Characterization of Unstructured Sparsification of Transformer LMs

ECE 718 - Compiler Design for HPC

December 2, 2025

Tahar HERRI

Outline

https://github.com/TaharHERRI/Edu-Sparsify-LLMs

Recap from Seminar 1

What is Sparsification?

- **Sparsification** refers to making a neural network more efficient by **removing or zeroing out parameters or activations**.
- Objectives:
  - reduce **memory footprint**,
  - reduce **FLOPs per token** and inference cost,
  - reduce **energy consumption**,
  - deploy models on **restricted hardware**.
- Possible pruning targets:
  - individual weights,
  - whole neurons/channels/heads,
  - activations (input-dependent),
  - entire blocks or layers.
- Why for LLMs?
  - Transformers contain **hundreds of millions to billions** of parameters.
  - Redundancy is high, inference cost scales with parameter count.
  - Complements quantization, distillation, and low-rank adaptation.

Four Families of Sparsification

1) Unstructured pruning

- Removes isolated weights (e.g., magnitude pruning, SparseGPT).
- Highest flexibility, minimal accuracy degradation.
- Produces **irregular masks** $\rightarrow$ CSR/CSC needed for acceleration.

2) Structured pruning

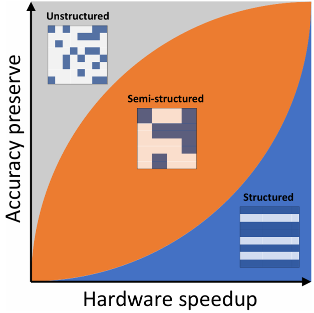
- Removes groups: neurons, channels, heads, layers.
- Produces a **smaller dense model**.
- Hardware-friendly but may hurt accuracy more.

3) Semi-structured pruning (N:M, blocks)

- Patterns such as **2:4**, or 8×8 blocks.
- Supported by NVIDIA Ampere/Hopper through cuSPARSELt.
- Compromise between flexibility and hardware efficiency.

4) Dynamic sparsity

- Sparsity mask changes over time or per input.
- Examples: RigL, SET, Mixture-of-Experts, activation pruning.
- Requires routing or scheduling logic.



Trade-offs of different sparsity schemes in terms of model accuracy and hardware acceleration.

Source: Xu *and al.*, *"LPViT: Low-Power Semi-structured Pruning for Vision Transformers" - arXiv 2025.*

2:4 vs 8×8 Sparsity (Previous Presentation Question)

"Does NVIDIA hardware support 8×8 block sparsity in the same way it supports 2:4 sparsity?"

Short answer: No — they are completely different mechanisms.

2:4 Fine-Grained Structured Sparsity (Ampere)

- Hardware-enforced: **exactly 2 non-zero weights out of 4**.
- Implemented in **Sparse Tensor Cores**.
- Provides up to **2× throughput** for GEMM / attention.

8×8 Block Sparsity

- **Software-level** block compression (BSR/CSR-block).
- Implemented in **cuSPARSE / cuSPARSELt**.
- Does **not** involve hardware Sparse Tensor Cores.
- Performance depends on sparsity level, not on hardware.

Conclusion:

- 2:4 = **hardware constraint**.
- 8×8 = **software kernel optimization**.

Source: NVIDIA Ampere Architecture Whitepaper (2020).

Objectives & Scope of the Project

Hardware and Constraints

- Applying sparsification requires **direct access to all model weights**, which means the entire model must be fully loaded into memory before pruning.
- In practice, this was the main bottleneck:
  - Whether on CPU (limited RAM) or on Google Colab T4 (limited VRAM), it was not possible to load a **sufficiently large model** to obtain meaningful inference results.
  - As a consequence, full text generation or perplexity evaluation could not be used as reliable metrics in this project.
- As a consequence, the project focuses on metrics that remain **independent of successful inference**:
  - structural analysis (per-layer sparsity, theoretical FLOPs, parameter counts),
  - behavioural similarity metrics based only on logits (top-1 agreement instead of text generation).

Objectives & Scope of This Project

- This work focuses **exclusively on unstructured weight pruning**.
  - Compatible with **CPU-only** environments,
  - Does not rely on GPU sparse tensor cores,
  - Allows detailed, layer-wise structural analysis.
- Other sparsification families are included only for context:
  - **Structured**: removes neurons/heads, yields smaller dense models,
  - **Semi-structured**: hardware patterns (e.g., 2:4), GPU-dependent,
  - **Dynamic**: sparsity evolving during training/inference.
- Objective of this project:
  - characterize the **behavioural impact** of pruning,
  - study its **internal structural effects** across model layers,
  - under strict hardware limitations (CPU, limited RAM).

Models and Methodology

Base Models and Variants

Base models used

- **facebook/opt-125m** — main model used on CPU nodes (light enough to load reliably under RAM constraints).
- **EleutherAI/pythia-410m** — used only on GPU (Google Colab T4) for testing larger-scale behaviour.
- Both are autoregressive, decoder-only Transformer LMs.

Three model variants studied

- **Dense**: original unmodified model.
- **Masked 30%**: global magnitude pruning (30% smallest weights set to zero; tensors remain dense).
- **CSR 30%**:
  - same pruning as Masked 30%,
  - selected linear layers replaced by `LinearCSRForward` using **true CSR sparse matrices**.

Source:

- Zhang *and al.*, *"OPT: Open Pre-trained Transformer Language Models" - arXiv 2022.*
- Biderman *and al.*, *"Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling" - arXiv 2023.*

Which Layers Are Pruned?

Pruned layers (global magnitude pruning):

- Linear layers inside **self-attention**:
  Q, K, V projections and output projection.
- Linear layers inside the **MLP/Feed-Forward** block.

Not pruned:

- Token and positional **embeddings**,
- Final **lm_head/linear** projection.

Why?

- Embeddings and output head (lm_head/linear) are often fragile w.r.t. pruning,
- Main computational cost lives in
  Attention + MLP/Feed-Forward
  → pruning there is most meaningful.

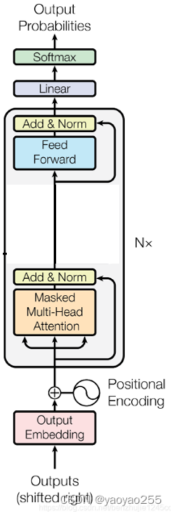


Illustration of a decoder-only Transformer (GPT-style architecture).

Notebook 1: Global Behaviour & Similarity

Global Behaviour Notebook: Goals

Goal: quantify how much pruning changes the model's predictions without relying on full perplexity or generation benchmarks.

- Use a fixed set of evaluation texts (`SAMPLE_TEXTS`).
- Compute logits for:
  - Dense model (reference),
  - Masked 30% model,
  - CSR 30% model.
- Compare predictions token-wise.

Global Metrics (1): Parameter Statistics

For each variant, we compute:

- Total number of parameters T.
- Number of non-zero parameters N.
- **Global sparsity:**

$$\text{sparsity} = 1 - \frac{N}{T}.$$

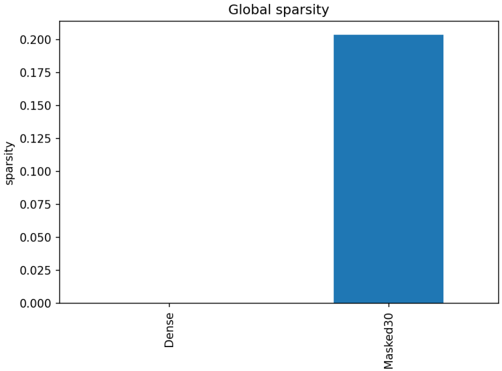
- Estimated model size in MB (based on tensor memory footprint).

- **Dense** and **Masked** share essentially the same size.
- **CSR** can reduce the effective storage for pruned layers.

Global Sparsity (Dense vs Masked)

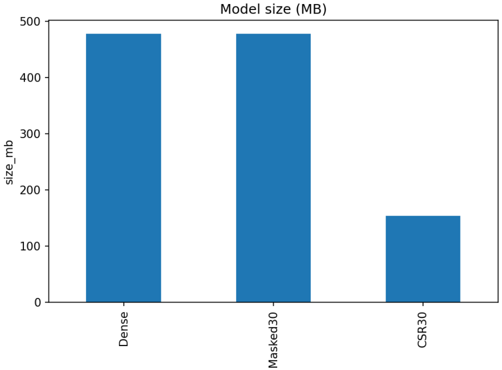
- The **Dense** model has virtually no zero weights.
- The **Masked 30%** variant reaches ∼20% global sparsity, since only attention and MLP linear layers are pruned.

Interpretation:
Global sparsity stays limited because large components (embeddings, LM head) are not pruned, even though the prunable layers do reach 30% sparsity internally.



Model Size After Pruning

- **Dense** and **Masked 30%** have exactly the same size: masking does *not* remove parameters from memory.
- **CSR 30%** is significantly smaller because pruned linear layers are stored using a **compressed sparse format**.
- Embeddings and LM head stay dense, which limits the total reduction.



Interpretation:
Only the CSR variant actually reduces memory footprint, since sparsity becomes "physical" instead of being stored as zeros.

Global Metrics (2): Top-1 Agreement

Definition

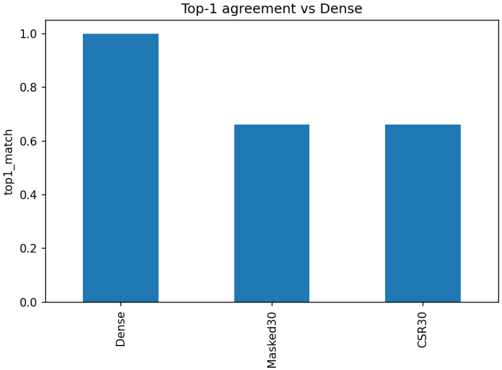
The **top-1 agreement** compares two models by checking, for each token, whether they predict the *same most likely next token*:

$$\text{top1\_match} = \frac{\#\{\arg\max_{\text{dense}} = \arg\max_{\text{variant}}\}}{\#\text{valid token positions}}.$$

- Measures how often the pruned model preserves the dense model's token-level decisions.
- Only compares the **top predicted token** at each position (not the entire probability distribution).
- Uses the same small evaluation corpus for all model variants.
- Values lie between 0 and 1:
    - 1 means identical predictions,
    - lower values mean more differences introduced by pruning.

Top-1 Agreement After Pruning

- **Top-1 agreement** = fraction of tokens where the pruned model keeps the same prediction as the dense one.
- **Masked 30%** and **CSR 30%** both achieve ≈ 0.66.
- Why this value?
  - Pruning 30% of weights in attention and MLP layers changes the internal activations enough to flip the top-1 token in about one third of positions.
  - CSR does **not** change predictions: it only changes the storage and execution format.
- The observed drift comes from **weight removal**, not from CSR conversion.



Global Behaviour: Summary of Results

- **30% pruning** (Masked or CSR) changes the top-1 prediction in roughly **one out of three** token positions. This could be improved with:
  - **larger models** (more redundancy),
  - **smarter pruning methods** (SparseGPT, Wanda),
  - a bit of **post-pruning fine-tuning**.
- The **global sparsity** of the model is *lower than 30%* because several components are **not pruned** (embeddings, `lm_head`, layer norms). Pruning applies only to attention and MLP/Feed-Forward layers.
- **Masked 30%** and **CSR 30%** show **the same behavioural impact**: pruning — not CSR conversion — explains the deviation.
- **CSR 30%** brings a tangible structural benefit:
  - strong reduction in **stored parameters** for pruned linear layers,

Notebook 2: Structural Layer-by-Layer View

Structural Notebook: Goals

Objective: analyse how parameters, sparsity, and compute are distributed across the model.

- **Per-layer inspection**: embeddings, attention projections, MLP/Feed-Forward layers, norms, output head.
- **Quantities computed for each layer**:
  - sparsity ratio,
  - approximate FLOPs per token,
  - share of parameters within the whole model.

Sparsity and Parameter Distribution Across Layers

To understand where pruning has the strongest effect, we measure two high-level quantities for each architectural layer (embeddings, attention, MLP, ...):

- **Layer sparsity**

$$\text{sparsity}_{\text{layer}} = 1 - \frac{\text{nonzero}_{\text{layer}}}{\text{total}_{\text{layer}}}.$$

  Indicates how many weights are removed inside each layer.

- **Parameter fraction**

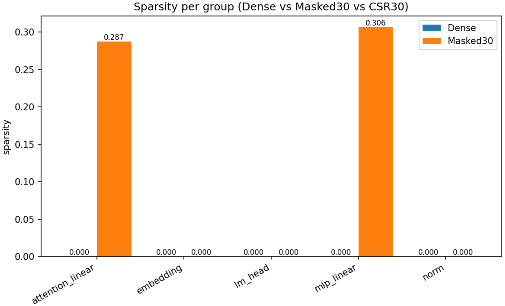
$$\text{param\_frac}_{\text{layer}} = \frac{\text{total}_{\text{layer}}}{\text{total parameters in model}}.$$

  Shows how much of the model's size each layer accounts for.

Together, these metrics reveal where parameters are concentrated and where sparsity actually appears after pruning.

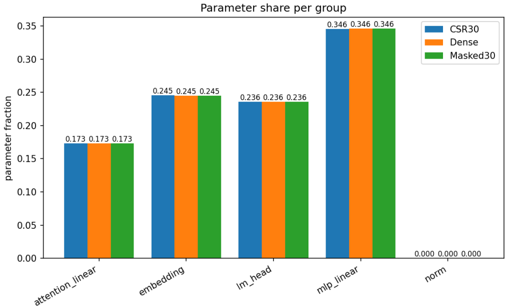
Where Does Sparsity Appear?

- Sparsity is concentrated in the **attention projections** and the **MLP (feed-forward) layers**.
- **Embeddings**, **LM head**, and **norm layers** show **0% sparsity** because they are *never pruned* in our pipeline.



Where Are the Parameters?

- The distribution of parameters is **identical** for Dense, Masked 30%, and CSR 30%: pruning does not change *where* parameters are located.
- Most parameters lie in:
  - the **MLP (feed-forward) layers** (35%),
  - the **embeddings** (25%),
  - the **LM head** (24%).
- Attention projections account for 17% of parameters.
- Norm layers contribute negligibly to the total, hence the fraction appears as zero on the plot.
- This decomposition explains why pruning mainly affects MLP and attention layers: they dominate the model size.



FLOPs Approximation

For a dense linear layer:

$$y = Wx, \quad W \in \mathbb{R}^{\text{out} \times \text{in}}, \ x \in \mathbb{R}^{\text{in}}.$$

- in $\times$ out multiplications,
- in $\times$ out additions,

so we use:

$$\text{FLOPs}_{\text{dense}} \approx 2 \cdot \text{in} \cdot \text{out}.$$

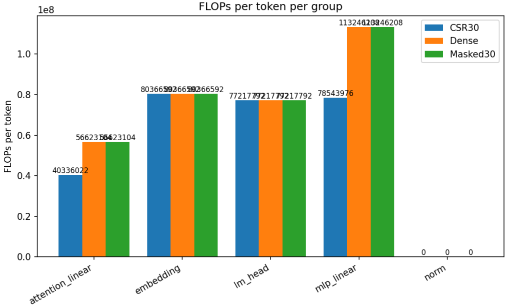
For a sparse matrix with *nnz* non-zero weights:

$$\text{FLOPs}_{\text{sparse}} \approx 2 \cdot nnz.$$

This directly connects **sparsity** and **theoretical compute**.

FLOPs per Token Across Groups

- Per-token compute is spread mainly across:
  - **MLP feed-forward layers** (largest contributor),
  - **Embedding projections**,
  - **LM head**.
- **Dense** and **Masked 30%** have identical FLOPs: masked weights stay in dense matrices $\rightarrow$ dense GEMM is still used.
- **CSR 30%** lowers FLOPs for pruned linear layers (attention + MLP), because computation scales with the number of **non-zero weights**.
- Embeddings, LayerNorm and lm_head have fixed cost and cannot be sparsified in this experiment $\rightarrow$ no FLOPs difference across variants for these groups.



Interpreting the Structural Results

- Embeddings and `lm_head`:
  - remain dense (not pruned),
  - represent a significant fraction of parameters.
- Attention and MLP linear layers:
  - main targets of pruning,
  - carry most of the sparsity,
  - account for a large fraction of FLOPs per token.
- CSR conversion:
  - does not change the *logical* architecture,
  - but changes how weights are stored and accessed,
  - reduces non-zero storage and theoretical compute in pruned layers.

Limitations and Discussion

Limitations

- **Model loading constraints**
  - Limited RAM (CPU) and limited VRAM (Colab T4) prevented loading models large enough for meaningful inference quality.
  - CSR inference also requires CUDA sparse kernels, unavailable on CPU-only nodes.
- **Consequences**
  - No reliable perplexity measurements,
  - No large-scale text generation evaluation,
  - Behaviour analysis restricted to **top-1 agreement** on small CPU/GPU-friendly inputs.
- **Despite this**
  - the study remains **systematic, controlled, and reproducible**,
  - structural metrics and pruning methods are rigorously compared across Dense, Masked, and CSR variants.

What the Study Shows

- **Behavioural impact**
  - At 30% sparsity, pruning alters roughly one-third of top-1 predictions, a result that would likely improve with larger models, more advanced pruning methods, or light post-pruning fine-tuning.
  - Masked and CSR models show **almost identical** behavioural drift: CSR conversion does not introduce additional change.
- **Structural impact**
  - Sparsity appears only in pruned linear layers (here attention projections and MLP feed-forward).
  - Embeddings and the output head stay dense, which limits global sparsity.
  - CSR representation **reduces storage** and **lowers theoretical FLOPs** in the pruned layers.
- **Overall insight**

  Pruning reduces memory/compute while maintaining a moderate level of behavioural stability.

Future Work

- **Sparsity exploration**
  - test higher sparsity levels (50%, 70%, 90%),
  - study the accuracy–sparsity tipping point.
- **Beyond unstructured pruning**
  - evaluate structured and semi-structured sparsity (N:M, 8$\times$8 blocks),
  - compare with dynamic sparsity and MoE-style routing.
- **Actual acceleration**
  - run experiments with GPU-supported sparse kernels (cuSPARSELt, Triton, PyTorch 2.x),
  - measure real speedups vs. theoretical FLOPs reductions.
- **Accuracy recovery**
  - post-pruning fine-tuning,
  - LoRA-based recovery,
  - quantization-aware sparsification.
- **Scaling up**
  - apply the full pipeline to larger models when hardware permits,
  - generate and benchmark optimized kernels (Cholesky, FFT, GEMM) using dense vs. sparsified models.

Questions

Thank you!

Questions?