.58	14.78	15.44	15.84	15.96	16.42
C4-subset	11.71	11.81	12.06	12.23	12.32	12.67

E. Sparsity Acceleration

Lastly, we perform a preliminary study of how well sparse language models can already be accelerated in practice with off-the-shelf tools, for both CPU and GPU inference. We think that these results can likely be improved significantly with more model specific optimization, which we think is an important topic for future work.

CPU Speedups. First, we investigate acceleration of *unstructured* sparsity for CPU inference. For that we utilize the state-of-the-art DeepSparse engine (NeuralMagic, 2022) and run end-to-end inference on OPT-2.7B (support for larger variants appears to be still under development) for a single batch of 400 tokens, on an Intel(R) Core(TM) i9-7980XE CPU @ 2.60GHz using 18 cores. Table 7 shows the end-to-end speedups of running sparse models over the dense one, executed in the same engine/environment. (For reference, dense DeepSparse is $1.5\times$ faster than the standard ONNXRuntime.) The achieved speedups are close to the theoretical optimum, which suggests that unstructured sparsity acceleration for LLM inference on CPUs is already quite practical.

Sparsity	40%	50%	60%
Speedup	$1.57\times$	$1.82\times$	$2.16\times$

Table 7. Speedup over dense version when running sparsified OPT-2.7 models in DeepSparse.

GPU Speedups. 2:4 sparsity as supported by NVIDIA GPUs of generation Ampere and newer theoretically offers $2\times$ acceleration of matrix multiplications. We now evaluate how big those speedups are in practice for the matmul problem sizes that occur in our specific models of interest. We use NVIDIA's official CUTLASS library (selecting the optimal kernel configuration returned by the corresponding profiler) and compare against the highly optimized dense cuBLAS numbers (also used by PyTorch). We assume a batch-size of 2048 tokens and benchmark the three matrix shapes that occur in OPT-175B; the results are shown in Table 8. We measure very respectable speedups through 2:4 sparsity between $54 - 79\%$, for individual layers (end-to-end speedups will likely be slightly lower due to some extra overheads from e.g. attention).

Weight	Q/K/V/Out	FC1	FC2
Dense	2.84ms	10.26ms	10.23ms
2:4 Sparse	1.59ms	6.15ms	6.64ms
Speedup	$1.79\times$	$1.67\times$	$1.54\times$

Table 8. Runtime and speedup for the different layer shapes occuring in OPT-175B using 2048 tokens.

F. Column Sparsity

In Section 3.1 we discussed, in the context of the row-Hessian challenge, how pruning entire columns is significantly more challenging than pruning individual weights, which we now demonstrate in a brief ablation study.

Concretely, we prune entire columns in all linear layers except for FC1 (as dropping a column in FC2 automatically removes a row in FC1 as well). For column mask selection, we use a modified version of `SparseGPT` which sums across rows (to also benefit from intermediate updates), followed by a complete reconstruction using all remaining columns, which is feasible in this structured setting. Table 9 summarizes our raw-WikiText2 results for several OPT models.

OPT	2.7B	6.7B	13B
25% cols	19.62	18.28	15.56
50% cols	70.12	72.90	57.55
50% unstr	13.48	11.55	11.17
50% 2:4	17.18	14.20	12.96

Table 9. Comparison of column sparsity (imposed via `SparseGPT`-like algorithm) and `SparseGPT` unstructured / 2:4; measuring perplexity on raw-Wikitext2.

As can be seen, 50% (cols) models are far from being useful and even only 25% column pruning is still significantly outperformed by full 2:4 pruning. Nevertheless, we think that one-shot structured pruning for such models is worth investigating further, e.g., in low amounts combined with unstructured/semi-structured sparsity.