PolySketchFormer: Fast Transformers via Sketching Polynomial Kernels

Praneeth Kacham [*] [1] [2] **Vahab Mirrokni** [*] [1] **Peilin Zhong** [*] [1]

Abstract

The quadratic time and memory complexity inherent to self-attention mechanisms, with respect to sequence length, presents a critical computational bottleneck in the training and deployment of large-scale Transformer-based language models. Recent theoretical results indicate the intractability of sub-quadratic softmax attention approximation under reasonable complexity assumptions. This paper addresses this challenge by first demonstrating that polynomial attention with high degree can effectively replace softmax without sacrificing model quality. Next, we develop polynomial sketching techniques from numerical linear algebra to achieve linear-time polynomial attention with approximation guarantees. Crucially, our approach achieves this speedup without requiring the sparsification of attention matrices. We also present a block-based algorithm to apply causal masking efficiently. Combining these techniques, we provide *PolySketchFormer*, a practical linear-time Transformer architecture for language modeling that offers provable guarantees. We validate PolySketchFormer empirically by training language models capable of handling long contexts. These experiments utilize both synthetic and real-world datasets (PG19, Wikipedia and C4) on Google Cloud TPUs. For context lengths of 32k and GPT-2 style models, our model achieves 2x speedup in training compared to FlashAttention of the fastest configuration, with no observed degradation in quality across our experiments.[1]

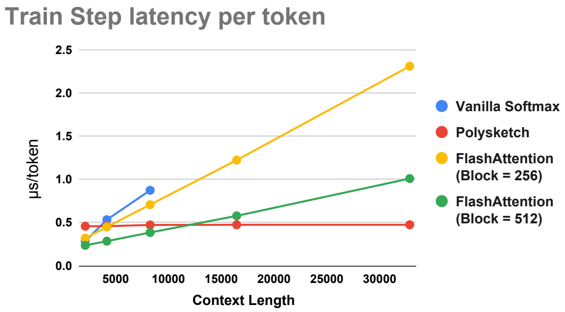

Figure 1. Train step latency per token in μs/token of GPT-2 small style models with softmax attention (FlashAttention) v.s. ours. Each model is trained with 1M tokens batches. Vanilla softmax attention goes out-of-memory (OOM) for context lengths > 8k.

1. Introduction

Transformer-based models (Vaswani et al., 2017) are state-of-the-art for many natural language tasks, leading to breakthroughs in machine translation, language understanding (Devlin et al., 2019), and language modeling (Brown et al., 2020; Chowdhery et al., 2023; OpenAI, 2023; Anil et al., 2023). However, the quadratic time and space complexity of the attention mechanism limits scalability for long context lengths. Numerous "efficient transformers" have been proposed to address this issue (Wang et al., 2020; Katharopoulos et al., 2020; Choromanski et al., 2020; Han et al., 2023). These variants approximate[2] the standard attention mechanism. A survey by Tay et al. (2022) provides a broad overview of these techniques. While many efficient transformer constructions achieve linear theoretical training complexity, the survey observes that practical training speedups are often less significant, with potential losses in model quality. This explains the continued dominance of vanilla transformers.

In this work, we focus on improving training latency for transformer models in decoding-only tasks, specifically language modeling trained via next-word prediction. We will first briefly discuss existing approaches to make training of transformer models faster and then place our contributions in context.

Memory efficient and I/O aware approach. Recent work

[*] Equal contribution [1] Google Research [2] Carnegie Mellon University. Correspondence to: Praneeth Kacham <pkacham@google.com>, Vahab Mirrokni <mirrokni@google.com>, Peilin Zhong <peilinz@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1] Our implementation is available at `https://github.com/google-research/google-research/tree/master/polysketchformer`

[2] "Approximation" is used informally here, since some "efficient transformers" deviate significantly from the vanilla model.

by (Dao et al., 2022; Dao, 2023) on FlashAttention and FlashAttention-2 seeks to enable vanilla transformer training on long contexts. This is achieved through I/O-aware optimizations like blocking/tiling and rematerialization, significantly improving memory efficiency. While this reduces the $O(n^2)$[3] HBM (High-Bandwidth Memory) requirements of accelerators (GPUs/TPUs), enabling training on thousands of tokens, the computational cost per step remains $O(n^2)$ (see Figure 1), and this remains a barrier to further scaling the context length.

Approximate softmax attention via sparsification. Another line of work tries to approximate softmax attention and avoid $n \times n$ attention matrix computation by focusing on a smaller set of pairs of *query* and *key* vectors. Techniques include utilizing locality/positional information (Child et al., 2019; Beltagy et al., 2020; Xiao et al., 2023; Zaheer et al., 2020; Roy et al., 2021; Ding et al., 2023), hashing/bucketing (Kitaev et al., 2020; Sun et al., 2021; Han et al., 2023), low-rank projection (Wang et al., 2020), or other sparsification methods. In these cases, there is usually some trade-off between model quality and sparsity, i.e., denser attentions improve quality but decrease speed. Hence, an efficient high-quality $n \times n$ attention mechanism may potentially improve on these sparsification-based techniques.

Efficient $n \times n$ attention by kernel-based methods. The kernel-based view of attention was taken by a series of earlier works (Tsai et al., 2019; Katharopoulos et al., 2020; Choromanski et al., 2020; Peng et al., 2021). In particular, let $\{\mathbf{q}_i \in \mathbb{R}^h\}_{i\in[n]}, \{\mathbf{k}_i \in \mathbb{R}^h\}_{i\in[n]}, \{\mathbf{v}_i \in \mathbb{R}^h\}_{i\in[n]}$ be sets of *query*, *key* and *value* vectors respectively, the output of the attention mechanism for query $\mathbf{q}_i$ is computed as $\mathsf{Attn}(\mathbf{q}_i, \{\mathbf{k}_j\}, \{\mathbf{v}_j\}) = \sum_{j\in[n]} \frac{\sigma(\mathbf{q}_i,\mathbf{k}_j)}{\sum_{j'\in[n]}\sigma(\mathbf{q}_i,\mathbf{k}_{j'})} \cdot \mathbf{v}_j^\top$. When the similarity kernel function $\sigma(\mathbf{x},\mathbf{y}) := \exp(\langle \mathbf{x},\mathbf{y}\rangle)$, the above attention is exactly the softmax attention[4]. If there exists a feature map ϕ such that $\sigma(\mathbf{x},\mathbf{y}) \equiv \langle \phi(\mathbf{x}), \phi(\mathbf{y})\rangle$, the attention output for query $\mathbf{q}_i$ can be rewritten as:

$$\begin{aligned}\mathsf{Attn}(\mathbf{q}_i, \{\mathbf{k}_j\}, \{\mathbf{v}_j\}) &= \sum_{j\in[n]} \frac{\phi(\mathbf{q}_i)^\top \cdot \phi(\mathbf{k}_j)}{\sum_{j'\in[n]} \phi(\mathbf{q}_i)^\top \cdot \phi(\mathbf{k}_{j'})} \cdot \mathbf{v}_j^\top \\ &= \frac{\phi(\mathbf{q}_i)^\top \cdot \sum_{j\in[n]} \phi(\mathbf{k}_j)\cdot \mathbf{v}_j^\top}{\phi(\mathbf{q}_i)^\top \cdot \sum_{j'\in[n]} \phi(\mathbf{k}_{j'})}.\end{aligned}$$

If $\phi(\cdot)$ has a finite dimension h', one can first compute $\sum_{j'\in[n]} \phi(\mathbf{k}_{j'})$ and $\sum_{j\in[n]} \phi(\mathbf{k}_j)\cdot \mathbf{v}_j^\top$ in $O(nhh')$ time, and then compute $\mathsf{Attn}(\mathbf{q}_i, \{\mathbf{k}_j\}, \{\mathbf{v}_j\})$ for all $i \in [n]$ in another $O(nhh')$ time, which is linear in the context length n.

Most of existing works such as (Katharopoulos et al., 2020; Bolya et al., 2022; Tsai et al., 2019; Babiloni et al., 2023; Yang et al., 2023; Kasai et al., 2021) only consider similarity functions $\sigma(\mathbf{x},\mathbf{y})$ with low dimensional feature mapping (e.g., $\sigma(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y}\rangle$, $\langle \mathbf{x},\mathbf{y}\rangle^2$, $\langle \mathrm{elu}(\mathbf{x}) + \mathbf{1}, \mathrm{elu}(\mathbf{y}) + \mathbf{1}\rangle$, etc.). Hua et al. (2022) proposed to use a mixed strategy based on the positions of the tokens: If positions $i, j \in [n]$ are close enough, they use $\sigma(\mathbf{q}_i,\mathbf{k}_j) = \mathrm{relu}^2(\langle \mathbf{q}_i,\mathbf{k}_j\rangle)$. Otherwise, they use $\sigma(\mathbf{q}_i,\mathbf{k}_j) = \langle \mathbf{q}_i,\mathbf{k}_j\rangle$, which again has a low dimensional feature mapping. These simple similarity kernel functions $\sigma(\cdot)$ either suffer from some loss of model quality (Katharopoulos et al., 2020) or require additional tweaks of network structures (e.g., significantly increasing the number of attention layers (Hua et al., 2022), introducing decay factors for earlier tokens (Yang et al., 2023)) to achieve comparable model quality as softmax attention.

Some other previous works try to approximate the regular softmax attention via approximate feature mappings for the exponential similarity function. Random Feature Attention (Peng et al., 2021) uses random Fourier features to produce an approximate feature mapping but without provable approximation guarantees. Performer (Choromanski et al., 2020) provides a low dimensional approximate non-negative feature mapping $\phi'(\cdot)$ via positive orthogonal random features. It has provable approximation to the pairwise similarities, i.e., the maximum error $\max_{i,j\in[n]} |\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j)\rangle - \exp(\langle \mathbf{q}_i,\mathbf{k}_j\rangle)|$ is small. However, the dimension of their feature mapping has to grow exponentially in $\|\mathbf{q}_i\|_2^2$ and $\|\mathbf{k}_j\|_2^2$ to have a small error. In other words, consider a single query $\mathbf{q}_i$ and two keys $\mathbf{k}_j$ and $\mathbf{k}_{j'}$ such that all $\|\mathbf{q}_i\|_2, \|\mathbf{k}_j\|_2, \|\mathbf{k}_{j'}\|_2 \le R$, then $\exp(\langle \mathbf{q}_i,\mathbf{k}_j\rangle)/\exp(\langle \mathbf{q}_i,\mathbf{k}_{j'}\rangle) \le \exp(2R^2)$. Thus, the maximum relative probability masses that can be assigned while guaranteeing the approximation factor is limited by the dimension of the feature mapping used. In fact, a recent work (Alman & Song, 2023) implies that it is actually impossible to get accurate approximations for pairwise exponential similarity under Strong Exponential Time Hypothesis (SETH (Impagliazzo et al., 2001)) when the query and key vectors have large entries. Furthermore, it was observed empirically (Choromanski et al., 2020; Hua et al., 2022) (also see Figure 2) that there is a clear model quality drop in comparison with the exact softmax attention.

Given barriers above, a natural question arises: *Does there exist a similarity kernel function that achieves similar model quality as softmax attention while also admitting proper approximation by a low-dimensional feature mapping?*

1.1. Our Contributions

Polynomial similarity kernel function of high degree. To tackle the first part of the above question, we explore the power of the polynomial kernel function $\sigma(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y}\rangle^p$ for large even degrees $p \ge 4$ empirically for language modeling tasks. In particular, we look at the standard GPT-2 (Rad-

[3] n denotes the context length – the number of input tokens.

[4] In standard softmax attention, $\sigma(\mathbf{x},\mathbf{y}) := \exp(\langle \mathbf{x},\mathbf{y}\rangle/\sqrt{h})$. We omit $\sqrt{h}$ here for simplicity of the presentation.

ford et al., 2019) architecture (from the small size to the large size) and the strongest known Transformer recipe (a.k.a. Transformer++) which is a common baseline model studied in many previous works as well (Hua et al., 2022; Gu & Dao, 2023; Yang et al., 2023). We compare the models with vanilla softmax attention to the models that simply replace the attention mechanism with degree-p polynomial attention. We consider context lengths ranging from 512 to 32k. As shown in Figure 2 and our other empirical studies (see Section 4 and Appendix), for all synthetic tasks (including tasks for measuring content aware reasoning and memorization capabilities, see Appendix F), autoregressive pre-training metrics (perplexity) and few-shot evaluations that we studied, the models with degree-p polynomial attention ($p \geq 4$) achieve comparable performance as the models with the vanilla softmax attention. In addition, we discuss the behavioral similarities between softmax attention and polynomial attention in Section 2.1 to provide more intuitions why they had similar empirical outcomes.

Approximate feature mapping for polynomial kernel. Unlike exponential kernel whose exact feature mapping has infinite dimension, the feature mapping of degree-p polynomial kernel over $\mathbb{R}^h$ has a finite feature mapping of dimension h^p (see e.g., (Avron et al., 2014)). In practice, the head size h is usually 64, 128 or even 256 (Chowdhery et al., 2023). Therefore, computing the exact feature mapping for $p \geq 4$ is still expensive. To address this issue, we apply the sketching technique from the numerical linear algebra literature to compute a low-dimensional approximate feature mapping ϕ' such that $\langle \phi'(\mathbf{x}), \phi'(\mathbf{y}) \rangle \approx \langle \mathbf{x}, \mathbf{y} \rangle^p$. Sketching polynomial kernels (Avron et al., 2014; Ahle et al., 2020; Song et al., 2021; Meister et al., 2019) has been extensively studied in the literature, and the techniques are used in many applications such as kernel regression (Song et al., 2021), kernel PCA (Avron et al., 2014), evaluating elementwise matrix functions (Han et al., 2020), and etc. However, though $\langle \mathbf{x}, \mathbf{y} \rangle^p$ is guaranteed to be non-negative for even integer p, none of the approximate feature mappings provided by previous work guarantees $\langle \phi'(\mathbf{x}), \phi'(\mathbf{y}) \rangle \geq 0$. This is undesirable in practice since the original normalized attention weights $\frac{\langle \mathbf{q}_i, \mathbf{k}_1 \rangle^p}{\sum_{j \in [n]} \langle \mathbf{q}_i, \mathbf{k}_j \rangle^p}, \frac{\langle \mathbf{q}_i, \mathbf{k}_2 \rangle^p}{\sum_{j \in [n]} \langle \mathbf{q}_i, \mathbf{k}_j \rangle^p}, \ldots, \frac{\langle \mathbf{q}_i, \mathbf{k}_n \rangle^p}{\sum_{j \in [n]} \langle \mathbf{q}_i, \mathbf{k}_j \rangle^p}$ naturally represent a probability distribution, but the property does not hold when there exists some negative attention weight $\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j) \rangle$. More importantly, previous work (Choromanski et al., 2020; Katharopoulos et al., 2020) found that negative attention weights make the training process unstable, potentially causing non-convergence. To address this issue, we open the construction of (Ahle et al., 2020) and develop an approximate feature mapping with desired non-negativity property.

Theorem 1.1. *Let $p \geq 2$ be an even integer, $\varepsilon \in (0, 0.5)$ be an error parameter. Let h be the dimension of the vectors to be mapped. There is a randomized feature mapping $\phi' : \mathbb{R}^h \to \mathbb{R}^{r^2}$ for $r = \Theta(p\varepsilon^{-2} \log 1/\delta)$, such that given any set of vectors $\{\mathbf{q}_i \in \mathbb{R}^h\}_{i \in [n]}, \{\mathbf{k}_j \in \mathbb{R}^h\}_{i \in [n]}$:*

1. *$\forall i, j \in [n], \langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j) \rangle \geq 0$.*
2. *$\sum_{i,j} |\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j) \rangle - \langle \mathbf{q}_i, \mathbf{k}_j \rangle^p|^2 \leq \varepsilon^2 \sum_{i,j} \|\mathbf{q}_i\|_2^{2p} \|\mathbf{k}_j\|_2^{2p}$ holds with probability $1 - \delta$.*
3. *Computing $\phi'(\mathbf{x})$ only requires $p/2$ matrix-vector multiplications of matrix size $h \times r$, $(p/2 - 2)$ matrix-vector multiplications of matrix size $r \times r$, $(p/2 - 1)$ Hadamard products of r-dimensional vectors, and 1 self-Kronecker product of an r-dimensional vector.*

The first property above is the desired non-negativity property that we discussed earlier. We achieve this property by providing a "self-tensoring" technique stated in Theorem 2.4. The second property states our error bound. Unlike the approximation guarantee of (Choromanski et al., 2020), though our error bound is still in terms of ℓ_2 norms query and key vectors, it allows a larger ratio between attention weights due to the difference between exponential kernel and polynomial kernel (See more discussions in Appendix B). The third property implies that the computation of $\phi'(\cdot)$ only requires a small number of standard matrix/vector operations which can be implemented to run quickly on accelerators (GPUs/TPUs).

Inspired by the literature of learned sketches (Hsu et al., 2019; Aamand et al., 2019), we also propose a heuristic which replaces each random projection matrix used in $\phi'(\cdot)$ constructed in Theorem 1.1 with a comparable size learnable multi-layer dense network. Since each random matrix used in $\phi'(\cdot)$ has size only $h \times r$ or $r \times r$, the number of parameters that we add to the model is negligible in comparison with the model size. We observe significant model quality improvements (see Figure 2) by learning the sketches through training instead of using randomly sampled sketches.

Block-based lower triangular multiplication for handling causal masks. Another bottleneck in applying attention linearization techniques in training transformer models with causal masking on long contexts is to handle a huge number of sequential gradient updates due to RNN-style sequential dependence (Hua et al., 2022). When causal masking is applied, the attention between the query $\mathbf{q}_i$ and the key $\mathbf{k}_j$ is masked out when $j > i$ (i.e., the j-th token appears later). More precisely, $\mathsf{Attn}(\mathbf{q}_i, \{\mathbf{k}_j\}, \{\mathbf{v}_j\}) =$

$$\sum_{j \in [i]} \frac{\sigma(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{j' \in [i]} \sigma(\mathbf{q}_i, \mathbf{k}_{j'})} \cdot \mathbf{v}_j^\top = \frac{\phi(\mathbf{q}_i)^\top \cdot \sum_{j \in [i]} \phi(\mathbf{k}_j) \cdot \mathbf{v}_j^\top}{\phi(\mathbf{q}_i)^\top \cdot \sum_{j' \in [i]} \phi(\mathbf{k}_{j'})}.$$

During the training, to compute the output of the attention mechanism in time linear in context length, one has to compute the prefix sums $\sum_{j \in [i]} \phi(\mathbf{k}_j) \cdot \mathbf{v}_j^\top$ for all i and then multiply the i-th prefix sum with the corresponding vector $\phi(\mathbf{q}_i)^\top$. This RNN-style sequential state updates make the training process fail in fully utilizing the parallelism strength of modern accelerators. To resolve the above issue, we propose a general block-based approach to compute $\mathsf{lt}_\triangle(\mathbf{A}\cdot\mathbf{B}^\top)\cdot\mathbf{C}$[5] for arbitrary matrices $\mathbf{A},\mathbf{B},\mathbf{C}$ without materializing $\mathbf{A}\cdot\mathbf{B}^\top$, and it only requires a small number of prefix updates. By working more carefully with our block-based approach, we observe that instead of using the approximate polynomial attention weight via approximate feature mapping, we are able to compute the exact polynomial attention weight between $\mathbf{q}_i$ and $\mathbf{k}_j$ if the i-th token and the j-th token are close in position. After applying exact polynomial attention weight for local tokens, we see improvements in model qualities (see Figure 2, Section 4 and other empirical results in Appendix).

Empirical studies. We empirically evaluate all our approaches. The models equipped with high degree polynomial attention and sketched polynomial attention achieve comparable or better quality on all our evaluation metrics in comparison with models equipped with vanilla softmax attention, and achieve significantly better quality than models with approximate softmax attention provided by Performer (Choromanski et al., 2020). For GPT-2 style small size models, the models with sketched polynomial attention achieve 2x speedup in comparison with FlashAttention (Dao et al., 2022; Dao, 2023) of the fastest configuration for 32k context length. Notice that we achieve such a speed-up without applying any advanced I/O aware optimization techniques. We believe that our running time can be further reduced by optimizing the implementation in a more careful manner.

1.2. Other Notation

$[n]$ denotes the set $\{1,2,3,\cdots,n\}$. Given a matrix $\mathbf{M}\in\mathbb{R}^{n\times m}$, we use $\mathbf{m}_i\in\mathbb{R}^m$ to denote the i-th row of $\mathbf{M}$. We also abuse the notation to use $\mathbf{M}$ to indicate the set of vectors $\{\mathbf{m}_1,\mathbf{m}_2,\cdots,\mathbf{m}_n\}$. We use $\mathbf{M}_{i,j}$ to denote the entry at the i-th row and j-th column of $\mathbf{M}$. We use $\mathbf{M}^p$ to indicate raising each entry of $\mathbf{M}$ to the power p. Let $f:\mathbb{R}^m\to\mathbb{R}^k$ be an arbitrary function over vectors, we use $f(\mathbf{M})\in\mathbb{R}^{n\times k}$ to denote the matrix obtained by replacing the i-th row of $\mathbf{M}$ with $f(\mathbf{m}_i)$. $\|\mathbf{x}\|_2$ denotes the ℓ_2 norm of $\mathbf{x}$, i.e., $\sqrt{\langle\mathbf{x},\mathbf{x}\rangle}$. $\|\mathbf{M}\|_\mathsf{F}$ denotes the Frobenius norm of $\mathbf{M}$, i.e., $\sqrt{\sum_{i,j}\mathbf{M}_{i,j}^2}$. Given vectors $\mathbf{a}=(a_1,a_2,\cdots,a_m)$ and $\mathbf{b}=(b_1,b_2,\cdots,b_m)$, $\mathbf{a}*\mathbf{b}=(a_1b_1,a_2b_2,\cdots,a_mb_m)$ denotes the entrywise product (Hadamard product), and $\mathbf{a}\otimes\mathbf{b}=(a_1b_1,a_1b_2,\cdots,a_1b_m,a_2b_1,a_2b_2,\cdots,a_2b_m,\cdots,a_mb_m)$ denotes the Kronecker product. $\mathrm{diag}(\mathbf{a})$ denotes a diagonal matrix where the i-th diagnoal entry is a_i. $\mathbf{A}*\mathbf{B}$ denotes the entrywise product between matrices $\mathbf{A}$ and $\mathbf{B}$. We define "self-tensoring" $\mathbf{a}^{\otimes p}:=\mathbf{a}\otimes\mathbf{a}^{\otimes(p-1)}\in\mathbb{R}^{m^p}$ where $\mathbf{a}^{\otimes 1}:=\mathbf{a}$. $\mathbf{A}^{\otimes p}$ indicates replacing each row $\mathbf{a}_i$ of $\mathbf{A}$ with $\mathbf{a}_i^{\otimes p}$. $\mathsf{lt}_\triangle(\mathbf{M})$ denotes the matrix obtained by only keeping the lower-triangular entries of $\mathbf{M}$ and zeroing the remaining. $\mathbf{1}_m\in\mathbb{R}^m$ denotes an all-one vector.

2. Polynomial Attention and Approximation

We discuss the polynomial attention in more detail in Section 2.1 and introduce the sketching techniques (Section 2.2, 2.3) for efficiently approximating the polynomial attention. We ignore causal masking in this section, and present how to efficiently handle causal masking in Section 3.

2.1. Softmax versus Normalized Polynomial

Let us revisit the softmax attention. Given sets of query, key vectors $\mathbf{Q}=\{\mathbf{q}_i\}_{i\in[n]},\mathbf{K}=\{\mathbf{k}_i\}_{i\in[n]}\subset\mathbb{R}^h$, and scaling parameter $\beta>0$, bias parameter $\alpha\in\mathbb{R}$, the normalized softmax attention weight between $\mathbf{q}_i$ and $\mathbf{k}_j$ is:

$$\mathbf{A}_{i,j}=\frac{\exp\left(\langle\mathbf{q}_i,\mathbf{k}_j\rangle/\beta-\alpha\right)}{\sum_{j'\in[n]}\exp\left(\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle/\beta-\alpha\right)}.$$

Note $\mathbf{A}_{i,j}$ is invariant in α. In practice, α is usually chosen to be $\max_{j'\in[n]}\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle/\beta$ to make the computation of both numerator and denominator numerically stable. β is a smoothness factor. When $\beta\to\infty$, then $\mathbf{A}_{i,j}\to 1/n$, i.e., the i-th row of $\mathbf{A}$ indicates a uniform distribution over all $j\in[n]$. When $\beta\to 0$, then $\mathbf{A}_{i,j}\to\begin{cases}0 & \langle\mathbf{q}_i,\mathbf{k}_j\rangle\neq\max_{j'\in[n]}\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle\\ 1\,/\,a & \langle\mathbf{q}_i,\mathbf{k}_j\rangle=\max_{j'\in[n]}\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle\end{cases}$ where a is the number of j satisfying $\langle\mathbf{q}_i,\mathbf{k}_j\rangle=\max_{j'\in[n]}\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle$, i.e., the i-th row of $\mathbf{A}$ indicates a uniform distribution only over j that provides the maximum inner product. The standard β is chosen to be $\sqrt{h}$ (Vaswani et al., 2017).

Interestingly, normalized polynomial function has a similar behavior of the interpolation nature between the uniform distribution and the argmax distribution discussed above. In particular, let p be an even integer and consider the following normalized weight computed between $\mathbf{q}_i$ and $\mathbf{k}_j$:

$$\frac{\left(\left(\langle\mathbf{q}_i,\mathbf{k}_j\rangle+\alpha\right)/\beta\right)^p}{\sum_{j'\in[n]}\left(\left(\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle+\alpha\right)/\beta\right)^p}. \tag{1}$$

It is clear the weight is invariant for different β. Choosing a proper β can make both numerator and denominator fall in a reasonable range and make the computation numerically stable. When $\alpha\to\infty$, the weight is close to $1/n$, i.e., these weights provide a uniform distribution. When $\alpha\geq-\min_{j'\in[n]}\langle\mathbf{q}_i,\mathbf{k}_{j'}\rangle$ and $p\to\infty$, the weight is close to 0 if $\langle\mathbf{q}_i,\mathbf{k}_j\rangle$ does not provide the maximum inner product, and the weight is close to $1/a$ otherwise, where a is the number of $\mathbf{k}_j$ that provides the maximum inner product.

[5] $\mathsf{lt}_\triangle(\mathbf{M})$ denotes the matrix obtained by only keeping the lower triangular entries of $\mathbf{M}$ and zeroing the rest of the entries.

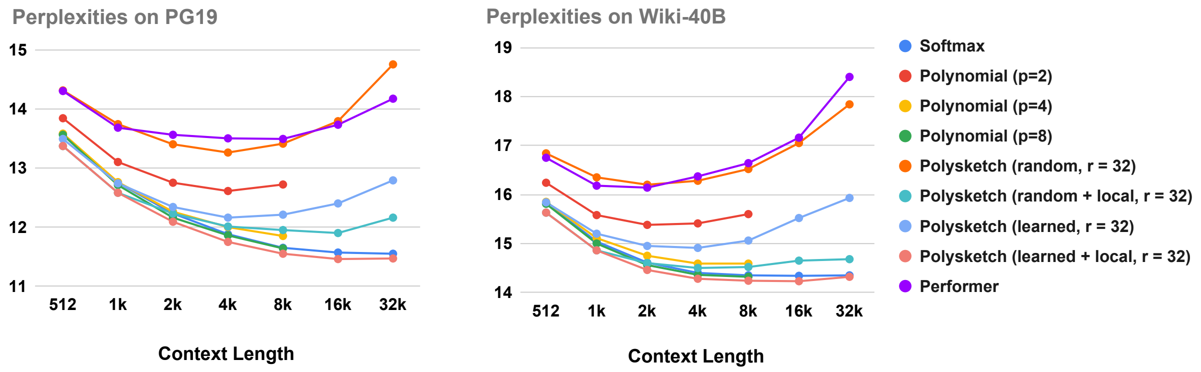

Figure 2. **Pre-training metric (perplexities). Lower is better.** GPT-2 small style models with various attention mechanisms are trained on PG-19 and Wiki-40B datasets at different context lengths up to 32k. Each batch contains 1M tokens in total. Polynomial attention with $p \geq 4$ has comparable model quality as softmax attention but OOM'ed when context length >8k. Polysketch attention (equipped with learned sketches (Section 2.3) + local exact polynomial attention (Section 3.2)) consistently outperforms all other mechanisms. The parameter r denotes the sketch size (see formal definition in Section 2.2). See Tables 2 and 3 in the Appendix for a full list of perplexity values.

Observe that if $\langle \mathbf{q}_i, \mathbf{1}_h \rangle = \langle \mathbf{k}_j, \mathbf{1}_h \rangle = 0$ for all $i, j \in [n]$, i.e., entries of $\mathbf{q}_i, \mathbf{k}_j$ always have mean 0, then we have $\forall i, j \in [n]$, $(\langle \mathbf{q}_i, \mathbf{k}_j \rangle + \alpha)/\beta = \langle \mathbf{q}'_i, \mathbf{k}'_j \rangle$, where $\mathbf{q}'_i = \mathbf{q}_i/\sqrt{\beta} + \sqrt{\alpha/(\beta h)} \cdot \mathbf{1}_h$, and $\mathbf{k}'_j = \mathbf{k}_j/\sqrt{\beta} + \sqrt{\alpha/(\beta h)} \cdot \mathbf{1}_h$, i.e., $\mathbf{q}'_i$ and $\mathbf{k}'_j$ are obtained by applying the same rescaling and bias to $\mathbf{q}_i$ and $\mathbf{k}_j$ respectively. Motivated by the above observation, we slightly tweak Equation 1 by applying an additional layer normalization[6] (Ba et al., 2016) to $\{\mathbf{q}_i\}, \{\mathbf{k}_j\}$, this gives the normalized degree-p polynomial attention weight matrix $\mathbf{A}^{(p)}$ considered in this paper:

$$\mathbf{A}^{(p)}_{i,j} = \frac{\langle \mathbf{q}'_i, \mathbf{k}'_j \rangle^p}{1 + \sum_{j' \in [n]} \langle \mathbf{q}'_i, \mathbf{k}'_{j'} \rangle^p}$$

where $\mathbf{q}'_i, \mathbf{k}'_j$ are obtained by applying the layer normalization layer to $\mathbf{q}_i, \mathbf{k}_j$ respectively. Unlike softmax attention matrix, it is possible that the term $\sum_{j' \in [n]} \langle \mathbf{q}'_i, \mathbf{k}'_{j'} \rangle^p$ is (close to) 0. We add 1 to the denominator to avoid dividing by zero. Given value vectors $\mathbf{V} = \{\mathbf{v}_i\}_{i \in [n]} \subset \mathbb{R}^h$, the full degree-$p$ polynomial attention $\mathsf{Attn}^{(p)}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}^{(p)} \cdot \mathbf{V} = \mathbf{D}^{-1} \cdot (\mathbf{Q}'\mathbf{K}'^\top)^p \cdot \mathbf{V}$, where $\mathbf{D} = \mathrm{diag}(\mathbf{1}_n + (\mathbf{Q}'\mathbf{K}'^\top)^p \mathbf{1}_n)$. In the rest of the paper, we abuse notation between $\mathbf{Q}, \mathbf{K}$ and $\mathbf{Q}', \mathbf{K}'$, and only consider $\mathbf{Q}, \mathbf{K}$ after layer normalization.

As presented in Figure 2 and other experiments in Section 4 and Appendix, the models with the degree-p polynomial attention described above achieve comparable model quality as vanilla softmax attention on all metrics as long as $p \geq 4$. To test the long range learning capabilities and in-context learning capabilities of attention mechanisms, we study the synthetic tasks of Selective Copying (Gu & Dao, 2023) and Induction heads (Olsson et al., 2022). The models with polynomial attention for $p \geq 4$ perform as well as models with softmax attention (see Appendix F for more details).

2.2. Random Sketches for Polynomial Attention with Theoretical Gaurantees

To compute $\mathsf{Attn}^{(p)}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, we only need to compute $(\mathbf{Q}\mathbf{K}^\top)^p \cdot \mathbf{V}$ and $(\mathbf{Q}\mathbf{K}^\top)^p \cdot \mathbf{1}_n$. Let us only focus on computing $(\mathbf{Q}\mathbf{K}^\top)^p \cdot \mathbf{V}$ since we can handle $(\mathbf{Q}\mathbf{K}^\top)^p \cdot \mathbf{1}_n$ in the same way. Due to a well-known fact $\forall \mathbf{x}, \mathbf{y}, \langle \mathbf{x}, \mathbf{y} \rangle^p = \langle \mathbf{x}^{\otimes p}, \mathbf{y}^{\otimes p} \rangle$, we have $(\mathbf{Q}\mathbf{K}^\top)^p \mathbf{V} = \mathbf{Q}^{\otimes p} (\mathbf{K}^{\otimes p})^\top \cdot \mathbf{V}$. If we reorder the computation and compute $(\mathbf{K}^{\otimes p})^\top \cdot \mathbf{V}$ first, we are able to compute $\mathbf{Q}^{\otimes p} \cdot (\mathbf{K}^{\otimes p})^\top \cdot \mathbf{V}$ in $O(nh^{p+1})$ time which is linear in the context length n. However h^{p+1} dependence is still expensive as we explained in Section 1.1. Thus, we resort to approximating $\mathbf{Q}^{\otimes p} (\mathbf{K}^{\otimes p})^\top$ using sketching techniques, which we formally describe ahead. We first state the definition of a sketch that has the "Approximate Matrix Multiplication (AMM)" guarantee.

Definition 2.1 (Approximate Matrix Multiplication (Woodruff et al., 2014))**.** Given parameters n, h and p, a randomized sketching matrix $\mathbf{S} \in \mathbb{R}^{h^p \times r}$ has the (ε, p)-AMM property if given any two $n \times h$ matrices $\mathbf{A}$ and $\mathbf{B}$, with probability $\geq 9/10$ over the randomized sketching matrix $\mathbf{S}$, we have that $\|(\mathbf{A}^{\otimes p}\mathbf{S})(\mathbf{B}^{\otimes p}\mathbf{S})^\top - \mathbf{A}^{\otimes p}(\mathbf{B}^{\otimes p})^\top\|_{\mathsf{F}} \leq \varepsilon \|\mathbf{A}^{\otimes p}\|_{\mathsf{F}} \|\mathbf{B}^{\otimes p}\|_{\mathsf{F}}$.

The parameter r above is referred to as the *sketch size*. Two important properties of a sketching distribution are (i) the sketch size r as a function of the accuracy parameter ε and (ii) the time required to compute $\mathbf{A}^{\otimes p}\mathbf{S}$ given an arbitrary matrix $\mathbf{A}$. Ideally, we want the matrix $\mathbf{S}$ to have a structure such that $\mathbf{A}^{\otimes p}\mathbf{S}$ can be computed without realizing the large matrix $\mathbf{A}^{\otimes p}$. Ahle et al. (2020) gave constructions of differ-

[6] Layer normalization shifts the entries of the input vector to make them have mean 0 and learns a suitable bias during training.

Algorithm 1 Polynomial Sketches

```
function PolySketchWithNegativity($\mathbf{A} \in \mathbb{R}^{k\times m}, r, p$)
    // Implementation of Theorem 2.2 (Ahle et al., 2020).
    // The output computes $\mathbf{A}^{\otimes p}\mathbf{S}$.
    If $p = 1$, return $\mathbf{A}$
    $\mathbf{M}_1$ = PolySketchWithNegativity($\mathbf{A}, r, p/2$)
    $\mathbf{M}_2$ = PolySketchWithNegativity($\mathbf{A}, r, p/2$)
    Sample Gaussian matrices $\mathbf{G}_1, \mathbf{G}_2$, each of $r$ columns
    Return $\sqrt{1/r} \cdot [(\mathbf{M}_1\mathbf{G}_1) * (\mathbf{M}_2\mathbf{G}_2)] \in \mathbb{R}^{k\times r}$
end function
function PolySketchNonNegative($\mathbf{A} \in \mathbb{R}^{k\times m}, r, p$)
    // Our approach based on Theorem 2.4.
    // The output computes $\phi'(\mathbf{A}) = (\mathbf{A}^{\otimes(p/2)}\mathbf{S})^{\otimes 2}$ where $\phi'(\cdot)$
    is the same mapping as mentioned in Theorem 1.1.
    $\mathbf{M}$ = PolySketchWithNegativity($\mathbf{A}, r, p/2$)
    Return $\mathbf{M}^{\otimes 2} \in \mathbb{R}^{k\times r^2}$.
end function
```

ent sketches that have both the properties that the sketch size r is small and the matrix $\mathbf{A}^{\otimes p}\mathbf{S}$ can be computed quickly. We describe the main properties of one of their sketches below and explain how to compute $\mathbf{A}^{\otimes p}\mathbf{S}$.

Theorem 2.2 ((Ahle et al., 2020))**.** *Given p and ε, there is a sketching matrix $\mathbf{S}$ with $r = \Theta(p/\varepsilon^2)$ columns such that $\mathbf{S}$ satisfies the (ε, p)-AMM property (Definition 2.1). Given an arbitrary vector $\mathbf{a} \in \mathbb{R}^h$, computing $(\mathbf{a}^{\otimes p})^\top\mathbf{S}$ only requires p matrix-vector multiplications of matrix size $h \times r$, $(p-2)$ matrix-vector multiplications of matrix size $r\times r$, and $(p-1)$ Hadamard products of r-dimensional vectors.*

To compute $\mathbf{A}^{\otimes p}\mathbf{S}$, we only need to compute $(\mathbf{a}_i^{\otimes p})^\top\mathbf{S}$ for each row $\mathbf{a}_i$ of $\mathbf{A}$. The number of matrix-vector multiplications and Hadamard products scales linearly in n. Let us focus on the construction of the sketch described in Theorem 2.2. We now explain how the sketch computation works for $p = 2$ and how it is extended to general values of p that are powers of 2. Let $\mathbf{G}_1 \in \mathbb{R}^{h\times r}$ and $\mathbf{G}_2 \in \mathbb{R}^{h\times r}$ denote two independently sampled random Gaussian matrices, i.e., each entry is drawn indepenently from a standard Gaussian distribution. Then the outcome of applying the sketch on $\mathbf{A}^{\otimes 2}$ is $\mathbf{A}^{\otimes 2}\mathbf{S} = \sqrt{1/r} \cdot [(\mathbf{A}\mathbf{G}_1) * (\mathbf{A}\mathbf{G}_2)]$. The construction extends to all p that are powers of 2 in a recursive way. PolySketchWithNegativity($\mathbf{A}, r, p$) (Algorithm 1) shows how to compute $\mathbf{A}^{\otimes p}\mathbf{S}$ in general.

The polynomial sketch described above can be used to approximate the matrix $(\mathbf{Q}\mathbf{K}^\top)^p = \mathbf{Q}^{\otimes p}(\mathbf{K}^{\otimes p})^\top$ with $(\mathbf{Q}^{\otimes p}\mathbf{S})(\mathbf{K}^{\otimes p}\mathbf{S})^\top$. However one issue is that they do not preserve nonnegativity: while for even p, the entries of the matrix $(\mathbf{Q}\mathbf{K}^\top)^p$ are nonnegative, the entries of the matrix $(\mathbf{Q}^{\otimes p}\mathbf{S})(\mathbf{K}^{\otimes p}\mathbf{S})^\top$ can be negative. This is not desired as discussed in Section 1.1. In the following, we propose a novel but simple approach to address this negativity issue.

Consider two arbitrary vectors $\mathbf{a}, \mathbf{b}$, we can see that the dot product $\langle \mathbf{a}^{\otimes 2}, \mathbf{b}^{\otimes 2}\rangle = \langle \mathbf{a}, \mathbf{b}\rangle^2 \geq 0$. Thus, given matrices $\mathbf{Q}^{\otimes(p/2)}\mathbf{S}$ and $\mathbf{K}^{\otimes(p/2)}\mathbf{S}$, consider the matrix $(\mathbf{Q}^{\otimes(p/2)}\mathbf{S})^{\otimes 2}((\mathbf{K}^{\otimes(p/2)}\mathbf{S})^{\otimes 2})^\top$. Since all the entries of the matrix are of the form $\langle \mathbf{a}^{\otimes 2}, \mathbf{b}^{\otimes 2}\rangle$ for some vectors $\mathbf{a}, \mathbf{b}$, all the entries of the matrix $(\mathbf{Q}^{\otimes(p/2)}\mathbf{S})^{\otimes 2}((\mathbf{K}^{\otimes(p/2)}\mathbf{S})^{\otimes 2})^\top$ are nonnegative as well. The "self-tensoring" trick ensures that all the entries in the approximate attention matrix are nonnegative at the cost of *squaring* the sketch size r.

Although $(\mathbf{Q}^{\otimes(p/2)}\mathbf{S})^{\otimes 2}((\mathbf{K}^{\otimes(p/2)}\mathbf{S})^{\otimes 2})^\top$ guarantees nonnegative property, it is not clear whether it is still a good approximation to $(\mathbf{Q}\mathbf{K}^\top)^p$ given that $\mathbf{S}$ is a polynomial sketch for degree $p/2$. One of our technical contributions is to provide a non-trivial analysis to show that it still provides a good approximation when the sketching matrix $\mathbf{S}$ is constructed as in (Ahle et al., 2020). The key is Theorem 2.4 which shows that a degree $p/2$ polynomial sketch followed by "self-tensoring" gives a degree p polynomial sketch.

To state Theorem 2.4 properly, we need to briefly introduce following concepts. The (ε, δ, t)-JL moment property is defined as follows. Given a scalar random variable $\mathbf{X}$ and $t \geq 1$, $\|\mathbf{X}\|_{L^t}$ is defined to be $\mathbf{E}[|\mathbf{X}|^t]^{1/t}$. $\|\cdot\|_{L^t}$ defines a norm over random variables defined over the same sample space and in particular satisfies $\|\mathbf{X} + \mathbf{Y}\|_{L^t} \leq \|\mathbf{X}\|_{L^t} + \|\mathbf{Y}\|_{L^t}$.

Definition 2.3 (JL-moment property (Woodruff et al., 2014))**.** Given $\varepsilon, \delta \geq 0, t \geq 1$, a random matrix $\mathbf{S}^{m\times r}$ has the (ε, δ, t)-JL moment property if for any $\mathbf{x} \in \mathbb{R}^m$ with $\|\mathbf{x}\|_2 = 1$, $\left\| \|\mathbf{x}^\top\mathbf{S}\|_2^2 - 1 \right\|_{L^t} \leq \varepsilon \cdot \delta^{1/t}$.

Theorem 2.4. *Let $\mathbf{S} \in \mathbb{R}^{h^{p/2}\times r}$ be a random sketch satisfying the (ε, δ, t)-JL moment and $(\varepsilon, \delta, 2t)$-JL moment properties for some even integer t. Given matrices $\mathbf{C}, \mathbf{D}$ with $h^{p/2}$ columns, $\|(\mathbf{C}\mathbf{S})^{\otimes 2}((\mathbf{D}\mathbf{S})^{\otimes 2})^\top - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\top\|_\mathsf{F} \leq \sqrt{5}\varepsilon\|\mathbf{C}^{\otimes 2}\|_\mathsf{F}\|\mathbf{D}^{\otimes 2}\|_\mathsf{F}$ holds with probability $\geq 1 - \delta$,*

Due to the page limit, we defer the proof to Appendix C.

Proof of Theorem 1.1. Results from Section 4 of (Ahle et al., 2020) implies that the polynomial sketch $\mathbf{S}$ as mentioned in Theorem 2.2 for degree $p/2$ with sketch size $r = \Theta(p/\varepsilon^2)$ satisfies the requirements of Theorem 2.4. By plugging $\mathbf{Q}^{\otimes(p/2)}$, $\mathbf{K}^{\otimes(p/2)}$ into $\mathbf{C}, \mathbf{D}$ of Theorem 2.4 respectively and scaling ε properly, we obtain $\|(\mathbf{Q}^{\otimes(p/2)}\mathbf{S})^{\otimes 2}((\mathbf{K}^{\otimes(p/2)}\mathbf{S})^{\otimes 2})^\top - (\mathbf{Q}\mathbf{K}^\top)^p\|_\mathsf{F} \leq \varepsilon\|\mathbf{Q}^{\otimes p}\|_\mathsf{F}\|\mathbf{K}^{\otimes p}\|_\mathsf{F}$ which concludes Theorem 1.1, i.e., the approximate feature mapping $\phi'(\mathbf{x}) = ((\mathbf{x}^{\otimes(p/2)})^\top\mathbf{S})^{\otimes 2} \in \mathbb{R}^{r^2}$ and $\phi'(\mathbf{Q}), \phi'(\mathbf{K})$ can be efficiently computed using PolySketchNonNegative($\cdot, r, p$) (see Algorithm 1).

Using $\phi'(\cdot)$, we get the following approximate polynomial attention $\widetilde{\mathsf{Attn}}^{(p)}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \tilde{\mathbf{D}}^{-1}\phi'(\mathbf{Q})\phi'(\mathbf{K})^\top\mathbf{V}$, where $\tilde{\mathbf{D}} = \mathrm{diag}(\mathbf{1}_n + \phi'(\mathbf{Q})\phi'(\mathbf{K})^\top\mathbf{1}_n)$. We call this attention mechanism Polysketch attention.

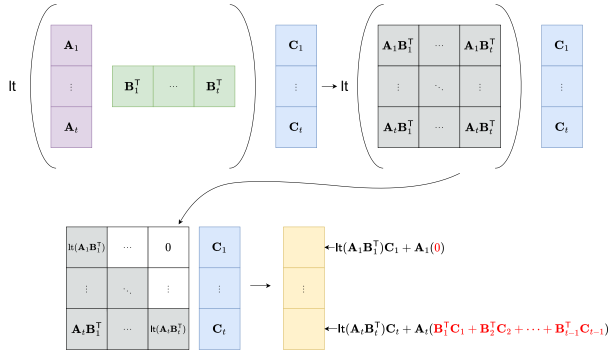
lt
$\mathbf{A}_1$
$\vdots$
$\mathbf{A}_t$
$\mathbf{B}_1^\top$
$\cdots$
$\mathbf{B}_t^\top$
$\mathbf{C}_1$
$\vdots$
$\mathbf{C}_t$
$\rightarrow$ lt
$\mathbf{A}_1\mathbf{B}_1^\top$
$\mathbf{A}_1\mathbf{B}_t^\top$
$\mathbf{A}_t\mathbf{B}_1^\top$
$\mathbf{A}_t\mathbf{B}_t^\top$
$\mathrm{lt}(\mathbf{A}_1\mathbf{B}_1^\top)$
0
$\mathrm{lt}(\mathbf{A}_t\mathbf{B}_t^\top)$
$\leftarrow \mathrm{lt}(\mathbf{A}_1\mathbf{B}_1^\top)\mathbf{C}_1 + \mathbf{A}_1(0)$
$\leftarrow \mathrm{lt}(\mathbf{A}_t\mathbf{B}_t^\top)\mathbf{C}_t + \mathbf{A}_t(\mathbf{B}_1^\top\mathbf{C}_1 + \mathbf{B}_2^\top\mathbf{C}_2 + \cdots + \mathbf{B}_{t-1}^\top\mathbf{C}_{t-1})$

Figure 3. **Block wise Lower Triangular Multiplication.** $\mathbf{A}_l$, $\mathbf{B}_l$, $\mathbf{C}_l$ are blocks of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. Each block has $b = n/t$ rows.

2.3. Learnable Sketches for Polynomial Attention

There are only $(p-2)$ random projections where each is introduced by a matrix multiplication with a small Gaussian matrix ($\mathbf{G}_1, \mathbf{G}_2$ in each recursion call in Algorithm 1) of size either $h \times r$ or $r \times r$ during the recursive computation of $\phi'(\mathbf{X}) = \textsc{PolySketchNonNegative}(\mathbf{X}, r, p)$ (Algorithm 1) for $\mathbf{X} \in \mathbb{R}^{n\times h}$. Inspired by the literature of learned sketches (Hsu et al., 2019; Aamand et al., 2019), a natural idea is to replace each random matrix $\mathbf{G}_1$, $\mathbf{G}_2$ in Algorithm 1 with learnable parameters. In practice, we found that replacing each of these random projections with a learnable *non-linear* transformation introduced by a dense neural network with size comparable to $\mathbf{G}_1, \mathbf{G}_2$ achieves a better model quality. We describe more details of our network structure for the learnable non-linear transformation in Appendix D. We also evaluate models with Polysketch attention with learned sketches on induction heads and selective copying synthetic tasks. We find that the models perform as well as models with softmax attention (See Appendix F).

3. Dealing with Causal Masks

When considering causal masks, the Polysketch attention with respect to $\mathbf{q}_i$ is $\sum_{j\le i} \frac{\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j)\rangle}{1+\sum_{j'\le i}\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_{j'})\rangle} \cdot \mathbf{v}_j^\top$. In this causal case, the full Polysketch attention can be written as $\widetilde{\mathsf{Attn}}^{(p)}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \tilde{\mathbf{D}}^{-1} \cdot \mathsf{lt}_{\triangle}(\phi'(\mathbf{Q})\phi'(\mathbf{K})^\top) \cdot \mathbf{V}$ where $\tilde{\mathbf{D}} = \mathrm{diag}(\mathbf{1}_n + \mathsf{lt}_{\triangle}(\phi'(\mathbf{Q})\phi'(\mathbf{K})^\top) \cdot \mathbf{1}_n)$. Therefore, it is crucial to efficiently compute $\mathsf{lt}_{\triangle}(\phi'(\mathbf{Q})\phi'(\mathbf{K})^\top) \cdot \mathbf{X}$ for $\mathbf{X} \in \{\mathbf{1}_n, \mathbf{V}\}$. In the next subsection, we present a block based algorithm to compute $\mathsf{lt}_{\triangle}(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C}$ for arbitrary matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times m}, \mathbf{C} \in \mathbb{R}^{n\times k}$ in time linear in n, while the number of sequentially dependent steps is small.

3.1. Fast Lower Triangular Multiplication

Let b be the block size and $t = n/b$ be the number of blocks where each block B_l ($l \in [t]$) contains indices $\{(l-1)b + 1, (l-1)b + 2, \cdots, l \cdot b\}$. Let $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ denote the i-th row vector of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ respectively. For each $l \in [t]$, let the rows of $\mathbf{A}_l \in \mathbb{R}^{b\times m}$ consist of $\mathbf{a}_i$ where $i \in B_l$. We define submatrices $\mathbf{B}_l, \mathbf{C}_l$ of $\mathbf{B}, \mathbf{C}$ respectively in a similar way. For $l \in [t]$, let us compute $\mathbf{H}_l = \sum_{i\in B_l} \mathbf{b}_i \mathbf{c}_i^\top$. Let $\mathbf{Z}_l$ indicate the prefix sum: $\mathbf{Z}_l = \sum_{j<l} \mathbf{H}_j$. In addition, let us compute $\mathbf{P}_l = \mathsf{lt}_{\triangle}(\mathbf{A}_l\mathbf{B}_l^\top)\mathbf{C}_l$ for each $l \in [t]$ in the direct way. For any $l \in [t]$, and any $i \in B_l$, if i is the i'-th index within the block B_l, it is easy to verify that the i-th row of $\mathsf{lt}_{\triangle}(\mathbf{A}\mathbf{B}^\top)\mathbf{C}$ can be obtained by $\mathbf{p} + \mathbf{a}_i^\top \mathbf{Z}_l$ where $\mathbf{p}$ is the i'-th row of $\mathbf{P}_l$. Figure 3 justifies the correctness of the above algorithm.

Since the prefix sum $\mathbf{Z}_t$ is over t matrices, the number of sequentially dependent steps is t. We can further reduce the number of sequential steps by using a parallel prefix sum algorithm (Blelloch, 1990) to exploit the parallelism. In our implementation, we only use the sequential prefix sum algorithm. Computing all $\mathbf{P}_l$ requires $O(t \cdot b^2(m+k))$ time. Computing all $\mathbf{H}_l, \mathbf{Z}_l$ requires $O(t \cdot bmk)$ time. Computing all $\mathbf{A}_l\mathbf{Z}_l + \mathbf{P}_l$ requires $O(t \cdot bmk)$ time. Therefore, the overall running time is $O(nb(m+k))$. When we set b as a constant, the running time is linear in n. If we directly plug in $\phi'(\mathbf{Q}), \phi'(\mathbf{K}), \mathbf{V}$ (or $\mathbf{1}_n$) into $\mathbf{A}, \mathbf{B}, \mathbf{C}$ above respectively, we compute causal Polysketch attention in time linear in the context length, n.

Let us take another look using the above process to compute Polysketch attention, the matrix $\mathbf{P}_l$ actually corresponds to $\mathsf{lt}_{\triangle}(\phi'(\mathbf{Q})_l\phi'(\mathbf{K})_l^\top)\mathbf{X}_l$ where $\mathbf{X}_l \in \{\mathbf{V}_l, \mathbf{1}_b\}$. $\phi'(\mathbf{Q})_l$, $\phi'(\mathbf{K})_l$ corresponds to approximate feature mapped query and key vectors within the block B_l, and $\mathbf{V}_l$ corresponds to the value vectors in B_l. Let $\mathbf{Q}_l, \mathbf{K}_l$ be the corresponding original query and key vectors in B_l. One observation is that $\phi'(\mathbf{Q})_l\phi'(\mathbf{K})_l^\top = \mathbf{L}^{\otimes 2}(\mathbf{R}^{\otimes 2})^\top = (\mathbf{L}\mathbf{R}^\top)^2$ where $\mathbf{L} = \textsc{PolySketchWithNegativity}(\mathbf{Q}_l, r, p/2) \in \mathbb{R}^{b\times r}$ and $\mathbf{R} = \textsc{PolySketchWithNegativity}(\mathbf{K}_l, r, p/2) \in \mathbb{R}^{b\times r}$ (recall Algorithm 1). Therefore $\mathsf{lt}_{\triangle}(\phi'(\mathbf{Q})_l\phi'(\mathbf{K})_l^\top)$ only takes $O(b^2 r)$ time instead of $O(b^2 r^2)$ time. The total time to compute Polysketch attention is $O(nb(r+h) + nr^2 h)$.

3.2. Applying Exact Attention Locally

We further observe that $\phi'(\mathbf{Q})_l\phi'(\mathbf{K})_l^\top$ is used to approximate $(\mathbf{Q}_l\mathbf{K}_l^\top)^p$. We can actually compute $\mathbf{P}_l$ as $\mathsf{lt}_{\triangle}\left((\mathbf{Q}_l\mathbf{K}_l^\top)^p\right)\mathbf{X}_l$. This means that when token i and j are within the same local block, we can use their exact polynomial attention weight instead of using the approximation. The time to compute $\mathsf{lt}_{\triangle}\left((\mathbf{Q}_l\mathbf{K}_l^\top)^p\right)\mathbf{X}_l$ is at most $O(b^2 h)$. In this case, the total time to compute our Polysketch attention is at most $O(nh(b+r^2))$. When $b \le r^2$, the running time is $O(nhr^2)$. As observed by our empirical studies (see Figure 2, Section 4 and other experiments in the appendix), using exact polynomial attention weights inside each local block further improves the model quality.

4. Experiments

To evaluate the effectiveness of the polynomial attention and Polysketch attention mechanisms, we train language models of various sizes with different attention mechanisms and look at both pre-training metrics and the performances on downstream tasks. Our implementations of all models are written in JAX. In our experiments, we use a Pallas implementation (JAX authors, 2023) of FlashAttention and a JAX implementation of Performer open-sourced by the authors (Choromanski et al., 2020). All the experiments are conducted on 32 Google Cloud TPUs (v5p).

Synthetic tasks. Selective Copying and Induction Heads are two well-known downstream synthetic tasks for measuring content aware reasoning capabilities and the memorization abilities of the models (see Gu & Dao (2023) for more discussions). We conduct both experiments and see both polynomial and Polysketch have similar performance as softmax attention. We include more details in Appendix F.

Models. For real world datasets, we train decoder-only models (only contain causal masked attention layers) of three different scales, mirroring the GPT-2 family (Radford et al., 2019): Small, Medium and Large. For small scale models, we train models using context lengths from 512 to 32k. For medium scale models, we only train using context length 8k. For large scale models, we only train using context length 2k. The reason that we did not train longer context length for medium and large scale models is that non-kernel based attention mechanisms (softmax, polynomial) are too slow or go out of memory (OOM). The detailed descriptions of model sizes can be found in Appendix H. We take the recipe of Transformer++ (see (Hua et al., 2022; Yang et al., 2023; Gu & Dao, 2023) as well). We refer readers to Appendix I for a detailed description of the Transformer++ used by us. If not specified otherwise, we use 10k warmup steps, 125k total training steps and a linear learning rate schedule. Depending on the original model scale, we also train kernel based attention models (Polysketch and Performer) with 0-3 additional layers, since these models are significantly faster than non-kernel based attention models so we can afford to train larger models compared to vanilla softmax. It only slightly increases model sizes.

Attention Mechanisms. We train models with the following 4 categories of attention mechanisms: (i) Softmax, (ii) Polynomial ($p = 2, 4, 8$), (iii) Polysketch (approximating polynomial attention of $p = 4$) with variants enabling learned sketches (Section 2.3) or local exact polynomial attention (Section 3.2) or both, and (iv) Performer equipped with our lower triangular multiplication approach (Section 3.1) for handling causal masks. For both Performer and Polysketch, all attention heads share the same ϕ' within the same attention layer.

Hyper-parameters. For FlashAttention, we try both block size 256 and 512[7]. For our fast lower triangular multiplication approach, we use $b = 1024$ for both Polysketch and Performer. We test both sketch sizes $r = 32$ and $r = 64$ for our Polysketch attention. We use 2048 features for Performer[8].

Pre-training metrics measurements (perplexities) over different context lengths. We train GPT-2 style small scale models equipped with different attention mechanisms on the Wiki-40B (Guo et al., 2020) and PG-19 (Rae et al., 2019) datasets with context length from 512 to 32k where each training batch contains 1M tokens. For all kernel based attentions (Performer and Polysketch), we use 13 layers instead of 12. More training details are mentioned in Appendix E. The perplexity results are shown in Figure 2 and training latencies are shown in Figure 4. Due to the space limit, we put all exact numbers in Appendix E including a detailed discussion. We observe that in the setting of 32k context length, Polysketch (learned + local, r=32) achieves **2x** speed-up in comparison with FlashAttention of the fastest setup. As shown in Table 2 and Table 3 in Appendix E, when we increase the sketch size r from 32 to 64, we further reduce the perplexities. In addition, as shown in Table 4, Polysketch (learned + local, r=64) still keeps $\sim 10\%$ speed-up in comparison with FlashAttention of the fastest setup. In addition, we observed that every kernel-based attention approach (Performer and Polysketch) with fast lower triangular matrix multiplication method almost keeps the same speed across different context lengths given that we use the same number of training tokens per step. See more discussions in Appendix E.

Downstream tasks of language models. We train our models at different scales on the C4 dataset where each training batch contains 0.5M tokens. The training details can be found in Appendix G. In Table 1, we report the perplexity on the validation split of C4 dataset and 0-shot and 5-shot accuracies on a random sample of 500 examples of HellaSwag (Zellers et al., 2019), 500 examples of PIQA (Bisk et al., 2020) and on the full Physics question answering dataset (Wang & Wang). In addition to training models using 125k steps, we also train models with 30k steps to observe how the performance of attention mechanisms evolve with increasing number of total tokens trained on. The results for 30k steps can be found in Appendix G. As observed from Table 1, Polysketch attention has a comparable performance and sometimes outperforms softmax attention. In addition, the model quality improved with increasing model sizes. We leave more discussions in Appendix G.

[7] We find a speed-up increasing the default 128 block size to 256 and 512 under our experimental setting. When increasing the block size to 1024, FlashAttention ran out of memory under our empirical setup.

[8] When using 4096 features, Performer ran out of memory in our experiments.

	C4	HellaSwag		PIQA		Physics	
	Perplexity ↓	0-shot ↑	5-shot ↑	0-shot ↑	5-shot ↑	0-shot ↑	5-shot ↑
GPT-2 Small style, 100M-scale, 12 layers default, Context Length 8192, 125k training steps							
Softmax	17.81	30.2	27.8	64.6	63.2	27.5	27.5
Polynomial (degree 4)	18.18	28.6	28.4	64.2	**65.0**	27.5	31.0
Polynomial (degree 8)	17.77	29.8	29.8	62.2	64.0	23.1	26.2
Polysketch (learned, r = 64)	18.79	29.6	28.6	60.0	60.0	24.8	30.5
Polysketch (learned, 13 layers, r = 64)	18.47	28.4	29.4	62.0	62.6	27.5	31.8
Polysketch (learned + local, r = 64)	17.98	29.8	**30.6**	62.4	63.6	**30.1**	32.3
Polysketch (learned + local, 13 layers, r = 64)	**17.68**	29.0	29.0	62.6	64.2	20.5	27.0
Polysketch (learned, r = 32)	19.09	28.0	28.4	60.6	62.0	28.3	27.5
Polysketch (learned, 13 layers, r = 32)	19.50	28.4	29.0	61.6	64.6	27.9	33.1
Polysketch (learned + local, r = 32)	18.04	29.0	29.2	63.4	62.8	26.6	**35.8**
Polysketch (learned + local, 13 layers, r = 32)	17.72	**31.2**	30.4	**64.8**	64.6	27.9	31.8
GPT-2 Medium style, 300M-scale, 24 layers default, Context Length 8192, 125k training steps							
Softmax	**13.98**	35.8	**36.6**	67.0	67.2	30.5	25.7
Polynomial (degree 4)	14.29	35.8	36.0	65.8	67.6	27.5	28.8
Polynomial (degree 8)	14.14	37.0	**36.6**	65.4	65.6	**33.1**	27.5
Polysketch (learned, r = 64)	14.64	34.6	33.4	63.2	65.4	31.0	26.2
Polysketch (learned, 26 layers, r = 64)	14.49	34.8	34.4	65.2	66.6	28.4	24.9
Polysketch (learned + local, r = 64)	14.16	35.0	35.0	65.8	**68.6**	29.6	**34.5**
Polysketch (learned + local, 26 layers, r = 64)	**13.98**	35.8	35.4	66.4	**68.6**	27.0	33.6
Polysketch (learned, r = 32)	14.94	32.2	33.8	65.6	67.6	32.7	33.6
Polysketch (learned, 26 layers, r = 32)	14.73	32.8	35.2	65.0	65.2	28.3	31.8
Polysketch (learned + local, r = 32)	14.15	36.0	35.8	65.2	67.6	27.5	27.9
Polysketch (learned + local, 26 layers, r = 32)	14.00	**37.2**	35.4	**68.0**	67.6	23.1	29.6
GPT-2 Large style, 700M-scale, 36 layers default, Context Length 2048, 125k training steps							
Softmax	12.71	40.2	40.2	68.8	**71.4**	34.4	24.4
Polynomial (degree 4)	12.82	40.0	**40.6**	67.8	66.6	31.8	31.4
Polynomial (degree 8)	12.85	40.0	39.8	66.8	70.4	34.4	29.6
Polysketch (learned, 39 layers, r = 64)	12.83	**41.0**	39.4	68.6	68.8	33.6	36.6
Polysketch (learned + local, 39 layers, r = 64)	**12.70**	40.6	40.0	**69.0**	69.0	**38.4**	**37.1**
Polysketch (learned, 39 layers, r = 32)	12.98	39.4	40.4	68.6	67.6	33.6	27.0
Polysketch (learned + local, 39 layers, r = 32)	12.74	39.6	**40.6**	66.8	69.4	35.3	31.8

Table 1. We compare the accuracies(%, higher the better) of different models on three different Q/A tasks. HellaSwag and Physics tasks have 4 choices and PIQA task has 2 choices. We also report the perplexities (lower the better) on the validation split of C4 dataset. **Bolding** indicates the best model in the task, underlining indicates beating softmax attention.

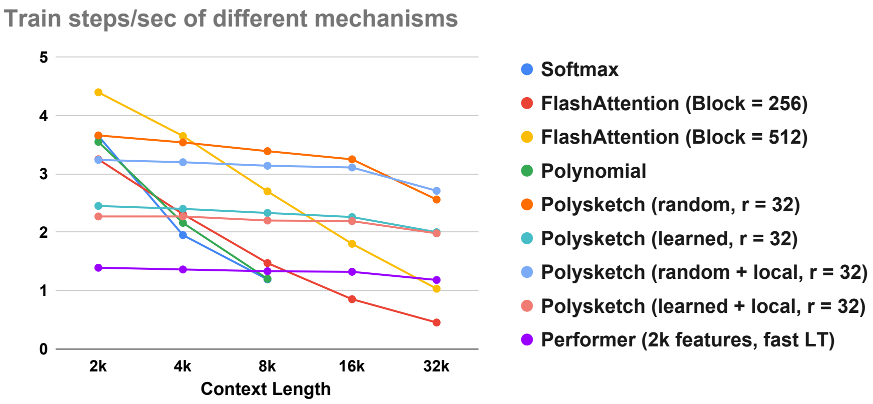

Figure 4. Training speed of models on PG-19 and Wiki-40B for different context lengths. Softmax and polynomial attentions OOM'ed when context length >8k.

Acknowledgements

We would like to thank Zeyuan Allen-Zhu, Krzysztof Choromanski, Insu Han, Yanping Huang, Weizhe Hua, Rajesh Jayaram, Zhipeng Jia, Amin Karbasi, Tamas Sarlos, and David P. Woodruff for helpful discussions and comments for improving the detailed implementations and presentations. We would like to also thank many other contributors in the JAX/Flax community for suggestions on additional implementation details.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here

References

Aamand, A., Indyk, P., and Vakilian, A. (learned) frequency estimation algorithms under zipfian distribution. *arXiv preprint arXiv:1908.05198*, 2019.

Ahle, T. D., Kapralov, M., Knudsen, J. B., Pagh, R., Velingker, A., Woodruff, D. P., and Zandieh, A. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 141–160. SIAM, 2020.

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

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Avron, H., Nguyen, H., and Woodruff, D. Subspace embeddings for the polynomial kernel. *Advances in neural information processing systems*, 27, 2014.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Babiloni, F., Marras, I., Deng, J., Kokkinos, F., Maggioni, M., Chrysos, G., Torr, P., and Zafeiriou, S. Linear complexity self-attention with 3rd order polynomials. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. PIQA: reasoning about physical commonsense in natural language. In *AAAI*, pp. 7432–7439. AAAI Press, 2020.

Blelloch, G. E. Prefix sums and their applications. 1990.

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., and Hoffman, J. Hydra attention: Efficient attention with many heads. In *European Conference on Computer Vision*, pp. 35–49. Springer, 2022.

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.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020. JAX implementation of Performer is available at `https://github.com/google-research/google-research/blob/master/performer/fast_attention/jax/fast_attention.py`.

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. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)*, 2019.

Ding, J., Ma, S., Dong, L., Zhang, X., Huang, S., Wang, W., Zheng, N., and Wei, F. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Guo, M., Dai, Z., Vrandečić, D., and Al-Rfou, R. Wiki-40b: Multilingual language model dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 2440–2452, 2020.

Han, I., Avron, H., and Shin, J. Polynomial tensor sketch for element-wise function of low-rank matrix. In *International Conference on Machine Learning*, pp. 3984–3993. PMLR, 2020.

Han, I., Jayaram, R., Karbasi, A., Mirrokni, V., Woodruff, D. P., and Zandieh, A. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

Hsu, C.-Y., Indyk, P., Katabi, D., and Vakilian, A. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

Hua, W., Dai, Z., Liu, H., and Le, Q. Transformer quality in linear time. In *International Conference on Machine Learning*, pp. 9099–9117. PMLR, 2022.

Impagliazzo, R., Paturi, R., and Zane, F. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

JAX authors. Implementation of FlashAttention in Pallas. https://github.com/google/jax/blob/main/jax/experimental/pallas/ops/attention.py, 2023.

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. Fine-tuning pretrained transformers into rnns. *arXiv preprint arXiv:2103.13076*, 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.

Meister, M., Sarlos, T., and Woodruff, D. Tight dimensionality reduction for sketching low degree polynomial kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

OpenAI. Gpt-4 technical report, 2023.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Song, Z., Woodruff, D., Yu, Z., and Zhang, L. Fast sketching of polynomial kernels of polynomial degree. In *International Conference on Machine Learning*, pp. 9812–9823. PMLR, 2021.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Sun, Z., Yang, Y., and Yoo, S. Sparse attention with learning to hash. In *International Conference on Learning Representations*, 2021.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2022.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Wang, G. and Wang, Z. Physics Multiple Choice. URL https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/physics.

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

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

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, M., Bhatia, K., Kumbong, H., and Ré, C. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. *arXiv preprint arXiv:2402.04347*, 2024.

A. Conclusion and Future Work

In this work, we empirically studied the performance of using high degree polynomial attention instead of softmax attention in training decoder-only models for language modeling tasks. Our empirical study shows that the polynomial attention can achieve a similar model quality as the vanilla softmax attention when degree $p \geq 4$. Then we developed an efficient approximate polynomial attention via polynomial sketching techniques which can be computed in linear time of context length with provable approximation guarantees. In addition, we presented a fast block based lower triangular matrix multiplication algorithm which can significantly boost the training time of any kernel based attention in the decoder based models.

There are several potential directions for future works. (1) Although we only empirically studied the performance of decoder-only models with polynomial attention for language modeling tasks, it is interesting to explore the potentials of encoder models with polynomial attention, and to understand whether it can be used in other fields such as vision. (2) In this work, empirically we mainly focus on reducing the training latency. The benefits of linear transformers also transfer to inference as the KV cache sizes are independent of the context length. The exact inference improvements using linear transformers have to be explored more thoroughly. (3) Polysketch attention is a kernel based method which can compute dense attention in linear time. It is interesting to see whether it can be combined with sparsification based efficient attention techniques such as HyperAttention proposed by (Han et al., 2023) recently.

B. Discussion of the error bound of Theorem 1.1

Let us look closely to the error bound stated in Theorem 1.1. Our error only has polynomial dependence in the ℓ_2 norm bounds of $\{\mathbf{q}_i\}$ and $\{\mathbf{k}_j\}$. In other word, to keep the same error, our sketching dimension r only has polynomial dependence in the ℓ_2 norm bounds of $\{\mathbf{q}_i\}$ and $\{\mathbf{k}_j\}$. In contrast, to approximate the exponential kernel, the sketching dimension of Performer (Choromanski et al., 2020) grows exponentially in the ℓ_2 norm bounds of $\{\mathbf{q}_i\}$ and $\{\mathbf{k}_j\}$.

Suppose the ℓ_2 norm of query and key vectors is bounded, i.e., $\max_i \max(\|\mathbf{q}_i\|_2, \|\mathbf{k}_i\|_2) \leq C$. In the softmax attention, the ratio between two attention weights can be at most $\exp(\langle \mathbf{q}_i, \mathbf{k}_j\rangle)/\exp(\langle \mathbf{q}_i, \mathbf{k}_{j'}\rangle) \leq \exp(C^2)/\exp(-C^2) = \exp(2C^2)$ which is bounded. In contrast, $\langle \mathbf{q}_i, \mathbf{k}_j\rangle^p/\langle \mathbf{q}_i, \mathbf{k}_{j'}\rangle^p$ can be arbitrarily large since $\langle \mathbf{q}_i, \mathbf{k}_{j'}\rangle$ can be close to 0. Therefore, in this bounded norm situation, polynomial attention is more capable for the operation of "taking the max".

Another difference between the approximation provided by Performer and ours is that Performer provides entry-wise approximation guarantee while we provide an approximation guarantee in average. Consider an example that all query and key vectors have ℓ_2 norm at most 1. By Markov inequality, we know 90% of pairs $(i,j) \in [n] \times [n]$ satisfies $\left|\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j)\rangle - \langle \mathbf{q}_i, \mathbf{k}_j\rangle^2\right| \leq \varepsilon'$, where $\varepsilon' = 10\varepsilon$. As long as both $\langle \mathbf{q}_i, \mathbf{k}_j\rangle^p, \langle \mathbf{q}_i, \mathbf{k}_{j'}\rangle^p \in (\varepsilon'/\varepsilon'', 1]$ for some arbitrary $\varepsilon'' > \varepsilon'$, $\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_j)\rangle/\langle \phi'(\mathbf{q}_i), \phi'(\mathbf{k}_{j'})\rangle$ is a $(1 \pm O(\varepsilon''))$-approximation to $\langle \mathbf{q}_i, \mathbf{k}_j\rangle^p/\langle \mathbf{q}_i, \mathbf{k}_{j'}\rangle^p$.

C. Proof of Theorem 2.4

We first note the following fact: If $\mathbf{S}$ has (ε, δ, t)-JL moment property, then for any two arbitrary vectors $\mathbf{x}$ and $\mathbf{y}$, we have that $\|\langle \mathbf{S}^\mathsf{T}\mathbf{x}, \mathbf{S}^\mathsf{T}\mathbf{y}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle\|_{L^t} \leq \varepsilon\delta^{1/t}\|\mathbf{x}\|_2\|\mathbf{y}\|_2$. For a proof see Lemma 9 from (Ahle et al., 2020).

Proof of Theorem 2.4. Let $\mathbf{c}_i$ denote the i-th row of $\mathbf{C}$ and $\mathbf{d}_j$ denote the j-th row of $\mathbf{D}$. Then the (i,j)-th entry of the matrix $\mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}$ is equal to $\langle \mathbf{c}_i, \mathbf{d}_j\rangle^2$. Similarly, the (i,j)-th coordinate of the matrix $(\mathbf{C}\mathbf{S})^{\otimes 2}((\mathbf{D}\mathbf{S})^{\otimes 2})^\mathsf{T}$ is equal to $\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2$ and therefore

$$\|(\mathbf{C}\mathbf{S})^{\otimes 2}((\mathbf{D}\mathbf{S})^{\otimes 2})^\mathsf{T} - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}\|_\mathsf{F}^2 = \sum_{i,j}(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2.$$

Recall that given an integer $t \geq 1$, for a random variable $\mathbf{X}$, we define $\|\mathbf{X}\|_{L^t}$ as $\mathbf{E}[|\mathbf{X}|^t]^{1/t}$. Also note that $\|\mathbf{X}\|_{L^t}$ is a norm over the random variables and in-particular satisfies the triangle inequality. Now,

$$\begin{aligned}
\|\|(\mathbf{C}\mathbf{S})^{\otimes 2}((\mathbf{D}\mathbf{S})^{\otimes 2})^\mathsf{T} - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}\|_\mathsf{F}\|_{L^t} &= \|\|(\mathbf{C}\mathbf{S})^{\otimes 2}((\mathbf{D}\mathbf{S})^{\otimes 2})^\mathsf{T} - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}\|_\mathsf{F}^2\|_{L^{t/2}}^{1/2} \\
&= \|\sum_{i,j}(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}}^{1/2} \\
&\leq (\sum_{i,j}\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}})^{1/2}
\end{aligned}$$

where we used the triangle inequality of $\|\cdot\|_{L^t}$ in the last inequality. Now consider a single term $\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}}$. First, we have

$$\begin{aligned}
&(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\\
&= (\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle + \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2\\
&= (\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle + 2\langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2\\
&\le (1+C)(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^4 + 4(1+1/C)\langle \mathbf{c}_i, \mathbf{d}_j\rangle^2(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2
\end{aligned}$$

with probability 1 for any $C \ge 1$. Since both LHS and RHS are *non-negative* random variables, we obtain that

$$\begin{aligned}
&\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}}\\
&\le (1+C)\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^4\|_{L^{t/2}} + 4(1+1/C)\langle \mathbf{c}_i, \mathbf{d}_j\rangle^2\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2\|_{L^{t/2}}.
\end{aligned}$$

Now,

$$\begin{aligned}
\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^4\|_{L^{t/2}} &= \|\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle\|_{L^{2t}}^4\\
&\le \varepsilon^4\delta^{2/t}\|\mathbf{c}_i\|_2^4\|\mathbf{d}_j\|_2^4
\end{aligned}$$

assuming that S has $(\varepsilon, \delta, 2t)$-JL moment property. We also have

$$\begin{aligned}
\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle)^2\|_{L^{t/2}} &= \|\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle - \langle \mathbf{c}_i, \mathbf{d}_j\rangle\|_{L^{t}}^2\\
&\le \varepsilon^2\delta^{2/t}\|\mathbf{c}_i\|_2^2\|\mathbf{d}_j\|_2^2
\end{aligned}$$

assuming that $\mathbf{S}$ has (ε, δ, t)-JL moment property. Overall, we get

$$\begin{aligned}
&\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i, \mathbf{S}^\mathsf{T}\mathbf{d}_j\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}}\\
&\le (1+C)\varepsilon^4\delta^{2/t}\|\mathbf{c}_i\|_2^4\|\mathbf{d}_j\|_2^4 + 4(1+1/C)\langle \mathbf{c}_i, \mathbf{d}_j\rangle^2\varepsilon^2\delta^{2/t}\|\mathbf{c}_i\|_2^2\|\mathbf{d}_j\|_2^2.
\end{aligned}$$

Picking $C = 1/\varepsilon$ and assuming $\varepsilon \le 1/5$, we get that

$$\|(\langle \mathbf{S}^\mathsf{T}\mathbf{c}_i\rangle^2 - \langle \mathbf{c}_i, \mathbf{d}_j\rangle^2)^2\|_{L^{t/2}} \le 5\varepsilon^2\delta^{2/t}\|\mathbf{c}_i\|_2^4\|\mathbf{d}_j\|_2^4.$$

Thus, we have

$$\begin{aligned}
\|\|(\mathbf{CS})^{\otimes 2}((\mathbf{DS})^{\otimes 2})^\mathsf{T} - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}\|_\mathsf{F}\|_{L^t} &\le \sqrt{5}\varepsilon\delta^{1/t}\sqrt{\sum_{i,j}\|\mathbf{c}_i\|_2^4\|\mathbf{d}_j\|_2^4}\\
&\le \sqrt{5}\varepsilon\delta^{1/t}\|\mathbf{C}^{\otimes 2}\|_\mathsf{F}\|\mathbf{D}^{\otimes 2}\|_\mathsf{F}.
\end{aligned}$$

By using Markov's inequality, we obtain that with probability $\ge 1 - \delta$,

$$\|(\mathbf{CS})^{\otimes 2}((\mathbf{DS})^{\otimes 2})^\mathsf{T} - \mathbf{C}^{\otimes 2}(\mathbf{D}^{\otimes 2})^\mathsf{T}\|_\mathsf{F} \le \sqrt{5}\varepsilon\|\mathbf{C}^{\otimes 2}\|_\mathsf{F}\|\mathbf{D}^{\otimes 2}\|_\mathsf{F}. \qquad \square$$

D. Replacing Random Projections with Learnable Transformations

Our learnable polynomial sketch algorithm is stated in Algorithm 2. It has a similar structure as our randomized polynomial sketch stated in Algorithm 1. The only differences are (1) we replace random projections $\mathbf{M}_1\mathbf{G}_1$ and $\mathbf{M}_2\mathbf{G}_2$ with $f_1(\mathbf{M}_1)$ and $f_2(\mathbf{M}_2)$ respectively. (2) We apply a $\tanh(\cdot)$ trick to each entry of $\sqrt{1/r}\cdot[f_1(\mathbf{M}_1) * f_2(\mathbf{M}_2)]$ to make the output within a reasonable range and thus make the optimization process stable and converge.

Each $f_1(\cdot), f_2(\cdot)$ has the same dense network structure but different learnable parameters. The network has output dimension r and 3 hidden layers with size $[8r, r, 8r]$. We apply an activation function $\mathrm{gelu}(\cdot)$ after the first and the third hidden layer. We apply an layer normalization before the input and the second hidden layer. Therefore, each network only has roughly $8hr + 24r^2$ or $32r^2$ number of parameters. The entire learnable polynomial sketch only contains $p - 2$ learnable networks.

Since all attention heads share the same learnable polynomial sketch within an attention layer, the number of increased learnable parameters is negeligible in comparison with the entire model.

Note that we did not take much time to optimize the network structure. It is likely that better network structures exist. We leave the question of finding a better network structure as a future work.

Algorithm 2 Learnable Polynomial Sketches

```
function LearnablePolysketchWithNegativity($\mathbf{A} \in \mathbb{R}^{k\times m}, r, p$)
    // Analog of PolysketchWithNegativity.
    If $p = 1$, return $\mathbf{A}$
    $\mathbf{M}_1$ = LearnablePolysketchWithNegativity($\mathbf{A}, r, p/2$)
    $\mathbf{M}_2$ = LearnablePolysketchWithNegativity($\mathbf{A}, r, p/2$)
    Return $\sqrt{r} \cdot \tanh\left(\sqrt{1/r} \cdot [f_1(\mathbf{M}_1) * f_2(\mathbf{M}_2)]\right) \in \mathbb{R}^{k\times r}$
end function
function LearnablePolysketchNonNegative($\mathbf{A} \in \mathbb{R}^{k\times m}, r, p$)
    // Analog of PolysketchNonNegative.
    $\mathbf{M}$ = LearnablePolysketchWithNegativity($\mathbf{A}, r, p/2$)
    Return $\mathbf{M}^{\otimes 2} \in \mathbb{R}^{k\times r^2}$.
end function
```

	512	1k	2k	4k	8k	16k	32k
Non-kernel based methods, 12 layers							
Softmax (using FlashAttention)	13.57	12.75	12.23	11.88	11.65	11.57	11.55
Polynomial (deg=2)	13.84	13.10	12.75	12.61	12.72	OOM	OOM
Polynomial (deg=4)	13.58	12.76	12.26	12.00	11.85	OOM	OOM
Polynomial (deg=8)	13.56	12.71	12.16	11.86	11.64	OOM	OOM
Kernel based methods, 13 layers							
Polysketch (random, r = 32)	14.31	13.74	13.40	13.26	13.41	13.79	14.75
Polysketch (random, r = 64)	14.00	13.35	13.03	12.84	12.92	13.18	13.66
Polysketch (learned, r = 32)	13.49	12.74	12.34	12.16	12.21	12.40	12.79
Polysketch (learned, r = 64)	**13.33**	12.60	12.10	11.90	11.86	11.94	12.19
Polysketch (random + local, r = 32)	13.37	**12.58**	12.23	12.01	11.95	11.90	12.16
Polysketch (random + local, r = 64)	13.37	**12.58**	12.24	11.98	11.93	11.96	11.91
Polysketch (learned + local, r = 32)	13.37	**12.58**	12.09	11.75	11.55	11.46	11.47
Polysketch (learned + local, r = 64)	13.37	**12.58**	**12.03**	**11.69**	**11.44**	**11.38**	**11.34**
Performer (2048 features)	14.30	13.68	13.56	13.50	13.49	13.73	14.17

Table 2. Perplexities on the test split of PG19 when the models are trained on PG19 dataset

E. Perplexity Results on PG-19 and Wiki-40B

We train GPT-2 small scale models on PG-19 and Wiki-40B datasets at various context lengths. We use the same training recipe that we described in Section 4 and train the models for 125k steps with a batch size of 1M tokens. For each of PG-19 and Wiki-40B datasets, we obtain a SentencePiece vocabulary of size 32k and train the models using the respective tokenizer.

We measure test perplexities of each of the models in Tables 2 and 3. We can see that the Polysketch attention model (learned + local) equipped with one additional layer beats the softmax attention models at all context lengths. We also note that Polysketch attention models, even without local attention, also achieve perplexities close to that of softmax models at all context lengths. These experiments show that our attention mechanism can scale to large context lengths without significant model quality loss.

The training latencies are shown in Table 4. As we observed that all kernel based approaches equipped with our fast lower triangular multiplication approach are significantly faster than non-kernel based methods. Notably, Polysketch (learned + local, r=64) achieves 1.1x speed up in comparison with the FlashAttention (block size 512) on 32k context length, and Polysketch (learned + local, r=32) achieves **2x** speed up in comparison with the FlashAttention (block size 512) on 32k context length. Both Polysketch (learned + local, r=32) and Polysketch (learned + local, r=64) have lower perplexity than the softmax attention. Polysketch (learned + local, r=64) has the lowest test perplexity.

	512	1k	2k	4k	8k	16k	32k
Non-kernel based methods, 12 layers							
Softmax (using FlashAttention)	15.82	15.04	14.61	14.40	14.35	14.34	14.35
Polynomial (p=2)	16.24	15.58	15.38	15.41	15.60	OOM	OOM
Polynomial (p=4)	15.85	15.11	14.75	14.59	14.59	OOM	OOM
Polynomial (p=8)	15.81	15.00	14.56	14.36	14.32	OOM	OOM
Kernel based methods, 13 layers							
Polysketch (random, r = 32)	16.84	16.35	16.20	16.28	16.52	17.05	17.84
Polysketch (random, r = 64)	16.32	15.73	15.72	15.88	16.01	16.44	17.45
Polysketch (learned, r = 32)	15.84	15.20	14.95	14.91	15.06	15.52	15.93
Polysketch (learned, r = 64)	15.65	14.96	14.62	14.58	14.70	14.95	15.35
Polysketch (random + local, r = 32)	**15.63**	**14.86**	14.60	14.50	14.52	14.65	14.68
Polysketch (random + local, r = 64)	**15.63**	**14.86**	14.58	14.52	14.43	14.62	14.54
Polysketch (learned + local, r = 32)	**15.63**	**14.86**	14.46	14.28	14.24	**14.23**	14.32
Polysketch (learned + local, r = 64)	**15.63**	**14.86**	**14.43**	**14.26**	**14.18**	14.24	**14.29**
Performer (2048 features)	16.75	16.18	16.14	16.37	16.64	17.16	18.40

Table 3. Perplexities on the test split of Wiki-40B when the models are trained on Wiki-40B dataset

E.1. Training Latency Comparison

The main advantage of linear transformers is that their training latency remains the same across different context lengths given that we use the same "batch size" (tokens per training step) for all the context lengths. To show that it is the case, we report the training latencies (in terms of steps/sec) of our models and other attention mechanisms in Table 4. Using the same batch size across different context lengths, we note that the steps/sec of linear transformers such as Polysketch and Performer remain almost constant whereas the steps/sec of quadratic-time transformers decreases with increasing context lengths. The results show that, depending on the model structure, models using our Polysketch attention mechanism are **significantly faster to train** than models using a quadratic attention mechanism such as softmax (implemented via FlashAttention) at **long context lengths**.

F. Experiments with Synthetic Tasks

For both synthetic experiments, We train a small 2-layer transformer. Each attention layer contains 8 attention heads where each has head size 16. For Polysketch attention, we choose r=32 and the block size $b = 1024$ for fast lower triangular multiplication (Section 3.1).

F.1. Selective Copying

Recently, Gu & Dao (2023) have used selective copying task as a yard stick for measuring content aware reasoning capabilities and the memorization abilities of the models. In this task, the model is required to memorize colored blocks that appear in the context and the model needs to output the colored blocks in the same order at the end. See (Gu & Dao, 2023) for a more detailed description of this task.

We generate 64k random examples used for training. Each batch has 64 examples. We train for 400k steps in total without otherwise specified. Using a similar training recipe as in their paper, we train small two layer models using different attention mechanisms to solve these tasks at context lengths 4k, 16k, and 32k. We report our results in Table 5. We see that polynomial and Polysketch attention manage to learn to solve the selective copying task though the accuracy of Polysketch is a bit worse at 16k context length with the same training recipe as other models. We found that with a different learning rate schedule, Polysketch attention also manages to solve the selective copying task at a context length 16k with an accuracy of 99.44% thus showing that there may not be any loss in matching the reasoning capabilities of the softmax attention mechanism. This suggests that Polysketch attention may require different learning rate schedules to obtain the optimal performance as compared to softmax transformers.

We also find that at a context length of 32k, Polysketch attention learns to solve 95.29% of test examples after 800k steps of training. In all our experiments, we observe sudden spike in the accuracies of the models indicating the point where the

	512	1k	2k	4k	8k	16k	32k
Softmax	**6.00**	4.95	3.65	1.95	1.19	OOM	OOM
FlashAttention (Block size 256 x 256)	4.78	4.09	3.25	2.31	1.47	0.85	0.45
FlashAttention (Block size 512 x 512)	5.46	**5.0**	**4.4**	**3.65**	**2.7**	1.8	1.03
Polynomial (p=2, 4, 8)	5.74	4.74	3.55	2.16	1.20	OOM	OOM
Polysketch (random, 13 layers, r = 32)	5.25	4.31	3.66	3.54	3.39	3.25	2.56
Polysketch (random, 13 layers, r = 64)	5.06	4.20	2.50	2.23	2.06	1.95	1.55
Polysketch (learned, 13 layers, r = 32)	3.16	2.82	2.45	2.40	2.33	2.26	2.00
Polysketch (learned, 13 layers, r = 64)	2.17	1.97	1.37	1.35	1.33	1.29	1.13
Polysketch (random + local, 13 layers, r = 32)	5.35	4.40	3.24	3.2	3.14	3.11	2.71
Polysketch (random + local, 13 layers, r = 64)	5.35	4.40	2.09	2.00	1.91	1.82	1.60
Polysketch (learned + local, 13 layers, r = 32)	5.35	4.40	2.27	2.27	2.20	**2.19**	**1.98**
Polysketch (learned + local, 13 layers, r = 64)	5.35	4.40	1.31	1.31	1.30	1.26	1.12
Performer (2k features + Fast lower triangular multiplications)	2.21	1.58	1.39	1.36	1.33	1.32	1.18
Performer (256 features (default) without Fast lower triangular multiplications)	0.44	0.40	0.36	0.29	0.21	0.14	0.08

Table 4. **Training steps/sec** of different attention mechanisms at various context lengths (**higher is faster**). For context lengths 512 and 1k, we compute the full attention matrix in Polysketch and Performer attention **without** using the linearization technique. These models are all GPT-2-like small scale models. Each batch contains 1M tokens in total.

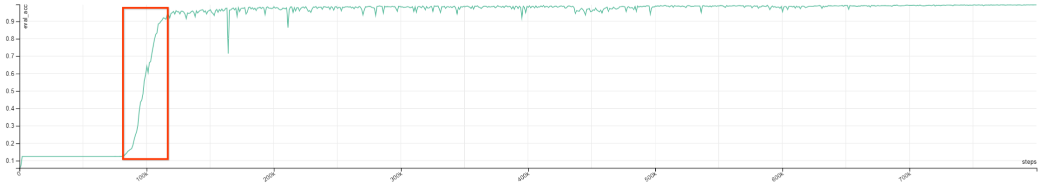

Figure 5. The test accuracy during training Polysketch (learned + local) on the selective copying task of 32k context length. The x-axis is the number of steps trained and the y-axis is the test accuracy. We observe that the model suddenly learns the pattern at some point during the training.

model learns to solve the task, see e.g., Figure 5.

	4k	16k	32k
Softmax	99.73 %	98.17%	0%
Polynomial (degree=4)	99.90%	97.97%	0%
Polynomial (degree=8)	99.90%	97.65%	0%
Polysketch (learned + local)	99.16%	92.75%	0%
Polysketch (learned + local) (different learning rate schedule)	-	99.44%	87.16%

Table 5. % of 4096 examples on which the models succeeded to perfectly output the colored blocks in the context in the same order.

F.2. Induction heads

Olsson et al. (2022) have proposed induction heads task as a way to identify and explain the in-context learning capabilities of language models. This task requires the model to output the token that appears immediately after a special token that appears in the context exactly once at an arbitrary position.

We generate 64k random examples used for training. Each batch contains 64 examples. Each position is a random token from a vocabulary of size 16. We replace a random position except the last 3 tokens with a special token. We replace the second to the last token with a special token. We replace the last token with the token appeared directly after the first special token.

The total number of training steps is 400k. We consider context lengths 128 and 256. We observe that all the models (softmax, degree 4 polynomial, degree 8 polynomial and Polysketch with sketch size 16 and 32) are able to solve (accuracy $> 99.95\%$) the task at a context length of 128 and all of them fail to solve (accuracy around 1/16, i.e., random guessing) the task at a context length 256 under the same optimization configuration.

G. Experiments on Downstream Tasks

Additional training details. We train all models from scratch on the C4 dataset using a SentencePiece tokenizer trained on C4 with a vocabulary size of 32,000. We use a batch size of 0.5M tokens per training step and a peak learning rate of 3e-4. We use a linear learning rate schedule to warmup the learning rate for the first 10% of iterations and then again use a linear learning rate schedule to decay the learning rate. We use Adam optimizer with weight decay parameters $(\beta_1 = 0.95, \beta_2 = 0.98)$ in all our experiments.

The training latencies of small scale models on 8k context length are:

1. Softmax (without FlashAttention): 2.40 step/sec.
2. Polynomial (p=4,8): 2.65 step/sec.
3. Polysketch (learned, r=64, 12 layers): 2.71 step/sec.
4. Polysketch (learned, r=32, 12 layers): 4.68 step/sec.
5. Polysketch (learned, r=64, 13 layers): 2.49 step/sec.

	C4	HellaSwag		PIQA		Physics	
	Perplexity ↓	0-shot ↑	5-shot ↑	0-shot ↑	5-shot ↑	0-shot ↑	5-shot ↑
GPT-2 Small style, 100M-scale, 12 layers default, Context Length 8192, 30k training steps							
Softmax	20.11	28.0	**28.8**	61.8	**62.8**	20.5	30.1
Polynomial (degree 4)	20.66	28.4	28.6	60.4	61.0	24.4	30.1
Polynomial (degree 8)	20.05	27.6	27.4	59.8	60.0	20.5	23.1
Polysketch (learned, r = 64)	21.16	27.8	28.0	60.4	61.2	29.6	33.6
Polysketch (learned, 13 layers, r = 64)	20.93	28.6	28.2	**62.0**	61.8	30.1	**35.8**
Polysketch (learned + local, r = 64)	20.30	28.4	27.8	61.6	**62.8**	27.9	34.9
Polysketch (learned + local, 13 layers, r = 64)	**19.91**	28.4	27.6	61.2	61.0	31.0	30.5
Polysketch (learned, r = 32)	22.31	27.0	26.6	59.8	61.8	**35.3**	34.4
Polysketch (learned, 13 layers, r = 32)	22.15	**28.8**	28.2	59.8	60.0	32.3	31.0
Polysketch (learned + local, r = 32)	20.28	28.4	28.4	59.6	61.4	29.6	31.4
Polysketch (learned + local, 13 layers, r = 32)	19.94	28.6	28.2	60.4	61.2	29.2	30.5
GPT-2 Medium style, 300M-scale, 24 layers default, Context Length 8192, 30k training steps							
Softmax	15.97	**32.0**	31.6	61.8	63.4	25.3	29.2
Polynomial (degree 4)	16.46	29.2	29.8	63.2	64.4	30.5	33.6
Polynomial (degree 8)	16.12	31.4	31.4	64.2	64.0	27.0	28.3
Polysketch (learned)	17.18	29.2	30.0	62.8	64.0	27.0	31.0
Polysketch (learned, 26 layers, r = 64)	17.06	29.8	31.0	64.8	64.6	28.3	30.1
Polysketch (learned + local, r = 64)	16.14	31.6	32.6	64.6	64.4	27.9	31.4
Polysketch (learned + local, 26 layers, r = 64)	**15.95**	31.8	31.4	63.8	**66.0**	20.0	27.9
Polysketch (learned, r = 32)	17.81	30.4	30.4	60.2	61.6	**34.4**	**34.0**
Polysketch (learned, 26 layers, r = 32)	17.47	29.2	30.2	61.6	62.0	28.3	31.4
Polysketch (learned + local, r = 32)	16.20	31.6	**33.0**	**65.8**	65.8	31.8	28.8
Polysketch (learned + local, 26 layers, r = 32)	16.02	31.8	31.8	64.4	65.0	27.5	31.4

Table 6. We compare the accuracies(%, higher the better) of different models (all trained for 30k training steps) on three different Q/A tasks. HellaSwag and Physics tasks have 4 choices and PIQA task has 2 choices. We also report the perplexities (lower the better) on the validation split of C4 dataset. **Bolding** indicates the best model in the task, underlining indicates beating softmax attention.

6. Polysketch (learned, r=32, 13 layers): 4.34 step/sec.
7. Polysketch (learned + local, r=64, 12 layers): 2.70 step/sec.
8. Polysketch (learned + local, r=32, 12 layers): 4.42 step/sec.
9. Polysketch (learned + local, r=64, 13 layers): 2.48 step/sec.
10. Polysketch (learned + local, r=32, 13 layers): 4.12 step/sec.

Note that the difference between above training latencies and those presented in Table 4 is due to the different number of tokens per batch (0.5M vs 1M).

The training latencies of medium scale models on 8k context length are reported as follows:

1. Softmax (without FlashAttention): 0.87 step/sec.
2. Polynomial (p=4,8): 0.89 step/sec.
3. Polysketch (learned, r=64, 24 layers): 0.99 step/sec.
4. Polysketch (learned, r=32, 24 layers): 1.62 step/sec.
5. Polysketch (learned, r=64, 26 layers): 0.92 step/sec.
6. Polysketch (learned, r=32, 26 layers): 1.52 step/sec.
7. Polysketch (learned + local, r=64, 24 layers): 0.98 step/sec.
8. Polysketch (learned + local, r=32, 24 layers): 1.59 step/sec.

9. Polysketch (learned + local, r=64, 26 layers): 0.91 step/sec.
10. Polysketch (learned + local, r=32, 26 layers): 1.46 step/sec.

For large scale models, since the context length is only 2k (recall that as we mentioned earlier, non-kernel based methods are either too slow or facing OOM issues for longer context length for the large-scale), the running time of kernel based methods do not take advantage from linearization. The purpose is to compare the model quality only. So we omit the training latencies of large scale models here.

Additional results. The results similar to Table 1 but for the models trained on only 30k steps is shown in Table 6.

G.1. Scaling with Model Sizes

From the results in Table 1 and Table 6, we have the following main observations: (i) Polysketch attention (learned + local) **closely matches** the performance and sometimes outperforms models trained with softmax attention. (ii) Models trained with Polysketch attention improve with increasing model sizes showing promise to be a replacement for softmax even in the largest models to **achieve lower training latencies** without significant **performance issues**. (iii) Strong performance of learned Polysketch attention, **without relying on local attention**, shows the capability of our proposed attention mechanism and that the results that we obtain with learned+local Polysketch attention are **not just due to using exact polynomial attention within the blocks**.

H. Model Sizes for Each Attention Mechanism

In this section, we include the sizes of all models that we trained.

H.1. Small scale models

The default configuration has 12 layers, 12 attention heads per layer, each attention head has head size 64. The number of parameters of each model is stated as the following.

12 layer models:

1. Softmax, Polynomial (degree = 2, 4 & 8): 110M
2. Polysketch (learned, sketch size = 64, 12 layers), Polysketch (learned + local, sketch size = 64, 12 layers): 113M
3. Polysketch (learned, sketch size = 32, 12 layers), Polysketch (learned + local, sketch size = 32, 12 layers): 111M

13 layer models:

1. Polysketch (learned, sketch size = 64, 13 layers), Polysketch (learned + local, sketch size = 64, 13 layers): 120M
2. Polysketch (learned, sketch size = 32, 13 layers), Polysketch (learned + local, sketch size = 32, 13 layers): 118M
3. Polysketch (random, sketch size = 32, 64, 13 layers), Polysketch (random + local, sketch size = 32, 64, 13 layers), Performer (2k features): 117M

H.2. Medium scale models

The default configuration has 24 layers, 16 attention heads per layer, each attention head has head size 64. The number of parameters of each model is stated as the following.

24 layer models:

1. Softmax, Polynomial (degree = 4, 8): 337M
2. Polysketch (learned, sketch size = 64, 24 layers), Polysketch (learned + local, sketch size = 64, 24 layers): 341M

3. Polysketch (learned, sketch size = 32, 24 layers), Polysketch (learned + local, sketch size = 32, 24 layers): 337M

26 layer models:

1. Polysketch (learned, sketch size = 64, 26 layers), Polysketch (learned + local, sketch size = 64, 26 layers): 367M
2. Polysketch (learned, sketch size = 32, 26 layers), Polysketch (learned + local, sketch size = 32, 26 layers): 362M

H.3. Large scale models

The default configuration has 36 layers, 20 attention heads per layer, each attention head has head size 64. The number of parameters of each model is stated as the following.

36 layer models:

1. Softmax, Polynomial (degree = 4, 8): 748M

39 layer models:

1. Polysketch (learned, sketch size = 64, 39 layers), Polysketch (learned + local, sketch size = 64, 39 layers): 817M
2. Polysketch (learned, sketch size = 32, 39 layers), Polysketch (learned + local, sketch size = 32, 39 layers): 811M

I. Reciepe of Transformer++

We add sinusoidal position embeddings (Vaswani et al., 2017) to the input embeddings and use Rotary Position Embeddings (RoPE) (Su et al., 2021) at all attention heads. We use Gated Linear Units (Dauphin et al., 2017; Shazeer, 2020) with an expansion factor of 4 as the FeedForward layer in the network. We use GELU as the non-linearity. All models are trained using Adam optimizer with weight decay and a peak learning rate of 7e-4.

J. Additional Experiments: Comparison with Sparsification based Method and Other Learnable Kernel based Method

In this section, we add additional comparisons with other attention mechanisms:

1. **Hedgehog (kernel-based method)**: A recent (concurrent work) learnable approximate attention mechanism approximating the softmax attention (Zhang et al., 2024). We implemented the learnable feature mapping and equipped it with our fast lower triangular multiplication (otherwise it is too slow during training for 32k context length, see our discussion in Section 1, and similar training latency without fast lower triangular multiplication can be found in the last line of Table 4).
2. **Sliding window attention (locality based method)**: This is a standard locality based softmax attention mechanism (see e.g., (Zaheer et al., 2020; Beltagy et al., 2020)) where we use a sliding window of length 1024 (the same as the block size b used in our fast lower triangular multiplication), i.e., each token only attends to the previous 1024 tokens (including itself).
3. **Local block based attention (locality based method)**: This is an attention mechanism that simply masks out all off-diagonal blocks of our PolySketch (learned/random + local) attention during our fast lower triangular multiplication, i.e., we include exact polynomial attention weights on the diagonal blocks and remove all approximate polynomial attention weights (obtained by approximate feature mapping) on the off-diagonal blocks. Unlike sliding window attention, the diagonal blocks of the above mechanism are always non-overlapping, and thus no information transfer occurs across blocks. This ablation study shows the capability of our approximate feature mapping to gather global information,

We train GPT-2 small style models on PG-19 with 32k context length and report the perplexities. Since all these attentions are sub-quadratic time, we use 13 layers for all of them. Additionally, we evaluate HedgeHog with various feature dimensions: (1) the original dimension described in their paper and (2) increased dimension $1024 = 32^2$ (which is equal to the feature mapping dimension of our PolySketch attention when $r = 32$).

PG-19, 32k context length	Perplexity (lower better)
PolySketch (local + learned, r=64)	11.34
PolySketch (local + learned, r=32)	11.47
PolySketch (learned, r=64)	12.19
PolySketch (learned, r=32)	12.79
Hedgehog (original)	13.58
Hedgehog (1024 features)	12.93
Sliding window	11.67
Polynomial (degree=4, local only)	12.78

Table 7. Comparison with Hedgehog and locality based attention mechanisms.

Comparison with Hedgehog: We observe that even allowing for 1024 features (comparable to r=32 PolySketch attention), the HedgeHog attention mechanism falls short of the performance of our learned attention mechanism (without using local attention) and is significantly worse than our local + learned attention mechanism. We observe that Polysketch Attention mechanism (2.00 step/sec for r=32) is additionally much faster than an implementation of HedgeHog with a comparable number (1024) of features (1.40 steps/sec).

In addition, as mentioned earlier, any mechanism (including HedgeHog) that tries to directly approximate the softmax attention mechanism is ruled out by a complexity theory lower bound (see (Alman & Song, 2023)).

Comparison with sliding window and local block based attention: We note that the sliding window attention mechanism obtains perplexities worse than polysketch (learned + local) in the above table. We also note that the local block based attention – polynomial (degree=4, local only) is also much worse than PolySketch (local + learned) showing that the global information carried by the PolySketch attention significantly improves model quality.

In addition, we also conduct selective copying synthetic tasks described in Appendix F for sliding window attention and local block based attention. None of them handles the task for 16k context length (the same experiment setup for Table 5 in Appendix F), i.e., 0% accuracy. This is an evidence that our PolySketch feature mapping can do better to process global information from a long context than these local attention approaches.

K. PolySketch vs Softmax When Using the Same Number of Layers

In the following (Table 8), we also include perplexity results of PolySketch attention with r=64 when training on PG-19 and using 12 layers. We observe (see the table below) that even when the number of layers is set to 12 (the same as softmax and polynomial) for PolySketch attention, it is competitive with softmax.

L. Additional Experiments on Downstream Tasks

Polynomial and softmax (without using FlashAttention) attentions go OOM for 8k context length when training GPT-2 large style models (using data parallelism). In the following we train GPT2-large models at 8k context lengths with softmax(FlashAttention)/polysketch(learned + local)/polysketch(learned) attention mechanism on the c4 dataset. We report a similar evaluations as done in Table 1 but now for 8k context lengths as follows (Table 9,Table 10,Table 11,Table 12):

PG-19, perplexities	context length = 16384	Context length = 32768
Softmax (FlashAttention, 12 layers)	11.57	11.55
PolySketch (local + learned, 12 layers)	11.80	11.57
PolySketch (local + learned, 13 layers)	11.38	11.34
PolySketch (learned, 12 layers)	12.43	12.43
PolySketch (learned, 13 layers)	11.94	12.19

Table 8. Comparison between PolySketch and softmax attentions using the same number of layers.

PT-2 Large, 8k context length	C4 Perplexity (lower better)
Softmax (FlashAttention)	12.47
PolySketch (learned)	12.99
PolySketch (learned + local)	12.45

Table 9. Perplexity results on C4 (8k context length).

GPT-2 Large, 8k context length	HellaSwag 0-shot (higher better)	HellaSwag 5-shot (higher better)
Softmax (FlashAttention)	0.402	0.408
PolySketch (learned)	0.398	0.386
PolySketch (learned + local)	0.404	0.396

Table 10. HellaSwag results (8k context length).

GPT-2 Large, 8k context length	PIQA 0-shot (higher better)	PIQA 5-shot (higher better)
Softmax (FlashAttention)	0.698	0.682
PolySketch (learned)	0.688	0.686
PolySketch (learned + local)	0.666	0.682

Table 11. PIQA results (8k context length).

GPT-2 Large, 8k context length	Physics 0-shot (higher better)	Physics 5-shot (higher better)
Softmax (FlashAttention)	0.366	0.323
PolySketch (learned)	0.296	0.336
PolySketch (learned + local)	0.327	0.34

Table 12. Physics results (8k context length).