Sparsifying Large Language Models
Seminar 1

ECE 718 - Compiler Design for HPC

November 4, 2025

Tahar HERRI

Outline

1. What is the sparsification of LLMs?
2. Why is sparsification important?
3. Overview of sparsification methods
4. Demo – Seminar 1

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

What is the sparsification of LLMs?

- **Sparsification** = reducing the number of active elements in a neural network (not only weights), to decrease memory and computation costs.
- This involves **removing or zeroing-out** less important components while striving to preserve model accuracy.
- What can be sparsified?
  - *Weights:* individual connections.
  - *Neurons or attention heads:* functional units.
  - *Layers or blocks:* higher-level architecture elements.
- Two complementary goals:
  - *Structural pruning:* determining which elements to remove.
  - *Sparse execution:* leveraging hardware/software to skip pruned parts and accelerate inference.

***Inspired by:* Han et al., "Learning both Weights and Connections," NeurIPS 2015**

Dense vs Sparse Representations

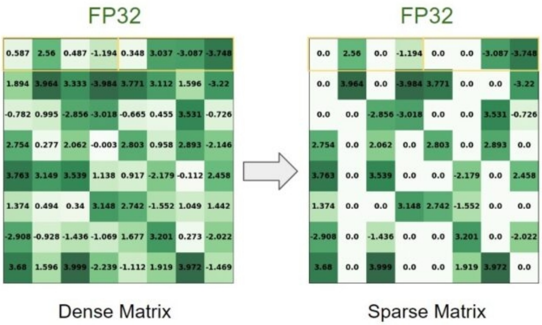

Figure: 1

Training sparsity workflow — NVIDIA (2023)
Source: NVIDIA Developer Blog

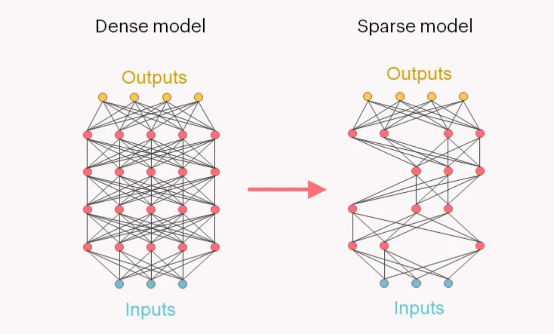

Figure: 2

Sparse LLM example — Graphcore (2021)
Source: Graphcore & Aleph Alpha

Why is sparsification important?

- LLMs have **hundreds of billions of parameters** (GPT-4, Gemini...).
- Inference cost scales roughly linearly with parameter count.
- Sparsification enables:
  - **Lower memory usage**
  - **Faster inference** (fewer FLOPs)
  - **Lower energy cost**
  - **Deployment on edge/limited hardware**
- Complements **quantization, distillation, and LoRA (Low-Rank Adaptation)**.

Refs: Kurtic et al., 2023; Mishra et al., 2021

Overview of Sparsification Methods

How we make large models sparse in practice

Unstructured Pruning

- Removes individual weights (typically smallest-magnitude).
- Very flexible: high accuracy retention.
- Irregular sparsity $\rightarrow$ hard to accelerate without special kernels (CSR, ...).
- **Recent methods:**
  - **SparseGPT (2023):** fast pruning without fine-tuning
  - **Wanda (2023), SPQR (2023):** post-training, quantization-friendly

Refs: Frantar et al., 2023; Su et al., 2023; Dettmers, 2023

Structured Pruning

- Removes **groups** of weights: neurons, channels, heads, or even layers.
- Produces a smaller **dense** model (no special execution format needed).
- Easy to run on CPUs/GPUs — widely supported.
- More aggressive pruning risks degrading accuracy.

Refs: He et al., "Channel Pruning for Accelerating Very Deep Networks," ICCV 2017.

Semi-Structured Pruning (N:M, Block)

- Enforces patterns: e.g., **2:4 sparsity** (2 non-zero out of 4), or **block sparsity** (e.g. 8×8).
- Supported by **NVIDIA Ampere/Hopper (cuSPARSELt)** and **PyTorch 2.x**.
- Trade-off: more regular = easier to accelerate, less flexible than unstructured.

Refs: Mishra et al., "Accelerating Sparse Deep Neural Networks," NVIDIA cuSPARSELt whitepaper, 2023.

Dynamic Sparsity

- Sparsity changes **during training over time or per input**.
- Examples:
  - **RigL, SET**: sparse training with evolving masks
  - **MoE (Mixture-of-Experts)**: only a few experts used per token
  - **Top-k activations**: only keep strongest activations
- Efficient but adds overhead for routing/scheduling.

Refs: Evci et al., "RigL: Efficient Sparse Training," NeurIPS 2020; Lepikhin et al., "GShard," 2021.

Recap of Sparsification Methods

Type	What is removed	How it speeds up	HW support
Unstructured	Individual weights	Sparse kernels (CSR, ...)	Limited
Structured	Neurons, heads, layers	Smaller dense model	Full
Semi-structured	N:M or block patterns	Specialized tensor kernels	High
Dynamic	Input/time-specific parts	Activates fewer units	Partial

Demo – Seminar 1

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

Steps and metrics

1. Load baseline model
   - `facebook/opt-125m` on CPU
   - `EleutherAI/pythia-410m` on GPU
2. Measure baseline: PPL, latency/token, model size.
3. Apply masked Pruning **(30%)** and Freeze then Measure.
4. Apply masked Pruning **(30%)**, Freeze and Convert to CSR Format.
5. Apply masked Pruning **(50%)**, Freeze and Convert to CSR Format.

Metrics

Perplexity, latency per token, model size, sparsity (%).

Expected results (30–70% sparsity)

Setup	Model size	Latency	Perplexity
Dense	100%	100%	ref
Masked 30%	≈100%	100%	≈
CSR 30%	85–95%	95–105%	≈
Masked 50%	≈100%	100%	≈
CSR 50%	60–70%	90–100%	≈
Masked 70%	≈100%	100%	$\approx + \varepsilon$
CSR 70%	40–50%	70–85%	$\approx + \varepsilon$

At low sparsity (30%), speedups are negligible. Real gains appear from ≥50–70% sparsity, where compute and memory reductions start to outweigh CSR overhead. Perplexity degradation (ε) usually stays minor if pruning is magnitude-based.

Demo & Analysis

Thank you and GitHub link

Thank you!

Full code, notebooks, and results:
`https://github.com/TaharHERRI/Edu-Sparsify-LLMs`