Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time

Zichang Liu [1] **Jue Wang** [2] **Tri Dao** [3] **Tianyi Zhou** [4] **Binhang Yuan** [5] **Zhao Song** [6] **Anshumali Shrivastava** [1]
Ce Zhang [5] **Yuandong Tian** [7] **Christopher Ré** [3] **Beidi Chen** [8 7]

Abstract

Large language models (LLMs) with hundreds of billions of parameters have sparked a new wave of exciting AI applications. However, they are computationally expensive at inference time. Sparsity is a natural approach to reduce this cost, but existing methods either require costly retraining, have to forgo LLM's in-context learning ability, or do not yield wall-clock time speedup on modern hardware. We hypothesize that *contextual sparsity*, which are small, input-dependent sets of attention heads and MLP parameters that yield approximately the same output as the dense model for a given input, can address these issues. We show that contextual sparsity exists, that it can be accurately predicted, and that we can exploit it to speed up LLM inference in wall-clock time without compromising LLM's quality or in-context learning ability. Based on these insights, we propose DEJAVU, a system that uses a low-cost algorithm to predict contextual sparsity on the fly given inputs to each layer, along with an asynchronous and hardware-aware implementation that speeds up LLM inference. We validate that DEJAVU can reduce the inference latency of OPT-175B by over 2× compared to the state-of-the-art FasterTransformer, and over 6× compared to the widely used Hugging Face implementation, without compromising model quality. The code is available at https://github.com/FMInference/DejaVu.

1 Introduction

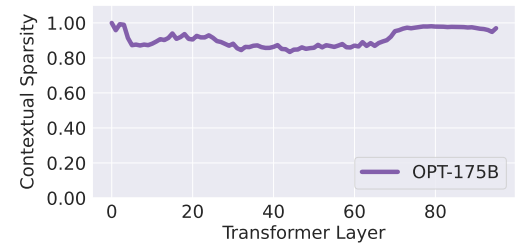
Large language models (LLMs), such as GPT-3, PaLM, and OPT have demonstrated that an immense number of parameters unleashes impressive performance and emergent in-context-learning abilities—they can perform a task by conditioning on input-output examples, without updating their parameters (Bommasani et al., 2021; Liang et al., 2022; Brown et al., 2020; Min et al., 2022; Chan et al., 2022). However, they are very expensive at inference time, especially for latency-sensitive applications (Pope et al., 2022). An ideal inference-time model should use less computation and memory while maintaining the performance and special abilities of pre-trained LLMs. The simplest and most natural approach is sparsification or pruning, which has a long history before the LLM era (LeCun et al., 1989). Unfortunately, speeding up inference-time sparse LLMs in wall-clock time while maintaining quality and in-context learning abilities remains a challenging problem.

While sparsity and pruning have been well-studied, they have not seen wide adoption on LLMs due to the poor quality and efficiency trade-offs on modern hardware such as GPUs. First, it is infeasible to retrain or iteratively prune models at the scale of hundreds of billions of parameters. Thus, methods in iterative pruning and lottery ticket hypothesis (Lee et al., 2018; Frankle & Carbin, 2018) can only be applied to smaller-scale models. Second, it is challenging to find sparsity that preserves the in-context learning ability of LLMs. Many works have shown the effectiveness of 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. Lastly, it is hard to achieve wall-clock time speed-up with unstructured sparsity due to its well-known difficulty with modern hardware (Hooker, 2021). For example, recent development in zero-shot pruning like SparseGPT (Frantar & Alistarh, 2023) finds 60% unstructured sparsity but does not yet lead to any wall-clock time speedup.

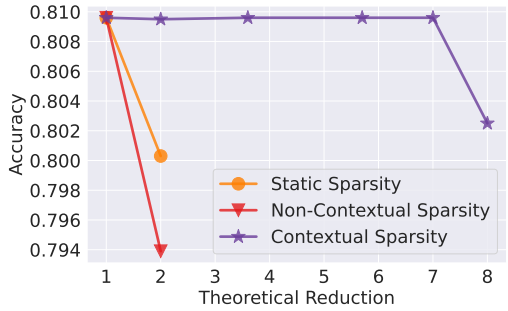
An ideal sparsity for LLMs should (i) not require model retraining, (ii) preserve quality and in-context learning ability, and (iii) lead to speed-up in wall-clock time on modern hardware. To achieve such demanding requirements, we go beyond *static* sparsity in previous works (e.g., structured/unstructured weight pruning). We instead envision *contextual sparsity*, which are small, input-dependent sets of attention heads and MLP parameters that lead to (approximately) the same output as the full model for an input. Inspired by the connections between LLMs, Hidden

[1]Rice University [2]Zhe Jiang University [3]Stanford University [4]University of California, San Diego [5]ETH Zurich [6]Adobe Research [7]Meta AI (FAIR) [8]Carnegie Mellon University. Correspondence to: Zichang Liu <zl71@rice.edu>, Tri Dao <trid@stanford.edu>, Tianyi Zhou <t8zhou@ucsd.edu>, Zhao Song <zsong@adobe.com>, Beidi Chen <beidic@andrew.cmu.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.



(a) Contextual Sparsity



(b) Accuracy-Efficiency Trade-offs

Figure 1. (1) LLMs have up to 85% contextual sparsity for a given input. (2) Contextual sparsity has much better efficiency-accuracy trade-offs (up to 7) than non-contextual sparsity or static sparsity.

Markov Models (Xie et al., 2022; Baum & Petrie, 1966), and the classic Viterbi algorithm (Viterbi, 1967), we hypothesize that for pre-trained LLMs,

contextual sparsity *exists given any input.*

The hypothesis, if true, would enable us to cut off specific attention heads and MLP parameters (structured sparsity) on the fly for inference-time, without modifying pre-trained models. However, there are three challenges.

Existence: It is nontrivial to verify if such contextual sparsity exists, and naive verification can be prohibitively expensive.

Prediction: Even if contextual sparsity exists, it is challenging to predict the sparsity for a given input in advance.

Efficiency: Even if the sparsity can be predicted, it might be difficult to achieve end-to-end wall-clock time speedup. Taking OPT-175B as an example, the latency of one MLP block is only 0.2 ms on an 8×A100 80GB machine. Without a fast prediction and optimized implementation, the overhead can easily increase the LLM latency rather than reduce it.

In this work, we address these challenges as follows:

Existence: Fortunately, we verify the existence of contextual sparsity with a surprisingly simple approach. To achieve essentially the same output, contextual sparsity is on average *85%* structured sparse and thereby potentially leads to a 7× parameter reduction for each specific input while maintaining accuracy (Figure 1(a)). During explorations of contextual sparsity, we make important empirical observations and build a theoretical understanding of major components in LLMs that help address the prediction and efficiency challenge.

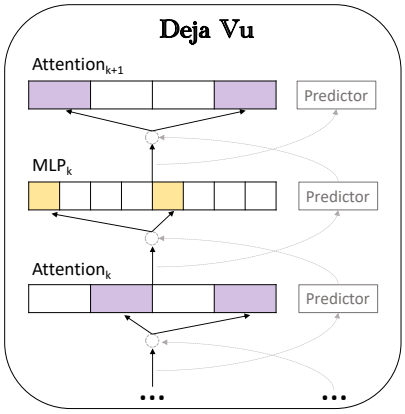


Figure 2. DEJAVU uses lookahead predictors to side-step prediction costs: given the input to the attention layer at block k, they (asynchronously) predict the contextual sparsity for the MLP at block k, and given the input to the MLP at block k, they predict the sparsity for the attention head at the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

Prediction: We discover that contextual sparsity depends not only on individual input tokens (i.e., *non-contextual dynamic* sparsity) but also on their interactions (*contextual dynamic* sparsity). Figure 1(b) shows that with pure dynamic information, sparsity prediction is inaccurate. Only with token embeddings with sufficient contextual information can we predict sparsity accurately. Another finding is that *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, which carries the immediate contextual mixture of token embeddings.

Efficiency: Because at inference time, model parameters are static, inspired by the classical nearest neighbor search (NNS) literature and its applications in efficient deep learning, it is possible to formulate the above similarity-based prediction as an NNS problem (Indyk & Motwani, 1998b; Zhang et al., 2018; Chen et al., 2020a). However, as mentioned, the overhead might be difficult to overcome as we would need to perform on-the-fly predictions before every layer. Luckily, we exploit a phenomenon of LLM where token embeddings change slowly across layers due to residual connections (well-known in computer vision (He et al., 2016)). Since the inputs to a few consecutive layers are very similar, we can design an asynchronous lookahead predictor (Figure 2).

Based on our findings, we present a system, DEJAVU, that exploits contextual sparsity and realizes efficient LLMs for latency-sensitive applications.

- In Section 4.1 and Section 4.2, we present a low-cost learning-based algorithm to predict sparsity on the fly. Given the input to a specific layer, it predicts a relevant subset of attention (heads) or MLP parameters in the next layer and only loads them for the computation.
- In Section 4.3, we propose an asynchronous predictor (similar to classic branch predictor (Smith, 1998)) to avoid the sequential overhead. A theoretical guarantee justifies that the

cross-layer design suffices for accurate sparsity prediction.

After integrating hardware-aware implementation of sparse matrix multiply (Section 4.4), DEJAVU (written mostly in Python) can reduce latency of open-source LLMs such as OPT-175B by over 2× end-to-end without quality degradation compared to the state-of-the-art library FasterTransformer from Nvidia (written entirely in C++/CUDA), and over 2× compared to the widely used Hugging Face implementation at small batch sizes. Furthermore, we show several ablations on different components of DEJAVU and its compatibility with quantization techniques.

2 Related Work and Problem Formulation

We first briefly discuss the rich literature on efficient inference. Then, we introduce the latency breakdown in our setting. Last, we provide a formal problem formulation.

2.1 Quantization, Pruning, Distillation for Inference

Various relaxations have been studied for decades for model inference in machine learning. There are three main techniques: quantization (Han et al., 2015; Jacob et al., 2018; Nagel et al., 2019; Zhao et al., 2019), pruning or sparsity (Molchanov et al., 2016; Liu et al., 2018; Hoefler et al., 2021), and distillation (Hinton et al., 2015; Tang et al., 2019; Touvron et al., 2021). They are orthogonal areas and usually excel in different settings. Recently, there is active research attempting to apply one or a combination of such techniques in LLM inference (Yao et al., 2022; Park et al., 2022; Dettmers et al., 2022; Frantar et al., 2022; Frantar & Alistarh, 2023; Bansal et al., 2022; Xiao et al., 2022). More discussion is presented in Appendix A.

2.2 LLM Inference Latency Breakdown

The generative procedure of LLMs consists of two phases: (i) the *prompt* phase takes an input sequence to generate the keys and values (KV cache) for each transformer block of LLMs, which is similar to the forwarding pass of LLMs training; and (ii) the *token generation* phase utilizes and updates the KV cache to generate tokens step by step, where the current token generation depends on previously generated tokens.

This paper studies the setting where the token generation phase easily dominates the end-to-end inference time. As shown in Table 1, generating a sequence of length 128 takes much longer time than processing a sequence of length 128 as prompt due to I/O latency of loading model parameters. In addition, Table 2 shows that attention and MLP are both bottlenecks in LLMs, e.g., in 175B models, loading MLP parameters takes around $\frac{2}{3}$ of the total I/O and attention heads take the other $\frac{1}{3}$. Further, in the tensor-parallel regime, there are two communications between GPUs, one after the attention block, and the other one after the MLP block. As shown in Table 3, communication between GPUs takes around 15 % token generation latency. This paper focuses on making attention and MLP more efficient. Communication cost implies that the upper bound of such speed-up is around 6× when skipping all transformer blocks.

Table 1. Theoretical breakdown for prompting versus token generation (tensor model parallelism on 8 A100-80G GPUs).

	TFLOPs	I/O	Compute Latency (ms)	I/O Latency (ms)
Prompting 128	44.6	330 GB	17.87	20.6
Token Generation 128	44.6	41 TB	17.87	2600

Table 2. Theoretical breakdown for Attention block versus MLP block in one transformer layer when generating one token (tensor model parallelism on 8 A100-80G GPUs).

	GFLOPs	I/O (GB)	Compute Latency (ms)	I/O Latency (ms)
Attention Block	1.21	1.12	0.00048	0.07
MLP Block	2.41	2.25	0.00096	0.14

Table 3. Latency breakdown of generating 1 token under the setting of batch size 1 and prompt length 128 on 8 A100-80GB.

All Reduce	MLP Block	Attention Block (ms)	Others
6 ms	19ms	13ms	2ms

2.3 Problem Formulation

The goal is to reduce the generation latency of LLMs by exploiting contextual sparsity. In the following, we formally define the sparsified attention and MLP blocks.

Sparsified MLP: There are two linear layers in one MLP block, $W^1, W^2 \in \mathsf{R}^{d \quad 4d}$. Denote $y \in \mathsf{R}^{1 \quad d}$ as the input to the MLP block in the current generation step. Let each column (the weight of i-th neuron) of linear layers be W^1_i, $W^2_i \in \mathsf{R}^{d \quad 1}$. With contextual sparsity, only a small set of them are required for computation. Let $S_M \subseteq [4d]$ denote such set of neurons for input y. The sparsified MLP computation is

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

where σ is the activation function, e.g., ReLU, GeLU. Note that since the computation in the first linear results in sparse activations, the second linear layer is also sparsified.

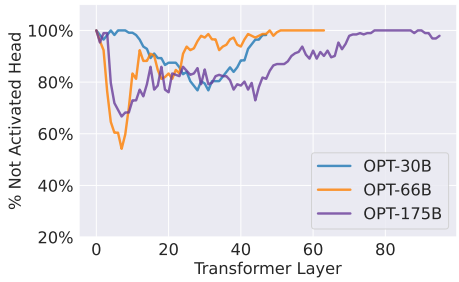
Sparsified Attention: Let $X \in \mathsf{R}^{n \quad d}$ denote the embeddings of all tokens (e.g., prompts and previously generated tokens). Let $y \in \mathsf{R}^{1 \quad d}$ be the input to the Multi-Head-Attention (MHA) in the current generation step. Suppose there are h heads. For each $i \in [h]$, we use $W^K_i, W^Q_i, W^V_i \in \mathsf{R}^{d \quad d_h}$ to denote key, query, value projections for the i-th head, and $W^O_i \in \mathsf{R}^{d_h \quad d}$ for output projections. With contextual sparsity, we denote S_A as a small set of attention heads leading to approximately the same output as the full attention for input y. Following the notation system in (Alman & Song, 2023), sparsified MHA computation can be formally written as

$$\mathsf{MHA}_{S_A}(y) = \sum_{i \in S_A} \underbrace{H_i(y)}_{1 \quad d_h} \underbrace{W^O_i}_{d_h \quad d},$$

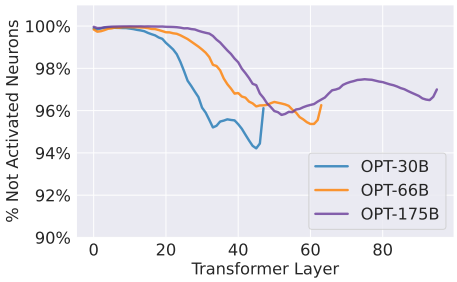
where $H_i(y) : \mathsf{R}^d \to \mathsf{R}^{d_h}$ and $D_i(y) \in \mathsf{R}$ can be written as

$$H_i(y) := D_i(y)^{-1} \exp(yW^Q_i (W^K_i)^\top X^\top) X W^V_i, \tag{2}$$
$$D_i(y) := \exp(yW^Q_i (W^K_i)^\top X^\top) \mathbf{1}_n.$$

For both MLP and Attention, given a compute budget, the goal is to find S_M and S_A that minimize the error between the sparse approximation and full computation.



(a) Contextual sparsity in Attention Head



(b) Contextual sparsity in MLP Block

Figure 3. In Figure (a), we plot the percentage of not-activated attention heads. By only keeping heads that yield large output norms, we can silence over 80% attention heads for a given token. In Figure (b), we plot the average sparsity we impose on MLP layers. We can zero out over 95% of MLP parameters for a given token.

3 Pre-trained LLMs are Contextually Sparse

In this section, we present several key observations and theoretical understandings of sparsity in LLMs, upon which the DEJAVU design is based. We first test the contextual sparsity hypothesis and verify that contextual sparsity exists in pre-trained LLMs in Section 3.1. Then, we build an understanding of why contextual sparsity happens naturally even when LLMs are densely trained in Section 3.2. Finally, we present an observation on residual connections and explain their relationship to contextual sparsity analytically in Section 3.3.

3.1 Contextual Sparsity Hypothesis

Inspired by prior pruning literature (Molchanov et al., 2016), we find a surprisingly simple method is sufficient to study and verify our hypothesis. In this section, we describe the testing procedure, observation details, and insights of this study.

Verification: Our test is performed on OPT-175B, 66B, and 30B models and various downstream datasets such as OpenBookQA (Mihaylov et al., 2018) and Wiki-Text (Merity et al., 2016). We find the contextual sparsity for every input example with two forward passes of the model. In the first pass, we record a subset of parameters, specifically which attention heads and MLP neurons yield large output norms for the input. In the second pass, each input example only uses the recorded subset of parameters for the computation. Surprisingly, these two forward passes lead to similar prediction or performance on all in-context learning and language modeling tasks.

Observation: Figure 3 shows that on average, we can impose up to 80% sparsity on attention heads and 95% sparsity on MLP neurons. As mentioned in Section 2, OPT-175B model has $2\times$ MLP parameters than those of attention blocks. Therefore total sparsity here is around 85%. Since these are all structured sparsity (heads and neurons), predicting them accurately could potentially lead to $7\times$ speedup.

Insight: It is intuitive that we can find contextual sparsity in MLP blocks at inference time because of their activation functions, e.g., ReLU or GeLU (Kurtz et al., 2020). Similar observations were made by (Li et al., 2022). However, it is surprising that we can find contextual sparsity in attention layers. Note that, finding contextual sparsity in attention is not the same as head pruning. We cross-check that different examples have different contextual sparsity. Although 80% of the parameters are not included in the paths for a given example, they might be used by other examples. Next, we will try to understand why contextual sparsity exists in attention blocks.

3.2 Token Clustering in Attention Layers

In the previous section, we have verified that there exists contextual sparsity for a given input in LLMs. In this section, we try to understand the reason for such phenomena, especially in attention layers. We first show an in-depth observation of attention. Then we present a hypothesis that self-attentions are conceptually clustering algorithms. Last we show analytical evidence to support this hypothesis.

Observation: Figure 4 shows the attention map of three different heads from the same layer for an example input. The next token it should predict is "Truck". Darker color represents higher attention scores. We observe that the middle head is a relatively uniform token-mixing head while the top and bottom ones are "heavy hitter" attention heads (with high attention to "like" and "shipping"). Unsurprisingly, only selecting heavy hitter heads but not uniform heads does not affect the prediction, since uniform heads do not model or encode important token interactions. In the next section, we will also explain in detail how the criteria for selecting uniform attention heads and heads with small output norms are highly correlated.

Hypothesis: We hypothesize that the attention head is performing mean-shift clustering (Derpanis, 2005).

Recall the notation defined in Section 2.3. For i-th head at current layer, $X = [x_1, \ldots, x_n]^\top \in \mathbb{R}^{n \times d}$ are the token embeddings in the previous time steps. XW_i^K and XW_i^V are the projection of embedding. For an input embedding y, the output $\text{~~}y_i = H_i(y)$, where $H_i(y)$ is defined in Eq. 2.

For each $i \in [h]$, if we let $K_i(x_j, y) := \exp(yW_i^Q (W_i^K)^\top x_j)$ measure the similarity between x_j and y, and define $m_i(y) := \frac{\sum_j K_i(x_j,y) x_j}{\sum_j K_i(x_j,y)}$, then we have $\text{~~}y_i = m_i(y) W_i^V$. Further, if we set $W_i^V = I$ and consider the residue connection followed by layer norm, then in the next layer, the embedding

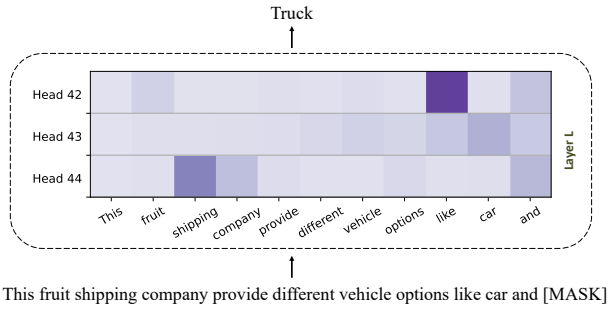


Figure 4. We visualize the attention scores of three different heads for an exemplary sentence. Head 42 and Head 44 give heavy attention scores on particular tokens while Head 43 is more uniform.

$\hat{y}_i$ of the current token becomes $\hat{y}_i = \text{Normalize}(y + \hat{y}_i) = \text{Normalize}(y + m_i(y))$, which has a fixed point $y = \gamma m_i(y)$ for any scalar γ. This iteration bears a resemblance to mean-shift clustering, which simply performs iteration $y \leftarrow m_i(y)$ until convergence. This has an obvious fixed point $y = m_i(y)$.

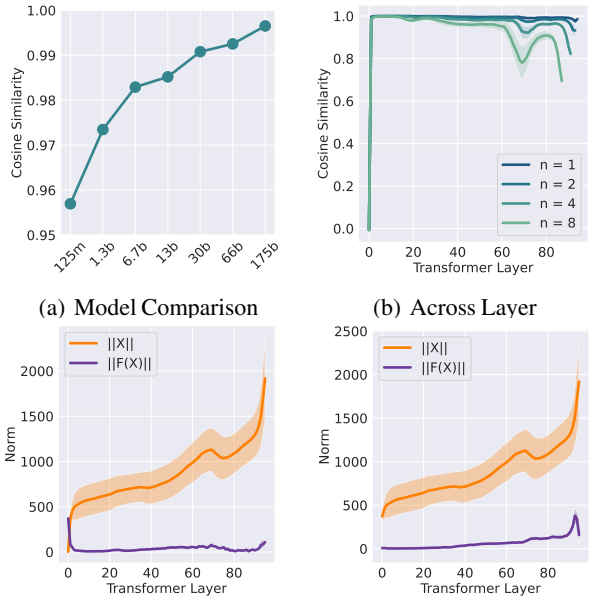
Therefore, the self-attention head can be regarded as *one mean-shift step* to push input embeddings of different tokens together, if they are already neighbors in a projection space specified by $W_i^Q (W_i^K)^\top$. Different heads learn different projection spaces to perform clustering. These dynamics explain the precise reason why token embeddings tend to cluster after going through more layers, resulting in high attention scores among cluster members, and low scores for non-members. Furthermore, the cluster patterns are different at different heads (More details in Appendix K).

The above analysis not only provides an understanding of why contextual sparsity exists naturally in pre-trained LLMs, but also inspires our design of "similarity"-based sparsity prediction for DEJAVU in Section 4.

3.3 Slowly Changing Embeddings across Layers

We first present our observation that embeddings change slowly across consecutive layers. Then we provide a detailed analysis on the phenomenon. Finally, we show its close connection with contextual sparsity. Details are in Section B.

High similar embeddings in consecutive layers: In Figure 5(a), we show that for the same given input, the cosine similarity between embeddings or activations in two consecutive layers is exceptionally high on 7 different sizes of OPT models. Specifically, we collect activations from each layer while performing OPT model inference on C4 validation set (Raffel et al., 2019). Taking OPT-175B as an example, starting from the second layer, the similarity between any two consecutive layers is around 0.99, which indicates that when an input is passed through the model, the direction of its embedding changes slowly. Interestingly, the most drastic change happens in the first layer. Furthermore, we increase the gap and investigate the similarity between the embedding at layer l and at layer $l + n$ shown in Figure 5(b). As we increase the gap, the similarity decreases as expected while the differences in cosine similarity between various choices



(a) Model Comparison (b) Across Layer

(c) Residual Around Attention (d) Residual Around MLP

Figure 5. **Slowly Changing Embedding.** Figure (a) shows the median cosine similarity between representations at two consecutive layers across all layers for different OPT models. All models show a similarity greater than 95%. Figure (b) shows cosine similarity stays high even a few layers apart. For the residual connection $X^0 = X + F(X)$ inside each block, we plot the ℓ_2 norm of X and $F(X)$ in Figure (c) and Figure (d). $\|X\|$ is significantly higher than $\|F(X)\|$, which explains the slowly changing embedding.

of n are smaller at the shallower layer. We plot the mean similarity, and the standard deviation is indicated by the shading. Similar plots on more models are presented in Appendix B.

Connection to residuals: We verify that the high similarity in embeddings in LLM inference is due to the residual connection. We first dissect the computation graph inside each transformer layer to understand the cause behind this phenomenon. There are two residual connections inside a transformer layer, one around the attention block, and the other one around the MLP block. The residual connection can be written as $X + F(X)$, where F is either the Multi-Head Attention or two MLP Layers. In Figure 5(c) and Figure 5(d), indeed we can see that $\|X\|$ is significantly greater than $\|F(X)\|$, confirming that embeddings are changing slowly because the residual norm is large.

Connection to Contextual Sparsity: We take a step deeper trying to understand the reason behind the large residual norm with mathematical modeling. We discover that one possible reason for small $\|F(X)\|$ is due to high sparsity. For the MLP Block, high sparsity may contribute to the small norm of $F(X)$ because a large portion of outputs have small norms. Similar reasoning applies to the Attention Block, and thus a large number of attention heads yield small norm outputs.

Residual Two Sides Bound: Besides empirical reasoning,

we formally define the computation of LLMs mathematically. Under our computation model, we can show that a shrinking property which is observed by our practical experiments. Proofs are in Appendix G, H, I.

Lemma 3.1 (Informal)**.** *Let $0 < \epsilon_1 < \epsilon_2 < 1$ be the lower and upper bound of the shrinking factor. Let x be the y be the output. We have the residual connection $y = x + F(x)$. For the MLP block $F(x)$, we have $\epsilon_1 \leq \|y - x\|_2 \leq \epsilon_2$. For the attention block $F(x)$, we have $\epsilon_1 \leq \|y - x\|_2 \leq \epsilon_2$.*

4 DEJAVU

In this section, we present our framework for inference-time contextual sparsity search for LLMs. We introduce the sparsity predictor for MLPs in Section 4.1 and for attention heads in Section 4.2. DEJAVU's workflow is shown in Figure 2. Section 4.3 discusses exploiting our observation on LLMs to avoid the sparse prediction overhead with theoretical guarantees. In Section 4.4, we present our optimized implementation that enables end-to-end latency reduction. More details are presented in Section D.

4.1 Contextual Sparsity Prediction in MLP Blocks

As explained in Section 2, MLP blocks are one of the major bottlenecks for the LLM generation ($\frac{2}{3}$ of the FLOPs and IOs). In this section, we discuss how we achieve wall-clock time speed-up with contextual sparsity in the MLP blocks.

Challenge Figure 3(b) shows that for a given token, the contextual sparsity of 95% is possible. The contextual sparsity in the MLP block can be identified after computing the activation. However, this only demonstrates the existence of contextual sparsity but brings no benefits in terms of efficiency. A fast and precise prediction is needed to exploit contextual sparsity for end-to-end efficiency. The naive way is to select a subset of neurons randomly. Unsurprisingly, random selection fails to identify the accurate contextual sparsity, resulting in drastic model degradation.

A Near-Neighbor Search Problem: Recall that we verify the existence of contextual sparsity by recording which neurons yield significant norms. Essentially, given the input, the goal is to search for the neurons that have high inner products with the input, because the activation function "filters" low activation. Thus, we formulate the contextual sparsity prediction of an MLP layer as the classical near-neighbor search problem under the inner product metric.

Definition 4.1 (Approximate MaxIP in MLP)**.** Let $c \in (0,1)$ and $\tau \in (0,1)$ denote two parameters. Given an n-vector dataset $W^1 \subset \mathbb{S}^{d-1}$ on a unit sphere, the objective of the (c,τ)-MaxIP is to construct a data structure that, given a query $y \in \mathbb{S}^{d-1}$ such that $\max_{w \in W^1} \langle y, w \rangle \geq \tau$, it retrieves a vector z from W^1 that satisfies $\langle y, z \rangle \geq c \cdot \max_{w \in W^1} \langle y, w \rangle$.

Remark 4.2. Our W^1 (first linear layer) and y (input embedding) in MLP blocks can be viewed as the dataset and query in Definition 4.1 respectively.

Design The standard state-of-the-art near-neighbor search methods and implementations slow down the computation. Take OPT-175B where d is 12288 as an example. HNSW (Malkov & Yashunin, 2018) requires more than 10ms, and FAISS (Johnson et al., 2019) requires more than 4ms, while the MLP computation is only 0.2ms. The high dimensionality and complications of data structure implementation on GPU make the search time longer than the MLP computation. Therefore, we choose a neural network classifier as our near-neighbor search method to exploit the fast matrix multiplication on GPU. For each MLP block, we train a small two-layer fully connected network to predict contextual sparsity. Collecting training data is straightforward because we know the contextual sparsity using dense computation. The training algorithm is summarized in Algorithm 1. The sparsified computation in W^1 has two steps: (1) Given y, the sparsity predictor SP_M predicts a set S_M of important neurons in weights W^1. (2) Compute the sparsified MLP defined in Eq. equation 1. Note here the sparsity in MLP is highly structured.

Algorithm 1 Sparse Predictor Training

Input: A pre-trained LLM block with parameter set M, token embedding set at block $M = \{x_i\}_{i2[N]}$, 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
$\mathcal{SP} \leftarrow \textsc{Train}(\mathcal{P}_+, \mathcal{P}^-, \mathcal{L})$ ▷ $\mathcal{L}$ is a loss function

4.2 Contextual Sparsity Prediction in Attention Blocks

Attention blocks take around 30% I/Os in the generation. In this section, we describe how DEJAVU exploits contextual sparsity to speed up the Attention blocks.

Challenge: As discussed in Section 3.1, only a few heads perform important computations for a given input token. Similar to the MLP blocks, a fast selection of attention heads without full computation is required to reduce end-to-end latency. Furthermore, one particular challenge of sparse prediction in attention blocks is attention's dependence on previous tokens. On the one hand, it is unclear whether the past token's key and value caches are needed for sparse prediction. On the other hand, it is unclear how to handle the missing KV cache of past tokens for the current token computation at the selected head.

A Near-Neighbor Search Problem: Head prediction can also be formulated as a near-neighbor search problem based on our understanding in Section 3.2. Since each head is performing mean-shift clustering, after the first few layers, the current token embedding alone is sufficient for the prediction thanks to the token-mixing nature of the transformer. Therefore, the prediction can be based on the similarity between y and head parameters.

Approach: We design our attention sparse predictor to be the same architecture as the MLP sparse predictor. Each head is regarded as one class and a similar training process is used (Algorithm 1). Then, similar to how MLP prediction is performed, the attention sparsity predictor SP_A selects a set S_A of heads H_i (see Eq. equation 2). To address the problem of missing KV cache for a past token, we exploit the fact that the generation latency is I/O bounded while computation is essentially "free". Specifically, for the predicted attention head of input y, we compute the corresponding keys, and values and store them in the KV cache. But we also save a copy of y for all the other non-selected heads. Then during the future token generation, if there is missing KV cache in the selected heads, we could load stored token embeddings and compute the keys and values together. This requires almost minimal extra memory access (the main cost is loading the weight matrices).

4.3 Reducing Overhead with Asynchronous Execution

Sparse prediction overhead may easily increase the end-to-end latency rather than reduce it despite the reduction in FLOPs. Therefore, we introduce a look-ahead sparse prediction method, inspired by our observations in Section 3.3.

Challenge: Denote $y_l \in \mathsf{R}^d$ as the input to transformer layer l. We can write the computation at layer l as $\widetilde{y}_l \leftarrow \mathsf{MHA}^l(y_l), \widehat{y}_l \leftarrow \mathsf{MLP}^l(\widetilde{y}_l)$. With predictors SP^l_A and SP^l_M, the computation at the transformer layer l can be re-written as

$$S^l_A \leftarrow \mathsf{SP}^l_A(y_l), \quad \widetilde{y}_l \leftarrow \mathsf{MHA}^l_{S^l_A}(y_l),$$
$$S^l_M \leftarrow \mathsf{SP}^l_M(\widetilde{y}_l), \quad \widehat{y}_l \leftarrow \mathsf{MLP}^l_{S^l_M}(\widetilde{y}_l)$$

where set S^l_A is the contextual sparsity for the Attention block, and set S^l_M is the contextual sparsity for the MLP block at l-th layer. Note that the computation at Attention and MLP blocks have to wait for the sparse predictor decision. This overhead potentially outweighs the saving from Attention and MLP blocks in terms of latency.

Approach: In Section 3.3, we present the slowly evolving embedding phenomenon, which provides opportunities to relax the sequential computation to parallel computation. Along with the observation of low computation intensity during generation, we parallel the sparse prediction with the computation of each block (See Figure 2). The computation can be written as follows:

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

We remark S^{l+1}_A and S^{l+1}_M can be computed in parallel with $\widetilde{y}_l$ or $\widehat{y}_l$, while the previous 4 steps are sequential.

Theoretical guarantee: The sparse predictor can make further cross-layer decisions because of the residual connection. We present an informal lemma statement regarding cross-layer prediction. It is well-known that MaxIP is equivalent to ℓ_2 nearest neighbor search. For convenience, we use MaxIP here. We include more discussions and proofs in Section J.

Lemma 4.3 (Informal)**.** *Let $\epsilon \in (0,1)$. Let y_l be input at l-th layer. Let $y_{l\ 1}$ be the input at $(l-1)$-th layer. Suppose that $\|y_l - y_{l\ 1}\|_2 \le \epsilon$. For any parameters c, τ such that $\epsilon < O(c\tau)$. Then we can show that, solving $\mathsf{MaxIP}(c,\tau)$ is sufficient to solve $\mathsf{MaxIP}(0.99c,\tau)$.*

4.4 Hardware-efficient Implementation

We describe how DEJAVU is implemented in a hardware-efficient manner to realize the theoretical speedup of contextual sparsity. Taking into account hardware characteristics leads to over 2× speedup compared to an optimized dense model, and 4× faster than a standard sparse implementation.

We highlight some hardware characteristics of GPUs:

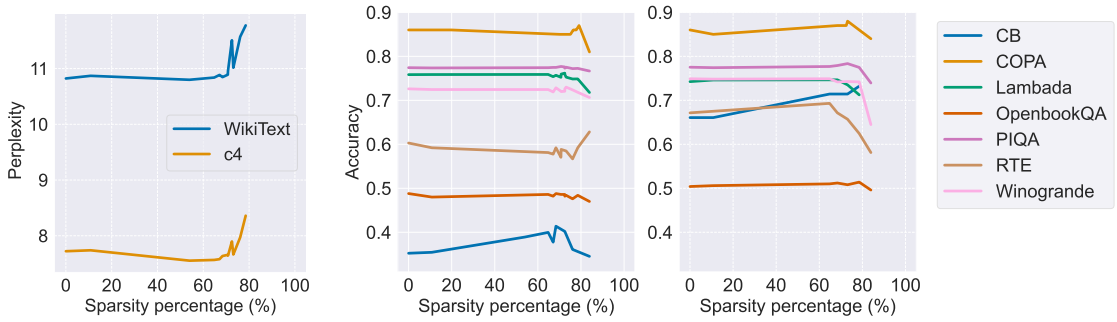
- Small-batch generation is bottlenecked by GPU memory I/Os (NVIDIA, 2022; Ivanov et al., 2021; Dao et al., 2022). This is because of low arithmetic intensity. For each element loaded from GPU memory, only a small number of floating point operations are performed.
- GPUs are block-oriented devices: loading a single byte of memory takes the same time as loading a block of memory around that same address (Harris, 2013). The block size is usually 128 bytes for NVIDIA GPUs (Cook, 2012).

These characteristics present some challenges in implementing contextual sparsity. However, they can be addressed with classical techniques in GPU programming.

Kernel fusion: A standard implementation of sparse matrix-vector multiply (e.g., in PyTorch) that separately indexes a subset of the matrix $W^1_{S_M}$ before multiplying with input y would incur 3× the amount of memory I/Os. Therefore, to avoid such overhead, we fuse the indexing and the multiplication step. Specifically, we load a subset of $W^1_{S_M}$ to memory, along with y, perform the multiply, then write down the result. This fused implementation (in Triton (Tillet et al., 2019)) yields up to 4× speedup compared to a standard PyTorch implementation (Appendix E).

Memory coalescing: In the dense implementation, the weight matrices of two linear layers in MLP are stored as $(W^1)^\top$ and W^2 so that no extra transpose operation is needed. They are conventionally stored in row-major format. In the sparse implementation, it allows us to load $(W^1_{S_M})^\top$ optimally (the second dimension is contiguous in memory). However, for cases where we need to load $(W^2_{S_M})$, this format significantly slows down memory loading, as indices in S_M point to non-contiguous memory. We simply store these matrices in column-major format (i.e., store $(W^2)^\top$ in row-major format), then use the same fused kernel above. Similarly, in attention blocks, we store attention output projection W^O column-major format.

These two techniques (kernel fusion and memory-coalescing) make DEJAVU hardware-efficient, yielding up to 2× speedup end-to-end compared to the state-of-the-art FasterTransformer (Section 5.1).



(a) Language Modeling (b) Zero-Shot(Left). Five-Shot(Right)

Figure 6. **Accuracy Trend for DEJAVU-OPT-175B**. This figure shows the accuracy of DEJAVU-OPT-175B on language modeling datasets and downstream tasks when we set different sparsity at test time. In general, DEJAVU-OPT-175B incurs no accuracy drop until 75% sparsity.

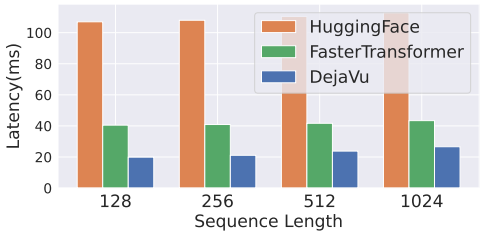


Figure 7. Average per-token latency (ms) with batch size 1 on 8 A100-80GB with NVLink when generating sequences with prompt lengths 128, 256, 512, and 1024, using FP16. DEJAVU speeds up generation by 1.8-2 compared to the state-of-the-art FT and by 4.8-6 compared to the widely used HF implementation.

5 Empirical Evaluation

In Section 5.1, we present the end-to-end results that show DEJAVU achieves over 2× reduction in token generation latency compared to the state-of-the-art FasterTransformer and over 6× compared to Hugging Face with no accuracy loss. In Section 5.2, we perform a list of ablation studies such as independent evaluation on the inference-time contextual sparsity of the MLP block and the Attention block (Details are presented in Section C). At last, we present the additional results to demonstrate the future possibility of sparsifying the entire LLMs via layer skipping in Section C.3.

5.1 End-to-End Result

Experiment Setting: We compare the accuracy of DEJAVU-OPT against the original OPT model on two language modeling datasets Wiki-Text (Merity et al., 2016) and C4 (Raffel et al., 2019) and seven few-shot downstream tasks: CB (de Marneffe et al., 2019), COPA (Gordon et al., 2012), Lambada (Radford et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), RTE (Giampiccolo et al., 2007), Winogrande (ai2, 2019). We use lm-eval-harness (Gao et al., 2021) for zero-shot and five-shot tasks. We collect training data for the sparsity predictor using 500 random data points from the C4 training dataset. Our experiments are conducted on NVIDIA A100 80GB GPU servers.

No accuracy drop until 75% sparsity: In Figure 6, we present DEJAVU-OPT-175B's accuracy trend. In a zero-shot setting, the average accuracy across tasks does not drop until 75% sparsity. A similar trend can be observed for the five-shot setting, which verifies the model's ability for in-context learning. This result is exceptionally encouraging given our observation in Figure 1(a), where we could impose 85% sparsity when allowed full computation.

Over 2× latency reduction: Figure 7 presents the latency speed-up for the token generation with OPT-175B at batch size 1, where DEJAVU achieves the best performance. At around 75% sparsity, DEJAVU speeds up generation by 1.8-2× compared to the state-of-the-art FasterTransformers (FT)[1] and by 4.8-6× to Hugging Face (HF) implementation[2].

5.2 Ablation Results

Contextual Sparsity for Larger Batches: Although this paper focuses on latency-sensitive settings, we demonstrate that DEJAVU generalizes to larger batches. we present the Union contextual sparsity (fraction of neurons/heads that are not used by any of the inputs in the batch) of different batches sizes for MLP and Attention blocks, respectively, in Figure 8 and 11. The union operation is essential to realize a fast sparse GEMM. Surprisingly the number of MLP neurons and Attention heads that DEJAVU activated does not grow linearly with the batch size. This suggests a power law distribution rather than a uniform distribution of parameter access from all input examples. This provides an opportunity for potentially extending Dejavu to the high-throughout setting. For example, we can first pre-process the inputs and batch similar inputs to enjoy a higher level of union contextual sparsity.

Contextual sparsity on MLP blocks: We study the contextual sparsification of the MLP block in OPT-175B. We leave the Attention block as dense computation. Table 4 shows the model performance at 85% sparsity. The MLP sparse predictor introduces no accuracy loss on both zero-shot tasks and language modeling. In the training of the MLP sparse predictor, we observe that the sparse predictor achieves high validation accuracy. The shallow layer seems easier to model

[1] http://github.com/NVIDIA/FasterTransformer
[2] http://github.com/huggingface/transformers

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

Table 5. DEJAVU-OPT66B on zero-shot downstream task.

Model	CB	COPA	Lambada	OpenBookQA	PIQA	RTE	Winogrande
OPT-66B	0.3928	0.87	0.7508	0.426	0.7921	0.6028	0.6890
DEJAVU-OPT-66B	0.4285	0.87	0.7458	0.434	0.7933	0.5884	0.6898

Table 6. DEJAVU-BLOOM on zero-shot downstream task.

	CB	COPA	OpenBookQA	PIQA	RTE	Winogrande	Lambada
BLOOM	0.455	0.8	0448	0.79	0.617	0.704	0.677
Dejavu-BLOOM	0.448	0.8	0.44	0.787	0.606	0.710	0.675

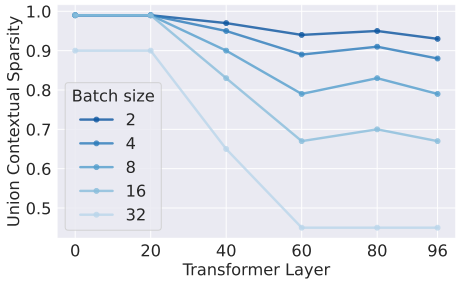


Figure 8. Union contextual sparsity with larger batch size.

because the predictor has validation accuracy over 99% in the shallow layers and drops to around 93% in the ending layers.

Contextual sparsity on attention blocks: In this section, we study the sparse predictor for the Attention block on OPT-175B and leave the MLP block as dense computation. Table 4 displays the test accuracy on zero-shot tasks and perplexity on the language modeling datasets. In summary, the Attention sparse predictor introduces no accuracy loss at around 50% sparsity. During the training of the Attention sparse predictor, we observe different trends compared to the MLP sparse predictor. The validation accuracy is around 93% in the middle layers and near 99% in the shallow and deep layers.

Contextual Sparsity on Smaller Models: Our main experiments focus on OPT-175B. Here, we verify DEJAVU's effectiveness on a smaller model, specifically OPT-66B. In Table 5, we summarize the accuracy on zero-shot task at 50% sparsity. Similar to DEJAVU-OPT-175B, we notice no accuracy loss.

Contextual Sparsity on Other Models: We expand the evaluation to another model family. In Table 6, we summarize the accuracy at attention sparsity 50% and MLP sparsity 30%. Similar to OPT family, we notice no accuracy loss. The lower sparsity level in MLP is due to the difference in activation function.

Non-Contextual Sparsity: As we mentioned in Section 1, one could predict sparsity without contextual information. For non-contextual sparsity, we rely on the original embedding at the input layer. At every block, we first pass the original embedding to record a subset of parameters yielding a large norm. In the second pass, the embedding at every layer only uses the recorded subset. As shown in Figure 1, non-contextual prediction is not sufficient and leads to accuracy losses even at 50% sparsity. This result verifies our design choices of relying on the activation at every layer as input to make contextual sparsity predictions.

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

Compatibility with Quantization: Quantization is another promising direction for efficient language models. We investigate the possibility of combining contextual sparsity with quantization techniques. For DEJAVU-OPT-175B, we set the entire model sparsity at 75%. For quantization, we apply 4-bit quantization on model weights (W4A16). As shown in Table 7, the combination of quantization and DEJAVU almost always achieves better accuracy than DEJAVU or quantization alone. This suggests that the approximation errors from these two directions do not get compounded.

6 Conclusion

Our main goal is to make LLM inference efficient so that their powerful in-context learning abilities can be used in more application domains. We observe that contextual sparsity can be accurately predicted with lightweight learning-based algorithms. This motivated us to design DEJAVU that uses asynchronous lookahead predictors and hardware-efficient sparsity to speed up LLM inference in wall-clock time. Our encouraging 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. Our method is a step towards making LLMs more accessible to the general community, which could unlock exciting new AI applications.

Acknowledgements

We would like to thank Ryan Spring, Laurel Orr, Guangxuan Xiao, Eric Han, Xun Huang, Daniel Y. Fu, Benjamin Spector, Ruan Silva, Diana Liskovich, and the anonymous reviewers for helpful discussions and feedback. We acknowledge the generous support by Together Computer, which enabled the necessary partial computations in this work.

References

Winogrande: An adversarial winograd schema challenge at scale. 2019.

Allen-Zhu, Z. and Li, Y. What can resnet learn efficiently, going beyond kernels? *Advances in Neural Information Processing Systems*, 32, 2019.

Alman, J. and Song, Z. Fast attention requires bounded entries. *arXiv preprint arXiv:2302.13214*, 2023.

Alman, J., Liang, J., Song, Z., Zhang, R., and Zhuo, D. Bypass exponential time preprocessing: Fast neural network training via weight-data correlation preprocessing. *arXiv preprint arXiv:2211.14227*, 2022.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29, 1996.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale. In *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 646–660. IEEE Computer Society, 2022.

Andoni, A. and Razenshteyn, I. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pp. 793–801, 2015.

Andoni, A., Indyk, P., Nguyen, H. L., and Razenshteyn, I. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1018–1028. SIAM, 2014.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1225–1233. Curran Associates, 2015.

Andoni, A., Laarhoven, T., Razenshteyn, I., and Waingarten, E. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 47–66. SIAM, 2017.

Andoni, A., Indyk, P., and Razenshteyn, I. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 7, 2018.

Arya, S. and Mount, D. M. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pp. 271–280. Citeseer, 1993.

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., and McWilliams, B. The shattered gradients problem: If resnets are the answer, then what is the question? In *International Conference on Machine Learning*, pp. 342–350. PMLR, 2017.

Bansal, H., Gopalakrishnan, K., Dingliwal, S., Bodapati, S., Kirchhoff, K., and Roth, D. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. *arXiv preprint arXiv:2212.09095*, 2022.

Baum, L. E. and Petrie, T. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Bello, I., Fedus, W., Du, X., Cubuk, E. D., Srinivas, A., Lin, T.-Y., Shlens, J., and Zoph, B. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, 2021.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of machine learning research (JMLR)*, 3(Feb):1137–1155, 2003.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., Pieler, M., Prashanth, U. S., Purohit, S., Reynolds, L., Tow, J., Wang, B., and Weinbach, S. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022. URL https://arxiv.org/abs/2204.06745.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Boutsidis, C., Woodruff, D. P., and Zhong, P. Optimal principal component analysis in distributed and streaming models. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, 2016.

Boytsov, L., Novak, D., Malkov, Y., and Nyberg, E. Off the beaten path: Let's replace term-based retrieval with k-nn search. In *Proceedings of the 25th ACM international on conference on information and knowledge management (CIKM)*, pp. 1099–1108, 2016.

Brand, J. v. d., Peng, B., Song, Z., and Weinstein, O. Training (overparametrized) neural networks in near-linear time. In *ITCS*, 2021.

Brand, J. v. d., Song, Z., and Zhou, T. Algorithm and hardness for dynamic attention maintenance in large language models. *arXiv preprint arXiv:2304.02207*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chan, S. C., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A. K., Richemond, P. H., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. In *Advances in Neural Information Processing Systems*, 2022.

Chang, W.-C., Yu, F. X., Chang, Y.-W., Yang, Y., and Kumar, S. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*, 2020.

Charikar, M., Chen, K., and Farach-Colton, M. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pp. 693–703. Springer, 2002.

Chen, B., Xu, Y., and Shrivastava, A. Fast and accurate stochastic gradient estimation. *Advances in Neural Information Processing Systems*, 32, 2019.

Chen, B., Medini, T., Farwell, J., Tai, C., Shrivastava, A., et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *Proceedings of Machine Learning and Systems*, 2:291–306, 2020a.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34: 17413–17426, 2021a.

Chen, B., Liu, Z., Peng, B., Xu, Z., Li, J. L., Dao, T., Song, Z., Shrivastava, A., and Re, C. Mongoose: A learnable lsh framework for efficient neural network training. In *International Conference on Learning Representations*, 2021b.

Chen, H., Chillotti, I., Dong, Y., Poburinnaya, O., Razenshteyn, I., and Riazi, M. S. {SANNS}: Scaling up secure approximate k-nearest neighbors search. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 2111–2128, 2020b.

Chen, L. On the hardness of approximate and exact (bichromatic) maximum inner product. In *33rd Computational Complexity Conference (CCC)*, 2018.

Cho, J. H. and Hariharan, B. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4794–4802, 2019.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Clarkson, K. L. and Woodruff, D. P. Low-rank approximation and regression in input sparsity time. In *STOC*, 2013.

Cohen, M. B. Nearly tight oblivious subspace embeddings by trace inequalities. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pp. 278–287. SIAM, 2016.

Cook, S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012. ISBN 9780124159334.

Cox, M. and Cox, T. Multidimensional scaling, 315–347. *Handbook of data visualization. Springer, Berlin, Germany*, 2008.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SoCG)*, pp. 253–262, 2004.

de Marneffe, M.-C., Simons, M., and Tonhauser, J. The commitmentbank: Investigating projection in naturally occurring discourse. 2019.

Deng, Y., Li, Z., and Song, Z. Attention scheme inspired softmax regression. *arXiv preprint arXiv:2304.10411*, 2023a.

Deng, Y., Mahadevan, S., and Song, Z. Randomized and deterministic attention sparsification algorithms for over-parameterized feature dimension. *arxiv preprint: arxiv 2304.03426*, 2023b.

Derpanis, K. G. Mean shift clustering. *Lecture Notes*, 32: 1–4, 2005.

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.

Dong, S., Lee, Y. T., and Ye, G. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1784–1797, 2021.

Dong, Y., Indyk, P., Razenshteyn, I., and Wagner, T. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations*, 2019.

Fang, J., Yu, Y., Zhao, C., and Zhou, J. Turbotransformers: an efficient gpu serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 389–402, 2021.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Frantar, E. and Alistarh, D. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Frei, S., Cao, Y., and Gu, Q. Algorithm-dependent generalization bounds for overparameterized deep residual networks. *Advances in neural information processing systems*, 32, 2019.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

Gao, Y., Mahadevan, S., and Song, Z. An over-parameterized exponential regression. *arXiv preprint arXiv:2303.16504*, 2023a.

Gao, Y., Song, Z., and Yang, X. Differentially private attention computation. *arXiv preprint arXiv:2305.04701*, 2023b.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 1–9, Prague, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/W07-1401.

Gionis, A., Indyk, P., Motwani, R., et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pp. 518–529, 1999.

Gordon, A., Kozareva, Z., and Roemmele, M. SemEval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 394–398, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics. URL https://aclanthology.org/S12-1052.

Gu, Y. and Song, Z. A faster small treewidth sdp solver. *arXiv preprint arXiv:2211.06033*, 2022.

Gu, Y., Song, Z., Yin, J., and Zhang, L. Low rank matrix completion via robust alternating minimization in nearly linear time. *arXiv preprint arXiv:2302.11068*, 2023.

Hall, R. and Attenberg, J. Fast and accurate maximum inner product recommendations on map-reduce. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, pp. 1263–1268, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Harris, M. How to access global memory efficiently in CUDA C/C++ kernels. *NVIDIA, Jan*, 2013.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4340–4349, 2019.

Hinton, G., Vinyals, O., Dean, J., et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

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. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.

Hooker, S. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021.

Hu, H., Song, Z., Weinstein, O., and Zhuo, D. Training overparametrized neural networks in sublinear time. *arXiv preprint arXiv:2208.04508*, 2022.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pp. 604–613, 1998a.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998b.

Indyk, P. and Wagner, T. Approximate nearest neighbors in limited space. In *Conference On Learning Theory*, pp. 2012–2036. PMLR, 2018.

Ivanov, A., Dryden, N., Ben-Nun, T., Li, S., and Hoefler, T. Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems*, 3:711–732, 2021.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

Jiang, S., Song, Z., Weinstein, O., and Zhang, H. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 823–832, 2021.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. In *ICLR*, 2020.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Shavit, N., and Alistarh, D. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5533–5543. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/kurtz20a.html.

Laurent, B. and Massart, P. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pp. 1302–1338, 2000.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lee, M., He, X., Yih, W.-t., Gao, J., Deng, L., and Smolensky, P. Reasoning in vector space: An exploratory study of question answering. In *ICLR*, 2016.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Lee, Y. T., Song, Z., and Zhang, Q. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory*, pp. 2140–2157. PMLR, 2019.

Li, P., Li, X., and Zhang, C.-H. Re-randomized densification for one permutation hashing and bin-wise consistent weighted sampling. *Advances in Neural Information Processing Systems*, 32, 2019.

Li, S., Song, Z., Xia, Y., Yu, T., and Zhou, T. The closeness of in-context learning and weight shifting for softmax regression. *arXiv preprint*, 2023a.

Li, X. and Li, P. C-MinHash: Improving minwise hashing with circulant permutation. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12857–12887. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/li22m.html.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S. J., Ye, K., Chern, F., Yu, F., Guo, R., and Kumar, S. Large models are parsimonious learners: Activation sparsity in trained transformers, 2022. URL https://arxiv.org/abs/2210.06313.

Li, Z., Song, Z., and Zhou, T. Solving regularized exp, cosh and sinh regression problems. *arXiv preprint, 2303.15725*, 2023b.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Liu, Z., Xu, Z., Ji, A., Zhang, J., Li, J., Chen, B., and Shrivastava, A. Halos: Hashing large output space for cheap inference. *Proceedings of Machine Learning and Systems*, 4:110–125, 2022.

Lu, Y., Dhillon, P., Foster, D. P., and Ungar, L. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in neural information processing systems (NIPS)*, pp. 369–377, 2013.

Malkov, Y., Ponomarenko, A., Logvinov, A., and Krylov, V. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45: 61–68, 2014.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

Meng, X. and Mahoney, M. W. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pp. 91–100, 2013.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Nagel, M., Baalen, M. v., Blankevoort, T., and Welling, M. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1325–1334, 2019.

Nelson, J. and Nguyên, H. L. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science*, pp. 117–126. IEEE, 2013.

Neyshabur, B. and Srebro, N. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning (ICML)*, pp. 1926–1934. PMLR, 2015.

NVIDIA. Fastertransformer. https://github.com/NVIDIA/FasterTransformer.

NVIDIA. Gpu performance background user's guide, 2022. URL https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html.

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*, 2022.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *arXiv preprint arXiv:2211.05102*, 2022.

Qin, L., Song, Z., and Wang, Y. Fast submodular function maximization. *CoRR*, abs/2305.08367, 2023a.

Qin, L., Song, Z., Zhang, L., and Zhuo, D. An online and unified algorithm for projection matrix vector multiplication with application to empirical risk minimization. In *AISTATS*, 2023b.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

Razenshteyn, I., Song, Z., and Woodruff, D. P. Weighted low rank approximations with provable guarantees. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 250–263, 2016.

Sarlos, T. Improved approximation algorithms for large matrices via random projections. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS)*, pp. 143–152. IEEE, 2006.

Seo, M., Lee, J., Kwiatkowski, T., Parikh, A. P., Farhadi, A., and Hajishirzi, H. Real-time open-domain question answering with dense-sparse phrase index. In *ACL*, pp. 4430–4441, 2019.

Shrivastava, A., Song, Z., and Xu, Z. Sublinear least-squares value iteration via locality sensitive hashing. *arXiv preprint arXiv:2105.08285*, 2021.

Smith, J. E. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, pp. 202–215, 1998.

Sohler, C. and Woodruff, D. P. Subspace embeddings for the l1-norm with applications. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 755–764, 2011.

Song, Z. and Ye, M. Efficient asynchronize stochastic gradient algorithm with structured data. *CoRR*, abs/2305.08001, 2023.

Song, Z. and Yu, Z. Oblivious sketching-based central path method for linear programming. In *International Conference on Machine Learning*, pp. 9835–9847. PMLR, 2021.

Song, Z., Woodruff, D. P., and Zhong, P. Low rank approximation with entrywise l1-norm error. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 688–701, 2017.

Song, Z., Woodruff, D. P., and Zhong, P. Relative error tensor low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2772–2789. SIAM, 2019.

Song, Z., Zhang, L., and Zhang, R. Training multi-layer over-parametrized neural network in subquadratic time. *arXiv preprint arXiv:2112.07628*, 2021.

Song, Z., Wang, W., and Yin, C. Fast and efficient matching algorithm with deadline instances. *CoRR*, abs/2305.08353, 2023a.

Song, Z., Yang, X., Yang, Y., and Zhang, L. Sketching meets differential privacy: fast algorithm for dynamic kronecker projection maintenance. In *International Conference on Machine Learning (ICML)*, 2023b.

Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Veit, A., Wilber, M. J., and Belongie, S. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

Wang, B. and Komatsuzaki, A. GPT-J-6B: A 6 billion parameter autoregressive language model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Wang, R. and Woodruff, D. P. Tight bounds for lp oblivious subspace embeddings. 2018.

Wang, X., Xiong, Y., Wei, Y., Wang, M., and Li, L. Lightseq: A high performance inference library for transformers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pp. 113–120, 2021.

Woodruff, D. P. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

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.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=RdJVFCHjUMI.

Xue, H.-J., Dai, X., Zhang, J., Huang, S., and Chen, J. Deep matrix factorization models for recommender systems. In *IJCAI*, pp. 3203–3209, 2017.

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.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Zandieh, A., Han, I., Daliri, M., and Karbasi, A. Kdeformer: Accelerating transformers via kernel density estimation. *arXiv preprint arXiv:2302.02451*, 2023.

Zhang, L. Speeding up optimizations via data structures: Faster search, sample and maintenance. Master's thesis, Carnegie Mellon University, 2022.

Zhang, M., Wang, W., Liu, X., Gao, J., and He, Y. Navigating with graph representations for fast and scalable decoding of neural language models. *Advances in neural information processing systems*, 31, 2018.

Zhao, R., Hu, Y., Dotzel, J., De Sa, C., and Zhang, Z. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pp. 7543–7552. PMLR, 2019.

Contents: In Section A, we present an extended discussion on LLM inference and related works. In Section B, we provide more observation plots for slowly changing activation and further observation on the possibility of sparsifying LLMs via layer skipping. In Section C, we provide experiment details. In Section D, we demonstrate implementation details. In Section E, we provide detailed benchmarks regarding our implementation. In Section F, we define some basic notations and definitions. In Section G, we define subspace embedding and show the norm preserving. In Section H, we introduce distances, angles, and inner product. In Section I, we provide the distance between different functions. In Section J, we provide the Near-neighbor Search data structure. In Section K, we discuss self-attention as a clustering algorithm in depth.

A Related Work

Generative LLM inference. Taking OPT-175B as an example, assume 6 A100 80GB PCIe, based on the hardware specifications, we compare two main phases of inference time LLM, namely prompting and token generation in Table 1, and two major components, namely Multi-Head-Attention block and MLP block in Table 2. In practice, the token generation phase usually dominates the end-to-end test latency due to IO latency. Generating only two tokens is about the same latency as prompting. Further, during token generation, the MLP block is $2\times$ more expensive in both FLOPs and IO access. The hardware is often at low utilization because memory reads and writes are more limited on modern hardware than tensor core computation.

Given the rapid development of LLM, there is an emergence of systems that are specialized for LLM inference, such as Faster Transformer (NVIDIA), Orca (Yu et al., 2022), LightSeq (Wang et al., 2021), PaLM inference (Pope et al., 2022), TurboTransformers (Fang et al., 2021), and Deepspeed-Inference (Aminabadi et al., 2022). In practice, the token generation phase usually dominates the end-to-end inference time. Although the state-of-the-art systems introduce some helpful system optimizations for speedup, there is a lack of careful algorithm and system co-design to unleash the full potential of hardware efficiency during the LLM inference computation.

Near-neighbor Search for Efficient Deep Neural Networks. Near-neighbor Search is a well-studied problem with wide applications in recommendation system (Xue et al., 2017; Hall & Attenberg, 2015), question answering (Boytsov et al., 2016; Seo et al., 2019; Chang et al., 2020) and natural language processing (Bengio et al., 2003; Lee et al., 2016). There has been a line of work using Near-neighbor Search techniques such as Locality-sensitive hashing (Gionis et al., 1999) and Graph-based indexing (Malkov et al., 2014) for efficient deep neural network training or inference (Zhang et al., 2018; Chen et al., 2019; 2020a; Kitaev et al., 2020; Chen et al., 2021b;a; Liu et al., 2022).

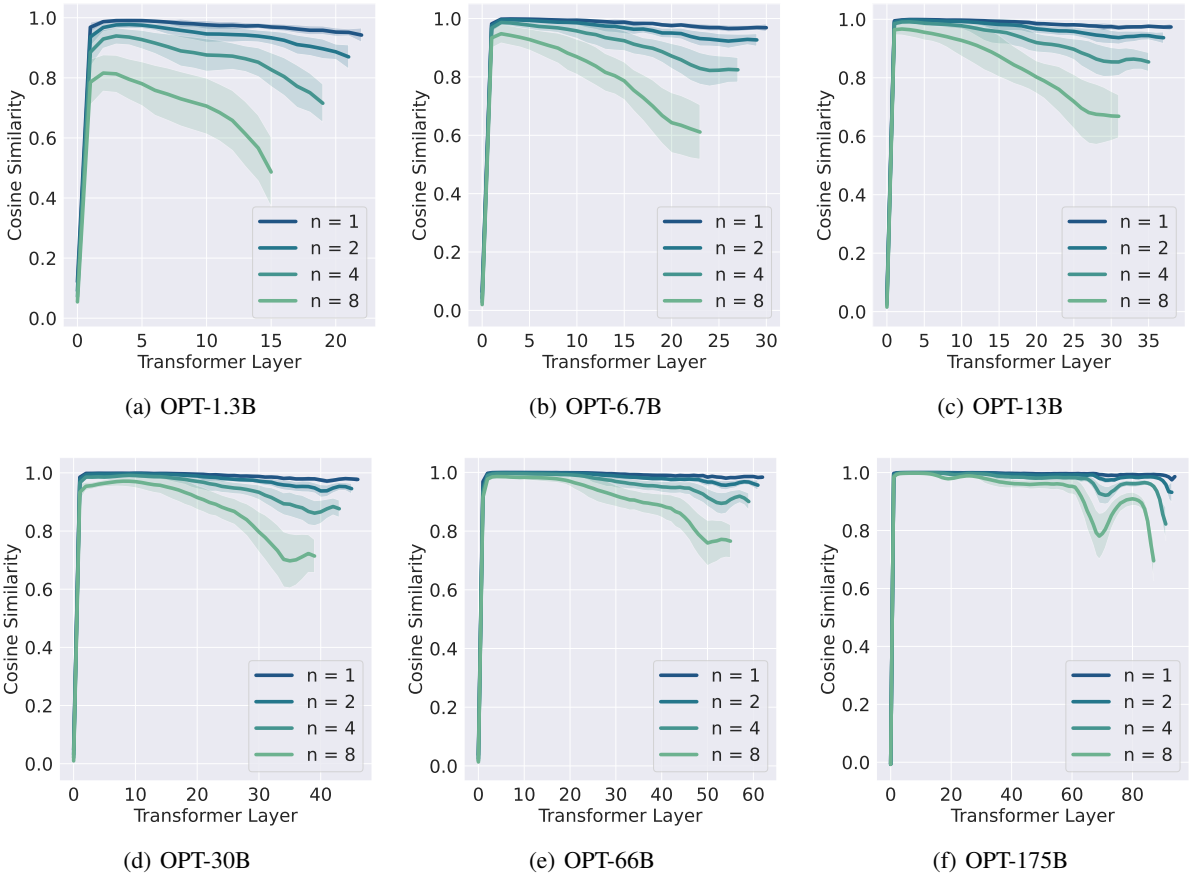
Quantization, pruning, distillation for LLM inference. Various system relaxations have been studied for decades for model inference in machine learning. For example, quantization (Han et al., 2015; Jacob et al., 2018; Nagel et al., 2019; Zhao et al., 2019), pruning (Molchanov et al., 2016; Liu et al., 2018; He et al., 2019; Hoefler et al., 2021), and distillation (Hinton et al., 2015; Cho & Hariharan, 2019; Tang et al., 2019; Touvron et al., 2021) have been applied to speed up the inference of the machine learning model. Active research has recently attempted to apply such techniques in LLM inference. For example, zeroQuant (Yao et al., 2022) and nuQmm (Park et al., 2022) implement customized CUDA kernels to support tenor-wise or group-wise quantization for LLM inference; LLM.int8 (Dettmers et al., 2022) adopts a mixed `INT8/FP16` computation to diminish the influence of activation outliers; SmoothQuant (Xiao et al., 2022) enables efficient 8-bit weight and activation for LLM inference; GPTQ (Frantar et al., 2022) adopts a one-shot weight quantization method based on approximate second-order information for accuracy and efficiency; SparseGPT (Frantar & Alistarh, 2023) introduces an approximate sparse regression solver to enable the sparsity in LLM inference; (Bansal et al., 2022) has reported that a small set of attention heads can perform primitive induction operations associated with in-context learning, and use this property to prune LLM for acceleration.

Residual connections in neural networks. Residual connection shows great advantages for neural network generalization, it provides additional paths for activations to reach the latter parts of the neural network by skipping some layers (He et al., 2016). The advancement of residual connections can be viewed as ensembles of multiple shallow neural networks (Veit et al., 2016). Plenty of active research has discussed the effectiveness of residual connections (Balduzzi et al., 2017; Bello et al., 2021; Allen-Zhu & Li, 2019; Frei et al., 2019). However, as far as we know, there is no former work that leverages the property of residual connections to improve the efficiency of LLM inference.

B Additional Observation on Slowly Changing Observation

First, we present more plots on the cosine similarity between representations. Figure 9 plots the cosine similarity between activation across layers on OPT family. It is evident that similarity is high for the larger models.

There are two residual connections inside a transformer layer, one around the attention block, and the other one around the MLP block. The residual connection can be written as $X + F(X)$, where F is either the Multi-Head Attention or two MLP Layer. Figure 10 plots the cosine similarity between X and $X + F(X)$, which is close to 1.0, and the cosine similarity between



(a) OPT-1.3B (b) OPT-6.7B (c) OPT-13B

(d) OPT-30B (e) OPT-66B (f) OPT-175B

Figure 9. Cosine similarity between layer *l* and layer *l* + 1 for various model.

X and $F(X)$, which is close to 0.0. This happens because $\|X\|$ is significantly greater than $\|F(X)\|$, shown in the purple. In the first layer, $\|F(X)\|$ is larger, which explains the low cosine similarity. The magnitude of the $L2$ norm is different across models, however, we observe a similar trend with models of different sizes. There exists a normalization layer before $F(X)$ and the layer normalization scale $\|X\|$ to a consistent magnitude across layers (e.g. 85 for OPT-30B, 110 for OPT175B), but not necessarily scale down $\|X\|$.

C Additional Experiment Detail

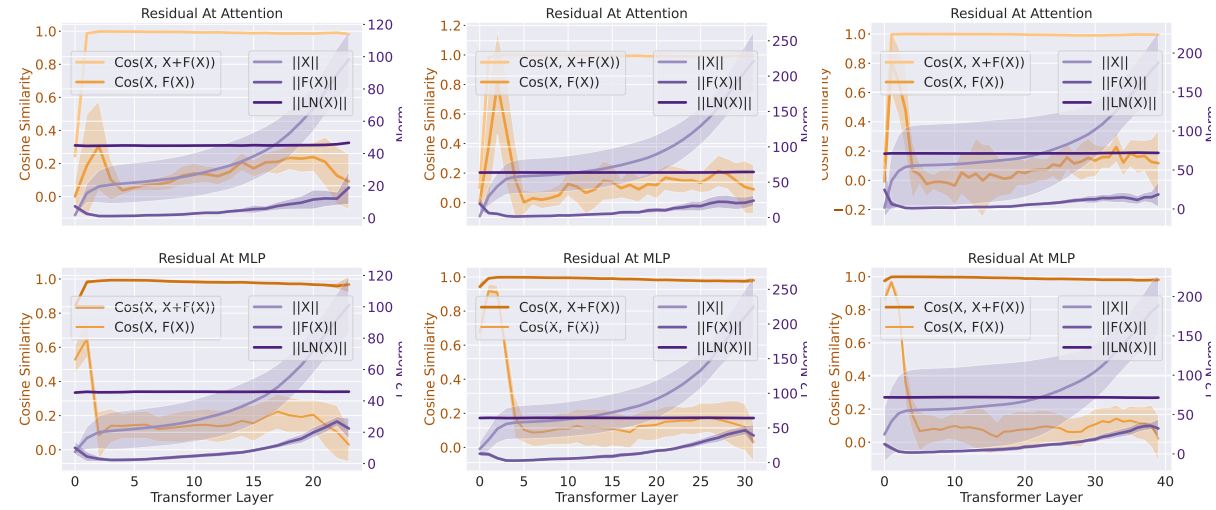
C.1 Large Batch Size

To help understand where the speed-up comes from when batch size is greater than 1, we present the Union Contextual Sparsity (fraction of neurons/heads that are not used by any of the inputs in the batch) of different batches sizes for MLP and Attention blocks, respectively, in Figure 11. Union Contextual Sparsity is calculated as 1.0 - the union of activated MLP neurons or Attention heads in the batch / total neurons or heads. The union operation is essential to realize a fast sparse GEMM.

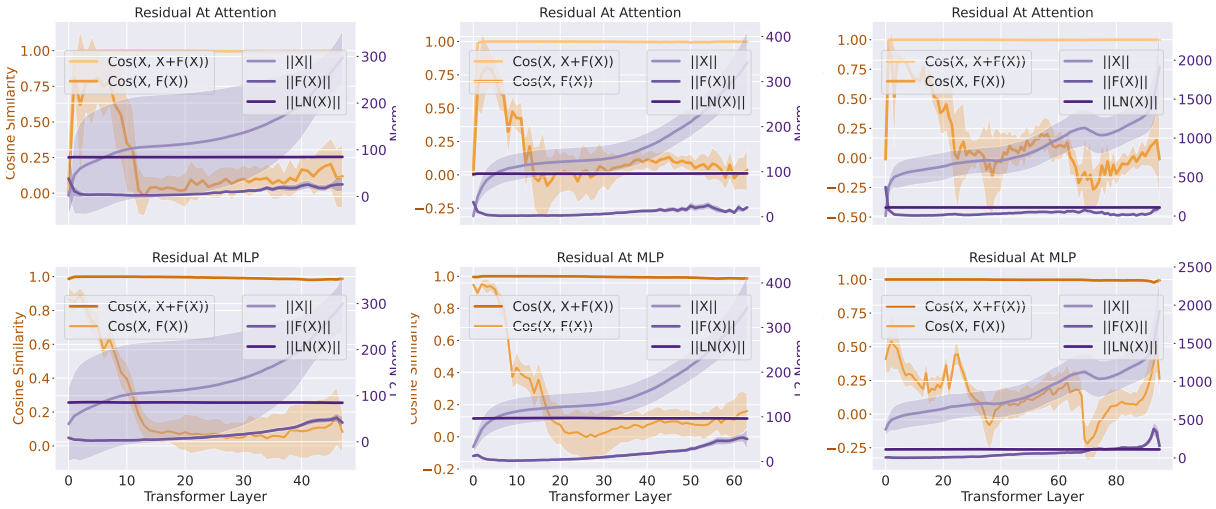
Surprisingly the number of MLP neurons/Attention heads that DEJAVU activated does not grow linearly with the batch size. This suggests a power law distribution rather than a uniform distribution of parameter access from all input examples. Further, a larger batch size can easily lead to out-of-memory for long sequence settings due to the limited GPU memory, the giant large model size, and the stored KV cache. For example, the total GPU memory of 8 80GB A100 is 640GB. Model parameters are around 350GB for OPT175B. The KV cache for a batch size 32 with a sequence longer than 1920 tokens has already filled up the GPU memory.

C.2 Near Neighbor classifier

In the DEJAVU framework, any near-neighbor search method under the inner product metric would be sufficient to predict a sparsity pattern. "Training predictor" is to reduce the cost of on-the-fly prediction, rather than training the model itself.



(a) OPT-1.3b (b) OPT-6.7b (c) OPT-13B

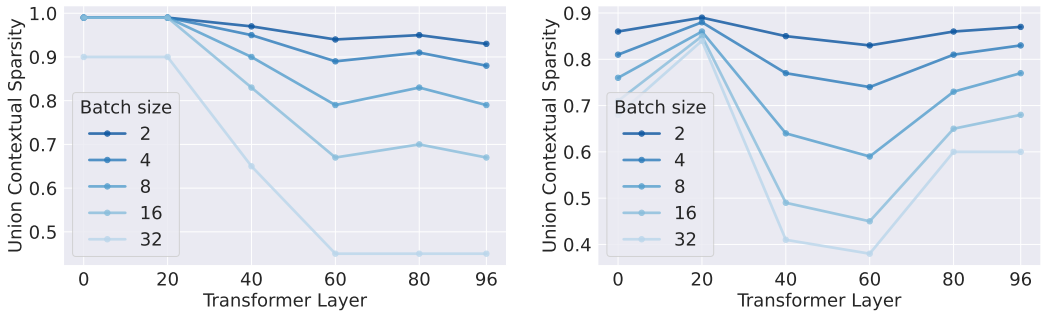


(d) OPT-30B (e) OPT-66B (f) OPT-175B

Figure 10. Cosine similarity between X and $F(X)$, and the cosine similarity between X and X^0 in orange color. $L2$ norm of X and $F(X)$ and X after layer normalization in purple on the right. Except on the first layer, kXk is significantly higher than $kF(X)k$. $kF(X)k$ is higher at the first layer, which corresponds to the low cosine similarity at the first layer.

For example, in our exploration stage mentioned in Section 4.1, we adopt HNSW, a state-of-art near-neighbor search method, to predict MLP sparse pattern, and we can see from the following table there is no drop in the perplexity at 90 % sparsity ratio. However, due to the high dimensionality of embedding and HNSW's reliance on CPU, the time HNSW took to identify the sparsity pattern is 10ms, which is longer than the MLP computation.

In our paper, we choose a neural network classifier as our near neighbor search method to take advantage of the fast matrix multiplication on GPU. And training such classifiers to predict sparsity patterns is not only cheaper in terms of training cost but also inherently different from the method concept.



(a) MLP (b) Attention

Figure 11. Union contextual sparsity with larger batch size.

	OPT-1.3B	OPT-1.3B + HNSW
Hellaswag	0.4154	0.4314
C4	14.2	14.4

Table 8. Sparsify from the Depth: Skipping or parallel entire transformer blocks may not lead to catastrophic drop in accuracy at test time.

Model	COPA	Hellaswag	Lambada	OpenBookQA	PIQA	Winogrande
OPT-175B	0.8600	0.7814	0.7584	0.4460	0.8096	0.7261
- Parallel 2	0.8300	0.7737	0.7762	0.4520	0.8030	0.7096
- Parallel 4	0.5200	0.2519	0	0.2720	0.5092	0.4870
- Skip 2/8	0.8000	0.7112	0.6387	0.4220	0.7840	0.6630
- Skip 2/4	0.6900	0.4409	0.0240	0.3400	0.6882	0.5383
Bloom	0.8000	0.7460	0.6771	0.4480	0.7949	0.7040
- Parallel 2	0.8100	0.7404	0.6992	0.4360	0.7813	0.7048
- Parallel 4	0.6200	0.3176	0.1325	0.2720	0.5593	0.5217
- Skip 2/8	0.7900	0.6829	0.5936	0.4120	0.7699	0.6614
- Skip 2/4	0.6600	0.5538	0.3023	0.3580	0.7046	0.5549

C.3 Future Possibility: Skipping Layer

Deja Vu currently sparsifies from the perspective of model width. Here, we explore the possibility of sparsification from model depth. As observed in Section 3, we show that the activation of large language models changes slowly across blocks. This property can be leveraged to increase the efficiency of a trained model by parallelizing, reordering, or skipping certain intermediate sub-blocks without significantly impacting the overall accuracy.

Setting	Wiki(ppl)	C4(ppl)
Baseline	11.57	10.17
Skip every 2 layers	21.16	16.58
Skip every 4 layers	13.45	11.37

Improving the inference efficiency of Transformer models is a challenging task due to their sequential execution of Transformer layers. Each sub-block depends on the output of the previous one, leading to low hardware efficiency, particularly during the token generation phase where each forward pass is computed for only one token. However, the sequential execution of blocks and sub-blocks yields computation bubbles, and the latter involves a large amount of communication overhead. Here, we present an interesting observation that can potentially alleviate these challenges. We found that the activation of the model changes slowly across blocks. Specifically, the cosine similarity of activations between adjacent blocks is often above 0.99. This suggests that the blocks might take the previous activation as input – parallelize or reorder the blocks – without significantly affecting the output. Slowly changing activations suggest that it may be possible to parallelize, reorder, or even skip blocks while maintaining a similar output. Some existing models, such as GPT-J (Wang & Komatsuzaki, 2021), GPT-NeoX (Black et al., 2022), and PaLM (Chowdhery et al., 2022) already placed the Attention block and MLP block

in parallel in training to facilitate parallel computation and reduce the communication overhead.

Here we investigate the possibility at inference time. And surprisingly, we found parallelizing those blocks for models that are trained in a sequence manner will not hurt the performance of downstream tasks significantly. And surprisingly, we found parallelizing those blocks for models that are trained in a sequence manner will not hurt the performance of downstream tasks significantly. TableC.3 presents some preliminary results of OPT-175B and Bloom

Given the activation y and Transformer layer l, we have:

$$\widetilde{y}_l \leftarrow y_l + \mathsf{MHA}^l(y_l)$$

$$\widehat{y}_l \leftarrow \widetilde{y}_l + \mathsf{MLP}^l(\widetilde{y}_l)$$

Parallelizing two blocks refers to placing the Attention and MLP blocks in parallel, i.e.:

$$\widehat{y}_l \leftarrow y + \mathsf{MHA}^l(y_l) + \mathsf{MLP}^l(y_l)$$

Parallelizing four blocks then parallelize the blocks of two Transformer layers, defined as follows:

$$\widehat{y}_{l+1} \leftarrow y_l + \mathsf{MHA}^l(y_l) + \mathsf{MLP}^l(y_l) + \mathsf{MHA}^{l+1}(y_l) + \mathsf{MLP}^{l+1}(y_l)$$

Skipping layers is straightforward, which drops an entire Transformer layer for every n layers.

We are surprised to find that parallel two layers preserve accuracy on a series of tasks across models. Besides, randomly skipping 25% layers doesn't lead to catastrophic quality. Our findings suggest from the downstream task perspective, the activation patterns within the model are relatively consistent across different blocks, providing a potential avenue for future research on model compression and optimization.

D Implementation Details

Figure 12 presents a more detailed workflow of DEJAVU. The left diagram shows how an input y performs the sparse MHA with selected indices 0,3, predicted by the head predictor. Similarly, the right diagram shows how an input y performs the sparse MLP with selected indices 0,2, predicted by the neuron predictor of that layer.

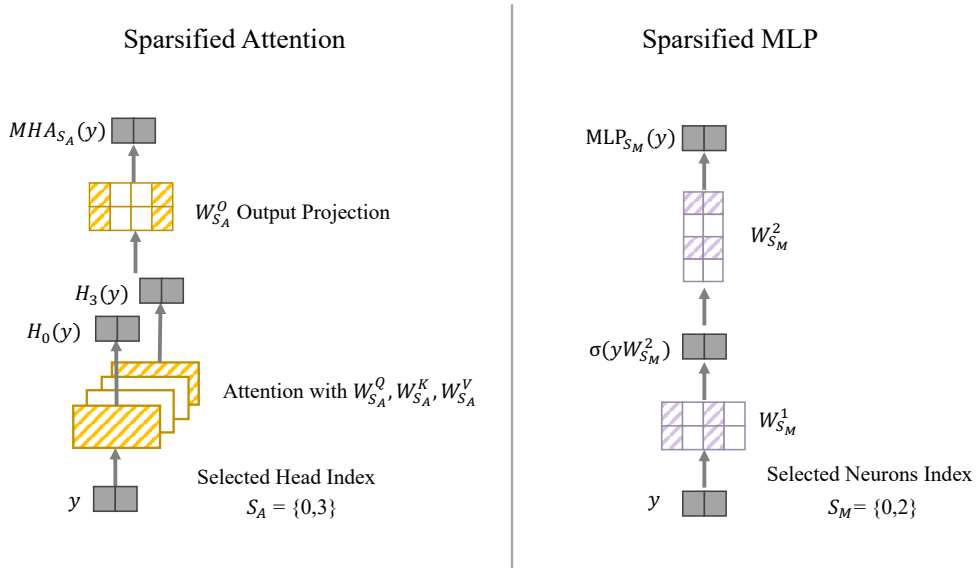


Figure 12. Detailed diagram on the sparsified computation process of MLP and Attention. Notation refers to Section 2.3

Next, we will present a general explanation of two optimizations we used in DEJAVU implementation. Kernel fusion: A standard implementation of sparse matrix-vector multiply (e.g., Wx in PyTorch) that separately indexes a subset of the matrix W[idx,:] before multiplying with input x would incur $3\times$ the amount of memory IOs: one to load a subset of W from GPU memory, one to write that subset to a different contiguous region in memory, and one to load that (now contiguous) subset in again to multiply with x. Similarly, to use sparse matrix multiply routines (e.g., cuSparse), we would first need to convert W[idx,:] to sparse format, again incurring more memory IOs. We instead fuse the indexing and the multiplication step: we load a subset of W[idx,:] to memory, along with x, perform the multiply, then write down the result. This fused implementation (in Triton (Tillet et al., 2019)) yields up to $4\times$ speedup compared to a standard PyTorch implementation (Section E). Memory coalescing: the weight matrices are conventionally stored in row-major format. This allows us to load

W[idx,:] optimally (as the second dimension is contiguous in memory). However, for cases where we need to load W[:,idx] (attention output projection and the 2nd weight matrix in the MLP) this format significantly slows down memory loading, as idx could contain indices pointing to non-contiguous memory. A simple solution is to store these matrices in column-major format (i.e., storing $W^\top$ in contiguous row-major format), then use the same fused kernel above. This transposition is done once when loading the model, and incurs no added cost during generation.

E Benchmarking Sparse MLP and Sparse Attention

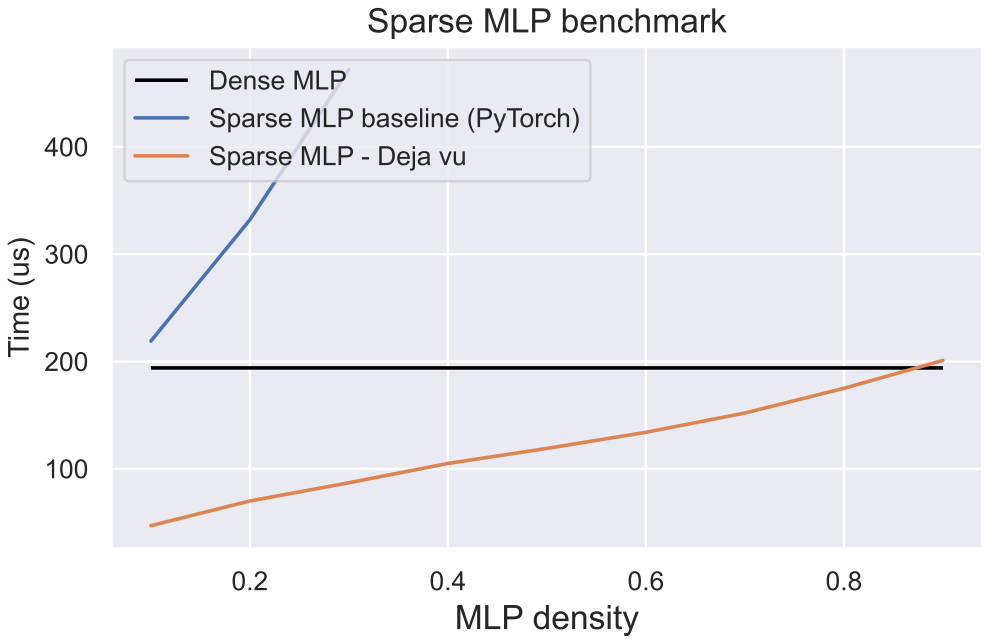


Figure 13. Speed benchmarking of the MLP layer of OPT-175B on 8xA100s. Our sparse implementation is up to 4.5 faster than the baseline implementation in PyTorch. Our sparse MLP implementation remains faster than dense MLP for density up to 0.8.

We validate that our hardware-aware implementation of sparse MLP and sparse attention (Section 4.4) yields wall-clock speed up compared to both dense MLP/attention and compared to the standard implementation in PyTorch.

Recall that our implementation fuses the sparse indexing and the multiplication $(W^1_{S_M})^\top y$ for weight matrices $(W^1)^\top$ and input vector y. In cases where we need to index $W^2_{S_M}$, we store the transpose of W^2 to ensure memory coalescing. For the baseline implementation in PyTorch, we index $(W^1_{S_M})^\top$ as a separate operation before multiplying with y, which incurs more memory reads/writes.

Similarly, we fuse the sparse indexing and the multiplication $(W^Q_{S_A})^\top y$, $(W^K_{S_A})^\top y$, $(W^V_{S_A})^\top y$ for weight matrices $(W^Q)^\top$, $(W^K)^\top$, $(W^V)^\top$ and input vector y. Note we usually concatenate all three matrices in the standard implementation, but we separate them here for clarity. In cases where we need to index $W^O_{S_A}$, we store the transpose of W^O to ensure memory coalescing.

In Figure 13 and Figure 14, our sparse MLP and attention implementations are 4-5$\times$ faster than the baseline implementation in Pytorch, and remains faster than the dense version for density up to 0.8.

F Notations and Basic Definitions

For a positive integer n, let $[n] := \{1,2,\cdots,n\}$. For a matrix $A \in \mathbb{R}^{n \quad n}$, let $A_{i,:}$ and $A_{:,j}$ be two column vectors corresponding to the i-th row and the j-th column of A respectively, and $A_{i,j}$ be the entry at the i-th row and the j-th column. For a vector $x \in \mathbb{R}^n$, let $\sqrt{x} \in \mathbb{R}^n$ denote the vector with the i-th entry being $\sqrt{x_i}$ and $\mathsf{diag}(x) \in \mathbb{R}^{n \quad n}$ denote the diagonal matrix with the i-th diagonal entry being x_i. For two matrices $A, W \in \mathbb{R}^{n \quad n}$, let $\|A\|_W := (\sum_{i=1}^n \sum_{j=1}^n W_{i,j} A_{i,j}^2)^{1/2}$ and $W \circ A$ denote the matrix where $(W \circ A)_{i,j} = W_{i,j} A_{i,j}$. For matrix $W \in \mathbb{R}^{n \quad n}$, let $D_{W_i} := \mathsf{diag}(W_{i,:})$ with $i \in [n]$.

For two vectors $x \in \mathbb{R}^n$ and $w \in \mathbb{R}^n{}_0$, let $\|x\|_w := (\sum_{i=1}^n w_i x_i^2)^{1/2}$. For a vector x, we denote $\|x\|_2 := (\sum_{i=1}^n x_i^2)^{1/2}$ as its

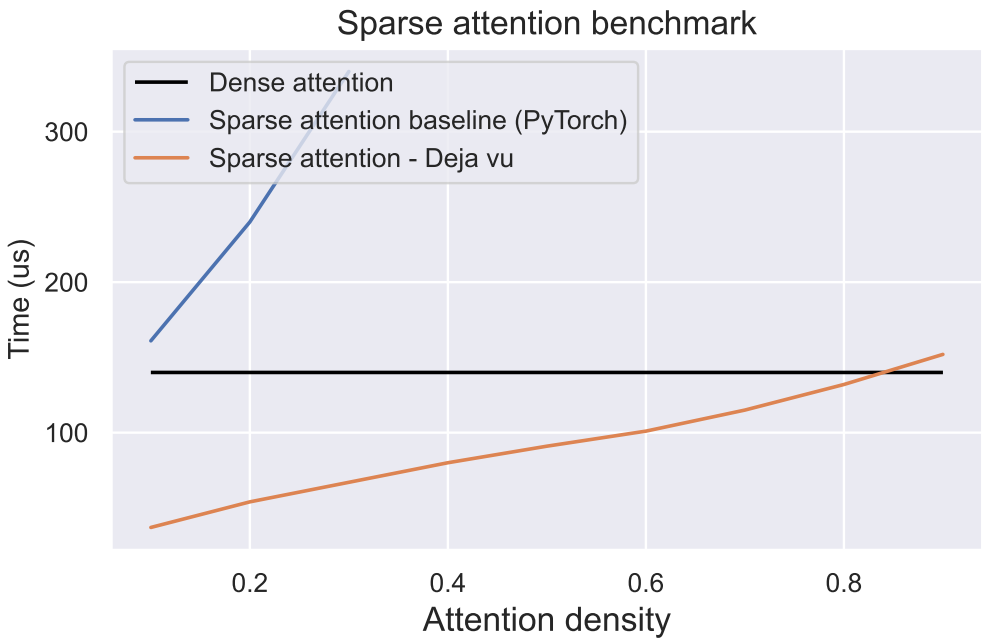


Figure 14. Speed benchmarking of the attention layer of OPT-175B on 8xA100s. Our sparse implementation is up to 5 faster than the baseline implementation in PyTorch. Our sparse attention implementation remains faster than dense MLP for density up to 0.8.

ℓ_2 norm. We denote $\|x\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$ as its ℓ_p norm. For a square matrix A, we denote $\mathsf{tr}[A]$ as the trace of matrix A. For a matrix $A \in \mathsf{R}^{n \quad k}$ (suppose $n \geq k$), we use $\|A\|$ to denote its spectral norm, i.e., $\|A\| = \sup_x \|Ax\|_2 / \|x\|_2$. We use $\|A\|_F$ to denote its Frobenius norm $\|A\|_F := (\sum_{i=1}^n \sum_{j=1}^k A_{i,j}^2)^{1/2}$.

Suppose matrix $A \in \mathsf{R}^{n \quad k}$ has SVD decomposition $U \quad V^\top$ where $U \in \mathsf{R}^{n \quad k}$ (this matrix has orthonormal columns), $\in \mathsf{R}^{k \quad k}$ is a diagonal matrix, and $V \in \mathsf{R}^{k \quad k}$. We call columns of U are singular vectors. We use $A^\dagger \in \mathsf{R}^{k \quad n}$ to denote the Moore-Penrose pseudoinverse, then $A^\dagger = V \quad ^1 U^\top$. Suppose $\in \mathsf{R}^{k \quad k}$ is sorted diagonal matrix, let $\sigma_1, \cdots, \sigma_k$ denote the diagonal entries of . Then we call σ_i the i-th singular value of matrix, and we write it as $\sigma_i(A)$.

For any symmetric matrix $B \in \mathsf{R}^{k \quad k}$, we define its eigenvalue decomposition as $U \quad U^\top$, where is a diagonal matrix. Let $\lambda_1, \cdots, \lambda_k$ denote the entries on diagonal of $\in \mathsf{R}^{k \quad k}$. We say λ_i is the i-th eigenvalue. Usually we write it as $\lambda_i(B)$.

The connection between eigenvalues and singular values is

$$\sigma_i^2(A) = \lambda_i(A^\top A)$$

We use notation $A \succeq 0$ to denote that matrix A is positive semidefinite (psd). Mathematically, $A \succeq 0$ means for all vectors x, we have $x^\top A x \geq 0$.

Similarly, for two squarer matrices A and B, we use $A \succeq B$ to denote the case where for all vectors x, $x^\top A x \geq x^\top B x$.

We use $\Pr[]$ and $\mathsf{E}[]$ for probability and expectation. We denote $\max\{a,b\}$ as the maximum between a and b. We denote $\min\{a,b\}$ (resp. $\max\{a,b\}$) as the minimum (reps. maximum) between a and b.

Throughout, for non-negative real numbers a and b, we use the notation $a = (1 \pm \epsilon) b$ if $a \in [(1-\epsilon)b, (1+\epsilon)b]$.

G Subspace Embeddings and Norm Preserving

In Section G.1, we show the norm preserving of the soft-max functions. In Section G.2, we show the norm preserving of the ReLU function. In Section G.3, we introduce the folded Guassian distribution. In Section G.4, we introduce the ℓ_2 subspace embedding. In Section G.5, we introduce the ℓ_1 subspace embedding. In Section G.6, we introduce different sketching matrices for subspace embedding.

G.1 Soft-Max Functions

Let $K \in \mathbb{R}^{s \times d}$ and $V \in \mathbb{R}^{d \times s}$. Inspired by the softmax unit in the attention scheme of large language models. The softmax related regression has been studied in many settings (Zandieh et al., 2023; Alman & Song, 2023; Brand et al., 2023; Li et al., 2023b; Deng et al., 2023b;a; Gao et al., 2023a; Li et al., 2023a; Gao et al., 2023b). In this work, we follow the standard softmax definition. We define $\sigma_1 : \mathbb{R}^s \to \mathbb{R}^s$ to be a softmax function, i.e., for any vector $y \in \mathbb{R}^s$, the $\sigma(y)$ can be written as

$$\sigma_1(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^d \exp(y_j)}, \quad \forall i \in [d]$$

The standard softmax is ℓ_1 version. In this work, we also consider the ℓ_2 generalization. We define $\sigma_2 : \mathbb{R}^s \to \mathbb{R}^s$ to be a softmax function (ℓ_2 version), i.e., for any vector $y \in \mathbb{R}^s$, the $\sigma(y)$ can be written as

$$\sigma_2(y)_i = \frac{\exp(y_i)}{(\sum_{j=1}^d \exp(2y_j))^{1/2}}, \quad \forall i \in [d]$$

We define function $f : \mathbb{R}^d \to \mathbb{R}^d$

$$f(x) = V \cdot (\sigma(K \cdot x)) \tag{3}$$

Definition G.1. We say $\mathcal{X} \subset \mathbb{R}^d$ is a rank-k subspace, if there is an orthonormal basis $U \in \mathbb{R}^{d \times k}$, for any $x \in \mathcal{X}$, there is $y \in \mathbb{R}^k$ such that

$$x = Uy.$$

We can have

Lemma G.2. *Let $\tau \in (0,1)$. Let $\mathcal{X} \subset \mathbb{R}^d$ denote a subspace with rank k. Let f be defined based on σ_2 function. Let $\overline{V}$ is a random Gaussian matrices with $d \geq (\epsilon^{-2}(k + \log(1/\delta)))$ rows. Let $V = \tau\overline{V}$, then we have with probability $1-\delta$*

$$(1-\epsilon)\tau\|x\|_2 \leq \|f(x)\| \leq (1+\epsilon)\tau\|x\|_2.$$

for all unit vectors $x \in \mathcal{X}$.

Further, if $d = O(k + \log(1/\delta))$, then we have

$$0.5\tau\|x\|_2 \leq \|f(x)\| \leq 2\tau\|x\|_2.$$

Remark G.3. The above condition implies that f is a shrinking operator but also not shrinking arbitrarily small.

Proof. Given $d \geq (\epsilon^{-2}(k + \log(1/\delta)))$, by using Lemma G.11 , we have

$$(1-\epsilon)\|y\|_2 \leq \|\overline{V}y\|_2 \leq (1+\epsilon)\|y\|_2$$

As the input of the function f here is the output of a softmax function (ℓ_2 version), we know that $\|y\|_2 = 1$.

Thus, we have

$$(1-\epsilon) \leq \|\overline{V}y\|_2 \leq (1+\epsilon)$$

By rescaling V, we have

$$(1-\epsilon)\|x\|_2 \leq \|Vy\|_2 \leq (1+\epsilon)\|x\|_2.$$

□

Lemma G.4. *Let $\tau \in (0,1)$. Let $\mathcal{X} \subset \mathbb{R}^d$ denote a subspace with rank k. Let f be defined based on σ_1 function. Suppose $\overline{V}$ is a random Gaussian matrix with $d \geq ((k + \log(1/\delta)))$ rows. Let $V = \frac{1}{2}\tau\overline{V}$.*

Then we have

$$\frac{1}{4\sqrt{s}}\tau \cdot \|x\|_2 \leq \|f(x)\|_2 \leq \tau \cdot \|x\|_2$$

for all unit vectors x.

Proof. By property of subspace embedding, we know that if $d \geq (\epsilon^{-2}(s + \log(1/\delta)))$,

$$(1-\epsilon)\|y\|_2 \leq \|\overline{V}y\|_2 \leq (1+\epsilon)\|y\|_2$$

By property of function of f, we know we only need to care $\|y\|_1 = 1$, this implies that

$$\frac{1}{\sqrt{s}}\|y\|_1 \leq \|y\|_2 \leq \|y\|_1$$

On one hand, we have

$$\begin{aligned}\|\overline{V}y\|_2 &\leq (1+\epsilon)\cdot\|y\|_2 \\ &\leq (1+\epsilon)\cdot\|y\|_1 \\ &= (1+\epsilon), \end{aligned} \tag{4}$$

where the first step follows from $\|\overline{V}y\|_2 \leq (1+\epsilon)\|y\|_2$, the second step follows from $\|y\|_2 \leq \|y\|_1$ and the last step follows from $\|y\|_1 = 1$.

On the other hand, we have

$$\begin{aligned}\|\overline{V}y\|_2 &\geq (1-\epsilon)\|y\|_2 \\ &\geq \frac{1}{\sqrt{s}}(1-\epsilon)\|y\|_1 \\ &= \frac{1}{\sqrt{s}}(1-\epsilon), \end{aligned} \tag{5}$$

where the first step follows from $(1-\epsilon)\|y\|_2 \leq \|\overline{V}y\|_2$, the second step follows from $\frac{1}{\sqrt{s}}\|y\|_1 \leq \|y\|_2$ and the last step follows from $\|y\|_1 = 1$.

Combining Eq. (5)and Eq. (4) together, we have

$$(1-\epsilon)\frac{1}{\sqrt{s}} \leq \|\overline{V}y\|_2 \leq (1+\epsilon)$$

Choosing $\epsilon = 1/2$, we have

$$\frac{1}{2\sqrt{s}} \leq \|\overline{V}y\|_2 \leq 2.$$

By $V = \frac{1}{2}\tau\overline{V}$ and $\|x\|_2 = 1$, we have

$$\frac{1}{4\sqrt{s}}\tau\|x\|_2 \leq \|Vy\|_2 \leq \tau\|x\|_2.$$

□

G.2 ReLU Functions

We use $\phi : \mathbb{R} \to \mathbb{R}$ to denote ReLU function, i.e., $\phi(z) = \max\{z, 0\}$.

We define function $g : \mathbb{R}^d \to \mathbb{R}^d$

$$g(x) = V \cdot (\phi(K \cdot x)) \tag{6}$$

Let $K \in \mathbb{R}^{s \quad d}$ and $V \in \mathbb{R}^{d \quad s}$.

Lemma G.5. *Let $\mathcal{X} \subset \mathbb{R}^d$ denote a rank-k subspace. Let K denote a random Gaussian matrix. Let V denote a random Gaussian matrix. Let $s \geq \quad (\epsilon \quad ^2 k \log(1/(\delta\epsilon)))$. Let $d \geq \quad (\epsilon \quad ^2 (k + \log(1/\delta)))$. Then we know with high probability $1-\delta$, for all unit vector $x \in \mathcal{X}$*

$$(1-\epsilon)\|x\|_2 \leq \|f(x)\|_2 \leq (1+\epsilon)\|x\|_2$$

Proof. Suppose $s \geq \quad (\epsilon \quad ^2 \log(1/\delta))$.

Using Lemma G.6, Fact G.7, we can show that for each fixed

$$(1-\epsilon)\|x\|_2 \leq \|\phi(Kx)\|_2 \leq (1+\epsilon)\|x\|_2$$

holds with probability $1-\delta$.

By a standard ϵ-net argument (Lemma G.9), the net points in $\mathcal{X}$ is at most $(10/\epsilon)^{O(k)}$.

Taking a union bound over all the net points, we can show that for all $x \in \mathcal{X}$

$$(1-\epsilon)\|x\|_2 \leq \|\phi(Kx)\|_2 \leq (1+\epsilon)\|x\|_2$$

holds with probability $1-\delta/2$ and $s \geq \quad (\epsilon \quad ^2 k \log(1/(\delta\epsilon)))$.

Further, we using Lemma G.11, we can show that

$$(1-\epsilon)\|\phi(Kx)\|_2 \leq \|f(x)\|_2 \leq (1+\epsilon)\|\phi(Kx)\|_2$$

holds with probability $1-\delta/2$.

Combining together,

$$(1-\epsilon)^2\|x\|_2 \leq \|f(x)\|_2 \leq (1+\epsilon)^2\|x\|_2$$

holds with probability $1-\delta$.

Rescaling the ϵ, we complete the proof.

□

G.3 Folded Gaussian Distribution

We state a standard tool from literature,

Lemma G.6 (Lemma 1 on page 1325 of Laurent and Massart (Laurent & Massart, 2000))**.** *Let $X \sim \chi^2_k$ be a chi-squared distributed random variable with k degrees of freedom. Each one has zero means and σ^2 variance.*

Then,

$$\Pr[X - k\sigma^2 \geq (2\sqrt{kt} + 2t)\sigma^2] \leq \exp(-t)$$
$$\Pr[k\sigma^2 - X \geq 2\sqrt{kt}\sigma^2] \leq \exp(-t)$$

Further if $k \geq$ (ϵ ^2t) and $t \geq$ ($\log(1/\delta)$), then we have

$$\Pr[|X - k\sigma^2| \leq \epsilon k\sigma^2] \leq \delta.$$

We prove the following property,

Fact G.7. *Let $h,q \in \mathbb{R}^p$ be fixed vectors and $h \neq 0, W \in \mathbb{R}^m$ ^{p} be random matrix with i.i.d. entries $W_{i,j} \sim \mathcal{N}(0, \frac{2}{m})$, and vector $v \in \mathbb{R}^m$ defined as $v_i = \phi((Wh)_i) = \mathbf{1}_{(W(h+q))_i}$ $_0(Wh)_i$. Then,*

- *$|v_i|$ follows i.i.d. from the following distribution: with half probability $|v_i| = 0$, and with the other half probability $|v_i|$ follows from folded Gaussian distributions $|\mathcal{N}(0, \frac{2khk^2}{m})|$.*
- *$\frac{mkvk^2}{2khk^2}$ is in distribution identical to χ^2_ω (chi-square distribution of order ω) where ω follows from binomial distribution $\mathcal{B}(m, 1/2)$.*

Proof. We assume each vector W_i is generated by first generating a gaussian vector $g \sim \mathcal{N}(0, \frac{2I}{m})$ and then setting $W_i = \pm g$ where the sign is chosen with half-half probability. Now, $|\langle W_i, h\rangle| = |\langle g, h\rangle|$ only depends on g, and is in distribution identical to $|\mathcal{N}(0, \frac{2khk^2}{m})|$. Next, after the sign is determined, the indicator $\mathbf{1}_{hW_i, h+qi}$ $_0$ is 1 with half probability and 0 with another half. Therefore, $|v_i|$ satisfies the aforementioned distribution. As for $\|v\|^2$, letting $\omega \in \{0,1,...,m\}$ be the variable indicator how many indicators are 1 , then $\omega \sim \mathcal{B}(m, 1/2)$ and $\frac{mkvk^2}{2khk^2} \sim \chi^2_\omega$. □

G.4 ℓ_2 subspace embedding

We define a standard notion in sketching technique.[3]

Definition G.8 (ℓ_2 subspace embedding (Sarlos, 2006))**.** A $(\epsilon, \delta, \ell_2)$-subspace embedding for the column space of an $n \times d$ matrix A is a matrix S for which

$$\Pr[\forall x \in \mathbb{R}^d, \|SAx\|_2^2 = (1 \pm \epsilon)\|Ax\|_2^2] \geq 1 - \delta.$$

The above condition is equivalent to

$$\Pr[\|U^\top U - U^\top S^\top SU\| \leq \epsilon] \geq 1 - \delta.$$

where the U is the orthonormal basis of A.

For the reason of above conditions are equivalent, we refer the readers to the survey (Woodruff, 2014).

We state a standard tool in literature,

Lemma G.9 (Lemma 5 in (Woodruff, 2014))**.** *Let $\mathcal{X} \subset \mathbb{R}^d$ be rank k. For any $\gamma \in (0,1)$, there is a γ-net N of $\mathcal{X}$ for which $|N| \leq (1 + 4/\gamma)^k$.*

[3] We remark that sketching technique has widely applied to many applications such as linear regression, low-rank approximation (Clarkson & Woodruff, 2013; Nelson & Nguyên, 2013; Lu et al., 2013; Boutsidis et al., 2016; Cohen, 2016; Razenshteyn et al., 2016; Song et al., 2017; 2019), linear programming (Song & Yu, 2021; Dong et al., 2021; Jiang et al., 2021; Gu & Song, 2022), semi-definite programming (Gu & Song, 2022; Song et al., 2023b), empirical risk minimization(Lee et al., 2019; Qin et al., 2023b), training over-parameterized neural network (Brand et al., 2021; Song et al., 2021; Alman et al., 2022; Hu et al., 2022; Zhang, 2022).

G.5 ℓ_1 subspace embedding

When $p = 1$, using Cauchy random variables, Sohler and Woodruff (Sohler & Woodruff, 2011) showed there exist ℓ_1 oblivious subspace embeddings with $O(d\log d)$ rows and $\kappa = O(d\log d)$. This approach was generalized by using p-stable random variables in work of Meng and Mahoney (Meng & Mahoney, 2013) to ℓ_p-norms when $1 < p < 2$, where they showed there exist ℓ_p oblivious subspace embeddings with $O(d\log d)$ rows and $\kappa = O((d\log d)^{1/p})$. Unlike the case when $p = 2$, due to the large distortion

In (Wang & Woodruff, 2018), they show for every $1 \le p < 2$, any oblivious subspace embedding with dimension r has distortion $\kappa = (\frac{1}{(\frac{1}{d})^{1/p}\ \log^{2/p} r + (\frac{r}{n})^{1/p\ \ 1/2}})$. They also give sparse oblivious subspace embeddings for every $1 \le p < 2$ which are optimal in dimension and distortion, up to poly $(\log d)$ factors. Importantly for $p = 1$, they achieve $r = O(d\log d), \kappa = O(d\log d)$ and $s = O(\log d)$ non-zero entries per column.

Definition G.10 (ℓ_1 subspace embedding)**.** Let $0 < \alpha < \beta$ be parameters. We will say a matrix S is an ℓ_1 subspace embedding for an $n \times d$ matrix A if there are constants $c_1, c_2 > 0$ so that for all $x \in \mathbb{R}^d$,

$$\|Ax\| \le \|SAx\|_1 \le d^{c_1}\|Ax\|_1,$$

and S has at most d^{c_2} rows.

G.6 Random Matrices

Matrices	b	Time for $R \cdot A$	Reference
Random Gaussian	$\epsilon^{\ 2}(d + \log(1/\delta))$	$\mathcal{T}_{\text{mat}}(b,n,d)$	Thm. 6 of (Woodruff, 2014)
SRHT	$\epsilon^{\ 2}(\sqrt{d} + \sqrt{\log n})^2 \log(d/\delta)$	$nd\log(\epsilon^{\ 1} d(\log n))$	Thm. 7 of (Woodruff, 2014)
AMS	$\epsilon^{\ 2}(d + \log(1/\delta))$	$\mathcal{T}_{\text{mat}}(b,n,d)$	Follow from JL guarantee
Count-sketch	$\epsilon^{\ 2}\delta^{\ 1}d^2$	$\text{nnz}(A)$	Thm. 9 of (Woodruff, 2014)
Sparse embedding	$\epsilon^{\ 2}d \cdot$ poly $\log(d/(\epsilon\delta))$	$\epsilon^{\ 1}\text{nnz}(A)$ poly $\log(d/(\epsilon\delta))$	Thm. 10 (2) of (Woodruff, 2014)
Sparse embedding	$\epsilon^{\ 2}d^{1+\gamma}$	$\epsilon^{\ 1}\text{nnz}(A)\text{poly}(1/\gamma)$	Thm. 10 (1) of (Woodruff, 2014)

Table 9. Summary for different sketching matrices for subspace embedding. The sketching matrix R has size $b\ \ n$. The vectors are from the column subspace of matrix A with size $n\ \ d$: $2\ (0;1)$ is the error parameter, and $2\ (0;1)$ is the probability parameter. $T_{\text{mat}}\ (a;b;c)$ denotes the running time of fast matrix multiplication of two matrices with size $a\ \ b$ and $b\ \ c$: In the first sparse embedding matrix, each column has $s\ \ ^1$ poly $\log(d=(\ \ ))$ non-zero entries; In the second sparse embedding matrix, each column has $s\ \ ^1$ poly $(1=\ \ )$ non-zero entries, > 0 is a tunable parameter that gives different trade-offs, and $\ \ $ can be as small as $1=$ poly (d): For count-sketch matrices, the subspace embedding guarantee is proved from JL moment property, instead of directly from JL guarantee.

Lemma G.11 (Theorem 6 of (Woodruff, 2014))**.** *Let $0 < \epsilon, \delta < 1$ and $S = \frac{1}{k}R \in \mathbb{R}^{k\ \ n}$ where the entries $R_{i,j}$ of R are independent standard normal random variables. Then if $k = \ \ (\epsilon^{\ 2}(d + \log(1/\delta)))$, then for any fixed $n \times d$ matrix A, with probability $1-\delta$, S is a $(1\pm\epsilon)\ell_2$-subspace embedding for A, that is, simultaneously for all $x \in \mathbb{R}^d$, $\|SAx\|_2 = (1\pm\epsilon)\|Ax\|_2$. Here $C > 0$ is an absolute constant.*

We consider several standard sketching matrices:

1. Random Gaussian matrices.
2. Subsampled randomized Hadamard/Fourier transform (SRHT) matrices (Lu et al., 2013).
3. AMS sketch matrices (Alon et al., 1996), random $\{-1, +1\}$ per entry.
4. Count-Sketch matrices (Charikar et al., 2002), each column only has one non-zero entry, and is $-1, +1$ half probability each.
5. Sparse embedding matrices (Nelson & Nguyên, 2013), each column only has s non-zero entries, and each entry is $-\frac{1}{s}, +\frac{1}{s}$ half probability each.
6. Uniform sampling matrices.

Definition G.12 (Random Gaussian matrix)**.** We say $R \in \mathbb{R}^{b\ \ n}$ is a random Gaussian matrix if all entries are sampled from $\mathcal{N}(0, 1/b)$ independently.

Definition G.13 (Subsampled randomized Hadamard/Fourier transform matrix (Lu et al., 2013))**.** We say $R \in \mathbb{R}^{b \times n}$ is a subsampled randomized Hadamard transform (SRHT) matrix [4] if it is of the form $R = \sqrt{n/b} SHD$, where $S \in \mathbb{R}^{b \times n}$ is a random matrix whose rows are b uniform samples (without replacement) from the standard basis of $\mathbb{R}^n$, $H \in \mathbb{R}^{n \times n}$ is a normalized Walsh-Hadamard matrix, and $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix whose diagonal elements are i.i.d. Rademacher random variables.

Definition G.14 (AMS sketch matrix (Alon et al., 1996))**.** Let $h_1, h_2, \cdots, h_b$ be b random hash functions picking from a 4-wise independent hash family $\mathcal{H} = \{h : [n] \to \{-\frac{1}{\sqrt{b}}, +\frac{1}{\sqrt{b}}\}\}$. Then $R \in \mathbb{R}^{b \times n}$ is a AMS sketch matrix if we set $R_{i,j} = h_i(j)$

Definition G.15 (Count-sketch matrix (Charikar et al., 2002))**.** Let $h : [n] \to [b]$ be a random 2-wise independent hash function and $\sigma : [n] \to \{-1, +1\}$ be a random 4-wise independent hash function. Then $R \in \mathbb{R}^{b \times n}$ is a count-sketch matrix if we set $R_{h(i),i} = \sigma(i)$ for all $i \in [n]$ and other entries to zero.

Definition G.16 (Sparse embedding matrix I (Nelson & Nguyên, 2013))**.** We say $R \in \mathbb{R}^{b \times n}$ is a sparse embedding matrix with parameter s if each column has exactly s non-zero elements being $\pm 1/\sqrt{s}$ uniformly at random, whose locations are picked uniformly at random without replacement (and independent across columns) [5].

Definition G.17 (Sparse embedding matrix II (Nelson & Nguyên, 2013))**.** Let $h : [n] \times [s] \to [b/s]$ be a random 2-wise independent hash function and $\sigma : [n] \times [s] \to \{-1, 1\}$ be a 4-wise independent. Then $R \in \mathbb{R}^{b \times n}$ is a sparse embedding matrix II with parameter s if we set $R_{(j-1)b/s + h(i,j), i} = \sigma(i,j)/\sqrt{s}$ for all $(i,j) \in [n] \times [s]$ and all other entries to zero [6].

Definition G.18 (Uniform sampling matrix)**.** We say $R \in \mathbb{R}^{b \times n}$ is a uniform sampling matrix if it is of the form $R = \sqrt{n/b} SD$, where $S \in \mathbb{R}^{b \times n}$ is a random matrix whose rows are b uniform samples (without replacement) from the standard basis of $\mathbb{R}^n$, and $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix whose diagonal elements are i.i.d. Rademacher random variables.

H Distances, Angles, and Inner Product

Most of the properties in this section are very standard in literature, e.g., see (Gu et al., 2023).

Let $U \in \mathbb{R}^{n \times k}$ denote an orthonormal basis, we use $U_\perp \in \mathbb{R}^{n \times (n-k)}$ denote the matrix such that $UU^\top + U_\perp U_\perp^\top = I_n$.

Definition H.1. Let $X \in \mathbb{R}^{n \times k}$ and $Y \in \mathbb{R}^{n \times k}$.

For any matrix X, and for orthogonal matrix Y ($Y^\top Y = I_k$) we define

- $\tan\theta(Y,X) := \|Y_\perp^\top X (Y^\top X)^{-1}\|$

For orthogonal matrices Y and X ($Y^\top Y = I_k$ and $X^\top X = I_k$), we define

- $\cos\theta(Y,X) := \sigma_{\min}(Y^\top X)$.
  - It is obvious that $\cos(Y,X) = 1/\|(Y^\top X)^{-1}\|$ and $\cos(Y,X) \le 1$.
- $\sin\theta(Y,X) := \|(I - YY^\top)X\|$.
  - It is obvious that $\sin\theta(Y,X) = \|Y_\perp Y_\perp^\top X\| = \|Y_\perp^\top X\|$ and $\sin\theta(Y,X) \le 1$.
- $\mathrm{dist}(Y,X) := \min_{Q \in O_k} \|YQ - X\|$

where O_k is the set of $k \times k$ orthogonal matrices.

Lemma H.2 (Structural lemma for orthogonal matrices)**.** *Let $X, Y \in \mathbb{R}^{n \times k}$ be orthogonal matrices. Then*

$$(Y^\top X)_\perp = Y_\perp^\top X.$$

Proof. Let us first compute the Gram of $Y^\top X$, which is

$$\begin{aligned} X^\top Y Y^\top X &= X^\top (I - Y_\perp Y_\perp^\top) X \\ &= X^\top X - X^\top Y_\perp Y_\perp^\top X \\ &= I_k - X^\top Y_\perp Y_\perp^\top X, \end{aligned}$$

[4] In this case, we require $\log n$ o be an integer.

[5] For our purposes the signs need only be $O(\log d)$-wise independent, and each column can be specified by a $O(\log d)$-wise independent permutation, and the seeds specifying the permutations in different columns need only be $O(\log d)$-wise independent.

[6] This definition has the same behavior as sparse embedding matrix I for our purpose

where the first step follows from $Y_? Y_?^\top + Y Y^\top = I$, the second step follows from simple algebra, and the last step follows from X is an orthogonal matrix, so $X^\top = X^{-1}$.

This means that $(Y^\top X)_? = Y_?^\top X$. □

Lemma H.3 (Orthogonal and inverse share singular vectors)**.** *Let $A \in \mathbb{R}^{k \times k}$ be non-singular, then $A_?$ and A^{-1} have the same set of singular vectors. Consequently, $\|A_? A^{-1}\| = \|A_?\| \|A^{-1}\|$.*

Proof. Let $A \in \mathbb{R}^{k \times k}$ and $A^\top A + A_?^\top A_? = I_k$, we will show that $\|A_? A^{-1}\| = \|A_?\| \|A^{-1}\|$. Let $x \in \mathbb{R}^k$ be the unit eigenvector of A that realizes the spectral norm, note that

$$\|A_? x\|_2^2 = 1 - \|A\|^2,$$

we argue that x corresponds to the smallest singular value of $A_?$ via contradiction. Suppose there exists some unit vector y with $\|A_? y\|_2 < \|A_? x\|_2$, by definition, we know that $\|A_? y\|_2^2 + \|Ay\|_2^2 = 1$, this means that $\|Ay\|_2 > \|Ax\|_2 = \|A\|$, contradicts the definition of spectral norm. Similarly, if z is the unit vector that realizes the spectral norm of $A_?$, then it is also singular vector corresponds to the smallest singular value of A, or equivalently, the spectral norm of A^{-1}. Our above argument essentially implies that $A_?$ and A^{-1} have the same set of singular vectors. The proof is then straightforward: suppose $A_? z = \lambda z$ and $A^{-1} z = \mu z$, then

$$\begin{aligned} A_? A^{-1} z &= A_? \mu z \\ &= \mu (A_? z) \\ &= \lambda \mu z, \end{aligned}$$

where the first step follows from our assumption, the second step follows from μ is a real number and a real number multiplying a matrix is commutative and follows from the associative property, and the third step follows from our assumption.

Thus, we have $\|A_? A^{-1}\| = \|A_?\| \|A^{-1}\|$, and we have proved the assertion. □

Lemma H.4. *Let $X, Y \in \mathbb{R}^{n \times k}$ be orthogonal matrices, then*

$$\tan\theta(Y,X) = \frac{\sin\theta(Y,X)}{\cos\theta(Y,X)}.$$

Proof. Due to Lemma H.2, we have $(Y^\top X)_? = Y_?^\top X$. Thus, $\tan\theta(Y,X) = \|(Y^\top X)_? (Y^\top X)^{-1}\|$. The proof then follows straightforwardly from Lemma H.3. □

Lemma H.5. *Let $X, Y \in \mathbb{R}^{n \times k}$ be orthogonal matrices, then* $\sin^2\theta(Y,X) + \cos^2\theta(Y,X) = 1$.

Proof. Recall that $\cos\theta(Y,X) = \frac{1}{\|(Y^\top X)^{-1}\|}$ and $\sin\theta(Y,X) = \|Y_?^\top X\|$, by Lemma H.2, we know that $(Y^\top X)_? = Y_?^\top X$, therefore $\sin\theta(Y,X) = \|(Y^\top X)_?\|$. Let $A := Y^\top X$, by Lemma H.3, we know that $A_?$ and A^{-1} have the same singular vectors, or equivalently, the singular vector realizing $\|A_?\|$ corresponds to the smallest singular value of A. Let $z \in \mathbb{R}^k$ be the unit singular vector with singular value $\|A_?\|$, then

$$\begin{aligned} z^\top A^\top A z + z^\top A_?^\top A_? z &= 1, \\ \|A_?\|^2 + \sigma_{\min}^2(A) &= 1, \\ \cos^2\theta(Y,X) + \sin^2\theta(Y,X) &= 1. \end{aligned}$$

This completes the proof. □

H.1 Angle is close

Lemma H.6. *Let $\epsilon \in (0, 0.1)$ Let x denote a unit vector, i.e., $\|x\|_2 = 1$.*

Let $z = (x+y)/\|x+y\|_2$.

If $\|y\|_2 \leq \epsilon \cdot \|x\|_2$, then

$$\sqrt{1 - \langle x, z \rangle^2} \leq 2\sqrt{\epsilon}$$

Proof. We have

$$\begin{aligned} \|x+y\|_2 &\geq \|x\|_2 - \|y\|_2 \\ &\geq 1 - \epsilon \end{aligned}$$

where the first step follows from triangle inequality.

We also have

$$\begin{aligned} \|x+y\|_2 &\leq \|x\|_2 + \|y\|_2 \\ &\leq 1+\epsilon \end{aligned} \tag{7}$$

We have

$$(1-\epsilon)^2 \geq 1-2\epsilon \tag{8}$$

We also have

$$\frac{1}{(1+\epsilon)^2} \geq 1-3\epsilon \tag{9}$$

where $\epsilon \in (0, 0.1)$.

Combining Eq. (8) and Eq. (9), we have

$$\begin{aligned} \frac{1}{(1+\epsilon)^2} \cdot (1-\epsilon)^2 &\geq (1-2\epsilon)\cdot(1-3\epsilon) \\ &= 1-5\epsilon+6\epsilon^2 \\ &\geq 1-5\epsilon+\epsilon \\ &= 1-4\epsilon \end{aligned} \tag{10}$$

where the first step follows from Eq. (8) and Eq. (9) and the rest of them follow from simple algebra.

Finally, we have

$$\begin{aligned} 1-\langle x,z\rangle^2 &= 1-\langle x, \frac{x+y}{\|x+y\|_2}\rangle^2 \\ &= 1-\frac{1}{\|x+y\|_2^2}\langle x, x+y\rangle^2 \\ &= 1-\frac{1}{\|x+y\|_2^2}\cdot(\|x\|_2^2+\langle x,y\rangle)^2 \\ &= 1-\frac{1}{\|x+y\|_2^2}\cdot(1+\langle x,y\rangle)^2 \\ &\leq 1-\frac{1}{(1+\epsilon)^2}\cdot(1+\langle x,y\rangle)^2 \\ &\leq 1-\frac{1}{(1+\epsilon)^2}\cdot(1-\epsilon)^2 \\ &\leq 1-(1-4\epsilon) \\ &= 4\epsilon, \end{aligned}$$

where the first step follow the definition of z, the second step follows from the reorganization, the third step follows from the definition of inner product, the fourth step follows from $\|x\|_2 = 1$, the fifth step follows from Eq. (7), the sixth step follows from $1+\langle x,y\rangle \geq 1-|\langle x,y\rangle| \geq 1-\|x\|_2\cdot\|y\|_2 \geq 1-\epsilon$, the seventh step follows from Eq. (10) and the last step follows from simple algebra.

□

I Function Approximations

We first we show the function approximation for two operators in Section I.1, which means that there are two functions. Then we show the function approximations for four operators in Section I.2.

I.1 Function Approximations for Two Operators

Lemma I.1. *Let $f_1 : \mathsf{R}^d \to \mathsf{R}^d$ and let $f_2 : \mathsf{R}^d \to \mathsf{R}^d$.*

Assume the the following conditions

- *Condition 1a. f_1 is a linear function*
- *Condition 1b. $\|f_1(x)\|_2 \leq \epsilon_1 \|x\|_2$ (f_1 is shrinking)*
- *Condition 1c. $\|f_1(x) - f_1(y)\|_2 \leq L_1 \|x-y\|_2$ (f_1 is Lipschitz)*

- *Condition 2a.* f_2 *is a linear function*
- *Condition 2b.* $\|f_2(x)\|_2 \leq \epsilon_2 \|x\|_2$ *(*f_2 *is shrinking)*
- *Condition 2c.* $\|f_2(x) - f_2(y)\|_2 \leq L_2 \|x-y\|_2$ *(*f_2 *is Lipschitz)*

We define three functions

- $$\begin{aligned} g_1(x) &=: (I + f_1) \cdot (I + f_2)(x) \\ &= x + f_2(x) + f_1(x + f_2(x)) \end{aligned}$$
- $$\begin{aligned} g_2(x) &=: (I + f_2) \cdot (I + f_1)(x) \\ &= x + f_1(x) + f_2(x + f_1(x)) \end{aligned}$$
- $$\begin{aligned} g_3(x) &=: (I + f_1 + f_2)(x) \\ &= x + f_1(x) + f_2(x) \end{aligned}$$

Then we can show that

- *Part 1.* $\|g_1(x) - g_2(x)\|_2 \leq 2\epsilon_1\epsilon_2\|x\|_2$*(if* f_1 *and* f_2 *are linear functions)*
- *Part 2.* $\|g_1(x) - g_2(x)\|_2 \leq (\epsilon_2 \cdot L_1 + \epsilon_1 \cdot L_2)\|x\|_2$ *(if* f_1 *and* f_2 *are Lipschitz functions)*
- *Part 3.* $\|g_1(x) - g_3(x)\|_2 \leq \epsilon_1\epsilon_2\|x\|_2$ *(if* f_1 *is a linear function)*
- *Part 4.* $\|g_1(x) - g_3(x)\|_2 \leq \epsilon_2 \cdot L_1\|x\|_2$ *(if* f_1 *is a Lipschitz function)*
- *Part 5.* $\|g_2(x) - g_3(x)\|_2 \leq \epsilon_1\epsilon_2\|x\|_2$ *(if* f_2 *is a linear function)*
- *Part 6.* $\|g_2(x) - g_3(x)\|_2 \leq \epsilon_1 \cdot L_2\|x\|_2$ *(if* f_2 *is a Lipschitz function)*

Proof. **Part 1.**

We have

$$\begin{aligned} \|g_1(x) - g_2(x)\|_2 &\leq \|g_1(x) - g_3(x)\|_2 + \|g_3(x) - g_2(x)\|_2 \\ &\leq \epsilon_1\epsilon_2\|x\|_2 + \epsilon_1\epsilon_2\|x\|_2 \\ &= 2\epsilon_1\epsilon_2\|x\|_2 \end{aligned}$$

where the first step follows from triangular inequality, the second step follows from Part 3 and Part 5 and the last step follows from simple algebra.

Part 2.

We have

$$\begin{aligned} \|g_1(x) - g_2(x)\|_2 &\leq \|g_1(x) - g_3(x)\|_2 + \|g_3(x) - g_2(x)\|_2 \\ &\leq \epsilon_2 \cdot L_1\|x\|_2 + \epsilon_1 \cdot L_2\|x\|_2 \\ &= (\epsilon_2 \cdot L_1 + \epsilon_1 \cdot L_2)\|x\|_2 \end{aligned}$$

where the first step follows from triangular inequality, the second step follows from Part 4 and Part 6 and the last step follows from simple algebra.

Part 3.

We have

$$\begin{aligned}\|g_1(x)-g_3(x)\|_2 &= \|f_1(x+f_2(x))-f_1(x)\|_2\\ &= \|f_1(x+f_2(x)-x)\|_2\\ &= \|f_1(f_2(x))\|_2\\ &\le \epsilon_1\cdot\|f_2(x)\|_2\\ &\le \epsilon_1\cdot\epsilon_2\cdot\|x\|_2,\end{aligned}$$

where the first step follows from the definition of g_1 and g_3, the second step follows from the fact that f_1 is a linear function, the third step follows from simple algebra, the fourth step follows from Condition 1b and the last step follows from Condition 2b.

Part 4.

$$\begin{aligned}\|g_1(x)-g_3(x)\|_2 &= \|f_1(x+f_2(x))-f_1(x)\|_2\\ &\le L_1\cdot\|x+f_2(x)-x\|_2\\ &= L_1\cdot\|f_2(x)\|_2\\ &\le L_1\cdot\epsilon_2\|x\|_2,\end{aligned}$$

where the first step follows from definition of g_1 and g_3, the second step follows from Condition 1c, the third step follows from simple algebra and the last step follows from Condition 2b. □

Part 5.

We have

$$\begin{aligned}\|g_2(x)-g_3(x)\|_2 &= \|f_2(x+f_1(x))-f_2(x)\|_2\\ &= \|f_2(x+f_1(x)-x)\|_2\\ &= \|f_2(f_1(x))\|_2\\ &\le \epsilon_2\cdot\|f_1(x)\|_2\\ &\le \epsilon_2\cdot\epsilon_1\cdot\|x\|_2,\end{aligned}$$

where the first step follows from the definition of g_2 and g_3, the second step follows from the fact that f_2 is a linear function, the third step follows from simple algebra, the fourth step follows from Condition 2b and the last step follows from Condition 1b.

Part 6.

$$\begin{aligned}\|g_2(x)-g_3(x)\|_2 &= \|f_2(x+f_1(x))-f_2(x)\|_2\\ &\le L_2\cdot\|x+f_1(x)-x\|_2\\ &= L_2\cdot\|f_1(x)\|_2\\ &\le L_2\cdot\epsilon_1\|x\|_2,\end{aligned}$$

where the first step follows from definition of g_1 and g_3, the second step follows from Condition 2c, the third step follows from simple algebra and the last step follows from Condition 1b.

I.2 Function Approximations for Four Operators

Lemma I.2. *For each $i\in[4]$, we assume the following conditions*

- *i(a) f_i is a linear function*
- *i(b) $\|f_i(x)\|_2\le\epsilon_i\|x\|_2$ (f_i is shriking)*
- *i(c) $\|f_i(x)-f_i(y)\|_2\le L_i\|x-y\|_2$ (f_i is Lipschitz)*

We define three functions

- $g_1(x) := (I+f_1)\cdot(I+f_2)\cdot(I+f_3)\cdot(I+f_4)(x)$
- $g_2(x) := (I+f_1)\cdot(I+f_3)\cdot(I+f_2)\cdot(I+f_4)(x)$
- $g_3(x) := (I+f_1+f_2+f_3+f_4)(x)$

Then, we can show that

- *Part 1.* $\|g_1(x)-g_2(x)\|_2 \leq 2(\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2$ *(if* $f_i, \forall i \in [4]$ *are linear functions)*
- *Part 2.* $\|g_1(x)-g_2(x)\|_2 \leq (2L_1\epsilon_2+2L_1\epsilon_3+2L_1\epsilon_4+L_2\epsilon_3+2L_2\epsilon_4+2L_3\epsilon_4+2L_1\epsilon_2\epsilon_3+2L_1\epsilon_2\epsilon_4+2L_1\epsilon_3\epsilon_4+L_2\epsilon_3\epsilon_4+2L_1\epsilon_2\epsilon_3\epsilon_4+L_3\epsilon_2+L_3\epsilon_2\epsilon_4)\|x\|_2$ *(if* $f_i, \forall i \in [4]$ *are Lipschitz functions)*
- *Part 3.* $\|g_1(x)-g_3(x)\|_2 \leq (\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2$ *(if* $f_i, \forall i \in [4]$ *are linear functions)*
- *Part 4.* $\|g_1(x)-g_3(x)\|_2 \leq (L_1\epsilon_2+L_1\epsilon_3+L_1\epsilon_4+L_2\epsilon_3+L_2\epsilon_4+L_3\epsilon_4+L_1\epsilon_2\epsilon_3+L_1\epsilon_2\epsilon_4+L_1\epsilon_3\epsilon_4+L_2\epsilon_3\epsilon_4+L_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2$ *(if* $f_i, \forall i \in [4]$ *are Lipschitz functions)*
- *Part 5.* $\|g_2(x)-g_3(x)\|_2 \leq (\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2$ *(if* $f_i, \forall i \in [4]$ *are linear functions)*
- *Part 6.* $\|g_2(x)-g_3(x)\|_2 \leq (L_1\epsilon_2+L_1\epsilon_3+L_1\epsilon_4+L_2\epsilon_4+L_3\epsilon_2+L_3\epsilon_4+L_1\epsilon_2\epsilon_3+L_1\epsilon_2\epsilon_4+L_1\epsilon_3\epsilon_4+L_3\epsilon_2\epsilon_4+L_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2$
  (if $f_i, \forall i \in [4]$ *are Lipschitz functions)*

Proof. **Part 1.**

We have

$$\begin{aligned}\|g_1(x)-g_2(x)\|_2 &\leq \|g_1(x)-g_3(x)\|_2+\|g_3(x)-g_2(x)\|_2\\ &\leq 2(\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2\end{aligned}$$

where the first step follows from triangular inequality and the last step follows from Part 3 and Part 5.

Part 2.

We have

$$\begin{aligned}\|g_1(x)-g_2(x)\|_2 &\leq \|g_1(x)-g_3(x)\|_2+\|g_3(x)-g_2(x)\|_2\\ &\leq (2L_1\epsilon_2+2L_1\epsilon_3+2L_1\epsilon_4+L_2\epsilon_3+2L_2\epsilon_4+2L_3\epsilon_4+2L_1\epsilon_2\epsilon_3+2L_1\epsilon_2\epsilon_4+2L_1\epsilon_3\epsilon_4+\\ &\quad L_2\epsilon_3\epsilon_4+2L_1\epsilon_2\epsilon_3\epsilon_4+L_3\epsilon_2+L_3\epsilon_2\epsilon_4)\|x\|_2\end{aligned}$$

where the first step follows from triangular inequality and the last step follows from Part 4 and Part 6.

Part 3. We have

$$\begin{aligned}\|g_1(x)-g_3(x)\|_2 &= \|(I+f_1)\cdot(I+f_2)\cdot(I+f_3)\cdot(x+f_4(x))-(I+f_1+f_2+f_3+f_4)(x))\|_2\\ &= \|(x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+\\ &\quad f_3(x+f_4(x)))+f_1(x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x))))\\ &\quad -(I+f_1+f_2+f_3+f_4)(x)))\|_2\\ &= \|f_3(f_4(x))+f_2(f_4(x)+f_3(x+f_4(x)))+f_1(f_4(x)+\\ &\quad f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x)))))\|_2\\ &= \|f_3(f_4(x))+f_2(f_4(x))+f_2(f_3(x))+f_2(f_3(f_4(x)))+\\ &\quad f_1(f_4(x))+f_1(f_3(x))+f_1(f_3(f_4(x)))+f_1(f_2(x))+f_1(f_2(f_4(x)))\\ &\quad +f_1(f_2(f_3(x)))+f_1(f_2(f_3(f_4(x))))\|_2\\ &\leq \|f_3(f_4(x))\|_2+\|f_2(f_4(x))\|_2+\|f_2(f_3(x))\|_2+\|f_2(f_3(f_4(x)))\|_2+\\ &\quad \|f_1(f_4(x))\|_2+\|f_1(f_3(x))\|_2+\|f_1(f_3(f_4(x)))\|_2+\|f_1(f_2(x))\|_2+\|f_1(f_2(f_4(x)))\|_2+\\ &\quad \|f_1(f_2(f_3(x)))\|_2+\|f_1(f_2(f_3(f_4(x))))\|_2\\ &\leq (\epsilon_3\epsilon_4+\epsilon_2\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_4+\epsilon_1\epsilon_3+\epsilon_1\epsilon_3\epsilon_4+\epsilon_1\epsilon_2+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2\\ &= (\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2,\end{aligned}$$

where the first step follows from the definition of g_1 and g_3, the second step follows from simple algebra, the third step follows from reorganization, the fourth step follows from the fact that all $f_i, \forall i \in [4]$ are linear function, the fifth step follows from triangular inequality, the sixth step follows from $i(b)$ and the last step follows from reorganization.

Part 4. We have

$$
\begin{aligned}
\|g_1(x)-g_3(x)\|_2 &= \|(I+f_1)\cdot(I+f_2)\cdot(I+f_3)\cdot(x+f_4(x))-(I+f_1+f_2+f_3+f_4)(x))\|_2 \\
&= \|x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+ \\
&\quad f_3(x+f_4(x)))+f_1(x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x))) \\
&\quad -(I+f_1+f_2+f_3+f_4)(x))\|_2 \\
&= \|f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x))) \\
&\quad +f_1(x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x)))) \\
&\quad -f_1(x)-f_2(x)-f_3(x))\|_2 \\
&= \|f_3(x+f_4(x))-f_3(x)+f_2(x+f_4(x)+f_3(x+f_4(x)))-f_2(x) \\
&\quad +f_1(x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x))))-f_1(x)\| \\
&\leq L_3\|x+f_4(x)-x\|_2+L_2\|x+f_4(x)+f_3(x+f_4(x))-x\|_2 \\
&\quad +L_1\|x+f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x)))-x\|_2 \\
&\leq L_3\|f_4(x)\|_2+L_2\|f_4(x)+f_3(x+f_4(x))\|_2 \\
&\quad +L_1\|f_4(x)+f_3(x+f_4(x))+f_2(x+f_4(x)+f_3(x+f_4(x)))\|_2 \\
&\leq L_3\epsilon_4\|x\|_2+L_2\epsilon_4\|x\|_2+L_2\epsilon_3\|x+f_4(x)\|_2 \\
&\quad +L_1\epsilon_4\|x\|_2+L_1\epsilon_3\|x+f_4(x)\|_2+L_1\epsilon_2\|x+f_4(x)+f_3(x+f_4(x))\|_2 \\
&\leq L_3\epsilon_4\|x\|_2+L_2\epsilon_4\|x\|_2+L_2\epsilon_3\|x\|+L_2\epsilon_3\epsilon_4\|x\|_2 \\
&\quad +L_1\epsilon_4\|x\|_2+L_1\epsilon_3\|x\|_2+L_1\epsilon_3\epsilon_4\|x\|_2+L_1\epsilon_2\|x\|_2+L_1\epsilon_2\epsilon_4\|x\|_2+L_1\epsilon_2\epsilon_3\|x+f_4(x)\|_2 \\
&\leq L_3\epsilon_4\|x\|_2+L_2\epsilon_4\|x\|_2+L_2\epsilon_3\|x\|+L_2\epsilon_3\epsilon_4\|x\|_2 \\
&\quad +L_1\epsilon_4\|x\|_2+L_1\epsilon_3\|x\|_2+L_1\epsilon_3\epsilon_4\|x\|_2+L_1\epsilon_2\|x\|_2+L_1\epsilon_2\epsilon_4\|x\|_2+L_1\epsilon_2\epsilon_4\|x\|_2+L_1\epsilon_2\epsilon_3\epsilon_4\|x\|_2 \\
&= (L_3\epsilon_4+L_2\epsilon_4+L_2\epsilon_3+L_2\epsilon_3\epsilon_4+L_1\epsilon_4+L_1\epsilon_3+L_1\epsilon_3\epsilon_4+L_1\epsilon_2+L_1\epsilon_2\epsilon_4+L_1\epsilon_2\epsilon_3+L_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2 \\
&= (L_1\epsilon_2+L_1\epsilon_3+L_1\epsilon_4+L_2\epsilon_3+L_2\epsilon_4+L_3\epsilon_4+L_1\epsilon_2\epsilon_3+L_1\epsilon_2\epsilon_4+L_1\epsilon_3\epsilon_4+L_2\epsilon_3\epsilon_4+L_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2
\end{aligned}
$$

where the first step follows from the definition of g_1 and g_3, the second step follows from simple algebra, the third step follows from simple algebra, the fourth step follows from reorganization, the fifth step follows from the fact that all $f_i, \forall i \in [4]$ are Lipschitz functions, the sixth step follows from simple algebra, the seventh step follows from $i(b)$, the eighth step follows from triangular inequality, the ninth step follows from $i(b)$, the tenth step follows from $i(b)$ and the last step follows from reorganization.

Part 5. We have

$$
\begin{aligned}
\|g_2(x)-g_3(x)\|_2 &= \|(I+f_1)\cdot(I+f_3)\cdot(I+f_2)\cdot(x+f_4(x))-(I+f_1+f_2+f_3+f_4)(x))\|_2 \\
&= \|(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+ \\
&\quad f_2(x+f_4(x)))+f_1(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+f_2(x+f_4(x)))) \\
&\quad -(I+f_1+f_2+f_3+f_4)(x)))\|_2 \\
&= \|f_2(f_4(x))+f_3(f_4(x))+ \\
&\quad f_3(f_2(x+f_4(x)))+f_1(f_4(x))+f_1(f_2(x+f_4(x)))+f_1(f_3(x+f_4(x)+f_2(x+f_4(x)))))\|_2 \\
&\leq (\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_3\epsilon_2+\epsilon_3\epsilon_2\epsilon_4+\epsilon_1\epsilon_4+\epsilon_1\epsilon_2+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3+\epsilon_1\epsilon_3\epsilon_4+\epsilon_1\epsilon_3\epsilon_2+\epsilon_1\epsilon_3\epsilon_2\epsilon_4)\|x\|_2 \\
&= (\epsilon_1\epsilon_2+\epsilon_1\epsilon_3+\epsilon_1\epsilon_4+\epsilon_2\epsilon_3+\epsilon_2\epsilon_4+\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_4+\epsilon_1\epsilon_3\epsilon_4+\epsilon_2\epsilon_3\epsilon_4+\epsilon_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2,
\end{aligned}
$$

where the first step follows from the definition of g_2 and g_3, the second step follows from simple algebra, the third step follows from the fact that all $f_i, \forall i \in [4]$ are linear function, the fourth step follows from triangular inequality and $i(b)$, and the last step follows from reorganization.

Part 6. We have

$$\begin{aligned}
\|g_2(x)-g_3(x)\|_2 &= \|(I+f_1)\cdot(I+f_3)\cdot(I+f_2)\cdot(x+f_4(x))-(I+f_1+f_2+f_3+f_4)(x))\|_2 \\
&= \|(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+f_2(x+f_4(x))) \\
&\quad +f_1(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+f_2(x+f_4(x)))) \\
&\quad -(I+f_1+f_2+f_3+f_4)(x)))\|_2 \\
&= \|f_2(x+f_4(x))-f_2(x)+f_3(x+f_4(x)+f_2(x+f_4(x)))-f_3(x) \\
&\quad +f_1(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+f_2(x+f_4(x))))-f_1(x)\|_2 \\
&\leq \|f_2(x+f_4(x))-f_2(x)\|_2+\|f_3(x+f_4(x)+f_2(x+f_4(x)))-f_3(x)\|_2 \\
&\quad +\|f_1(x+f_4(x)+f_2(x+f_4(x))+f_3(x+f_4(x)+f_2(x+f_4(x))))-f_1(x)\|_2 \\
&\leq L_2\epsilon_4\|x\|_2+L_3\epsilon_4\|x\|_2+L_3\epsilon_2\|x+f_4(x)\|_2 \\
&\quad +L_1\epsilon_4\|x\|_2+L_1\epsilon_2\|x+f_4(x)\|_2+L_1\epsilon_3\|x+f_4(x)+f_2(x+f_4(x))\|_2 \\
&\leq L_2\epsilon_4\|x\|_2+L_3\epsilon_4\|x\|_2+L_3\epsilon_2\|x\|_2+L_3\epsilon_2\epsilon_4\|x\|_2 \\
&\quad +L_1\epsilon_4\|x\|_2+L_1\epsilon_2\|x\|_2+L_1\epsilon_2\epsilon_4\|x\|_2+L_1\epsilon_3\|x\|+L_1\epsilon_3\epsilon_4\|x\|_2+L_1\epsilon_3\epsilon_2\|x\|_2+L_1\epsilon_3\epsilon_2\epsilon_4\|x\|_2 \\
&= (L_1\epsilon_2+L_1\epsilon_3+L_1\epsilon_4+L_2\epsilon_4+L_3\epsilon_2+L_3\epsilon_4+L_1\epsilon_2\epsilon_3+L_1\epsilon_2\epsilon_4+L_1\epsilon_3\epsilon_4+L_3\epsilon_2\epsilon_4+L_1\epsilon_2\epsilon_3\epsilon_4)\|x\|_2
\end{aligned}$$

where the first step follows from the definition of g_2 and g_3, the second step follows from simple algebra, the third step follows from reorganization, the fourth step follows from triangular inequality, the fifth step follows from the fact that all $f_i, \forall i \in [4]$ are Lipschitz functions and $i(b)$, the sixth step follows from triangular inequality, and the last step follows from reorganization.

□

J Nearest Neighbor Search Data Structure

We use the reduction-based approximate MaxIP method with LSH data-structure to achieve sublinear iteration cost. Note that we choose this method due to its clear theoretical guarantee on the retrieval results. It is well-known that an LSH data-structures is used for approximate nearest neighbor problem. The following definition of approximate nearest neighbor search is very standard in literature (Arya & Mount, 1993; Indyk & Motwani, 1998a; Datar et al., 2004; Andoni et al., 2014; 2015; Andoni & Razenshteyn, 2015; Indyk & Wagner, 2018; Andoni et al., 2017; 2018; Dong et al., 2019; Chen et al., 2020b; Li & Li, 2022; Li et al., 2019).

J.1 LSH and MaxIP

We start with the defining the Approximate Nearest Neighbor (ANN) problem (Arya & Mount, 1993; Indyk & Motwani, 1998a; Datar et al., 2004; Andoni et al., 2014; 2015; Andoni & Razenshteyn, 2015; Indyk & Wagner, 2018; Andoni et al., 2017; 2018; Dong et al., 2019; Chen et al., 2020b) as:

Definition J.1 (Approximate Nearest Neighbor (ANN))**.** Let $\overline{c} > 1$ and $r \in (0,2)$ denote two parameters. Given an n-vector set $Y \subset \mathbb{S}^{d-1}$ on a unit sphere, the objective of the $(\overline{c},r)$-Approximate Nearest Neighbor (ANN) is to construct a data structure that, for any query $x \in \mathbb{S}^{d-1}$ such that $\min_{y \in Y} \|y-x\|_2 \leq r$, it returns a vector z from Y that satisfies $\|z-x\|_2 \leq \overline{c} \cdot r$.

The ANN problem can be solved via locality sensitive hashing (LSH) (Indyk & Motwani, 1998a; Datar et al., 2004; Indyk & Wagner, 2018). In this paper, we use the standard definitions of LSH (see Indyk and Motwani (Indyk & Motwani, 1998a)).

Definition J.2 (Locality Sensitive Hashing)**.** Let $\overline{c} > 1$ denote a parameter. Let $p_1, p_2 \in (0,1)$ denote two parameters and $p_1 > p_2$. We say a function family $\mathcal{H}$ is $(r, \overline{c} \cdot r, p_1, p_2)$-sensitive if and only if, for any vectors $x, y \in \mathbb{R}^d$, for any h chosen uniformly at random from $\mathcal{H}$, we have:

- if $\|x-y\|_2 \leq r$, then $\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \geq p_1$,
- if $\|x-y\|_2 \geq \overline{c} \cdot r$, then $\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \leq p_2$.

Next, we show that LSH solves ANN problem with sublinear query time complexity.

Theorem J.3 (Andoni, Laarhoven, Razenshteyn and Waingarten (Andoni et al., 2017))**.** *Let $\overline{c} > 1$ and $r \in (0,2)$ denote two parameters. One can solve $(\overline{c},r)$-ANN on a unit sphere in query time $O(d \cdot n^{\rho})$ using preprocessing time $O(dn^{1+o(1)})$ and space $O(n^{1+o(1)} + dn)$, where $\rho = \frac{2}{\overline{c}^2} - \frac{1}{\overline{c}^4} + o(1)$.*

Here we write $o(1)$ is equivalent to $O(1/\sqrt{\log n})$. Note that we could reduce d to $n^{o(1)}$ with Johnson–Lindenstrauss Lemma (Johnson & Lindenstrauss, 1984). Besides, we could achieve better ρ using LSH in (Andoni & Razenshteyn, 2015) if we allowed to have more proprocessing time.

In this work, we focus on a well-known problem in computational complexity: approximate MaxIP. In this work, we follow the standard notation in (Chen, 2018) and define the approximate MaxIP problem as follows:

Definition J.4 (Approximate MaxIP)**.** Let $c \in (0,1)$ and $\tau \in (0,1)$ denote two parameters. Given an n-vector dataset $Y \subset \mathsf{S}^{d-1}$ on a unit sphere, the objective of the (c,τ)-MaxIP is to construct a data structure that, given a query $x \in \mathsf{S}^{d-1}$ such that $\max_{y \in Y} \langle x,y \rangle \geq \tau$, it retrieves a vector z from Y that satisfies $\langle x,z \rangle \geq c \cdot \max_{y \in Y} \langle x,y \rangle$.

In many applications, it is more convenient to doing inner product search in a transformed/projected space compared to doing inner product search in the original space. Thus, we propose the following definitions (Definition J.5 and Definition J.6)

Definition J.5 (Projected MaxIP)**.** Let $\phi, \psi : \mathsf{R}^d \to \mathsf{R}^k$ denote two transforms. Given a data set $Y \subseteq \mathsf{R}^d$ and a point $x \in \mathsf{R}^d$, we define (ϕ,ψ)-MaxIP as follows:

$$(\phi,\psi)\text{-MaxIP}(x,Y) := \max_{y \in Y} \langle \phi(x), \psi(y) \rangle$$

Definition J.6 (Projected approximate MaxIP)**.** Let $\phi, \psi : \mathsf{R}^d \to \mathsf{R}^k$ denote two transforms. Given an n-vector dataset $Y \subset \mathsf{R}^d$ so that $\psi(Y) \subset \mathsf{S}^{k-1}$, the goal of the (c,ϕ,ψ,τ)-MaxIP is to construct a data structure that, given a query $x \in \mathsf{R}^d$ and $\phi(x) \in \mathsf{S}^{k-1}$ such that $\max_{y \in Y} \langle \phi(x), \psi(y) \rangle \geq \tau$, it retrieves a vector $z \in Y$ that satisfies $\langle \phi(x), \psi(z) \rangle \geq c \cdot (\phi,\psi)\text{-MaxIP}(x,Y)$.

J.2 Connections

Fact J.7. *Let $\widetilde{x}$ denote the vector that $\langle \widetilde{x}, x \rangle \geq 1 - \frac{1}{2}\epsilon^2$, where both $\widetilde{x}$ and x are unit vectors. We have*

$$\|\widetilde{x} - x\|_2 \leq \epsilon$$

Proof.

$$\begin{aligned} \|\widetilde{x} - x\|_2 &= (\|\widetilde{x}\|_2^2 + \|x\|_2^2 - 2\langle x, \widetilde{x} \rangle)^{1/2} \\ &= (2 - 2\langle x, \widetilde{x} \rangle)^{1/2} \\ &\leq (2 - 2(1 - \frac{1}{2}\epsilon^2))^{1/2} \\ &= \epsilon \end{aligned}$$

Now, we complete the proof. □

Lemma J.8. *Let $\widetilde{x}$ denote the vector that $\langle \widetilde{x}, x \rangle \geq 1 - \frac{1}{2}\epsilon^2$, where both $\widetilde{x}$ and x are unit vectors. Let $0.01c \cdot \tau > \epsilon$. Suppose there is a $z \in Y$, where $\|z\|_2 = 1$, such that*

$$\langle x, z \rangle \geq c \cdot \max_{y \in Y} \langle x, y \rangle$$

Note that $\max_{y \in Y} \langle x,y \rangle \geq \tau$. Then, we can find a $z \in Y$ such that

$$\langle \widetilde{x}, z \rangle \geq \frac{1}{2} c \cdot \max_{y \in Y} \langle x, y \rangle$$

Proof. We have

$$\begin{aligned} \langle \widetilde{x}, z \rangle &= \langle x, z \rangle + \langle \widetilde{x} - x, z \rangle \\ &\geq \langle x, z \rangle - |\langle \widetilde{x} - x, z \rangle| \\ &\geq \langle x, z \rangle - \|\widetilde{x} - x\|_2 \cdot \|z\|_2 \\ &\geq \langle x, z \rangle - \epsilon \\ &\geq c \cdot \max_{y \in Y} \langle x, y \rangle - \epsilon \\ &\geq 0.99 \cdot c \cdot \max_{y \in Y} \langle x, y \rangle \end{aligned}$$

where the first step follows from simple algebra, the second step follows from the fact that $\langle x, y \rangle \geq -|\langle x, y \rangle|$, the third step follows from the property of inner product, the fourth step follows from Fact J.7, the fifth step follows from $\langle x,z \rangle \geq c \cdot \max_{y \in Y} \langle x,y \rangle$ and the final step follows from the fact that $0.01c \cdot \tau > \epsilon$.

□

J.3 Efficient Transformations

We have learned from that (c,τ)-MaxIP on a unit sphere $\mathcal{S}^{d-1}$ using LSH for ANN. Therefore, the next step is to transform the direction search procedure in iterative optimization algorithm into a MaxIP on a unit sphere. To achieve this, we formulate the direction search as a projected approximate MaxIP (see Definition J.5). We start with presenting a pair of transformation $\phi_0,\psi_0:\mathbb{R}^d\to\mathbb{R}^{d+1}$ such that, given a function $g:\mathbb{R}^d\to\mathbb{R}$, for any x,y in a convex set $\mathcal{K}$, we have

$$\phi_0(x) := [\nabla g(x)^\top, x^\top \nabla g(x)]^\top, \quad \psi_0(y) := [-y^\top, 1]^\top. \tag{11}$$

In this way, we show that

$$\begin{aligned} \langle y-x, \nabla g(x)\rangle &= -\langle \phi_0(x), \psi_0(y)\rangle, \\ \operatorname{argmin}_{y\in Y}\langle y-x, \nabla g(x)\rangle &= \operatorname{argmax}_{y\in Y}\langle \phi_0(x), \psi_0(y)\rangle \end{aligned} \tag{12}$$

Therefore, we could transform the direction search problem into a MaxIP problem.

Next, we present a standard transformations (Neyshabur & Srebro, 2015) that connects the MaxIP to ANN in unit sphere. For any $x,y\in\mathbb{R}^d$, we propose transformation $\phi_1,\psi_1:\mathbb{R}^d\to\mathbb{R}^{d+2}$ such that

$$\begin{aligned} \phi_1(x) &= \left[(D_x^{-1}x)^\top \quad 0 \quad \sqrt{1-\|xD_x^{-1}\|_2^2}\right]^\top \\ \psi_1(y) &= \left[(D_y^{-1}y)^\top \quad \sqrt{1-\|yD_y^{-1}\|_2^2} \quad 0\right]^\top \end{aligned} \tag{13}$$

Here D_x, D_y are some constant that make sure both x/D_x and y/D_y have norms less than 1. Under these transformations, both $\phi_1(x)$ and $\psi_1(y)$ have norm 1 and $\operatorname{argmax}_{y\in Y}\langle \phi_1(x), \psi_1(y)\rangle = \operatorname{argmax}_{y\in Y}\langle x,y\rangle$.

Combining transformations in Eq. (11) and Eq. (13), we obtain query transform $\phi:\mathbb{R}^d\to\mathbb{R}^{d+3}$ with form $\phi(x)=\phi_1(\phi_0(x))$ and data transform $\phi:\mathbb{R}^d\to\mathbb{R}^{d+3}$ with form $\psi(y)=\psi_1(\psi_0(y))$. Using ϕ and ψ, we transform the direction search problem in optimization into a MaxIP in unit sphere. Moreover, given a set $Y\subset\mathbb{R}^d$ and a query $x\in\mathbb{R}^d$, the solution z of (c,ϕ,ψ,τ)-MaxIP over (x,Y) has the propriety that $\langle z-x,\nabla g(x)\rangle \le c\cdot\min_{y\in Y}\langle y-x,\nabla g(x)\rangle$. Thus, we could approximate the direction search with LSH based MaxIP data-structure.

Note that only MaxIP problem with positive inner product values could be solved by LSH. We found the direction search problem naturally satisfies this condition. We show that if g is convex, given a set $S\subset\mathbb{R}^d$, we have $\min_{s\in S}\langle \nabla g(x), s-x\rangle \le 0$ for any $x\in\mathcal{B}(S)$, where $\mathcal{B}$ is the convex hull of S. Thus, $\max_{y\in Y}\langle \phi_0(x),\psi_0(y)\rangle$ is non-negative following Eq. (12).

J.4 Data Structures

In this section, we present a formal statement that solves (c,τ)-MaxIP problem on unit sphere using LSH for $(\bar{c},r)$-ANN.

Theorem J.9. *Let $c\in(0,1)$ and $\tau\in(0,1)$. Given a set of n-vector set $Y\subset\mathcal{S}^{d-1}$ on the unit sphere, there exists a data structure with $O(dn^{1+o(1)})$ preprocessing time and $O(n^{1+o(1)}+dn)$ space so that for any query $x\in\mathcal{S}^{d-1}$, we take $O(d\cdot n^\rho)$ query time to retrieve the (c,τ)-MaxIP of x in Y with probability at least 0.9[7], where $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$*

Proof. We know that $\|x-y\|_2^2 = 2-2\langle x,y\rangle$ for all $x,y\in\mathcal{S}^{d-1}$. In this way, if we have a LSH data-structure for $(\bar{c},r)$-ANN. It could be used to solve (c,τ)-MaxIP with $\tau = 1-0.5r^2$ and $c = \frac{1-0.5\bar{c}^2r^2}{1-0.5r^2}$. Next, we write $\bar{c}^2$ as

$$\bar{c}^2 = \frac{1-c(1-0.5r^2)}{0.5r^2} = \frac{1-c\tau}{1-\tau}.$$

Next, we show that if the LSH is initialized following Theorem J.3, it takes query time $O(d\cdot n^\rho)$, space $O(n^{1+o(1)}+dn)$ and preprocessing time $O(dn^{1+o(1)})$ to solve (c,τ)-MaxIP through solving $(\bar{c},r)$-ANN, where

$$\rho = \frac{2}{\bar{c}^2} - \frac{1}{\bar{c}^4} + o(1) = \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1).$$

□

In practice, c is increasing as we set parameter τ close to $\text{MaxIP}(x,Y)$. There is also another LSH data structure (Andoni & Razenshteyn, 2015) with longer preprocessing time and larger space that could solve the (c,τ)-MaxIP with similar query

[7] It is obvious to boost probability from constant to by repeating the data structure log(1=) times.

time complexity. We refer readers to Section 8.2 in (Shrivastava et al., 2021) for more details[8]. Moreover, Corollary J.9 could be applied to projected MaxIP problem.

Theorem J.10. *Let $c \in (0,1)$ and $\tau \in (0,1)$. Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denote two transforms. Let $\mathcal{T}_\phi$ denote the time to compute $\phi(x)$ and $\mathcal{T}_\psi$ denote the time to compute $\psi(y)$. Given a set of n-points $Y \in \mathbb{R}^d$ with $\psi(Y) \subset \mathcal{S}^{k-1}$ on the sphere, one can construct a data structure with $O(dn^{1+o(1)} + \mathcal{T}_\psi n)$ preprocessing time and $O(n^{1+o(1)} + dn)$ space so that for any query $x \in \mathbb{R}^d$ with $\phi(x) \in \mathcal{S}^{k-1}$, we take query time complexity $O(d \cdot n^\rho + \mathcal{T}_\phi)$ to solve (c, ϕ, ψ, τ)-MaxIP with respect to (x, Y) with probability at least 0.9, where $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.*

Proof. The preprocessing phase can be decomposed in two parts.

- It takes $O(\mathcal{T}_\psi n)$ time to transform every $y \in Y$ into $\psi(y)$.
- It takes $O(O(dn^{1+o(1)})$ time and $O(dn^{1+o(1)} + dn)$ to index every $\psi(y)$ into LSH using Theorem J.9.

The query phase can be decomposed in two parts.

- It takes $O(\mathcal{T}_\phi)$ time to transform every $x \in \mathbb{R}^d$ into $\phi(x)$.
- It takes $O(d \cdot n^\rho)$ time perform query for $\phi(x)$ in LSH using Theorem J.9.

□

K Self-attention layer as a clustering algorithm

The self-attention layer in the Transformer looks like mean-shift clustering. Suppose $\{(x_j, v_j)\}$ are a bunch of key and value pairs and q is the query. Note that $q = W_q x$, $k = W_k x$ and $v = W_v x$ are computed by three projection matrices W_k, W_q and W_v from a common x. Then from self-attention we have:

$$v = \sum_j p_j v_j = \frac{\sum_j \exp(x^\top W_q^\top W_k x_j) W_v x_j}{\sum_j \exp(x^\top W_q^\top W_k x_j)} = W_v \frac{\sum_j \exp(x^\top W_q^\top W_k x_j) x_j}{\sum_j \exp(x^\top W_q^\top W_k x_j)} \tag{14}$$

where $\sim(q, k_j) := \exp(q^\top k_j) = \exp(x^\top W_q^\top W_k x_j)$ and $p_j = \sim(q, k_j) / \sum_j \sim(q, k_j)$.

On the other hand, mean-shift clustering looks like the following:

$$m(x) = \frac{\sum_j K(x_j, x) x_j}{\sum_j K(x_j, x)} \tag{15}$$

where $K(x_j, x)$ is a kernel matrix that measure the similarity between x_j and x. According to the mean-shift algorithm, in the next iteration, we will simply replace x with $m(x)$.

So in some sense, self-attention is just to do some kind of clustering for the input embedding q and k, plus a transformation of the embedding to another place. The term "projection" is due to the fact that there is a projection matrix W_v on x for the next level.

Residue connection and LayerNorm. Compared to mean-shift, Transformer layer has residue connection. Therefore, for single-headed attention, what you actually get is $v + x$, followed by a LayerNorm. For the residue connection, the mean-shift analog already shows the output $m(x)$ contains $x +$ part. The reason why we need residue connection is that the self-attention part might only model the "change" of x in the mean-shift picture, rather than the full update of x.

L The role of self-attention

Consider we have a vocabulary of size m and d dimensional embedding space. In practice, many papers in NLP have reported clustering behaviors of word embeddings: such a clustering of word embedding naturally occurs after training.

An explanation for the above phenomenon is that, by grouping these word embedding together, we might generalize better, since similarity in word now can transfer (e.g., A linked to B, B linked to C, then A might link to C as well) and generalization follows.

Let's treat it as a fact and focus on how this is achieved and how self-attention plays a role here.

[8] Recently, there a line of work that use fast MaxIP data structure to speedup the iterative-type optimization algorithms (Shrivastava et al., 2021; Song & Ye, 2023; Qin et al., 2023a; Song et al., 2023a).

L.1 The capacity of embedding layer

First let us take a look at the following pairwise distance constraints between word embedding (e.g., some words should be close to each other, some should be far away from each other) as the following:

$$\|x_i - x_j\| = D(i,j) \tag{16}$$

where $D(i,j)$ is large for i and j that should be far apart and $D(i,j)$ is small for i and j that are close to each other. In visualization, this is called Multidimensional Scaling (MDS) (Cox & Cox, 2008).

Note that in neural network training, the constraint (Eqn. 16) is not directly enforced during training, but the clustering naturally happens. Since we talk about capacity, how we achieve Eqn. 16 doesn't matter for now.

In general we cannot find a *fixed* low-dimensional embedding ($d \ll m$) to satisfy these constraints, since we only have md parameters (m vectors, each has d entries), but m^2 constraint. So two vectors that are supposed to be close may not be close enough (but hopefully they remain close to each other).

L.2 The role of self-attention

For this, the self-attention mechanism comes to the rescue, trading model-size with additional computation. It fulfills what (static) embedding cannot achieve: to further group the embedding vectors together in a multi-layer structure.

Note that one sentence never covers all d vocabularies. Once the words in the sentence are picked, they are grouped together via self-attention layers to collectively represent a concept that can be useful for the task.

L.3 How the clustering happens through self-attention?

Now one fundamental questions arise: How the static clustering of embedding happens during end-to-end training? In practice, no one explicitly enforces the MDS constraint (Eqn. 16).

Let's start with a simple example. we have two unit embedding: x and y with the normalization condition that $\|x\|_2 = 1$ and $\|y\|_2 = 1$, and a simple self-attention layer (without projection) which output z:

$$z = (1-p)x + py \tag{17}$$

Where the attention map is:

$$p = \frac{e^{x^\top y}}{e^{x^\top x} + e^{x^\top y}} = \frac{1}{1 + e^{1 \quad x^\top y}} \tag{18}$$

Note that here we attend to x so $0 < p < 1/2$ always. The last two is due to normalization condition.

Now we consider a loss function $L = -\frac{1}{2}\|z\|_2^2$. The intuition behind is that "for some reason, we found that z is a good representation for our task, and want to make sure its length is as long as possible".

Under this context, what would be the gradient rule for x and y? Will they cluster together?

The answer is yes! We could compute

$$\frac{\partial z}{\partial x} = (1-p)I + \frac{\partial p}{\partial x}(y-x)^\top \tag{19}$$

$$\frac{\partial z}{\partial y} = pI + \frac{\partial p}{\partial y}(y-x)^\top \tag{20}$$

Let $t := 1 - x^\top y$ and define the following function with respect to t:

$$f(t) := (x-y)^\top z = (1-2p)(1-x^\top y) > 0 \tag{21}$$

Therefore, we can compute the gradient for x and gradient for y:

$$-g_x := -\frac{\partial L}{\partial x} = -\frac{\partial z}{\partial x}\frac{\partial L}{\partial z} = (1-p)^2 x + p(1-p)(1-f(t))y \tag{22}$$

$$-g_y := -\frac{\partial L}{\partial y} = -\frac{\partial z}{\partial y}\frac{\partial L}{\partial z} = p^2 y + p(1-p)(1-f(t))x \tag{23}$$

Note that since x and y are kept to be normalized, the term $(1-p)^2 x$ in $\partial L/\partial x$ is gone (and similarly $p^2 y$ for g_y). So how x and y move depends on the sign of $1 - f(t)$.

With some computation, we could see $0 < f(t) < 1$ when $t < 1.5424$. In summary, if $x^\top y > -0.4576$, then the (negative) gradient of x pushes it towards y and pushes x towards y, and the clustering of static embedding happens during training. Note that since both x and y are normalized, $-1 \le x^\top y \le 1$, so this is a quite loose condition and can be easily satisfied.

L.4 Multiple embeddings

People might wonder what happen to multiple unit embeddings $x, y_1, y_2, \ldots, y_K$? In this case, we can similarly define self-attention probability p_i (note that here we consider the case that every embedding attends to x):

$$p_i := \frac{e^{x^\top y_i}}{e^{x^\top x} + \sum_j e^{x^\top y_j}} = \frac{e^{x^\top y_i}}{1 + \sum_j e^{x^\top y_j}} \tag{24}$$

Define $p_S := \sum_{i=1}^K p_i = 1 - \frac{1}{1+\sum_j e^{x^\top y_j}} < 1$ and we have:

$$z = (1-p_S)x + \sum_i p_i y_i \tag{25}$$

Let $\tilde{p}_i := p_i/p_S$ be the (normalized) probability on y_i and $y := \frac{1}{p_S}\sum_i p_i y_i = \sum_i \tilde{p}_i y_i$ be the weighted mean of $\{y_i\}$ other than x, then we have:

$$z = (1-p_S)x + p_S y \tag{26}$$

Now we can still compute the partial derivative:

$$\frac{\partial p_j}{\partial x} = p_j[-p_S y + y_j] \tag{27}$$

$$\frac{\partial p_j}{\partial y_i} = p_i[-p_j + \mathsf{I}(i=j)]x \tag{28}$$

which gives

$$\frac{\partial z}{\partial x} = (1-p_S)I + \sum_j \frac{\partial p_j}{\partial x}(y_j - x)^\top \tag{29}$$

$$\frac{\partial z}{\partial y_i} = p_i I + \sum_j \frac{\partial p_j}{\partial y_i}(y_j - x)^\top \tag{30}$$

After some manipulation, we have:

$$\frac{\partial z}{\partial x} = (1-p_S)[I + p_S y(y-x)^\top] + p_S Q \tag{31}$$

where $Q := \sum_j \tilde{p}_j (y_j - y)(y_j - y)^\top$ is the weighted covariance matrix of data points $\{y_j\}$.

Similar to the two unit case, we want to check $-g_x$ to see how the embedding x changes over time.

$$\begin{aligned} -g_x &= -\frac{\partial L}{\partial x} = -\frac{\partial z}{\partial x}\frac{\partial L}{\partial z} \\ &= (1-p_S)^2 x + p_S\left[(1-2p_S)x^\top y - (1-p_S) + p_S\|y\|^2\right]y + p_S Q z \end{aligned} \tag{32}$$

If things are already quite clustered, then $\|y\| \approx 1$ (usually $\|y\|_2 < 1$ since sphere is a convex set), $Qz \approx 0$ (since Q spans on the tangent space of z at the sphere and z is perpendicular to it), and we have:

$$-g_x \approx (1-p_S)^2 x + p_S(1-2p_S)(x^\top y - 1)y \tag{33}$$

It is clear that $x^\top y < 1$. When $p_S > 1/2$, which is high likely for large K, then $-g_x$ has positive component of y and x will move towards y.

On the other hand, we could also check

$$\frac{\partial z}{\partial y_i} = p_i[I + (1-p_S)x(y-x)^\top] + p_i x(y_i - y)^\top \tag{34}$$

which gives an expression of $-g_y$:

. (35)

With the same argument, it moves towards y (so all y_i will cluster together) and towards x.

When there is a W_k and W_q before the embedding, following the same logic, only the column subspace of W_k (or W_q) will be clustered together. On the other hand, the value part will be different in order to enable encoding of more complicated concepts based on co-occurrence of multiple tokens.

M Link self-attention with generative models.

Consider the following self-attention structure. Consider an embedding matrix $X \in \mathsf{R}^{n \quad d}$ and for embedding x_i and x_j, let

$$y_{ij} = \phi(x_i; x_j) := (1-\beta_{ij})x_i + \beta_{ij} x_j, \quad \beta_{ij} := \frac{e^{x_i^\top x_j}}{e^{x_i^\top x_i} + e^{x_i^\top x_j}} \tag{36}$$

Here $\phi(x_i;x_j) := x_i + \beta_{ij}(x_j - x_i)$ is the self-attention operation. More properties of this operator ϕ need to be explored. Then we want to maximize the following objective:

$$\max_{X,kx_ik_2=1} \sum_{ijk} P(k|i,j) y_{ij}^\top x_k \tag{37}$$

or more formally, using a softmax to avoid trivial solution $x_i \equiv x$, we have:

$$\max_{X,kx_ik_2=1} J := \max_{X,kx_ik_2=1} \sum_{ijk} P(k|i,j) \log \delta_{ijk}, \quad \delta_{ijk} := \frac{e^{y_{ij}^\top x_k}}{\sum_k e^{y_{ij}^\top x_k}} \tag{38}$$

which is:

$$\max_{X,kx_ik_2=1} \sum_{ijk} P(k|i,j) \left[y_{ij}^\top x_k - \log \sum_k e^{y_{ij}^\top x_k} \right] \tag{39}$$

We can compute its gradient update. Here we assume the index k never appears in index i and j (encoding and decoding matrices are decoupled), then by gradient rule, we have:

$$\underline{x}_k = \frac{\partial L}{\partial x_k} = P_{x_k}^? \sum_{ij} P(k|i,j)(1-\delta_{ijk}) y_{ij} \tag{40}$$

where $P_{x_k}^?$ is the projection matrix that projects a vector to the orthogonal complement space of x_k. The projection is due to the constraint $\|x_k\|_2 = 1$. If the training converges ($\underline{x}_k = 0$), then we know that

$$\sum_{ij} P(k|i,j)(1-\delta_{ijk}) y_{ij} = \gamma x_k \tag{41}$$

for some $\gamma > 0$ (note that $\gamma < 0$ will be an unstable stationary point).

Depending on different structure of the generative model specified by $P(k|i,j)$, we might end up learning different embedding matrix X.

The first thing we want to check is independency. Assume that for some specific token k and i, we have $P(k|i,j) = P(k|i)$ for any j, which means that the frequency of token k has nothing to do with the second entry j. Furthermore, token k is not connected with other token $i^0 \neq i$, i.e, $P(k|i^0,j) \equiv 0$. If we just let $\delta_{ijk} = \delta > 0$, then we have:

$$P(k|i) \sum_j y_{ij} = \gamma^0 x_k \tag{42}$$

which yields

$$P(k|i) n x_i + \sum_j \beta_{ij}(x_j - x_i) = \gamma^0 x_k \tag{43}$$

And we could possibly show that $\sum_j \beta_{ij}(x_j - x_i) \approx 0$ since $\beta_{ij} = 1/(1 + e^{1\ \ x_i^\top x_j})$ applies equal weights for embeddings around x_i and they cancel out. Therefore, x_k is aligned with x_i.

Another thing we might want to check is identification of two tokens. Assume that there exists two tokens j_1 and j_2 and specific k and i, so that $P(k|i,j_1) = P(k|i,j_2)$. For other k,i,j combination $P(k|i,j) \equiv 0$, then we have:

$$P(k|i,j_1) y_{ij_1} = \gamma_1 x_k \tag{44}$$

(not sure how to continue).

If we have W_q, W_k and W_v, then the formulation doesn't change that much. The only difference here is that now

$$\beta_{ij} := \frac{e^{x_i^\top W_{pq} x_j}}{e^{x_i^\top W_{pq} x_i} + e^{x_i^\top W_{pq} x_j}} \tag{45}$$

and $y_{ij}^\top x_k$ now becomes $y_{ij}^\top W_v x_k$.