Algorithm and Hardness for Dynamic Attention Maintenance in Large Language Models

Jan van den Brand [1] **Zhao Song** [2] **Tianyi Zhou** [3]

Abstract

The attention scheme is one of the key components over all the LLMs, such as BERT, GPT-1, Transformers, GPT-2, 3, 3.5 and 4. Inspired by previous theoretical study of static version of the attention multiplication problem [Zandieh, Han, Daliri, and Karbasi ICML 2023, Alman and Song NeurIPS 2023], we formally define a dynamic version of attention matrix multiplication problem. In each iteration we update one entry in key matrix $K \in \mathbb{R}^{n \times d}$ or value matrix $V \in \mathbb{R}^{n \times d}$. In the query stage, we receive $(i, j) \in [n] \times [d]$ as input, and want to answer $(D^{-1}AV)_{i,j}$, where $A := \exp(QK^\top) \in \mathbb{R}^{n \times n}$ is a square matrix and $D := \mathrm{diag}(A\mathbf{1}_n) \in \mathbb{R}^{n \times n}$ is a diagonal matrix and $\mathbf{1}_n$ denotes a length-n vector that all the entries are ones. We provide two results: an algorithm and a conditional lower bound. Inspired by the lazy update idea from [Demetrescu and Italiano FOCS 2000, Sankowski FOCS 2004, Cohen, Lee and Song STOC 2019, Brand SODA 2020], we provide a data-structure that uses $O(n^{\omega(1,1,\tau)-\tau})$ amortized update time, and $O(n^{1+\tau})$ worst-case query time, where $n^{\omega(1,1,\tau)}$ denotes $\mathcal{T}_{\mathrm{mat}}(n, n, n^\tau)$ with matrix multiplication exponent ω and τ denotes a constant in $(0, 1]$. We also show that unless the hinted matrix vector multiplication conjecture [Brand, Nanongkai and Saranurak FOCS 2019] is false, there is no algorithm that can use both $O(n^{\omega(1,1,\tau)-\tau-\Omega(1)})$ amortized update time, and $O(n^{1+\tau-\Omega(1)})$ worst query time.

[1]Georgia Tech, Atlanta, GA, USA [2]Adobe Research, San Jose, CA, USA [3]University of Southern California, Los Angeles, CA, USA. Correspondence to: Jan van den Brand <vdbrand@gatech.edu>, Zhao Song <zsong@adobe.com>, Tianyi Zhou <tzhou029@usc.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Large language models (LLMs) such as Transformer (Vaswani et al., 2017), BERT (Devlin et al., 2018), GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and OPT (Zhang et al., 2022a) offer better results when processing natural language compared to smaller models or traditional techniques. These models possess the capability to understand and produce complex language, which is beneficial for a wide range of applications like language translation, sentiment analysis, and question answering. LLMs can be adjusted to multiple purposes without requiring them to be built from scratch. A prime example of this is ChatGPT, a chat software developed by OpenAI utilizing GPT-3's potential to its fullest. GPT-4 (OpenAI, 2023), the latest iteration, has the potential to surpass the already impressive abilities of GPT-3, including tasks such as language translation, question answering, and text generation. As such, the impact of GPT-4 on NLP could be significant, with new applications potentially arising in areas like virtual assistants, chatbots, and automated content creation.

The primary technical foundation behind LLMs is the attention matrix (Vaswani et al., 2017; Radford et al., 2018; Devlin et al., 2018; Brown et al., 2020). Essentially, an attention matrix is a square matrix with corresponding rows and columns representing individual words or "tokens," and entries indicating their correlations within a given text. This matrix is then utilized to gauge the essentiality of each token in a sequence, relative to the desired output. As part of the attention mechanism, each input token is assigned a score or weight based on its significance or relevance to the current output, which is determined by comparing the current output state and input states through a similarity function.

More formally, the attention matrix can be expressed as follows: Suppose we have two matrices, Q and K, comprising query and key tokens respectively, where $Q \in \mathbb{R}^{n \times d}$ and $K \in \mathbb{R}^{n \times d}$. The attention matrix is a square $n \times n$ matrix denoted by A that relates the input tokens in the sequence. After normalizing using the softmax function, each entry in this matrix quantifies the attention weight or score between a specific input token (query token Q) and an output token (key token K). Notably, entries along the diagonal reflect self-attention scores, indicating the significance of

each token in relation to itself.

When modeling long sequences with large n, the most significant hindrance to accelerating LLM operations is the duration required for carrying out attention matrix calculations (Kitaev et al., 2020; Wang et al., 2020). These calculations involve multiplying the attention matrix A with another value token matrix $V \in \mathbb{R}^{n \times d}$. In (Wang et al., 2020), they demonstrate that the self-attention mechanism can be approximated by a low-rank matrix. They propose a new self-attention mechanism and used it in their Linformer model. In (Kitaev et al., 2020), they replace dot-product attention with one that uses locality-sensitive hashing, which also improves the time complexity.

Furthermore, the static attention computation and approximation has been studied by (Alman & Song, 2023) from both algorithmic and hardness perspectives. However, in practice, the attention matrix needs to be trained and keeps changing. In this work, we study the dynamic version of the attention computation problem. By using a dynamic approach, the attention weights can be updated on-the-fly as new information is introduced, enabling the model to adapt more effectively to changes in the input. This is particularly beneficial in cases where the input data is highly dynamic and subject to frequent changes, such as in natural language processing applications where the meaning and context of words and phrases can be influenced by the surrounding text.

Following the prior work (Zandieh et al., 2023; Alman & Song, 2023; Deng et al., 2023d;e;b), we formally define the standard attention computation problem as follows. To distinguish their standard model with the dynamic version studied in this paper, we call the problem defined in (Zandieh et al., 2023; Alman & Song, 2023) "static" version of attention multiplication. Another major difference between previous work (Zandieh et al., 2023; Alman & Song, 2023) and our work is that they studied an approximate version, whereas we study the exact version.

Definition 1.1 (Static Attention Multiplication)**.** *Given three matrices $Q, K, V \in \mathbb{R}^{n \times d}$, we define attention computation*

$$\mathsf{Att}(Q, K, V) = D^{-1}AV$$

where square matrix $A \in \mathbb{R}^{n \times n}$ and diagonal matrix $D \in \mathbb{R}^{n \times n}$ are

$$A := \exp(QK^\top), D := \mathrm{diag}(A\mathbf{1}_n)$$

Here we apply the $\exp(\cdot)$ function entry-wise[1]*. We use $\mathbf{1}_n$ to denote a length-n vector where all the entries are ones. The $\mathrm{diag}()$ function is taking a length-n vector as input and outputs an $n \times n$ diagonal matrix by copying that vector on the diagonal of the output matrix. See Figure 1 and Figure 2 for an illustration.*

In applied LLMs training, the model parameters are changing slowly during training (Chen et al., 2021). In addition, deep neural network architectures frequently exhibit significant redundancy, and empirical evidence supports the capacity of deep neural networks to tolerate substantial levels of sparsity (Han et al., 2015; Gale et al., 2019). In downstream fine-tuning tasks, the dimensions of the model often make the fine-tuning infeasible. Over the past few years, numerous techniques for inducing sparsity have been proposed to sparsify the neural network such as magnitude pruning (Zhu & Gupta, 2017), RegL (Evci et al., 2020) and dynamic sparse reparameterization (Mostafa & Wang, 2019). Thus, it is worth considering the dynamic version of Attention multiplication problem which update the attention matrix entry-wise. Next, we formally define the "dynamic" or "online" version of attention multiplication problem, we call it ODAMV[2]. For consistency of the discussion, we will use the word "online" in the rest of the paper.

Definition 1.2 ($\mathsf{ODAMV}(n, d)$)**.** *The goal of **O**nline **D**iagonal-based normalized **A**ttention **M**atrix **V**ector multiplication problem $\mathsf{ODAMV}(n, d)$ is to design a data-structure that satisfies the following operations:*

1. INIT*: Initialize on three $n \times d$ matrices Q, K, V.*
2. UPDATE*: Change any entry of K, or V.*
3. QUERY*: For any given $i \in [n]$, $j \in [d]$, return $(D^{-1}\exp(QK^\top)V)_{i,j}$.*
   - *Here $D := \mathrm{diag}(\exp(QK^\top)\mathbf{1}_n) \in \mathbb{R}^{n \times n}$ is a positive diagonal matrix.*
   - *Here $[n]$ denotes the set $\{1, 2, \cdots, n\}$.*

In this paper, we first propose a data-structure that efficiently solves the ODAMV problem (Definition 1.2) by using lazy update techniques. We then complement our result by a conditional lower bound. On the positive side, we use lazy update technique in the area of dynamic algorithms to provide an upper bound. In the area of theoretical computer science, it is very common to assume some conjecture in complexity when proving a lower bound. For example, $\mathsf{P} \neq \mathsf{NP}$, (strong) exponential time hypothesis, orthogonal vector and so on (Abboud & Williams, 2014; Henzinger et al., 2015; Backurs & Indyk, 2015; Backurs et al., 2017; Chen, 2018; Rubinstein, 2018; Alman et al., 2020; 2023; Alman & Song, 2023). To prove our conditional lower bound, we use a conjecture which is called **H**inted **M**atrix **V**ector

[1] For a matrix $M \in \mathbb{R}^{n \times n}$, following the transformer literature, we use $\exp(M)_{i,j} := \exp(M_{i,j})$.

[2] The name of our problem is inspired by a well-known problem in theoretical computer science which is called **O**nline **M**atrix **V**ector multiplication problem (OMV) (Henzinger et al., 2015; Larsen & Williams, 2017; Chakraborty et al., 2018).

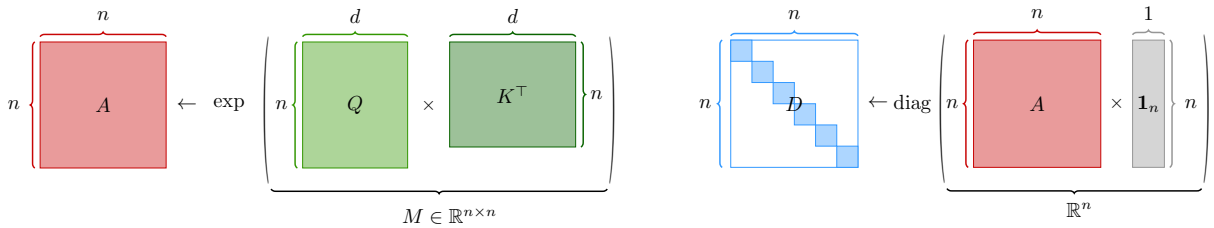

Figure 1. Computation of the attention matrix $A = \exp(QK^\top)$ and the diagonal matrix $D \in \mathbb{R}^{n \times n}$ (defined in Definition 1.1). Here $\exp()$ is the entry-wise function.

multiplication (HMV) conjecture 5.2 of (Brand et al., 2019). On the negative side, we show a lower bound of computing solving ODAMV assuming the HMV conjecture holds. One notable difference between prior work (Alman & Song, 2023) and our work is, their techniques are from the area of fine-grained complexity, and our techniques are not. Our algorithmic techniques are from recent work in convex optimization, e.g. solving linear programming. Our hardness techniques are from the area of dynamic algorithms.

1.1. Our Results

We first show our upper bound result making use of the lazy update strategy.

Theorem 1.3 (Upper bound, informal version of Theorem B.1)**.** *For any constant $a \in (0, 1]$. Let $d = O(n)$. Let $\delta \in \mathbb{R}$ denote the update to the matrix. There is a dynamic data structure that uses $O(n^2)$ space and supports the following operations:*

- INIT(Q, K, V). *It runs in $O(\mathcal{T}_{\mathrm{mat}}(n, n, n))$ time.*[3]
- UPDATEK$(i \in [n], j \in [d], \delta \in \mathbb{R})$. *This operation updates one entry in K, and it runs in $O(\mathcal{T}_{\mathrm{mat}}(n, n^a, n)/n^a)$ amortized*[4] *time.*
- UPDATEV$(i \in [n], j \in [d], \delta \in \mathbb{R})$. *This operation takes same amortized*[4] *time as* UPDATEK.
- QUERY$(i \in [n], j \in [d])$. *This operation outputs $(D^{-1}(\exp(QK^\top))V)_{i,j}$ and takes $O(n^a)$ worst-case time.*

Remark 1.4. *The amortized time in* UPDATEK *and* UPDATEV *can be made into worst case time by using standard techniques, e.g. see Section B of (Brand et al., 2019).*

[3]We use $\mathcal{T}_{\mathrm{mat}}(n, d, m)$ to denote the time of multiplying a $n \times d$ matrix with another $d \times m$ matrix. For more details, we refer the readers to Section 2.

[4]We remark that the presented data structure can be made worst-case via standard techniques (sometimes referred to as "global rebuilding") from the dynamic algorithm area (Overmars, 1983; Sankowski, 2004; Goranci et al., 2017; Frandsen & Frandsen, 2009).

The parameter a allows for a trade-off between update and query time. For example, $a = 1$ leads to $O(n^{1.373})$ update time and $O(n)$ query time whereas $a = 1/2$ leads to $O(n^{1.55})$ update and $O(\sqrt{n})$ query time, using current bounds on $\mathcal{T}_{\mathrm{mat}}(\cdot, \cdot, \cdot)$ (Gall & Urrutia, 2018; Williams et al., 2023). We remark that our results beat the naive $O(n^2)$ update time regardless of which fast matrix multiplication algorithm is used[5]. E.g., when using Strassen's algorithm (Strassen et al., 1969) we get an update time of $O(n^{2-0.192a})$.

Our second result makes use of a variation of the popular online matrix vector multiplication (OMV) conjecture which is called hinted matrix vector multiplication conjecture (see Definition C.2 and (Brand et al., 2019)). Next, we present a lower bound for the problem of dynamically maintaining the attention computation $\mathrm{Att}(Q, K, V)$ that matches our upper bound from Theorem 1.3.

Lemma 1.5 (Lower bound, informal version of Lemma C.5)**.** *Assuming the* HMV *conjecture is true. For every constant $0 < \tau \leq 1$, there is no algorithm that solves the* ODAMV(n, d) *problem (see formal version in Definition C.4) with*

- *polynomial initialization time, and*
- *amortized update time $O(\mathcal{T}_{\mathrm{mat}}(n, n^\tau, d)/n^{\tau+\Omega(1)})$, and*
- *worst query time $O(n^{\tau-\Omega(1)})$.*

Conditional lower bounds identify the nature/origin of the hardness. E.g., problems with hardness from the OV (orthogonal vector) conjecture (Williams, 2005; Abboud et al., 2014) boil down to the fundamental bottleneck of searching, hardness from the BMM (boolean matrix multiplication) conjecture (Abboud & Williams, 2014) show that hardness comes from matrix multiplication, and problems with hardness from the HMV conjecture boil down to the trade-off between matrix-vector multiplication vs fast matrix multiplication. We show that dynamic attention maintenance

[5]This is because $\mathcal{T}_{\mathrm{mat}}(n, n^a, n) \leq n^{2+(\omega-2)a}$ for $0 \leq a \leq 1$.

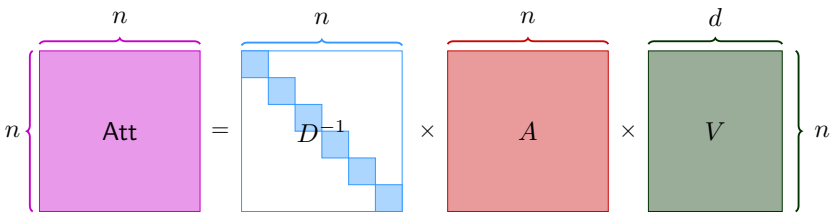

Figure 2. Computation of the target matrix $\mathsf{Att}(Q, K, V) = D^{-1}AV$ (defined in Definition 1.1)

belongs to the latter class by providing tight upper and conditional lower bounds.

1.2. Related Work

Static Attention Computation (Zandieh et al., 2023) was the first to give an algorithm with provable guarantees for approximating the attention computation. Their algorithm makes use of locality sensitive hashing (LSH) techniques (Charikar et al., 2020). They show that the computation of partition functions in the denominator of softmax function can be reduced to a variant of the kernel density estimation (KDE) problem, and an efficient KDE solver can be employed through subsampling-based swift matrix products. They propose the KDEformer which can approximate the attention within sub-quadratic time and substantiated with provable spectral norm bounds. In contrast, earlier findings only procure entry-wise error bounds. Based on empirical evidence, it was confirmed that KDEformer outperforms other attention approximations in different pre-trained models, in accuracy, memory, and runtime. There are also works (Deng et al., 2023c;a; 2024; Song et al., 2024) that optimize the attention computation.

In another recent work (Alman & Song, 2023), they focus on the long-sequence setting with $d = O(\log n)$. The authors established that the existence of a fast algorithm for approximating the attention computation is dependent on the value of B, given the guarantees of $\|Q\|_\infty \leq B$, $\|K\|_\infty \leq B$, and $\|V\|_\infty \leq B$. They derived their lower bound proof by building upon a different line of work that dealt with the fine-grained complexity of KDE problems, which was previously studied in (Backurs et al., 2017; Alman et al., 2020). Their proof was based on a fine-grained reduction from the Approximate Nearest Neighbor search problem ANN. Additionally, their findings explained how LLM computations can be made faster by assuming that matrix entries are bounded or can be well-approximated by a small number of bits, as previously discussed in (Zafrir et al., 2019), Section 2 and (Katharopoulos et al., 2020), Section 3.2.1. Specifically, they (Alman & Song, 2023) showed a lower bound stating that when $B \geq \Omega(\sqrt{\log n})$, there is no algorithm that can approximate the computation in subquadratic time. However, when $B < o(\sqrt{\log n})$, they proposed an algorithm that can approximate the attention computation almost linearly.

Transformer Theory Although the achievements of transformers in various fields are undeniable, there is still a significant gap in our precise comprehension of their learning mechanisms. Although these models have been examined on benchmarks incorporating numerous structured and reasoning activities, comprehending the mathematical aspects of transformers still considerably lags behind. Prior studies have posited that the success of transformer-based models, such as BERT (Devlin et al., 2018), can be attributed to the information contained within its components, specifically the attention heads. These components have been found to hold a significant amount of information that can aid in solving various probing tasks related to syntax and semantics, as noted by empirical evidence found in several studies (Hewitt & Manning, 2019; Clark et al., 2019; Tenney et al., 2019; Hewitt & Liang, 2019; Vig & Belinkov, 2019; Belinkov, 2022; Xu et al., 2024; Gu et al., 2024; Shi et al., 2024).

Various recent studies have delved into the representational power of transformers and have attempted to provide substantial evidence to justify their expressive capabilities. These studies have employed both theoretical as well as controlled experimental methodologies through the lens of Turing completeness (Bhattamishra et al., 2020b), function approximation (Yun et al., 2020), formal language representation (Bhattamishra et al., 2020a; Ebrahimi et al., 2020; Yao et al., 2021), abstract algebraic operation learning (Zhang et al., 2022b), and statistical sample complexity (Wei et al., 2021; Edelman et al., 2022) aspects. According to the research conducted by (Yun et al., 2020), transformers possess the capability of functioning as universal approximators for sequence-to-sequence operations. Similarly, the studies carried out by (Pérez et al., 2019; Bhattamishra et al., 2020b) have demonstrated that attention models may effectively imitate Turing machines. In addition to these recent works, there have been several previous studies that aimed to assess the capacity of neural network models by testing their learning abilities on simplistic data models (Siegelmann & Sontag, 1992; Yao et al., 2021; Zhang et al., 2022b). Furthermore, (Li et al., 2023a) conducted a formal analysis of the training dynamics to further understand the

type of knowledge that the model learns from such data models. According to findings from a recent study (Zhao et al., 2023), moderately sized masked language models have demonstrated the ability to parse with satisfactory results. Additionally, the study utilized BERT-like models that were pre-trained using the masked language modeling loss function on the synthetic text generated with probabilistic context-free grammar. They empirically validated that these models can recognize syntactic information that aids in partially reconstructing a parse tree. (Li et al., 2023b) studied the computation of regularized version of exponential regression problem (without normalization factor). In (Zhang et al., 2023; Liu et al., 2023), they speedup the inference time from both theoretical perspective and experimental perspective by leverage the property of attention. In (Wu et al., 2023), they develop an information-theoretic framework that formulates soft prompt tuning as maximizing mutual information between prompts and other model parameters.

Dynamic Maintenance In recent years, projection maintenance has emerged as a crucial data structure problem. The effectiveness and efficiency of several cutting-edge convex programming algorithms greatly hinge upon a sturdy and streamlined projection maintenance data structure (Cohen et al., 2019; Lee et al., 2019; Brand, 2020; Jiang et al., 2020b; Brand et al., 2020; Jiang et al., 2021; Song & Yu, 2021; Brand, 2021; Jiang et al., 2020a; Huang et al., 2022; Gu & Song, 2022). There are two major differences between the problem in the dynamic data structure for optimization and our dynamic attention matrix maintenance problem. The first notable difference is that, in the optimization task, the inverse of a full rank square matrix is typically computed, whereas, in the attention problem, we care about the inverse of a positive diagonal matrix which behaves the normalization role in LLMs. The second major difference is, in the standard optimization task, all the matrix matrix operations are linear operations. However, in LLMs, nonlinearity such as softmax/exp function is required to make the model achieve good performance. Therefore, we need to apply an entry-wise nonlinear function to the corresponding matrix. In particular, to compute $f(QK^\top)V$ when f is linear function, we can pre-compute $K^\top V$. However when f is exp function, we are not allowed to compute $K^\top V$ directly.

Next, we will give more detailed reviews for classical optimization dynamic matrix maintenance problems. Let $B \in \mathbb{R}^{m\times n}$, consider the projection matrix $P = B^\top (BB^\top)^{-1} B$. The projection maintenance problem asks the following data structure problem: it can preprocess and compute an initial projection. At each iteration, B receives a low rank or sparse change, and the data structure needs to update B to reflect these changes. It will then be asked to approximately compute the matrix-vector product, between the updated P and an online vector h. For example, in linear programming, one sets $B = \sqrt{W}A$, where $A \in \mathbb{R}^{m\times n}$ is the constraint matrix and W is a diagonal matrix. In each iteration, W receives relatively small perturbations. Then, the data structure needs to output an approximate vector to $\sqrt{W}A^\top (AWA^\top)^{-1} A\sqrt{W}h$, for an online vector $h \in \mathbb{R}^n$.

Roadmap The rest of the paper is organized as follows. In Section 2, we give some preliminaries. In Section 3, we explain the techniques used to show our upper bound and lower bound results. In Section 4, we provide a lower bound proof for the simplified version of dynamic attention problem. In Section 5, we provide the conclusion for our paper. We defer the full proofs of upper bound in Appendix B. We defer the full proofs of lower bound in Appendix C.

2. Preliminary

For a matrix A, we use $A^\top$ to denote its transpose. For a matrix A, use $A_{i,j}$ to denote its entry at i-th row and j-th column. For a non-zero diagonal matrix $D \in \mathbb{R}^{n\times n}$, we use $D^{-1} \in \mathbb{R}^{n\times n}$ to denote the matrix where the (i,i)-th diagonal entry is $(D_{i,i})^{-1}$ for all $i \in [n]$. For a vector $x \in \mathbb{R}^n$, we use $\mathrm{diag}(x) \in \mathbb{R}^{n\times n}$ to denote an $n \times n$ matrix where the i,i-th entry on the diagonal is x_i and zero everywhere else for all $i \in [n]$. We use $\exp(M)$ to denote the entry-wise exponential, i.e., $\exp(M)_{i,j} := \exp(M_{i,j})$. We use $\mathbf{1}_n$ to denote the length-n vector where all the entries are ones. We use $\mathbf{0}_n$ to denote the length-n vector where all entries are zeros.

We define a standard notation for describing the running time of matrix multiplication.

Definition 2.1. *For any three positive integers, we use $\mathcal{T}_{\mathrm{mat}}(a,b,c)$ to denote the time of multiplying an $a \times b$ matrix with another $b \times c$ matrix.*

We use ω to denote the time that $n^\omega = \mathcal{T}_{\mathrm{mat}}(n,n,n)$. Currently $\omega \approx 2.372$ (Duan et al., 2023; Williams et al., 2023).

Definition 2.2. *We define $\omega(\cdot,\cdot,\cdot)$ function as follows, for any a, b and c, we use $\omega(a,b,c)$ to denote that $n^{\omega(a,b,c)} = \mathcal{T}_{\mathrm{mat}}(n^a, n^b, n^c)$.*

We give a standard fact that is used in our proof.

Fact 2.3 (folklore)**.** *Given a set of vectors $a_1, \cdots, a_k \in \mathbb{R}^n$ and $b_1, \cdots b_k \in \mathbb{R}^d$, then we have $\sum_{i=1}^k a_i b_i^\top = AB^\top$ where $A \in \mathbb{R}^{n\times k}$ and a_i is i-th column of A, and $B \in \mathbb{R}^{d\times k}$ and b_i is the i-th column of B for all $i \in [k]$. Further, we have*

- *Part 1. Computing $AB^\top$*
  - *takes $O(nkd)$ time, if we do it naively*
  - *takes $\mathcal{T}_{\mathrm{mat}}(n,k,d)$ time, if we use fast matrix multiplication*

- *Part 2. For any matrix $C \in \mathbb{R}^{d \times d}$, computing $AB^\top C$*
  - *takes $\mathcal{T}_{\mathrm{mat}}(n,k,d) + \mathcal{T}_{\mathrm{mat}}(n,d,d)$, if we use fast matrix multiplication, first compute $AB^\top$ then compute $(AB^\top)C$*
  - *takes $\mathcal{T}_{\mathrm{mat}}(k,d,d) + \mathcal{T}_{\mathrm{mat}}(n,k,d)$ time, if we use fast matrix multiplication, first compute $B^\top C$, then compute $A(B^\top C)$*

3. Technique Overview

Given three matrices $Q, K, V \in \mathbb{R}^{n \times d}$, we need to compute the attention given by $\mathsf{Att}(Q,K,V) = D^{-1}AV$ where square matrix $A \in \mathbb{R}^{n \times n}$ and diagonal matrix $D \in \mathbb{R}^{n \times n}$ are $A := \exp(QK^\top)$, $D := \mathrm{diag}(A\mathbf{1}_n)$. The static problem (Alman & Song, 2023) is just computing Att for given Q, K and V. In the dynamic problem, we can get updates for K and V in each iteration.

Due to space limitation, we only describe the core ideas and proof sketch of upper bound in Section 3.1. For the complete proofs, we refer the readers to read the Appendix B. Similarly, we only give high description for lower bound in Section 3.2 and defer the details into Appendix C.

3.1. Algorithm

Problem Formulation For each update, we receive δ as input and update one entry in either matrix K or V. In the query function, we take index $i \in [n], j \in [d]$ as input, and return the $\{i,j\}$-th element in the target matrix $B := D^{-1}AV$.

Let C denote AV. Let $\widetilde{B}$ denote the updated target matrix B. We notice that the computation of the attention can be written as $\widetilde{B} = (D^{-1} + \Delta_D)(C + \Delta_C)$. Let $\Delta^{(t)}$ denote the change in the t-th iteration. In a lazy-update fashion, we write $\widetilde{B}$ in the implicit form

$$\widetilde{B} = (D^{-1} + \sum_{t=1}^{\mathrm{ct}} \Delta_D^{(t)})(C + \sum_{t=1}^{\mathrm{ct}} \Delta_C^{(t)})$$

where ct denotes the number of updates since the last time we recomputed D and C.

Lazy Update We propose a lazy-update algorithm (Algorithm 2) that does not compute the attention matrix when there is an update on the key matrix K. We also propose a lazy-update algorithm (Algorithm 3) that does not compute the attention matrix when there is an update on the value matrix V. Instead, we maintain a data-structure (Algorithm 1) that uses $\mathrm{List}_C, \mathrm{List}_D$ and List_V to record the update by storing rank-1 matrices before the iteration count reaches the threshold n^a for some constant a. For the initialization (Algorithm 1), we compute the exact target matrix $D^{-1}AV$ and other intermediate matrices, which takes $O(\mathcal{T}_{\mathrm{mat}}(n,d,n))$ time (Lemma B.2).

Re-compute When the iteration count reaches the threshold n^a, we re-compute all the variables in the data-structure as follows (Lemma B.7). By using Fact 2.3, we first stack all the rank-1 matrices in List_C and compute the matrix multiplication once to get $\sum_{t=1}^{\mathrm{ct}} \Delta_C^{(t)}$ using $\mathcal{T}_{\mathrm{mat}}(n, n^a, d) = n^{\omega(1,1,a)}$ time (Lemma B.8). Then, we compute $C + \sum_{t=1}^{\mathrm{ct}} \Delta_C^{(t)}$ to get the re-computed $\widetilde{C}$. Similarly, to re-compute V, we stack all the rank-1 matrices in List_V and compute the matrix multiplication once to get $\sum_{t=1}^{\mathrm{ct}} \Delta_V^{(t)}$ using $\mathcal{T}_{\mathrm{mat}}(n, n^a, d) = n^{\omega(1,1,a)}$ time. Then, we compute $V + \sum_{t=1}^{\mathrm{ct}} \Delta_V^{(t)}$ to get the re-computed $\widetilde{V}$. To re-compute the diagonal matrix D, we sum up all the updates by $\sum_{t=1}^{\mathrm{ct}} \Delta_D^{(t)}$ and add it to the old D^{-1} (detail can be found in Algorithm 5). Hence, our algorithm takes $n^{\omega(1,1,a)}/n^a$ amortized time to update K and V (Lemma B.3, Lemma B.4).

Fast Query Recall that the query function takes index $i \in [n], j \in [d]$ as input, and returns the $\{i,j\}$-th element in the target matrix $B := D^{-1}AV$. Let $\widetilde{D}^{-1}$ denote the lates D^{-1} obtained from List_D. Let $\Delta_{V,1}$ and $\Delta_{V,2}$ be stacked matrix obtained from list from V. We can rewrite the output by

$$\begin{aligned}
&((\widetilde{D}^{-1}) \cdot (A) \cdot (V + \Delta_{V,1}\Delta_{V,2}))_{i,j} \\
=\ &((\widetilde{D}^{-1}) \cdot (A \cdot V))_{i,j} + ((\widetilde{D}^{-1}) \cdot A \cdot (\Delta_{V,1}\Delta_{V,2}))_{i,j} \\
=\ &(\widetilde{D})_i^{-1}(C_{i,j} + (\Delta_{C,1}\Delta_{C,2})_{i,j}) \\
&+ (\widetilde{D})_i^{-1} A_{i,*} \Delta_{V,1} (\Delta_{V,2})_{*,j}.
\end{aligned}$$

Note that we maintain C in our re-compute function. Hence, computing the first part takes $O(n^a)$ time. As each column of $\Delta_{V,1}$ and row of $\Delta_{V,2}$ is 1-sparse, computing the second part takes $O(n^a)$ time. The total running time needed for the query function is $O(n^a)$ (Lemma B.6, Lemma B.5).

3.2. Hardness

We now turn to our lower bound result, which is inspired by the HMV conjecture 5.2 of (Brand et al., 2019). Let us firstly define the HMV problem (see formal definition in Definition C.2).

Let the computation be performed over the boolean semiring. For any $0 < \tau \leq 1$, the HMV problem has the following three phases

- **Phase 1.** Input two $n \times n$ matrices M and V
- **Phase 2.** Input an $n \times n$ matrix P with at most n^τ non-zero entries
- **Phase 3.** Input a single index $i \in [n]$
  - We need to answer $MPV_{*,i}$
  - Here $V_{*,i} \in \mathbb{R}^n$ is the i-th column of matrix V

Algorithm 1 Dynamic Data Structure

data structure DYNAMICATTENTION ▷ Theorem B.1
members
  $Q, K, V \in \mathbb{R}^{n \times d}$ ▷ Query token, Key token, Value token
  $M \in \mathbb{R}^{n \times n}$ ▷ The logits matrix, $M = QK^\top$
  $A \in \mathbb{R}^{n \times n}$ ▷ The attention matrix, $A = \exp(QK^\top)$
  $D \in \mathbb{R}^{n \times n}$ ▷ The diagonal matrix,
  $C \in \mathbb{R}^{n \times d}$ ▷ Intermediate matrix, $C = \exp(QK^\top)V$
  $B \in \mathbb{R}^{n \times d}$ ▷ Target matrix, $B = D^{-1}AV$
  $\mathrm{List}_A, \mathrm{List}_C, \mathrm{List}_D$ ▷ List with size n^a
  $\mathrm{ct}_K, \mathrm{ct}_V$
end members

procedure INIT(Q, K, V) ▷ Lemma B.2
  $Q \leftarrow Q, K \leftarrow K, V \leftarrow V$
  $M \leftarrow QK^\top, A \leftarrow \exp(QK^\top)$
  $C \leftarrow \exp(QK^\top)V$
  $B \leftarrow D^{-1}AV$
  $\mathrm{ct}_K \leftarrow 0$
  $\mathrm{ct}_V \leftarrow 0$
end procedure
end data structure

Algorithm 2 Algorithm that update K and maintain the data structure

data structure DYNAMICATTENTION ▷ Theorem B.1
procedure UPDATEK($i \in [n], j \in [d], \delta$) ▷ Lemma B.3
  $\mathrm{ct}_K \leftarrow \mathrm{ct}_K + 1$
  $\widetilde{K}_{i,j} \leftarrow K_{i,j} + \delta$
  $(\Delta_M)_{*,i} \leftarrow \delta \cdot \underbrace{Q}_{n \times d} \underbrace{e_j}_{d \times 1}$ ▷ Δ_M only have entries in i-th column
  ▷ Here $\circ$ denotes entry-wise product
  $(\Delta_A)_{*,i} \leftarrow (A_{*,i} \circ (\exp((\Delta_M)_{*,i}) - \mathbf{1}_n))$
  $\widetilde{M} \leftarrow M + (\Delta_M)_{*,i} e_i^\top$ ▷ We only update i-th column of M
  $\widetilde{A} \leftarrow A + (\Delta_A)_{*,i} e_i^\top$ ▷ We only update i-th column of A
  Obtain diagonal vector D_{tmp} from $\mathrm{List}_D[\mathrm{ct}_K - 1]$.GETB ▷ It takes $O(n)$ time
  $\widetilde{D} \leftarrow D_{\mathrm{tmp}}^{-1} + \mathrm{diag}(\Delta_A)_{*,i}$
  for $j = 1 \to n$ **do**
    $(\Delta_D)_{j,j} \leftarrow (D_{\mathrm{tmp}})_{j,j}^{-1} - \widetilde{D}_{j,j}^{-1}$
  end for
  if $\mathrm{ct}_K < n^a$ **then**
    $\mathrm{List}_C[\mathrm{ct}_K - 1].(a, b) \leftarrow ((\Delta_A)_{*,i} \in \mathbb{R}^n, V^\top e_i \in \mathbb{R}^d)$
    $\mathrm{List}_D[\mathrm{ct}_K - 1].(a, b) \leftarrow (\Delta_D \in \mathbb{R}^{n \times n}, \widetilde{D}^{-1} \in \mathbb{R}^{n \times n})$ ▷ Diagonal matrices
  else ▷ $\mathcal{T}_{\mathrm{mat}}(n, n^a, d) = n^{\omega(1,1,a)}$ time
    RECOMPUTE() ▷ Algorithm 5. Re-compute everything
  end if
  /*Referesh the memory*/
  $K \leftarrow \widetilde{K}$
  $A \leftarrow \widetilde{A}$
  $M \leftarrow \widetilde{M}$
end procedure
end data structure

Algorithm 3

data structure DYNAMICATTENTION ▷ Theorem B.1
procedure UPDATEV($i \in [n], j \in [d], \delta$) ▷ Lemma B.4
  $\mathrm{ct}_V \leftarrow \mathrm{ct}_V + 1$
  if $\mathrm{ct}_V < n^a$ **then**
    $\mathrm{List}_V[\mathrm{ct}_V - 1].(a, b) \leftarrow (e_i \in \mathbb{R}^n, \delta e_j \in \mathbb{R}^d)$
  else
    RECOMPUTE() ▷ Algorithm 5. Re-compute everything
  end if
end procedure
end data structure

In (Brand et al., 2019), the above problem is conjectured to be hard in the following sense,

Conjecture 3.1 (Hinted MV (HMV), Conjecture 5.2 of (Brand et al., 2019))**.** *For every constant $0 < \tau \leq 1$ no algorithm for the hinted Mv problem (Definition C.2) can simultaneously satisfy*

- *polynomial time in* **Phase 1.**
- $O(n^{\omega(1,1,\tau)-\epsilon})$ *time complexity in* **Phase 2.** *and*
- $O(n^{1+\tau-\epsilon})$ *in* **Phase 3.**

for some constant $\epsilon > 0$.

Our primary contribution lies in demonstrating how to reduce HMV problem (Definition C.2) to OAMV (Definition 4.1) and ODAMV (Definition C.4). To achieve this, we have adopted a contradiction-based approach. Essentially, we begin by assuming the existence of an algorithm that can solve the OAMV problem with polynomial initialization time and amortized update time of $O(\mathcal{T}_{\mathrm{mat}}(n, n^\tau, d)/n^{\tau+\Omega(1)})$, while worst-case query time is $O(n^{\tau-\Omega(1)})$ for all $\tau \in (0, 1]$. Our assumption implies that there exists a data structure that is faster than our result (Theorem B.1). We subsequently proceed to demonstrate that using this algorithm enables us to solve the HMV problem too quickly, which contradicts the HMV conjecture.

Specifically, let us take an instance for the HMV problem (Definition C.2)

- Let $\mathsf{M}, \mathsf{V} \in \{0,1\}^{n \times n}$ denote two matrices from **Phase 1.** from HMV.

We create a new instance $\mathsf{OAMV}(\widetilde{n} = n, \widetilde{d} = n)$ where $\widetilde{Q} = \mathsf{M}, \quad \widetilde{K} = 0, \quad \widetilde{V} = \mathsf{V}$.

In Claim 4.3 and Claim 4.4, by making use of our construction of $\widetilde{Q}, \widetilde{K}$ and $\widetilde{V}$, we show that for each $i \in [n]$ and $j \in [n]$,

$$\text{If } ((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n \times n})\widetilde{V})_{j,i} > 0, \text{ then } (\mathsf{MPV})_{j,i} = 1.$$

$$\text{If } ((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n \times n})\widetilde{V})_{j,i} = 0, \text{ then } (\mathsf{MPV})_{j,i} = 0.$$

By using the above two statements, we know that $\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V}_{*,i}$ is enough to reconstruct $\mathsf{MPV}_{*,i}$ for the HMV problem (Definition C.2). Then, solving $\mathsf{MPV}_{*,i}$ takes polynomial initialization time and amortized update time of

$$O(\mathcal{T}_{\mathrm{mat}}(n, n^\tau, d)/n^{\tau+\Omega(1)}),$$

while worst-case query time is $O(n^{\tau-\Omega(1)})$ for every $\tau \in (0, 1]$. The contradiction of the HMV conjecture shows that there is no such algorithm. Similarly, for the normalized case ODAMV (Definition C.4) problem, we show how to reconstruct another instance of the HMV problem and complete the proof by contradiction.

4. The Lower Bound for A Simplified Version

We define the dynamic attention matrix vector problem here. For the following definition, we ignore the effect by the normalization factor for simplicity. We will show how to handle the normalization factor in the Appendix (see Appendix C).

Definition 4.1 ($\mathsf{OAMV}(n, d)$)**.** *The goal of the **O**nline **A**ttention **M**atrix **V**ector Multiplication problem $\mathsf{OAMV}(n, d)$ is to design a data structure that satisfies the following operations:*

1. INIT*: Initialize on $n \times d$ matrices Q, K, V.*
2. UPDATE*: Change any entry of Q, K, or V.*
3. QUERY*: For any given $i \in [n]$, $j \in [d]$, return $(\exp(QK^\top)V)_{i,j}$.*

Next, we present our lower bound result ignoring the normalization factor.

Lemma 4.2. *Assuming the hinted Mv conjecture (Conjecture C.3): For every constant $0 < \tau \leq 1$, there is no dynamic algorithm for $\mathsf{OAMV}(n, d)$ problem (Definition 4.1) with*

- *polynomial initialization time, and*
- *amortized update time $O(\mathcal{T}_{\mathrm{mat}}(n, n^\tau, d)/n^{\tau+\Omega(1)})$, and*
- *worst query time $O(n^{\tau-\Omega(1)})$.*

Proof. Assume there was a dynamic algorithm faster than what is stated in Lemma 4.2 for some parameter τ, i.e. update time

$$O(\mathcal{T}_{\mathrm{mat}}(n, n^\tau, d)/n^{\tau+\epsilon})$$

and query time $O(n^{\tau-\epsilon})$ for some constant $\epsilon > 0$. We show that this would contradict the hinted Mv conjecture (Conjecture C.3).

Let us take an instance for the v-hinted Mv problem (Definition C.2) with $\mathsf{M}, \mathsf{V} \in \{0,1\}^{n \times n}$. We create a new instance $\mathsf{OAMV}(\widetilde{n} = n, \widetilde{d} = n)$ where

$$\widetilde{Q} = \mathsf{M}, \quad \widetilde{K} = 0, \quad \widetilde{V} = \mathsf{V}$$

During phase 1, we give this input to the dynamic algorithm for the OAMV problem (Definition 4.1). During phase 2, when we receive the $n \times n$ matrix P with n^τ non-zero entries, we perform n^τ updates to the data structure to set $\widetilde{K}^\top = \mathsf{P}$. This time is bounded by

$$O(\widetilde{n}^\tau \cdot (\mathcal{T}_{\mathrm{mat}}(\widetilde{n}, \widetilde{n}^\tau, \widetilde{d})/\widetilde{n}^{\tau+\epsilon})) = O(n^{\omega(1,1,\tau)-\epsilon}).$$

At last, in phase 3, we perform $\widetilde{n}$ queries to obtain the column $\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V}_{*,i}$ in $O(\widetilde{n} \cdot \widetilde{n}^{\tau-\epsilon}) = O(n^{1+\tau-\epsilon})$ time.

Using Claim 4.3, and Claim 4.4, we know that $\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V}_{*,i}$ is enough to reconstruct $\mathsf{MPV}_{*,i}$ for the hinted Mv problem. □

Claim 4.3. *For each $i \in [n]$ and $j \in [n]$, if $((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n})\widetilde{V})_{j,i}$ is > 0, then $(\mathsf{MPV})_{j,i} = 1$,*

Proof. Assume we have

$$((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n})\widetilde{V})_{j,i} > 0,$$

We defined $\widetilde{Q} = \mathsf{M}, \widetilde{K} = \mathsf{P}, \widetilde{V} = \mathsf{V}$, so we can rewrite it as

$$((\exp(\mathsf{MP}) - \mathbf{1}_{n\times n})\mathsf{V})_{j,i} > 0.$$

Using the definition of matrix multiplication, and the fact that $\exp(x) > 1$ for all $x > 0$, we have some $k \in [n]$ with

$$((\exp(\mathsf{MP}) - \mathbf{1}_{n\times n})_{j,k}(\mathsf{V})_{k,i} > 0$$
$$((\exp(\mathsf{MP})_{j,k} - 1)(\mathsf{V})_{k,i} > 0$$

We can conclude that for each $i \in [n], j \in [n]$, there is at least one $k \in [n]$ such that $\mathsf{V}_{k,i} > 0$ and $(\mathsf{MP})_{j,k} > 0$. Therefore, by using the definition of boolean semi-ring, we can conclude that $(\mathsf{MPV})_{j,i} = 1$ □

Claim 4.4. *For each $i \in [n]$ and $j \in [n]$, if $((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n})\widetilde{V})_{j,i}$ is 0 then $(\mathsf{MPV})_{j,i} = 0$.*

Proof. We have

$$\begin{aligned}
&((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n})\widetilde{V})_{j,k} \\
=\ &((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n}))_{j,*}\widetilde{V}_{*,i} \\
=\ &((\exp(\mathsf{MP}) - \mathbf{1}_{n\times n}))_{j,*}\mathsf{V}_{*,i}
\end{aligned}$$

where the first step follows from the definition of matrix multiplication and the second step follows from the definition of $\widetilde{Q}, \widetilde{K}$ and $\widetilde{V}$.

By using the above equation, if

$$((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{n\times n})\widetilde{V})_{j,k} = 0,$$

we have

$$(\exp(\mathsf{MP}) - \mathbf{1}_{n\times n})_{j,*}\mathsf{V}_{*,i} = 0. \tag{1}$$

Eq. (1) implies that, for all $k \in [n]$ such that $\mathsf{V}_{k,i} = 1$, we have

$$(\exp(\mathsf{MP}) - \mathbf{1}_{n\times n})_{j,k} = 0,$$

which also implies that $(\mathsf{MP})_{j,k} = 0$.

Now, we can conclude that $(\mathsf{MPV})_{j,i} = 0$ for each $i \in [n]$ and $j \in [n]$. □

5. Conclusion

The development of Large Language Models (LLMs) has had a profound impact on society, with the attention mechanism being a critical aspect of LLMs. This study introduces the dynamic version of the attention matrix multiplication and delivers two outcomes - an algorithm and a conditional lower bound. The algorithmic outcome presents a data structure that supports the dynamic maintenance of attention computations, with a $O(n^{\omega(1,1,\tau)-\tau})$ amortized update time, and $O(n^{1+\tau})$ worst-case query time. The lower bound illustrates that the algorithm is conditionally optimal unless the conjecture on hinted matrix vector multiplication is incorrect. It is an interesting future direction to prove an unconditional lower bound. The problem of dynamic attention matrix multiplication, as proposed, focuses on updating only one entry at a time in either the K or V matrix during each iteration. It is possible to update multiple entries simultaneously in both matrices in practice. Therefore, further research could expand the scope of the problem formulation to include such situations.

Impact Statement

Our approach seeks to balance the computational demands with environmental considerations, acknowledging the potential for increased energy consumption. We advocate for the judicious use of resources in model training and deployment, aiming to set a precedent for sustainable practices in the field. Our findings hold particular promise for large-scale data analysis applications, where they can contribute to more informed and efficient decision-making processes. We are dedicated to continuous assessment and improvement of our methods to ensure they align with both technological advancements and ecological sustainability.

References

Abboud, A. and Williams, V. V. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pp. 434–443. IEEE, 2014.

Abboud, A., Williams, V. V., and Weimann, O. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pp. 39–51. Springer, 2014.

Alman, J. and Song, Z. Fast attention requires bounded entries. In *NeurIPS*, 2023.

Alman, J., Chu, T., Schild, A., and Song, Z. Algorithms and hardness for linear algebra on geometric graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 541–552. IEEE, 2020.

Alman, J., Liang, J., Song, Z., Zhang, R., and Zhuo, D. Bypass exponential time preprocessing: Fast neural network training via weight-data correlation preprocessing. In *NeurIPS*, 2023.

Backurs, A. and Indyk, P. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 51–58, 2015.

Backurs, A., Indyk, P., and Schmidt, L. On the fine-grained complexity of empirical risk minimization: Kernel methods and neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Belinkov, Y. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7.

Bhattamishra, S., Ahuja, K., and Goyal, N. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116, Online, November 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL https://aclanthology.org/2020.emnlp-main.576.

Bhattamishra, S., Patel, A., and Goyal, N. On the computational power of transformers and its implications in sequence modeling. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pp. 455–475, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.37. URL https://aclanthology.org/2020.conll-1.37.

Brand, J. v. d. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 259–278. SIAM, 2020.

Brand, J. v. d. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pp. 1–13. SIAM, 2021.

Brand, J. v. d. and Nanongkai, D. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 436–455. IEEE, 2019.

Brand, J. v. d., Nanongkai, D., and Saranurak, T. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 456–480. IEEE, 2019.

Brand, J. v. d., Lee, Y. T., Sidford, A., and Song, Z. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 775–788, 2020.

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.

Chakraborty, D., Kamma, L., and Larsen, K. G. Tight cell probe bounds for succinct boolean matrix-vector multiplication. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pp. 1297–1306, 2018.

Charikar, M., Kapralov, M., Nouri, N., and Siminelakis, P. Kernel density estimation through density constrained near neighbor search. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 172–183. IEEE, 2020.

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*, 2021.

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

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.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL https://aclanthology.org/W19-4828.

Cohen, M. B., Lee, Y. T., and Song, Z. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

Demetrescu, C. and Italiano, G. F. Fully dynamic transitive closure: breaking through the o (n/sup 2/) barrier. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 381–389. IEEE, 2000.

Deng, Y., Li, Z., Mahadevan, S., and Song, Z. Zero-th order algorithm for softmax attention optimization. *arXiv preprint arXiv:2307.08352*, 2023a.

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

Deng, Y., Mahadevan, S., and Song, Z. Randomized and deterministic attention sparsification algorithms for over-parameterized feature dimension. *arXiv preprint arXiv:2304.04397*, 2023c.

Deng, Y., Song, Z., Xie, S., and Yang, C. Unmasking transformers: A theoretical approach to data recovery via attention weights. *arXiv preprint arXiv:2310.12462*, 2023d.

Deng, Y., Song, Z., and Zhou, T. Superiority of softmax: Unveiling the performance edge over linear attention. *arXiv preprint arXiv:2310.11685*, 2023e.

Deng, Y., Song, Z., and Yang, C. Attention is naturally sparse with gaussian distributed input. *arXiv preprint arXiv:2404.02690*, 2024.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Duan, R., Wu, H., and Zhou, R. Faster matrix multiplication via asymmetric hashing. In *FOCS*, pp. 2129–2138. IEEE, 2023.

Ebrahimi, J., Gelda, D., and Zhang, W. How can self-attention networks recognize Dyck-n languages? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4301–4306, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.384. URL https://aclanthology.org/2020.findings-emnlp.384.

Edelman, B. L., Goel, S., Kakade, S., and Zhang, C. Inductive biases and variable creation in self-attention mechanisms. 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. 5793–5831. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/edelman22a.html.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Frandsen, G. S. and Frandsen, P. F. Dynamic matrix rank. *Theor. Comput. Sci.*, 410(41):4085–4093, 2009.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gall, F. L. and Urrutia, F. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *SODA*, pp. 1029–1046. SIAM, 2018.

Goranci, G., Henzinger, M., and Peng, P. The power of vertex sparsifiers in dynamic graph algorithms. In *ESA*, volume 87 of *LIPIcs*, pp. 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

Gu, J., Li, C., Liang, Y., Shi, Z., Song, Z., and Zhou, T. Fourier circuits in neural networks: Unlocking the potential of large language models in mathematical reasoning and modular arithmetic. *arXiv preprint arXiv:2402.09469*, 2024.

Gu, Y. and Ren, H. Constructing a distance sensitivity oracle in $o(n^{2.5794}m)$ time. *arXiv preprint arXiv:2102.08569*, 2021.

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

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Henzinger, M., Krinninger, S., Nanongkai, D., and Saranurak, T. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pp. 21–30, 2015.

Hewitt, J. and Liang, P. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2733–2743, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275.

Hewitt, J. and Manning, C. D. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://www.aclweb.org/anthology/N19-1419.

Huang, B., Jiang, S., Song, Z., Tao, R., and Zhang, R. Solving sdp faster: A robust ipm framework and efficient implementation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 233–244. IEEE, 2022.

Jiang, H., Kathuria, T., Lee, Y. T., Padmanabhan, S., and Song, Z. A faster interior point method for semidefinite programming. In *2020 IEEE 61st annual symposium on foundations of computer science (FOCS)*, pp. 910–918. IEEE, 2020a.

Jiang, H., Lee, Y. T., Song, Z., and Wong, S. C.-w. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 944–953, 2020b.

Jiang, S., Song, Z., Weinstein, O., and Zhang, H. Faster dynamic matrix inverse for faster lps. In *STOC*. arXiv preprint arXiv:2004.07470, 2021.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Larsen, K. G. and Williams, R. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2182–2189, 2017.

Lee, Y. T., Song, Z., and Zhang, Q. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, 2019.

Li, Y., Li, Y., and Risteski, A. How do transformers learn topic structure: Towards a mechanistic understanding. *arXiv preprint arXiv:2303.04245*, 2023a.

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

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

OpenAI. Gpt-4 technical report, 2023.

Overmars, M. H. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983.

Pérez, J., Marinković, J., and Barceló, P. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Rubinstein, A. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pp. 1260–1268, 2018.

Sankowski, P. Dynamic transitive closure via dynamic matrix inverse. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 509–517. IEEE, 2004.

Sankowski, P. Subquadratic algorithm for dynamic shortest distances. In *Computing and Combinatorics: 11th Annual International Conference, COCOON 2005 Kunming, China, August 16–19, 2005 Proceedings 11*, pp. 461–470. Springer, 2005.

Shi, Z., Wei, J., Xu, Z., and Liang, Y. Why larger language models do in-context learning differently? *arXiv preprint arXiv:2405.19592*, 2024.

Siegelmann, H. T. and Sontag, E. D. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pp. 440–449, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130432. URL https://doi.org/10.1145/130385.130432.

Song, Z. and Yu, Z. Oblivious sketching-based central path method for solving linear programming problems. In *38th International Conference on Machine Learning (ICML)*, 2021.

Song, Z., Yin, J., and Zhang, L. Solving attention kernel regression problem via pre-conditioner. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2024.

Strassen, V. et al. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

Tenney, I., Das, D., and Pavlick, E. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vig, J. and Belinkov, Y. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 63–76, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4808. URL https://aclanthology.org/W19-4808.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wei, C., Chen, Y., and Ma, T. Statistically meaningful approximation: a case study on approximating turing machines with transformers, 2021. URL https://arxiv.org/abs/2107.13163.

Williams, R. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

Williams, V. V., Xu, Y., Xu, Z., and Zhou, R. New bounds for matrix multiplication: from alpha to omega. *CoRR*, abs/2307.07970, 2023.

Wu, J., Yu, T., Wang, R., Song, Z., Zhang, R., Zhao, H., Lu, C., Li, S., and Henao, R. Infoprompt: Information-theoretic soft prompt tuning for natural language understanding. *arXiv preprint arXiv:2306.04933*, 2023.

Xu, Z., Shi, Z., and Liang, Y. Do large language models have compositional ability? an investigation into limitations and scalability. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.

Yao, S., Peng, B., Papadimitriou, C., and Narasimhan, K. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3770–3785, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.292. URL https://aclanthology.org/2021.acl-long.292.

Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S., and Kumar, S. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pp. 36–39. IEEE, 2019.

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

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022a.

Zhang, Y., Backurs, A., Bubeck, S., Eldan, R., Gunasekar, S., and Wagner, T. Unveiling transformers with lego: a synthetic reasoning task, 2022b. URL https://arxiv.org/abs/2206.04301.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

Zhao, H., Panigrahi, A., Ge, R., and Arora, S. Do transformers parse while predicting the masked word? *arXiv preprint arXiv:2303.08117*, 2023.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Zwick, U. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)*, 49(3):289–317, 2002.

Appendix

Roadmap.

In Section A, we provide several basic notations, definitions and more related work. In Section B, we present our dynamic data-structure. Our algorithm shows the upper bound results. In Section C, we give our conditional lower bound result by assuming the Hinted MV conjecture.

A. Preliminary

Notations For a matrix A, we use $A^\top$ to denote its transpose. For a non-zero diagonal matrix $D \in \mathbb{R}^{n \times n}$, we use $D^{-1} \in \mathbb{R}^{n \times n}$ to denote the matrix where the (i,i)-th diagonal entry is $(D_{i,i})^{-1}$ for all $i \in [n]$.

For a vector $x \in \mathbb{R}^n$, we use $\mathrm{diag}(x) \in \mathbb{R}^{n \times n}$ to denote an $n \times n$ matrix where the i,i-th entry on the diagonal is x_i and zero everywhere else for all $i \in [n]$.

In many theoretical computer science (TCS)/machine learning (ML) literature, $\exp(M)$ denotes the matrix exponential, i.e., $\exp(M) = \sum_{i=0}^{\infty} \frac{1}{i!} M^i$. However, in this paper, we use $\exp(M)$ to denote the entry-wise exponential, i.e.,

$$\exp(M)_{i,j} := \exp(M_{i,j}).$$

We use $\mathbf{1}_n$ to denote the length-n vector where all the entries are ones. We use $\mathbf{0}_n$ to denote the length-n vector where all entries are zeros.

In this work, we use standard notation $\mathcal{T}_{\mathrm{mat}}(\cdot,\cdot,\cdot)$ (see Definition 2.1) and $\omega(\cdot,\cdot,\cdot)$ (see Definition 2.2) for describing the running time of matrix multiplication, see literature (Demetrescu & Italiano, 2000; Zwick, 2002; Sankowski, 2004; 2005; Gall & Urrutia, 2018; Brand & Nanongkai, 2019; Cohen et al., 2019; Lee et al., 2019; Brand et al., 2019; Brand, 2020; Gu & Ren, 2021; Jiang et al., 2021; Brand, 2021; Williams et al., 2023) for examples.

Detailed Comparison with (Alman & Song, 2023) In (Alman & Song, 2023), from the upper bound side, they make use of the 'polynomial method in algorithm design'. The polynomial method is a technique for finding low-rank approximations of $f(M)$, where M is a matrix and f is an entry-wise function. They apply a polynomial method to decompose $\exp(QK^\top)$ to $U_1 U_2$, where U_1 and U_2 are low rank matrices. Hence, for the follow-up attention computation (i.e., $\exp(QK^\top)V$) , they can first compute $U_2 V$, and then compute $U_1(U_2 V)$. As U_1 and U_2 are low rank matrices, these two steps can be computed efficiently. From the lower bound perspective, they give a fine-grained reduction from the Approximate Nearest Neighbor search (ANN) to attention problems. The hypothesis uses the Strong exponential time hypothesis.

In our case, from the upper bound side, we first proposed a data-structure that efficiently solves the Online Diagonal-based normalized Attention Matrix Vector multiplication problem by using the lazy update techniques. Instead of updating the target matrix every time, we set a hyperparameter a that lets the user strike the balance between the query time and the update time. From the lower bound side, we make use of a variation of the popular online matrix vector multiplication conjecture which is called hinted matrix vector multiplication conjecture. Notably, our work achieves congruence between upper and lower bound results for dynamically maintaining attention computations.

B. Main Upper Bound

In Section B.1, we show the running time of initializing our data structure. In Section B.2, we show the running time of updating K and V. In Section B.3, we show the correctness and the running time of querying the target matrix. In Section B.4, we show the correctness and the running time of recomputing the variables in our data-structure.

We propose our upper bound result as the following:

Theorem B.1 (Main algorithm, formal version of Theorem 1.3)**.** *For any constant $a \in (0,1]$. Let $d = O(n)$. There is a dynamic data structure that uses $O(n^2)$ space and supports the following operations:*

- INIT(Q,K,V). *It runs in $O(\mathcal{T}_{\mathrm{mat}}(n,d,n))$ time.*
- UPDATEK$(i \in [n], j \in [d], \delta \in \mathbb{R})$. *This operation updates one entry in K, and it runs in $O(\mathcal{T}_{\mathrm{mat}}(n,n^a,n)/n^a)$ amortized time.*

Algorithm 4 Algorithm that query the $\{i, j\}$-th element in the target matrix

data structure DYNAMICATTENTION ▷ Theorem B.1
procedure QUERY($i \in [n], j \in [d]$) ▷ Lemma B.6, B.5
  Let $\Delta_{V,1}$ and $\Delta_{V,2}$ be rectangular matrix obtained from list from V
  Let $(D_{\mathrm{tmp}})_i^{-1}$ denote the list of diagonal matrices obtained from $\mathrm{List}_D[\mathrm{ct}_K]$.GETB ▷ This takes $O(1)$ time
  /*Below is the target*/
  answer $\leftarrow ((D_{\mathrm{tmp}}^{-1}) \cdot (A) \cdot (V + \Delta_{V,1}\Delta_{V,2}))_{i,j}$
  /*The actual computation*/
  /*Part 1. Answer, This is fast because we store $C = AV$*/
  $\mathrm{answer}_1 \leftarrow (D_{\mathrm{tmp}})_i^{-1}(C_{i,j} + (\Delta_{C,1}\Delta_{C,2})_{i,j})$ ▷ $O(n^a)$ time
  /*Part 2. Answer, this is fast because each column of $\Delta_{V,1}$ and row of $\Delta_{V,2}$ is 1-sparse*/
  $\mathrm{answer}_2 \leftarrow (D_{\mathrm{tmp}})_i^{-1} A_{i,*}\Delta_{V,1}(\Delta_{V,2})_{*,j}$ ▷ $O(n^a)$ time
  answer $\leftarrow \sum_{j=1}^{2} \mathrm{answer}_j$
  return answer
end procedure
end data structure

Algorithm 5 Algorithm that re-compute evreything

data structure DYNAMICATTENTION ▷ Theorem B.1
procedure RECOMPUTE() ▷ Lemma B.8, Lemma B.7
  Let $\Delta_{C,1}$ and $\Delta_{C,2}$ be rectangular matrix obtained from List_C
  Let $\Delta_{V,1}$ and $\Delta_{V,2}$ be rectangular matrix obtained from List_V
  Let $\Delta_D(i)$ denote the list of diagonal matrices obtained from $\mathrm{List}_D[i]$.GETA
  $\widetilde{C} \leftarrow C + \Delta_{C,1} \cdot \Delta_{C,2} + A\Delta_{V,1} \cdot \Delta_{V,2}$ ▷ It takes $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$ time
  $\widetilde{V} \leftarrow V + \Delta_{V,1} \cdot \Delta_{V,2}$ ▷ It takes $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$ time
  $\Delta_D \leftarrow \sum_{i=1}^{\mathrm{ct}_K} \Delta_D(i)$ ▷ It takes n^{1+a} time
  $\widetilde{D}^{-1} \leftarrow D^{-1} + \Delta_D$ ▷ It takes n time
  $\widetilde{B} \leftarrow \widetilde{D}^{-1} \cdot \widetilde{C}$ ▷ This takes nd
  /*Refresh the memory*/
  $D \leftarrow \widetilde{D}, C \leftarrow \widetilde{C}, B \leftarrow \widetilde{B}, V \leftarrow \widetilde{V}$
  /*Reset the counter*/
  $\mathrm{ct}_K \leftarrow 0, \mathrm{ct}_V \leftarrow 0$
end procedure
end data structure

- UPDATEV($i \in [n], j \in [d], \delta \in \mathbb{R}$). *This operation takes same amortized time as K update.*
- QUERY($i \in [n], j \in [d]$). *This operation outputs $(D^{-1}(\exp(QK^\top))V)_{i,j}$ operation takes in $O(n^a)$ worst case time.*

B.1. Initialization

We first give the running time of the initialization procedure.

Lemma B.2 (Init)**.** *The procedure* INIT *(Algorithm 1) takes $\mathcal{T}_{\mathrm{mat}}(n, d, n)$ time.*

Proof. It is trivially from applying fast matrix multiplication. □

B.2. Update

Next, we give the running time of updating K.

Lemma B.3 (Running time of UPDATEK)**.** *The procedure* UPDATEK *(Algorithm 2) takes*

- *Part 1. $\mathcal{T}_{\mathrm{mat}}(n, n, n^a)$ time in the worst case*

- *Part 2.* $\mathcal{T}_{\mathrm{mat}}(n,n,n^a)/n^a$ *time in the amortized case*

Proof. **Part 1.** It trivially from Lemma B.8

Part 2. If the $\mathrm{ct}_K < n^a$, we pay $O(n)$ time. If $\mathrm{ct}_K = n^a$, we pay $n^{\omega(1,1,a)}$. So the amortized time is

$$\frac{n(n^a-1)+n^{\omega(1,1,a)}}{n^a} = O(n^{\omega(1,1,a)-a})$$

Note that, by using fast matrix multiplication and the fact that $d = O(n)$, we have $n^{\omega(1,1,a)} = \mathcal{T}_{\mathrm{mat}}(n,n^a,d)$. Thus we complete the proof. □

Now, we give the running time of updating V.

Lemma B.4 (Running time of UPDATEV)**.** *The procedure* UPDATEV *(Algorithm 3) takes*

- *Part 1.* $\mathcal{T}_{\mathrm{mat}}(n,n,n^a)$ *time in the worst case.*
- *Part 2.* $\mathcal{T}_{\mathrm{mat}}(n,n,n^a)/n^a$ *time in the amortized case.*

Proof. **Part 1.** It trivially from Lemma B.8.

Part 2. If the $\mathrm{ct}_K < n^a$, we pay $O(n)$ time. If $\mathrm{ct}_K = n^a$, we pay $n^{\omega(1,1,a)}$. So the amortized time is

$$\frac{n(n^a-1)+n^{\omega(1,1,a)}}{n^a} = O(n^{\omega(1,1,a)-a})$$

Note that, by using fast matrix multiplication and the fact that $d = O(n)$, we have $n^{\omega(1,1,a)} = \mathcal{T}_{\mathrm{mat}}(n,n^a,d)$. Thus we complete the proof. □

B.3. Query

We show the correctness of our QUERY that queries only one element in the target matrix.

Lemma B.5 (Correctness of QUERY)**.** *The procedure* QUERY *(Algorithm 4) outputs*

$$\begin{aligned}\widetilde{B}_{i,j} &= (D^{-1}\cdot A\cdot (V+\Delta_V))_{i,j}\\ &= (D^{-1}AV + D^{-1}A\Delta_V)_{i,j}\end{aligned}$$

Proof. Let $\Delta_{V,1}$ denote the vector obtained from $\mathrm{List}_D[\mathrm{ct}_K]$.GETA.

Let $\Delta_{V,2}$ denote the vector obtained from $\mathrm{List}_D[\mathrm{ct}_K]$.GETB

Let $(D_{\mathrm{tmp}})_i^{-1}$ denote the list of diagonal matrices obtained from $\mathrm{List}_D[\mathrm{ct}_K]$.GETB

We know

$$\begin{aligned}\widetilde{B} &= ((D_{\mathrm{tmp}}^{-1})\cdot(A)\cdot(V+\Delta_{V,1}\Delta_{V,2}))\\ &= (D_{\mathrm{tmp}})^{-1}(AV) + (D_{\mathrm{tmp}})^{-1}(A\Delta_{V,1}\Delta_{V,2})\end{aligned}$$

For the $\{i,j\}$-th element, by using simple algebra, we have

$$\begin{aligned}\widetilde{B}_{i,j} &= (D_{\mathrm{tmp}})_i^{-1}(AV)_{i,j} + (D_{\mathrm{tmp}})_i^{-1}(A\Delta_{V,1}\Delta_{V,2})\\ &= (D_{\mathrm{tmp}})_i^{-1}(C+\Delta_{C,1}\cdot\Delta_{C,2})_{i,j} + (D_{\mathrm{tmp}})_i^{-1}(A\Delta_{V,1}\Delta_{V,2})_{i,j}\\ &= (D_{\mathrm{tmp}})_i^{-1}(C+\Delta_{C,1}\cdot\Delta_{C,2})_{i,j} + (D_{\mathrm{tmp}})_i^{-1}A_{i,*}\Delta_{V,1}(\Delta_{V,2})_{*,j}\end{aligned}$$

We know

$$\text{answer}_1 = (D_{\text{tmp}})_i^{-1}(C + \Delta_{C,1} \cdot \Delta_{C,2})_{i,j}$$

and

$$\text{answer}_2 = (D_{\text{tmp}})_i^{-1} A_{i,*} \Delta_{V,1} (\Delta_{V,2})_{*,j}$$

By summing up answer_1 and answer_2, we have

$$\widetilde{B}_{i,j} = (D^{-1}AV + D^{-1}A\Delta_V)_{i,j}.$$

Now, we complete the proof. □

Next, we give the running time of it.

Lemma B.6 (Running time of QUERY)**.** *The running time of procedure* QUERY *(Algorithm 4) is* $O(n^a)$.

Proof. We first stack all the vectors in List_V to $\Delta_{V,1} \in \mathbb{R}^{n \times n^a}$ and $\Delta_{V,2} \in \mathbb{R}^{n^a \times d}$, which takes $O(1)$ time.

- Computing $(D_{\text{tmp}})_i^{-1}(C + \Delta_{C,1} \cdot \Delta_{C,2})_{i,j}$ takes $O(n^a)$ time.
- Computing $(\Delta_{V,1}\Delta_{V,2})$ takes $O(n^a)$ time as $\Delta_{V,1}$ is 1-sparse in columns and $(\Delta_{V,2})$ is 1-sparse in rows.
- Computing $(D_{\text{tmp}})_i^{-1} A_{i,*}(\Delta_{V,1}\Delta_{V,2})_{*,j}$ takes $O(n^a)$ time as $\text{nnz}((\Delta_{V,1}\Delta_{V,2})_{*,j}) \leq n^a$.

Hence, the total running time needed is $O(n^a)$ □

B.4. Re-compute

We show the correctness of our re-compute function.

Lemma B.7 (Correctness of RECOMPUTE)**.** *The procedure* RECOMPUTE *(Algorithm 5) correctly re-compute* D, C, B, V.

Proof. **Part 1.** Re-compute D

Let $\Delta_D(i)$ denote the list of diagonal matrices obtained from $\text{List}_D[i]$.GETA. Then, the total difference between the updated $\widetilde{D}$ and D is $\sum_{i=1}^{\text{ct}_K} \Delta_D(i)$.

By computing $\widetilde{D}^{-1} \leftarrow D^{-1} + \Delta_D$, we correctly get the updated $\widetilde{D}^{-1}$. By computing the inverse of a diagonal matrix we get $\widetilde{D}$.

Part 2. Re-compute V

We first stack all the vectors in List_V to $\Delta_{V,1} \in \mathbb{R}^{n \times n^a}$ and $\Delta_{V,2} \in \mathbb{R}^{n^a \times d}$.

By using Fact 2.3, we have $\widetilde{V} = V + \Delta_{V,1} \cdot \Delta_{V,2}$.

Part 3. Re-compute C

Similar to the proof of re-computing V.

We first stack all the vectors in List_C to $\Delta_{C,1} \in \mathbb{R}^{n \times n^a}$ and $\Delta_{C,2} \in \mathbb{R}^{n^a \times d}$.

By using Fact 2.3, we have $\widetilde{C} = C + \Delta_{C,1} \cdot \Delta_{C,2} + A\Delta_{V,1} \cdot \Delta_{V,2}$.

Part 4. Re-compute B

By using the definition of $B = D^{-1}C$, we can update B by using $\widetilde{B} = \widetilde{D}^{-1} \cdot \widetilde{C}$.

Now, we complete the proof. □

Next, we give the running time of it.

Lemma B.8 (Running time of RECOMPUTE)**.** *The running time of procedure* RECOMPUTE *(Algorithm 5) is* $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$.

Proof. We first stack all the vectors in List_V to $\Delta_{V,1} \in \mathbb{R}^{n \times n^a}$ and $\Delta_{V,2} \in \mathbb{R}^{n^a \times d}$, which takes $O(1)$ time.

We stack all the vectors in List_C to $\Delta_{C,1} \in \mathbb{R}^{n \times n^a}$ and $\Delta_{C,2} \in \mathbb{R}^{n^a \times d}$, which takes $O(1)$ time.

- Computing $C + \Delta_{C,1} \cdot \Delta_{C,2} + A\Delta_{V,1} \cdot \Delta_{V,2}$ takes $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$ time.
- Computing $V + \Delta_{V,1} \cdot \Delta_{V,2}$ takes $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$ time.
- Computing $\sum_{i=1}^{\mathrm{ct}_K} \Delta_D(i)$ takes $O(n^{a+1})$ time as $\mathrm{nnz}(\Delta_D(i)) = O(n)$ and $\mathrm{ct}_K = O(n^a)$.
- Computing $D^{-1} + \Delta_D$ takes $O(n)$ time as $\mathrm{nnz}(\Delta_D) = O(n)$.
- Computing $\widetilde{D}^{-1} \cdot \widetilde{C}$ takes $O(nd)$ time as $\widetilde{D}^{-1}$ is a diagonal matrix. Hence, the total running time is $\mathcal{T}_{\mathrm{mat}}(n, n^a, d)$.

□

C. Main Lower Bound

In Section C.1, we give the definition of Online Matrix Vector (OMV) problem. In Section C.2, we introduce the definition of Hinted MV and its conjecture (from previous work (Brand et al., 2019)). In Section C.3, we show the hardness of computing the target matrix with the normalization factor.

C.1. Online Matrix Vector Multiplication

Before studying the hardness of our problem, we first review a famous problem in theoretical computer science which is called online matrix vector multiplication problem. Here is the definition of online matrix vector multiplication, which has been a crucial task in many fundamental optimization problems.

Definition C.1 (Online Matrix Vector (OMV) (Henzinger et al., 2015; Larsen & Williams, 2017; Chakraborty et al., 2018))**.** *Given a matrix $A \in \{0,1\}^{n \times n}$, let $T = O(n)$, there is an online sequence of vectors $u_1, \cdots, u_T \in \{0,1\}^n$. The goal is to design a structure that whenever receives a new vector u_t and output Au_t.*

Such a problem is widely believed in the community that there is no algorithm to solve it in truly subquadratic time per vector and there is no algorithm to solve it in truly subcubic time over all vectors.

C.2. Hardness from Previous Work

We define the hinted Mv problem from previous work (Brand et al., 2019).

Definition C.2 (Hinted MV (HMV) Definition 5.6 of (Brand et al., 2019))**.** *Let the computations be performed over the boolean semi-ring and let $m = n^\tau$, $0 < \tau \leq 1$. The hinted Mv problem consists of the following phases:*

1. *Input two $n \times n$ matrices M and V*
2. *Input an $n \times n$ matrix P with at most n^τ non-zero entries*
3. *Input a single index $i \in [n]$*
   - *We need to answer $MPV_{*,i}$*
   - *Here $V_{*,i} \in \mathbb{R}^n$ is the i-th column of matrix V*

We give the hinted Mv conjecture which is from prior work (Brand et al., 2019).

Conjecture C.3 (HMV conjecture 5.2 of (Brand et al., 2019), restatement of Conjecture 3.1)**.** *For every constant $0 < \tau \leq 1$ no algorithm for the hinted Mv problem (Definition C.2) can simultaneously satisfy*

- *polynomial time in phase 1*

- $O(n^{\omega(1,1,\tau)-\epsilon})$ *time complexity in phase 2 and*
- $O(n^{1+\tau-\epsilon})$ *in phase 3*

for some constant $\epsilon > 0$.

C.3. Online Diagonal-normalized Attention Matrix Vector Multiplication

Next, we consider the normalization factor and defined the problem as the following.

Definition C.4 ($\mathsf{ODAMV}(n,d)$, restatement of Definition 1.2)**.** *The goal of* ***O****nline* ***D****iagonal-based normalized* ***A****ttention* ***M****atrix* ***V****ector Multiplication problem* $\mathsf{ODAMV}(n,d)$ *is to design a data structure that satisfies the following operations:*

1. INIT*: Initialize on* $n \times d$ *matrices* Q, K, V.
2. UPDATE*: Change any entry of* Q, K, *or* V.
3. QUERY*: For any given* $i \in [n]$, $j \in [d]$, *return* $(D^{-1}\exp(QK^\top)V)_{i,j}$, *where* $D = \mathrm{diag}(\exp(QK^\top)\mathbf{1}_n)$.

Next, we present our lower bound result with the normalization factor.

Lemma C.5. *Assuming the hinted* Mv *conjecture (Conjecture C.3): For every constant* $0 < \tau \leq 1$, *there is no algorithm that solve* $\mathsf{ODAMV}(n,d)$ *problem (Definition C.4) with*

- *polynomial initialization time, and*
- *amortized update time* $O(\mathcal{T}_{\mathrm{mat}}(n,n^\tau,d)/n^{\tau+\Omega(1)})$, *and*
- *worst query time* $O(n^{\tau-\Omega(1)})$.

Proof. Assume there was a dynamic algorithm faster than what is stated in Lemma C.5 for some parameter τ, i.e. update time $O(\mathcal{T}_{\mathrm{mat}}(n,n^\tau,d)/n^{\tau+\epsilon})$ and query time $O(n^{\tau-\epsilon})$ for some constant $\epsilon > 0$. We show that this would contradict the hinted Mv conjecture (Conjecture C.3).

Let us take an instance for the v-hinted Mv problem (Definition C.2) with $M \in \{0,1\}^{n\times n}, V \in \{0,1\}^{n\times n}$.

We can construct matrix $\mathsf{M} \in \{0,1\}^{n\times 2n}$ and $\mathsf{V} \in \{0,1\}^{2n\times n}$ as follows

$$\mathsf{M} := \begin{bmatrix} M & \overline{M} \end{bmatrix} \quad \text{and} \quad \mathsf{V} := \begin{bmatrix} V \\ \mathbf{0}_{n\times n} \end{bmatrix}$$

where $\overline{M}$ is a matrix that $\overline{M}_{i,j} = 1 - M_{i,j}$.

Note that $\|\mathsf{M}_{i,*}\|_1 = n$, for each $i \in [n]$.

Based on the above construction, we will create a new instance $\mathsf{ODAMV}(\widetilde{n} = 2n, \widetilde{d} = 2n)$, where

$$\widetilde{Q} = \begin{bmatrix} \mathsf{M} \\ \mathbf{0}_{n\times 2n} \end{bmatrix}, \quad \widetilde{K} = \mathbf{0}_{2n\times 2n}, \quad \widetilde{V} = \begin{bmatrix} \mathsf{V} & \mathbf{0}_{2n\times n} \end{bmatrix}$$

During phase 1, we give this input to the dynamic algorithm for the ODAMV problem (Definition C.4).

Let $D \in \{0,1\}^{n\times n}$ denote a diagonal matrix, where $\mathrm{nnz}(D) = n^\tau$

During phase 2, we receive the $2n \times 2n$ diagonal matrix P, where

$$\mathsf{P} = \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix}$$

and $\mathrm{nnz}(\mathsf{P}) = 2n^\tau$.

We perform $2n^\tau$ updates to the data structure to set $\widetilde{K}^\top = \mathsf{P}$. This takes

$$O(\widetilde{n}^\tau \cdot (\mathcal{T}_{\mathrm{mat}}(\widetilde{n}, \widetilde{n}^\tau, \widetilde{d})/\widetilde{n}^{\tau+\epsilon})) = O(n^{\omega(1,1,\tau)-\epsilon})$$

time.

Note that

- $\|\widetilde{Q}_{i,*}\|_1 = n$, for each $i \in [n]$.
- $\|\widetilde{Q}_{i,*}\|_1 = 0$, for each $i \in [n+1, 2n]$.

By using the definition of P, we know that, for each $i \in [n]$

$$\widetilde{D}_{i,i} = n^\tau \exp(1) + n^\tau \exp(0) = n^\tau (e+1).$$

For each $i \in [n+1, 2n]$

$$\widetilde{D}_{i,i} = n^\tau \exp(0) = n^\tau. \tag{2}$$

Hence, we don't need to update $\widetilde{D}$.

At last, in phase 3, we perform $\widetilde{n}$ queries to obtain the column $\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V}_{*,i}$ in $O(\widetilde{n} \cdot \widetilde{n}^{\tau-\epsilon}) = O(n^{1+\tau-\epsilon})$ time.

Using Claim C.7 and Claim C.6, we know that, for any $i \in [n]$ and for any $j \in [n]$, if there is an algorithm that can find $(\widetilde{D}^{-1}\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V})_{j,i}$, then using $(\widetilde{D}^{-1}\exp(\widetilde{Q}\widetilde{K}^\top)\widetilde{V})_{j,i} - (\widetilde{D}^{-1}\widetilde{V})_{j,i}$ is enough to reconstruct $(\mathsf{MPV})_{j,i}$. Here $\widetilde{D}^{-1}\widetilde{V}$ can be computed in just $O(1)$ time via Eq. (2). Thus, we can know the $(MDV)_{j,i}$ for the hinted Mv problem in $O(n^{1+\tau\epsilon})$ time, contradicting the hinted Mv conjecture.

□

Claim C.6. *For each $i \in [n]$ and $j \in [n]$, if $(\widetilde{D}^{-1}(\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i}$ is > 0, then $(\mathsf{MPV})_{j,i} = 1$,*

Proof. By using the fact that $n^\tau(e+1) > 0$ and $n^\tau > 0$, we have

$$\begin{aligned}\widetilde{D}^{-1}(\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i} > 0 \\ ((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i} > 0\end{aligned}$$

We know

$$\widetilde{Q} = \begin{bmatrix} \mathsf{M} \\ \mathbf{0}_{n\times 2n} \end{bmatrix}, \quad \widetilde{K}^\top = \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix}, \quad \widetilde{V} = \begin{bmatrix} \mathsf{V} & \mathbf{0}_{2n\times n} \end{bmatrix},$$

so we have

$$((\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})\mathsf{V})_{j,i} > 0.$$

For $k \in [n+1, 2n]$, as $\mathsf{V} = \begin{bmatrix} V \\ \mathbf{0}_{n\times n} \end{bmatrix}$, we know $(\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})_{j,k}(\mathsf{V})_{k,i} = 0$.

Using the definition of matrix multiplication, and the fact that $\exp(x) > 1$ for all $x > 0$, we have some $k \in [n]$ with

$$\begin{aligned}(\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})_{j,k}(\mathsf{V})_{k,i} > 0 \\ (\exp(\mathsf{MP})_{j,k} - 1)(\mathsf{V})_{k,i} > 0\end{aligned}$$

We can conclude that for each $i \in [n], j \in [n]$, there is at least one $k \in [n]$ such that

- $\mathsf{V}_{k,i} > 0$

- $(\mathsf{MP})_{j,k} > 0$

Therefore, by using the definition of boolean semi-ring, we can conclude that $(\mathsf{MPV})_{j,i} = 1$

□

Claim C.7. *For each $i \in [n]$ and $j \in [n]$, if $(\widetilde{D}^{-1}(\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i}$ is 0 then $(\mathsf{MPV})_{j,i} = 0$.*

Proof. By using the fact that $n^\tau(e+1) > 0$ and $n^\tau > 0$, we have

$$\widetilde{D}^{-1}(\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i} = 0$$
$$((\exp(\widetilde{Q}\widetilde{K}^\top) - \mathbf{1}_{\widetilde{n}\times\widetilde{n}})\widetilde{V})_{j,i} = 0$$

We know

$$\widetilde{Q} = \begin{bmatrix} \mathsf{M} \\ \mathbf{0}_{n\times 2n} \end{bmatrix}, \quad \widetilde{K}^\top = \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix}, \quad \widetilde{V} = \begin{bmatrix} \mathsf{V} & \mathbf{0}_{2n\times n} \end{bmatrix},$$

so we have

$$((\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})\mathsf{V})_{j,i} = 0.$$

For $k \in [n+1, 2n]$, as $\mathsf{V} = \begin{bmatrix} V \\ \mathbf{0}_{n\times n} \end{bmatrix}$, we know $(\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})_{j,k}(\mathsf{V})_{k,i} = 0$.

For all $k \in [n]$ such that $\mathsf{V}_{k,i} = 1$, we have $(\exp(\mathsf{MP}) - \mathbf{1}_{n\times 2n})_{j,k} = 0$, which also implies that $(\mathsf{MP})_{j,k} = 0$.

Now, we can conclude that $(\mathsf{MPV})_{j,i} = 0$ for each $i \in [n]$ and $j \in [n]$. □