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

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

**Large-scale generative pretrained transformer (GPT) family models can be pruned to at least 50% sparsity in one-shot at minimal loss of accuracy.**

郑沥杨

2023/12/12

# Introduction

- The best-performing pruning methods so far require extensive retraining of the model to recover accuracy.
- SparseGPT is efficient enough to execute in a few hours on the largest openly-available GPT models (175B parameters), on a single GPU.
- SparseGPT is accurate enough to drop negligible accuracy post-pruning, without any fine-tuning.

# Introduction

- SparseGPT can induce uniform layerwise sparsity of up to 60%

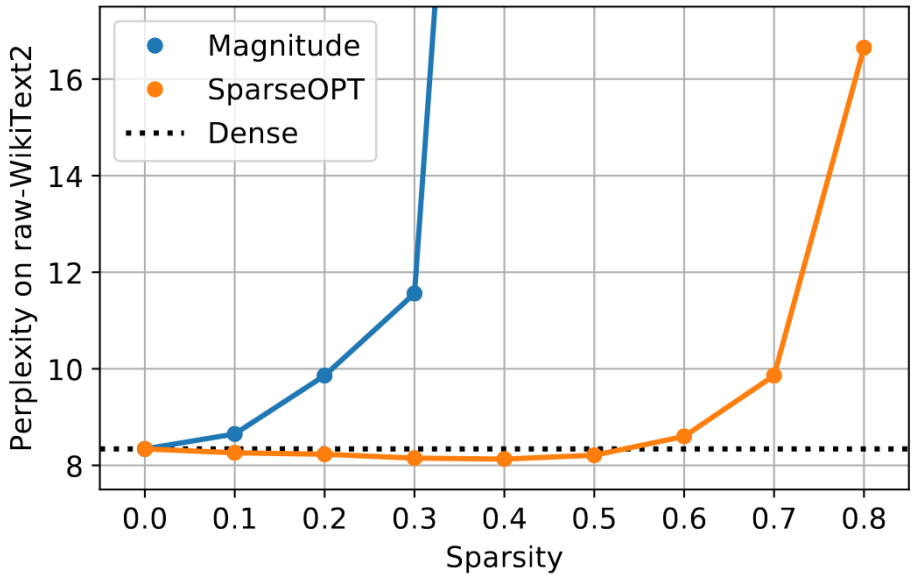



*Figure 1.* Sparsity-vs-perplexity comparison of `SparseGPT` against magnitude pruning on OPT-175B, when pruning to different *uniform* per-layer sparsities.

# Introduction

- SparseGPT can also accurately impose sparsity in the more stringent, but hardware-friendly, **2:4 and 4:8 semi-structured sparsity patterns**, although this comes at an accuracy loss relative to the dense baseline for smaller models.

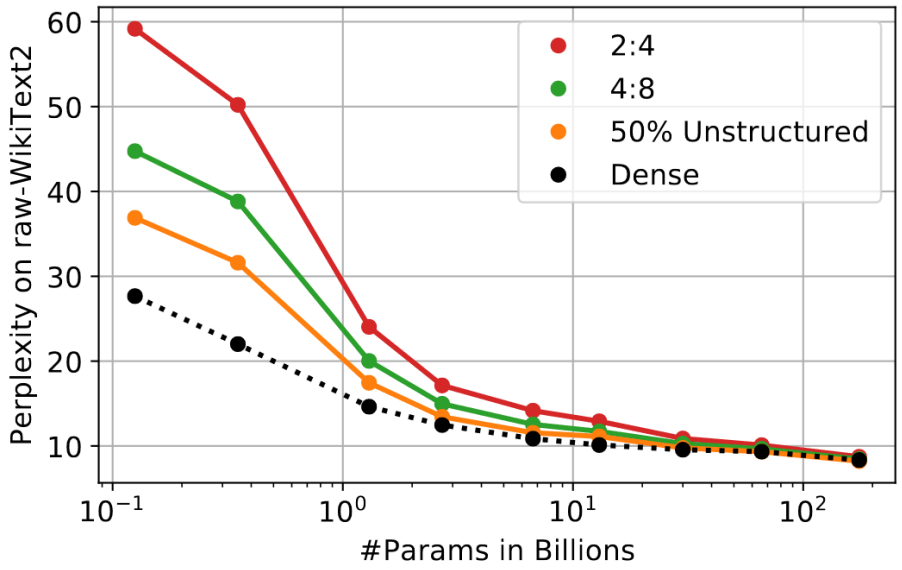



*Figure 2.* Perplexity vs. model and sparsity type when compressing the entire OPT model family (135M, 350M, ..., 66B, 175B) to different sparsity patterns using `SparseGPT`.

# Background

- **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 $\ell_2 - \boldsymbol{error}$ between the output, for given **inputs $\boldsymbol{X}_\ell$**, of the **uncompressed layer with weights $\boldsymbol{W}_\ell$** and that of the compressed one.
  - The overall compressed model is then obtained by "stitching together" the individually compressed layers.

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

NP-Hard Optimization problem

# Background

- **Mask Selection & Weight Reconstruction**
  - A particularly popular approach is to separate the problem into mask selection and weight reconstruction
  - First choose a pruning mask $M$ according to some saliency criterion, like the weight magnitude, 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 have difficulty of scaling to 100+ billion parameters.**

# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Motivation**

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

    - The optimal values of all weights in the mask can be calculated exactly by solving the sparse reconstruction problem
    - This requires inverting the Hessian matrix $\mathbf{H}_{\mathbf{M_i}} = \mathbf{X}_{\mathbf{M_i}}\mathbf{X}_{\mathbf{M_i}}^{\top}$

    - For a Transformer model, this means that the overall runtime is $O(d_{hidden}^4)$
    - We need a speedup by at least a full factor of $d_{hidden}$ to arrive at a practical algorithm.

# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Different Row-Hessian Challenge**

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

  - Solving each row requires the individual inversion of a $O(d_{col} \times d_{col})$ matrix.
  - 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.

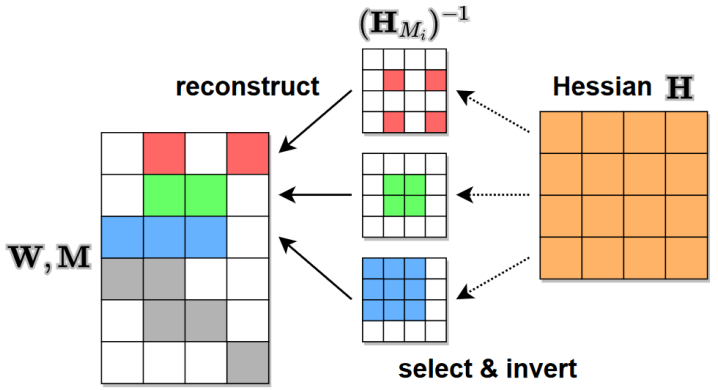



*Figure 3.* Illustration of the row-Hessian challenge: rows are sparsified independently, pruned weights are in white.

# The SparseGPT Algorithm

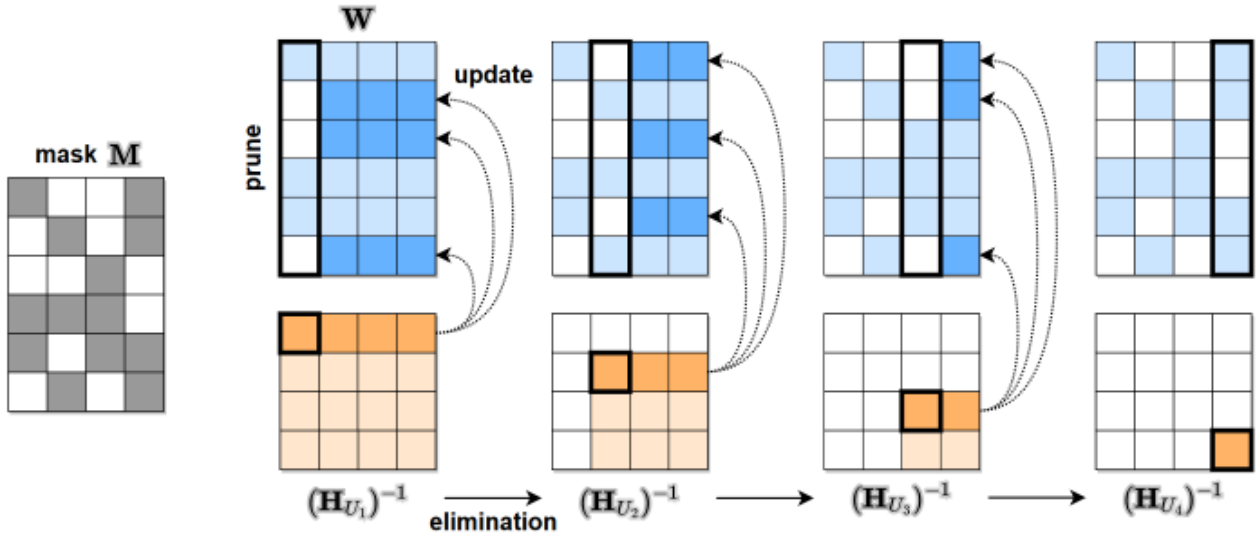
- **Fast Approximate Reconstruction**
  - **Equivalent Iterative Perspective**
    - Row-wise weight reconstruction from a different iterative perspective, using the classic **OBS(Optimal Brain Surgeon)** update.
      - Assuming the current weights $w$ are optimal, the OBS update $\delta_m$ provides the optimal adjustment of the remaining weights to compensate for the removal of the weight at index $m$, incurring error $\varepsilon_m$:

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

Hence, $w + \delta_m$ is the optimal weight reconstruction corresponding to mask $\{m\}^C$.

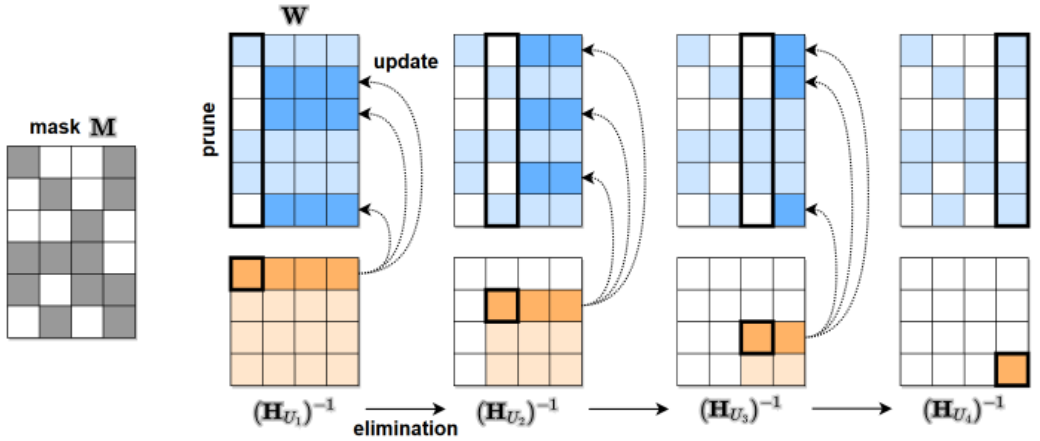
# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Equivalent Iterative Perspective**
    - For solving for a full mask $\mathrm{M} = \{m_1, \ldots, m_p\}^c$, we could iteratively apply OBS to individually prune the weights $m_1$ up until , $m_p$ in order, one-at-a-time, and will ultimately arrive at the same optimal solution as applying the closed-form regression reconstruction with the full M directly.



# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Optimal Partial Updates**
    - Only update the weights in a subset $U \subseteq M$ among remaining unpruned weights
    - The restriction to $U$ 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 $|U| < |M|$, then inverting $H_U$ will be a lot faster than inverting $H_M$.



# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Hessian Synchronization**
    - Asuming a fixed ordering of the input features $j = 1, \ldots, d_{col}$
    - Define a sequence of $d_{col}$ index subsets $U_j$ recursively as:

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

    - 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 $W$.

# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Hessian Synchronization**
    - 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 $H$, from $\mathbf{B} = (\mathbf{H}_{U_j})^{-1}$ in $O(\mathrm{d}_{\mathrm{col}}^2)$ time via one step of Gaussian elimination:

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

$$(\mathbf{H}_{U_1})^{-1} = \mathbf{H}^{-1}$$

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

# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Hessian Synchronization**

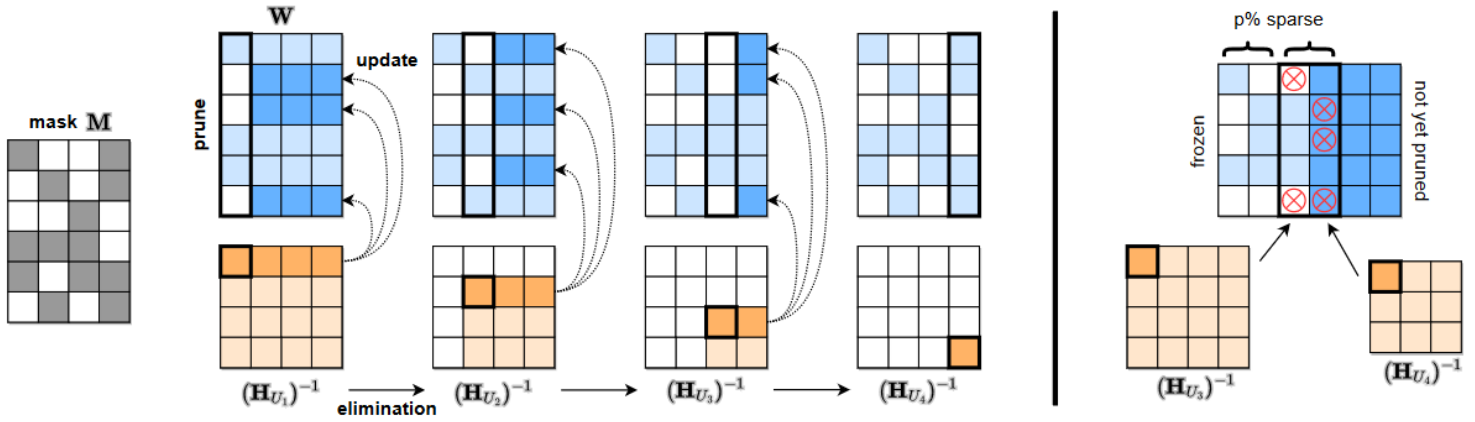


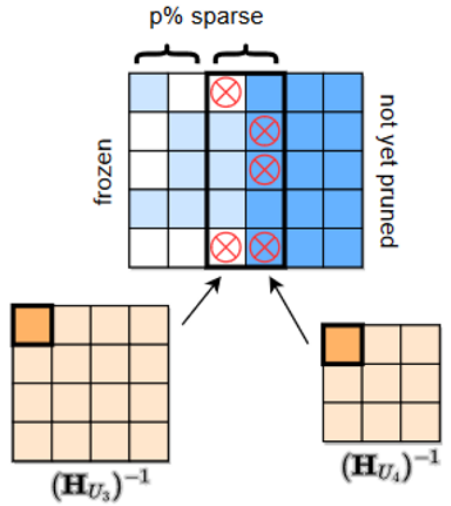

*Figure 4.* [Left] Visualization of the `SparseGPT` reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights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

# The SparseGPT Algorithm

- **Fast Approximate Reconstruction**
  - **Computational Complexity**
    a) The initial Hessian, which takes time $\Theta(n \cdot d_{col}^2)$ where $n$ is the number of input samples used
    b) Iterating through the inverse Hessian sequence in time $O(d_{col}^3)$
    c) The reconstruction/pruning itself $O(d_{col} d_{row} d_{col})$
  - In total, $O\left(d_{col}^3 + d_{row} d_{col}^2\right)$
  - For Transformer models $O(d_{hidden}^3)$

# The SparseGPT Algorithm

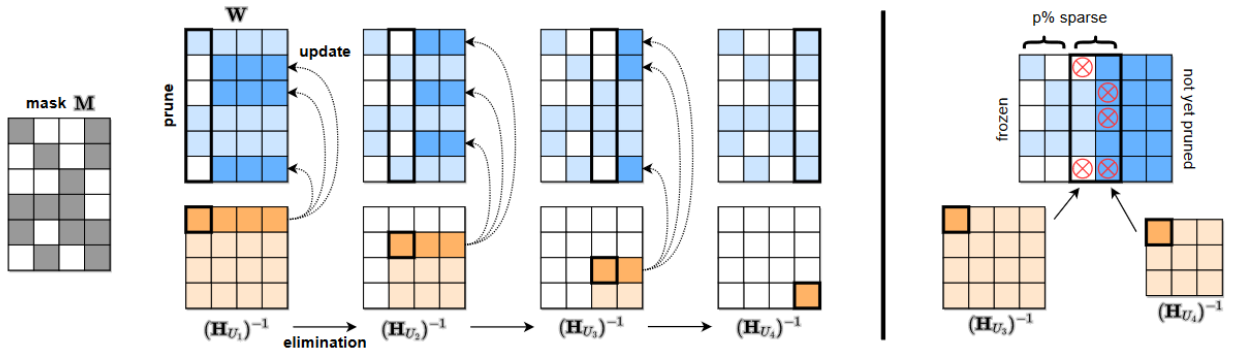
- **Adaptive Mask Selection**
  - So far, we have assumed a fixed pruning mask M.
  - Recent work (Frantar et al., 2022b) shows that updates during pruning change weights significantly due to correlations, and that taking this into account in the mask selection yields better results.
  - Adaptively choosing the mask while running the reconstruction.
  - Select the pruning mask for $\mathrm{B}_s = 128$ columns at a time.
  - Based on the OBS reconstruction error $\varepsilon$ from Equation (3), using the diagonal values in our Hessian sequence. We then perform the next $\mathrm{B}_s$ weight updates, before selecting the mask for the next block, and so on.
- **Extended to Semi-Structured Sparsity**



# The SparseGPT Algorithm

**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 block-size $B$ 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 each block
  **for** $j = i, \ldots, i + B - 1$ **do** — Inside the block, for each column
    **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 $w_c^2/[\mathbf{H}^{-1}]_{cc}^2$ → Select the mask for current block according to the error information
    **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* → Only updates the weights in some specific rows
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update* → $w + \delta_m$ is the optimal weight reconstruction corresponding to mask $\{m\}^C$
  **end for**
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update* → Perform the next $B_s$ weight updates, before selecting the mask for the next block.
**end for**
$\mathbf{W} \leftarrow \mathbf{W} \cdot \mathbf{M}$ *// set pruned weights to 0*




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

# Experiments

- **Setup**
  - Single NVIDIA A100 GPU, 80GB memory
  - Focus on 175-billion-parameter models
  - In approximately 4 hours
  - Compressing Transformer layers sequentially in order, which significantly reduces memory requirements.

- **Models, Datasets & Evaluation**
  - **Models:** OPT family, 176B BLOOM
  - Evaluation on **perplexity**
  - **Test sets:** raw-WikiText2, PTB, C4 validation subset
  - **ZeroShot accuracy benchmarks:** Lambada, ARC, PIQA, StoryCloze

# Results & Discussion

- **Results**
  - **Pruning vs. Model Size**

*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 |

**Larger Models Are More Compressible**

# Results & Discussion

- **Results**
  - **Pruning vs. Model Size**

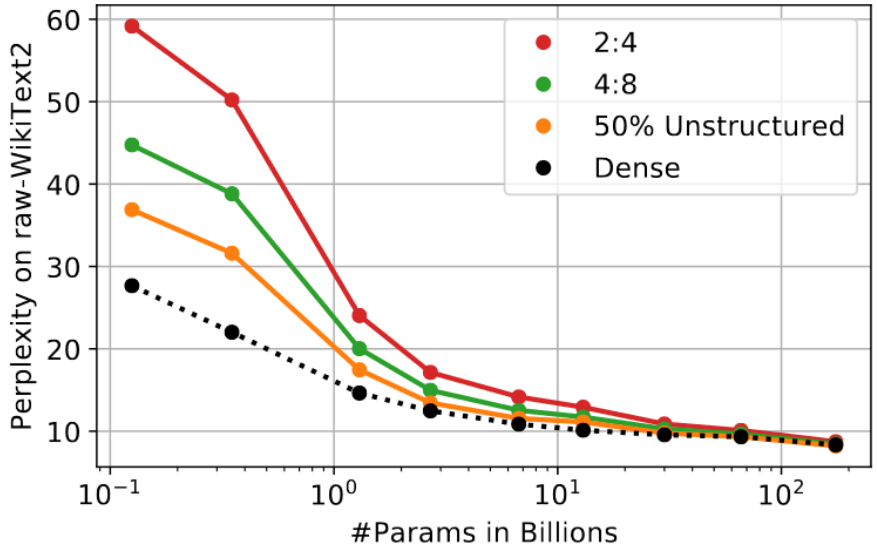



*Figure 2.* Perplexity vs. model and sparsity type when compressing the entire OPT model family (135M, 350M, . . . , 66B, 175B) to different sparsity patterns using `SparseGPT`.

**Larger Models Are More Compressible**

# Results & Discussion

- **Results**
  - **ZeroShot Experiments**

*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** |

SparseGPT models maintained close to original accuracy in ZeroShot tasks, while magnitude-pruned models showed a significant drop in performance. Interestingly, 2:4 pruning sometimes achieved higher accuracy than the dense model.

# Results & Discussion

- **Results**
  - **Joint Sparsification & Quantization**

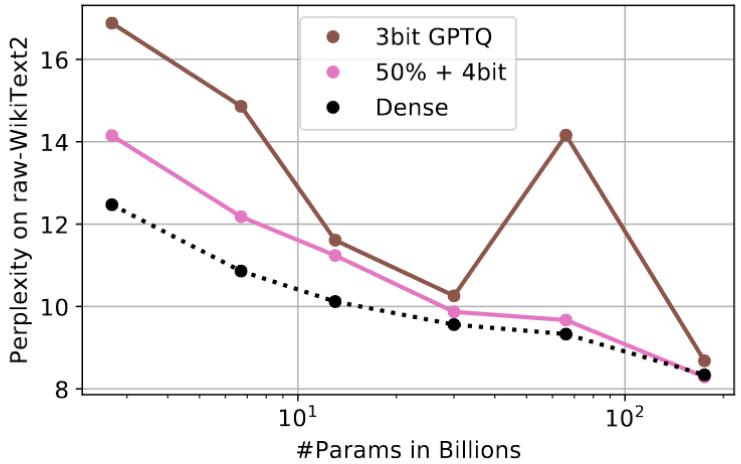



*Figure 6.* Comparing joint 50% sparsity + 4-bit quantization with size-equivalent 3-bit on the OPT family for $\geq$ 2.7B params.

Combining sparsity and quantization, SparseGPT with 50% sparsity + 4-bit quantization outperformed the 3-bit versions for models larger than 2.7B parameters, including the 175B model

# Results & Discussion

- **Results**
  - **Sensitivity & Partial N:M Sparsity**

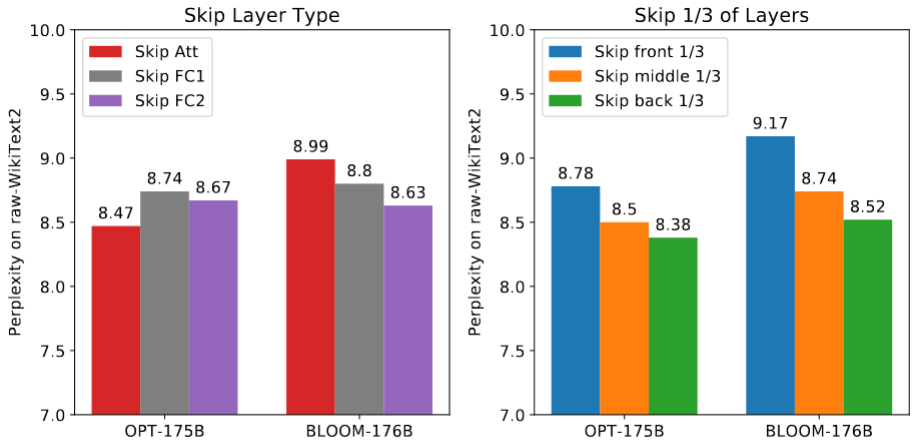



*Figure 7.* Sensitivity results for partial 2:4 pruning.

Full sparsity is not accurate enough and how to choose a subset of layers for n:m sparsification. It was found that skipping the last third of the model layers yields the best accuracy, indicating that later layers are more sensitive than earlier ones.

# Results & Discussion

- **Dicussion：**
  - This paper focus primarily on uniform per-layer sparsity but non-uniform distributions are a promising topic for future work.
  - SparseGPT is currently not quite as accurate on smaller and medium sized variants as on the very largest ones.