# *Deja Vu*: Contextual Sparsity for Efficient LLMS at Inference Time

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., ... & Chen, B. (2023, July). *International Conference on Machine Learning* (pp. 22137-22176).

Presented by Selin Yildirim
in UIUC CS 598 AI Efficiency, Spring 2024

# Outline

- Problem
- Existing Solutions
- Contextual Sparsity
    i. Existence
    ii. Prediction
    iii. Efficiency
- Results
- Conclusion

# Background

Sparsity for achieving the real-world deployment of large scale language models.

Many of the parameters, weights or connections, in the network are zero (or close to zero)

Competitive prediction accuracy at a reduced resource utilization footprint i.e., model size, inference FLOPs, and working memory.

| Unstructured Sparsity | Structured Sparsity |
| --- | --- |
| yields highly sparse and accurate models | enables massive parallelization |
| by pruning out weights based on their importance | enforce sparse structure on |

# Problem

LLM generation latency with memory I/O bottleneck

Earlier on efficient Inference: Quantization, Distillation, Sparsification

- ➢ Challenging to find sparsity that preserves *in-context learning ability*
- ➢ Hard to achieve *wall-clock time speedup* with sparsity due hardware difficulties

→ **Objective:** Speeding up inference-time sparse LLMs in wall-clock time while maintaining quality and in-context learning abilities

# Existing Solutions

- Preserving *in-context learning ability* ?

Effective *task-dependent* pruning (Michel et al., 2019; Bansal et al., 2022), but maintaining different models for each task conflicts with the task independence goal of LLMs

- Achieving *wall-clock time speedup* ?

Recent development in *zero-shot pruning* like SparseGPT (Frantar & Alistarh, 2023) finds 60% unstructured sparsity, but does not lead to any wall-clock time speedup

- Infeasible pruning (sparsification)

*Iterative pruning* (Lee et al., 2018; Frankle & Carbin, 2018) only applies to smaller-scale models.

- Expensive fine-tuning and retraining

# Contextual Sparsity

*Ideal Sparsity* for LLMs,

- ➢ not require model retraining
- ➢ maintains accuracy and in-context learning ability
- ➢ speed-up in wall-clock time on modern hardware

→ *Hypothesis*: Contextual Sparsity

Small, input-dependent sets of attention heads and MLP parameters that lead to nearly the same output as the full model for an input

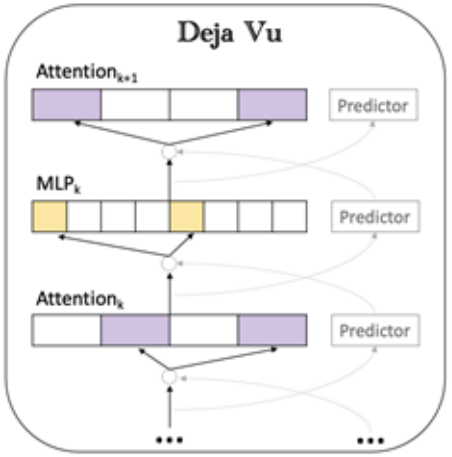



*Figure 2.* DEJAVU uses lookahead predictors to side-step prediction costs: given the input to the attention layer at block $k$, they (asynchronously) predict the contextual sparsity for the MLP at block $k$, and given the input to the MLP at block $k$, they predict the sparsity for the attention head at the next layer.

# Contextual Sparsity : Existence

To verify, 2 forward passes of the model :

- Record attention heads and MLP neurons that yield **large output norms** for the input (because the activation function ReLU or GeLU zeros out low activation)
- Each input example only uses the recorded subset of parameters for the computation (pruning)

$$\mathrm{MLP}_{S_M}(y) = \sigma(yW^1_{S_M})(W^2_{S_M})^\top$$

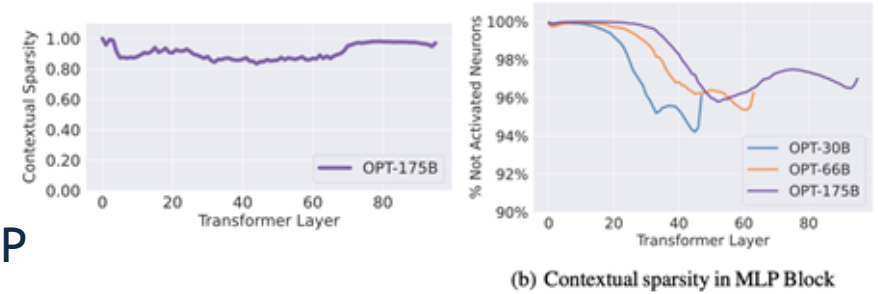
$$\mathrm{MHA}_{S_A}(y) = \sum_{i \in S_A} \underbrace{H_i(y)}_{1 \times d_h} \underbrace{W_i^O}_{d_h \times d},$$

→ Leads to similar prediction on ***all in-context learning and language modeling tasks*** (Widely exists in pre-trained LLMs, e.g., OPT, Llama, GPT…)

# Contextual Sparsity : Existence

## Observation #1

- ➢ results in 85% contextual sparsity
  - On avg. 85% in attention, 95% in MLP

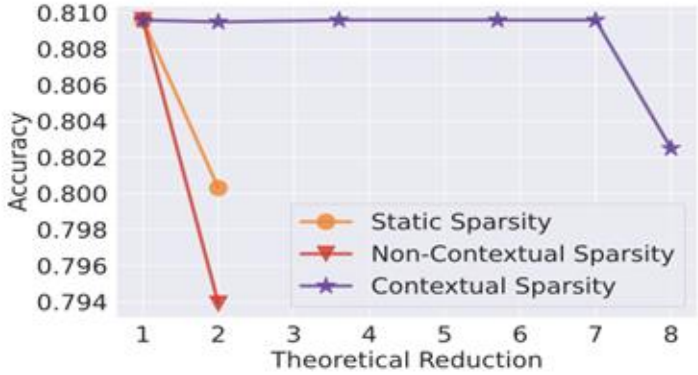



(b) Contextual sparsity in MLP Block

## Observation #2

- ➢ 7× parameter reduction for each input

Contextual sparsity depends not only on individual input tokens (*non-contextual dynamic* sparsity), but also on their interactions (*contextual dynamic* sparsity)



# Contextual Sparsity : Prediction

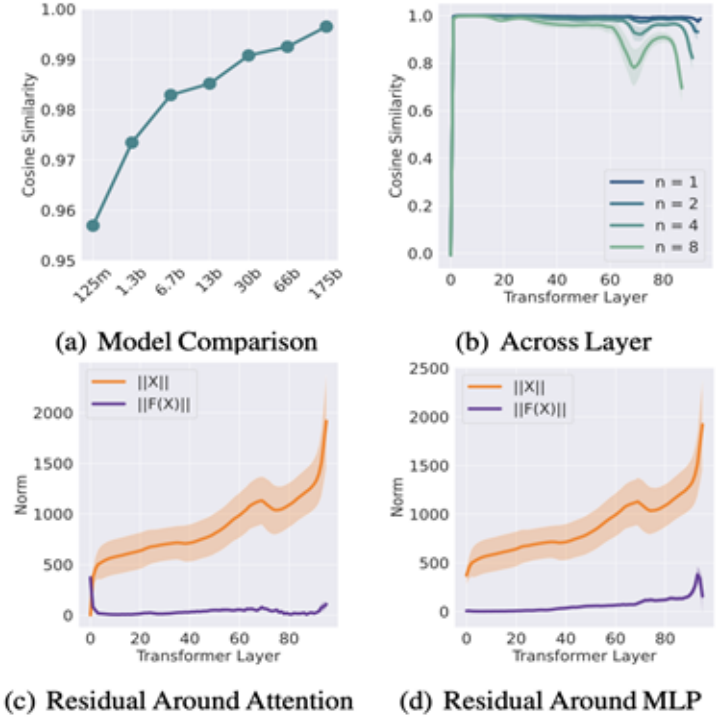
**Evaluation #3**

A large portion of attention head/MLP outputs yield small norm ~ High contextual sparsity

Token embeddings change slowly across layers due to high residual norm, *residual connections*

**Observation #3**

Contextual dynamic sparsity for every layer can be predicted based on the ***similarity between layer parameters (heads/MLP)*** and ***the output from the previous layer***

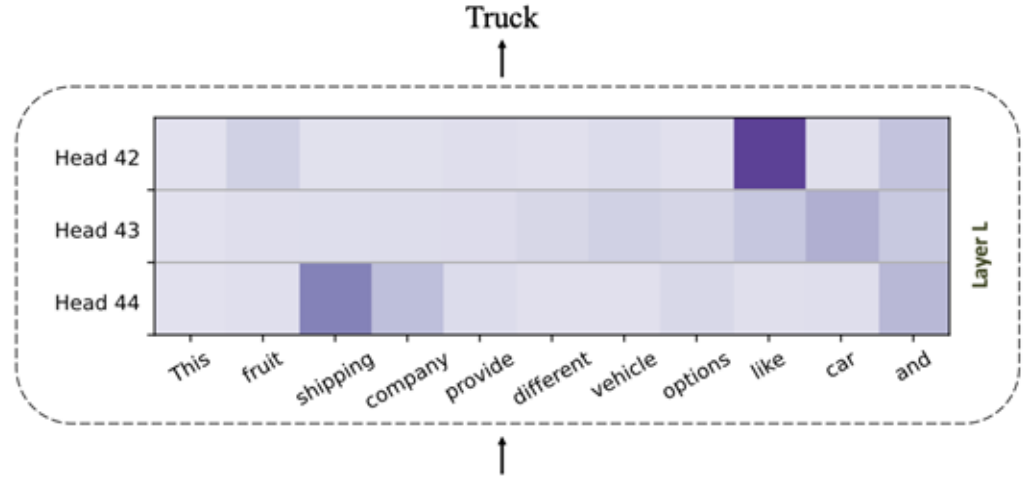


# Contextual Sparsity : Prediction

Different heads learn different projection spaces to perform clustering.

Token embeddings tend to cluster after going through more layers

*Hypothesis:* Self-attention head is regarded as *one **mean-shift step*** to push input embeddings of different tokens together, if they are already neighbors in a projection space by $W_i^Q (W_i^K)^\top$

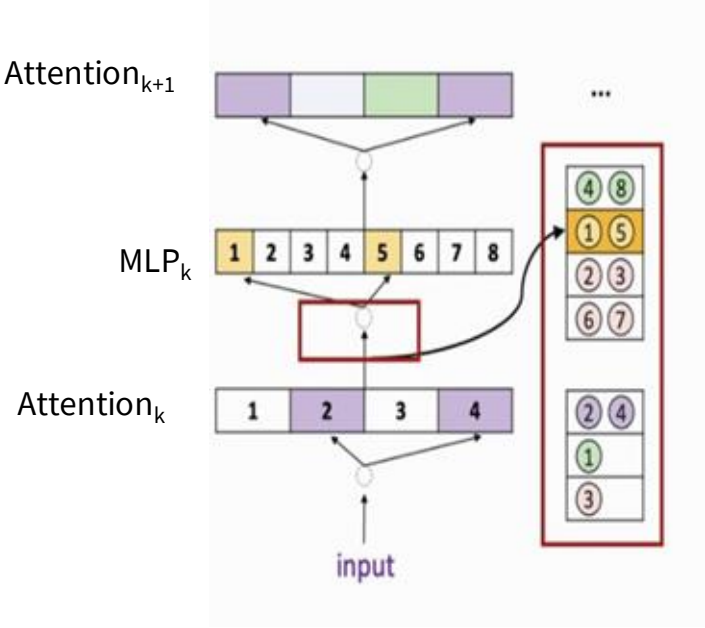


# Contextual Sparsity : Prediction

→ CS Prediction of an MLP layer can be formulated as the *classical nearest-neighbor search* problem, based on similarity between input & parameters,

- Query: input at each layer
- Data: neurons or attention heads

Propose an asynch. lookahead predictor to avoid the sequential overhead, since cross-layer design suffices for accurate sparsity prediction



# Contextual Sparsity : Efficiency (MLP)

One of the major bottlenecks for the LLM generation : 2/3 of the FLOPs and IOs

*Example:* For OPT-175B, one MLP block is only 0.2 ms on 8×A100 80GB

**How efficient would applying nearest-neighbor search problem be?**

High dimensionality and complications of data structure implementation on GPUs make the search time longer than the MLP computation.

→ A low-cost small trainable MLP (two-layer fully connected network) to predict CS on the fly.

# Contextual Sparsity : Efficiency (MLP)

- Given y, the sparsity predictor $SP_M$ predicts set $S_M$ of important neurons in weights $W^1$
- Compute the sparsified MLP defined in

$$\text{MLP}_{S_M}(y) = \sigma(yW^1_{S_M})(W^2_{S_M})^\top$$

Since slowly evolving embedding phenomenon provides opportunities to relax the sequential computation to parallel asynchronously,

$$\widetilde{y}_l \leftarrow \text{MHA}^l_{S^l_A}(y_l), \quad \widehat{y}_l \leftarrow \text{MLP}^l_{S^l_M}(\widetilde{y}_l),$$
$$S^{l+1}_A \leftarrow \text{SP}^l_A(y_l), \quad S^{l+1}_M \leftarrow \text{SP}^l_M(y_l),$$

**Algorithm 1** Sparse Predictor Training

**Input**: A pre-trained LLM block with parameter set $M$, token embedding set at block $M = \{x_i\}_{i\in[N]}$, threshold $t$
**Sparse Predictor** $\mathcal{SP}$
$\mathcal{P}_+ \leftarrow \emptyset, \mathcal{P}_- \leftarrow \emptyset$
**for** $i = 1 \rightarrow N$ **do**
  $\mathcal{P}_+ \leftarrow \mathcal{P}_+ \cup \{(x_i, m_r) \mid m_r \in M, m_r(x_i) \geq t\}$
  $\mathcal{P}_- \leftarrow \mathcal{P}_- \cup \{(x_i, m_r) \mid m_r \in M, m_r(x_i) < t\}$
**end for**
$\mathcal{SP} \leftarrow \text{TRAIN}(\mathcal{P}_+, \mathcal{P}_-, \mathcal{L})$ ▷ $\mathcal{L}$ is a loss function

→ Overall latency now also includes prediction latency. CS for Attention and MLP blocks pays off.

# Contextual Sparsity : Efficiency (Attention)

Similar to the MLP blocks, a fast selection of attention heads without full computation

Only a few heads perform important computations for a given input token.

**Challenges:**

- ➢ It is unclear whether the past token's key and value caches are needed for sparse prediction.
- ➢ It is unclear how to handle the missing KV cache of past tokens for the current token computation at the selected head.

# Contextual Sparsity : Efficiency (Attention)

**Method:**

- For the predicted attention head of input y, compute the corresponding keys & values to store them in the KV cache. Also save a copy of y for all the other non-selected heads.
- In the future token generation, if there is missing KV cache in the selected heads, load stored token embeddings and recompute the keys and values together. (This requires almost minimal extra memory access).

# Implementation

**Kernel Fusion:**

For SpMV, fuse the indexing and the multiplication step. (e.g., load a subset of W1 to memory, along with y, perform the multiply, $S_M$ then write down the result.)

→ 4x improvement over PyTorch's SpMV with 3x memory I/O

**Memory Coalescing :**

- When loading $W^2_{SM}$, indices in $S_M$ point to non-contiguous memory. These are stored in column-major format.
- In attention blocks, attention output projection $W_O$ is stored in column-major.

→ These two techniques make DejaVu hardware-efficient, yielding up to 2× speedup in end-to-end time compared to *FasterTransformer*

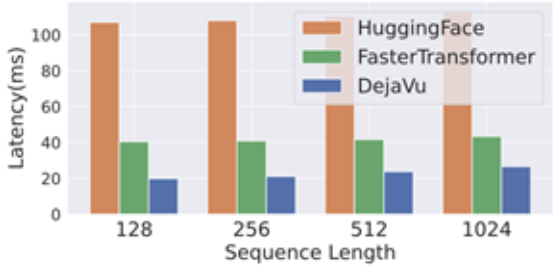



*Figure 7.* Average per-token latency (ms) with batch size 1 on 8 A100-80GB with NVLink when generating sequences with prompt lengths 128, 256, 512, and 1024, using FP16. DEJAVU speeds up generation by 1.8-2× compared to the state-of-the-art FT and by 4.8-6× compared to the widely used HF implementation.

# Results

**Ablation on MLP:**

- ➢ MLP sparse predictor introduces no accuracy loss.
- ➢ In its training, it achieves high validation accuracy.

**Ablation on Attention:**

- ➢ Attention sparse predictor introduces no accuracy loss at around 50% sparsity.
- ➢ During its training, the validation accuracy is around 93% in the middle layers and near 99% in the shallow and deep layers.

*Table 4.* Accuracy of zero-shot tasks and language modeling when sparsifying the MLP block and the Attention block separately. The sparsity is set at 85% for MLP-block and 50% for Attention-block. DEJAVU incurs no accuracy drop across the boards.

| Model | CB | COPA | Lambada | OpenBookQA | PIQA | RTE | Winogrande | Wikitext | C4 |
|---|---|---|---|---|---|---|---|---|---|
| OPT-175B | 0.3523 | 0.86 | 0.7584 | 0.446 | 0.8096 | 0.6029 | 0.7261 | 10.8221 | 7.7224 |
| DEJAVU-MLP-OPT-175B | 0.3544 | 0.85 | 0.7619 | 0.446 | 0.8096 | 0.6065 | 0.7206 | 10.7988 | 7.7393 |
| DEJAVU-Attention-OPT-175B | 0.3544 | 0.86 | 0.7586 | 0.4460 | 0.8063 | 0.5921 | 0.7245 | 10.8696 | 7.7393 |

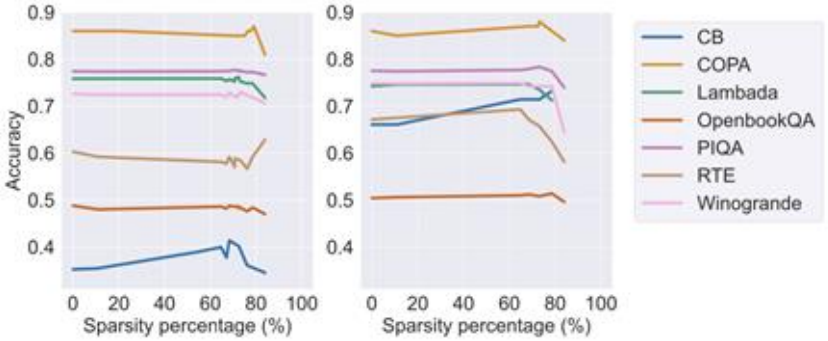
# Results - DejaVu OPT-175B

**Accuracy:** the average accuracy across tasks does not drop until 75% sparsity.

→ Verifies the model's ability for in-context learning.

**Added Quantization:** Combination achieves better accuracy than DejaVu or quantization only.

→ This suggests that the approximation errors from these two directions do not get compounded.




(b) Zero-Shot(Left). Five-Shot(Right)

*Table 7.* DEJAVU-OPT-175B with 4-bit quantization.

| | CB | COPA | OpenBookQA | PIQA | RTE | Winogrande | Lambada |
|---|---|---|---|---|---|---|---|
| OPT-175B | 0.352 | 0.86 | 0.446 | 0.809 | 0.602 | 0.726 | 0.758 |
| Dejavu-OPT-175B | 0.402 | 0.85 | 0.450 | 0.802 | 0.592 | 0.726 | 0.753 |
| OPT-175B + W4A16 | 0.356 | 0.85 | 0.44 | 0.806 | 0.574 | 0.714 | 0.757 |
| Dejavu-OPT-175B + W4A16 | 0.365 | 0.86 | 0.452 | 0.805 | 0.592 | 0.726 | 0.754 |

# Conclusion

✓ Paper observes contextual sparsity can be *accurately predicted* with lightweight learning-based algorithms.

✓ DejaVu is designed to use asynchronous lookahead predictors and hardware-efficient sparsity to *speed up LLM inference in wall-clock time*.

✓ Empirical results validate that contextual sparsity can reduce inference latency by over 2× compared to the state-of-the-art FasterTransformer *without model quality drops*.

→ Paper Discussion…

# References

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., ... & Chen, B. (2023, July). *International Conference on Machine Learning* (pp. 22137-22176).

Xia, H., Zheng, Z., Li, Y., Zhuang, D., Zhou, Z., Qiu, X., ... & Song, S. L. (2023). Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *arXiv preprint arXiv:2309.10285*.

Kusupati, A. (2020). Adapting Unstructured Sparsity Techniques for Structured Sparsity.

https://slideslive.com/39003934/deja-vu-contextual-sparsity-for-efficient-llms-at-inference-time?ref=speaker-19019