QUASI LINEAR KERNEL ATTENTION WITH INFINITE CAPACITY

Anonymous authors
Paper under double-blind review

ABSTRACT

The evaluation cost of transformers with softmax attention scales quadratically with sequence length. Kernel attention addresses this by replacing softmax with a more general kernel function. In this paper, we aim to identify kernels that retain the expressivity of attention while enabling quasi linear computation. To quantify expressivity, we introduce a capacity for each kernel, measuring the maximum sequence length for which the attention matrix can approximate the identity. A higher capacity thus indicates greater expressivity. We show that expressive kernels like softmax, Gauss, and Laplace have infinite capacity. In contrast, common quasi linear kernels, such as those derived from finite dimensional feature maps, exhibit finite capacity. As a solution, we propose additive kernels constructed from univariate spline and polynomial exponential kernels. We prove that these maintain infinite capacity while allowing quasi linear computation via sorting. Finally, we implement additive sorting kernels efficiently and benchmark them against modern softmax backends, demonstrating advantages for long sequences.

1 INTRODUCTION

Transformers (Vaswani et al., 2017) are the dominant architecture in language processing (Brown et al., 2020) and computer vision (Dosovitskiy et al., 2021). Their central building block is attention, which computes for *query vectors* $\boldsymbol{q} = (q_m)_{m=1}^M \subseteq \mathbb{R}^D$, *key vectors* $\boldsymbol{k} = (k_n)_{n=1}^N \subseteq \mathbb{R}^D$, and *value vectors* $\boldsymbol{v} = (v_n)_{n=1}^N \subseteq \mathbb{R}^C$ the M *attention* vectors $(y_m)_{m=1}^M$ as

$$y_m = \sum_{n=1}^N \Phi(q_m, k_n) v_n \Big/ \sum_{n=1}^N \Phi(q_m, k_n), \qquad m = 1, \dots, M. \tag{1}$$

Most implementations use the *softmax* kernel $\Phi(q, k) = \mathrm{e}^{\tau q^\top k}$ with inverse temperature $\tau > 0$, which acts as similarity score between the vectors q and k. For $M = N$, the computational effort scales quadratically with the *sequence length* N. This has become the main computational bottleneck in practice. Current models are trained and deployed with context windows of 128k tokens and beyond (Llama Team, AI @ Meta, 2024), which involves substantial engineering effort (Dao et al., 2022; Dao, 2024). There have been several attempts to escape the quadratic complexity through approximations and windowed attention patterns; see Tay et al. (2022) for an overview.

In this paper, we focus on *exact* subquadratic attention, where the structure of the kernel Φ allows the computation of (1) in quasi linear time (Tsai et al., 2019). Within this class, Katharopoulos et al. (2020) achieved linear complexity by choosing kernels that factor over a *finite feature map* (FFM) $\varphi\colon \mathbb{R}^D \to \mathbb{R}^L$, i.e., $\Phi(q, k) = \varphi(q)^\top \varphi(k)$. This construction yields symmetric positive definite (spd) kernels, which can be evaluated in linear time. However, for small L, these approaches have been shown to admit only limited expressivity compared to softmax attention, see Schlag et al. (2021). At the same time, the cost of evaluating the finite feature map increases linearly with L.

On the other hand, there exist several univariate kernels (namely $D = 1$), for which the corresponding kernel sums can be evaluated exactly via quasi linear sorting algorithms. Examples are polynomial exponential kernels like the Laplace kernel (Hofmeyr, 2021) and piecewise linear kernels (Hertrich, 2024; Hertrich et al., 2024; Vialard & Boufadène, 2025). However, most of these algorithms do not generalize to higher dimensions. An exception is the Laplace kernel, for which the sums can be evaluated in quasi linear time for arbitrary D, see Langrené & Warin (2021), but the complexity depends exponentially on D, making the approach practically infeasible for $D > 6$.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Contribution Our goal is to identify kernels Φ such that the attention (1) is both expressive and computable in quasi linear time. To this end, we introduce the *capacity* of a kernel. Its definition is inspired by the associative recall experiment of Schlag et al. (2021) and measures up to which sequence length the identity can be approximated by the attention algorithm. Intuitively, this corresponds to the number of tokens that can be distinguished. Consequently, a higher capacity indicates higher expressivity. We prove that common choices have infinite capacity, including the softmax, Gauss, and Laplace kernels. This resembles the observation that their associated attention is expressive in practice. For kernels based on an FFM, we prove that the capacity is upper-bounded by the number L of considered features, reflecting their limited expressivity.

In order to achieve quasi linear attention with infinite capacity, we consider additive kernels of the form $\Phi(q,k) = \phi(q_1,k_1) + \ldots + \phi(q_D,k_D)$, where ϕ is a univariate kernel with quasi linear attention. For the univariate Laplace kernel $\phi(s,t) = \exp(-|s-t|)$ and the bump kernel $\phi(s,t) = \max(0, 1-|s-t|)$, we prove that the capacity of both ϕ and Φ is infinite, so that we obtain quasi linear attention with infinite capacity. We verify our findings in numerical experiments and provide an efficient CUDA implementation. For sequence lengths $N \gtrsim 2{,}048$, our implementation is faster than conventional single-precision PyTorch attention, which offers a comparable level of numerical precision and code optimization. Compared to the highly optimized half-precision FlashAttention, we reach the break-even point for $N \gtrsim 20{,}000$. Thanks to the quasi linear complexity of our algorithm, the advantage increases rapidly with the sequence length.

Related work Several works reduce the quadratic cost of attention by approximation, either through sparse and windowed attention patterns (Kitaev et al., 2020; Beltagy et al., 2020; Zaheer et al., 2020), or through low-rank approximations of the attention matrix (Wang et al., 2020; Xiong et al., 2021). For an extensive overview, see Tay et al. (2022). The FFM idea was refined by Choromanski et al. (2021); Peng et al. (2021); Qin et al. (2022) with feature maps tailored to the softmax kernel and by Schlag et al. (2021), who increased the *feature dimension* L after recognizing that it caps the attention capacity. Lower bounds on the recurrent state size required for multi-query associative recall were proven by Arora et al. (2024a). Building on this, Arora et al. (2024b) combine linear attention with sliding windows, while gated linear attention (Yang et al., 2024a) and DeltaNet (Yang et al., 2024b) use data-dependent gating and delta-rule updates of the recurrent state, respectively, to better use a fixed-size state. These models are recurrent and go beyond the normalized kernel sums considered in (1). Alternative notions of expressivity for attention mechanisms were considered in Yun et al. (2020); Furuya et al. (2025). Our proposed bump and Laplace variants have been previously considered in different contexts: Feng et al. (2026) approximate the Laplace kernel attention via a finite feature map/Nyström approach, and Vialard & Boufadène (2025) combine the univariate bump kernel with a linear or MLP layer before each attention module to project onto a single dimension. Finally, the additive kernels are related to kernel slicing, see also Appendix B.

2 Capacity of Kernel Attention

In order to determine which kernels Φ lead to an expressive attention mechanism, we assign to each Φ a capacity $\mathrm{Cap}(\Phi)$. Intuitively, this measures the maximal number of tokens that can be distinguished by the attention mechanism, which corresponds to the largest sequence length such that the identity can be approximated. Consequently, a larger capacity corresponds to more expressive attention. In this section, we formally define $\mathrm{Cap}(\Phi)$ and derive sufficient conditions such that $\mathrm{Cap}(\Phi) = \infty$. These conditions apply in particular to the Gauss, softmax, and Laplace kernels, which are empirically known to correspond to expressive attention mechanisms.

Definition of Capacity For a kernel Φ, keys $\boldsymbol{k} = (k_n)_{n=1}^N$ and queries $\boldsymbol{q} = (q_m)_{m=1}^M$, we define the *Gram matrix* $G(\boldsymbol{q},\boldsymbol{k}) = (G_{m,n})_{m,n=1}^{M,N}$ and the *attention matrix* $A(\boldsymbol{q},\boldsymbol{k}) = (A_{m,n})_{m,n=1}^{M,N}$ by

$$G_{m,n} = \Phi(q_m,k_n) \qquad \text{and} \qquad A_{m,n} = G_{m,n}/\textstyle\sum_{l=1}^N G_{m,l}. \tag{2}$$

For $\boldsymbol{v} = (v_n)_{n=1}^N$ in $\mathbb{R}^C$, the attention mechanism can be rewritten as the matrix-vector multiplication $y_c = A(\boldsymbol{q},\boldsymbol{k})v_c$ for $c = 1,\ldots,C$. We denote the *capacity* $\mathrm{Cap}(\Phi)$ of Φ as the largest sequence length N such that there exist keys $\boldsymbol{k}$ and queries $\boldsymbol{q}$ for which $A(\boldsymbol{q},\boldsymbol{k}) \approx \mathrm{Id}_N$, where Id_N is the $N \times N$ identity matrix. Formally, we define

$$\mathrm{Cap}(\Phi) = \sup\left\{ N \in \mathbb{N} : \inf_{\boldsymbol{q},\boldsymbol{k} \in (\mathbb{R}^D)^N} \|A(\boldsymbol{q},\boldsymbol{k}) - \mathrm{Id}_N\|_F^2 = 0 \right\}. \tag{3}$$

In other words, if the sequence length N is larger than $\mathrm{Cap}(\Phi)$, the attention mechanism is no longer able to represent the identity, which indicates a lack of expressiveness.

Kernels with Infinite Capacity A kernel Φ is stationary if $\Phi(q,k) = F(q-k)$ for some function F. If F is continuous and fulfills $F(x) \to 0$ as $\|x\|_2 \to \infty$, we call Φ a stationary $\mathcal{C}_0$ kernel.

Theorem 1. *Let Φ be a stationary $\mathcal{C}_0$ kernel with $F(0) > 0$, then it holds* $\mathrm{Cap}(\Phi) = \infty$.

The theorem applies in particular to the Gauss kernel $\Phi(q,k) = \exp(-\frac{1}{2}\|q-k\|_2^2)$ and many others including Laplace and Matérn kernels. Even though the softmax kernel $\Phi(q,k) = \exp(q^\top k)$ is not stationary, we can exploit its close relation to the Gauss kernel to show that $\mathrm{Cap}(\Phi) = \infty$.

Proposition 2. *For $D \geq 2$ and $\Phi(q,k) = \exp(q^\top k)$, we have* $\mathrm{Cap}(\Phi) = \infty$.

Upper Bounding the Capacity Next, we derive a criterion which upper bounds the capacity of an spd kernel Φ by the dimension of the corresponding reproducing kernel Hilbert space (RKHS). To define the RKHS, we consider the space $\mathcal{H}_\Phi^0 \coloneqq \mathrm{span}\{\Phi(\cdot,k) : k \in \mathbb{R}^D\}$ with the bilinear form

$$\left\langle \sum_{m=1}^{M} a_m \Phi(\cdot, q_m), \sum_{n=1}^{N} b_n \Phi(\cdot, k_n) \right\rangle_{\mathcal{H}_\Phi} = \sum_{m=1}^{M}\sum_{n=1}^{N} a_m b_n \Phi(q_m, k_n). \tag{4}$$

Then, the RKHS $\mathcal{H}_\Phi$ is defined as the completion of $\mathcal{H}_\Phi^0$ with respect to the norm induced by $\langle\cdot,\cdot\rangle_{\mathcal{H}_\Phi}$. The columns of the Gram matrix $G(\boldsymbol{q},\boldsymbol{k})$ can be seen as functions $\Phi(\cdot,k_n) \in \mathcal{H}_\Phi^0$ evaluated at queries $\boldsymbol{q}$. Therefore, the rank of $G(\boldsymbol{q},\boldsymbol{k})$ can be bounded via

$$\mathrm{rank}(G(\boldsymbol{q},\boldsymbol{k})) \leq \dim(\mathrm{span}\{\Phi(\cdot,k_1),\dots,\Phi(\cdot,k_N)\}) \leq \dim(\mathcal{H}_\Phi). \tag{5}$$

As the attention matrix $A(\boldsymbol{q},\boldsymbol{k})$ arises from $G(\boldsymbol{q},\boldsymbol{k})$ by rescaling the rows, we also find that $\mathrm{rank}(A(\boldsymbol{q},\boldsymbol{k})) = \mathrm{rank}(G(\boldsymbol{q},\boldsymbol{k})) \leq \dim(\mathcal{H}_\Phi)$. In particular, we have for any $\boldsymbol{q}$ and $\boldsymbol{k}$ by the Eckart-Young theorem (Eckart & Young, 1936) that

$$\inf_{\boldsymbol{q},\boldsymbol{k}\in(\mathbb{R}^D)^N} \|A(\boldsymbol{q},\boldsymbol{k}) - \mathrm{Id}_N\|_F^2 \geq N - \dim(\mathcal{H}_\Phi). \tag{6}$$

Thus, the left hand side can only be zero if $N \leq \dim(\mathcal{H}_\Phi)$ such that $\mathrm{Cap}(\Phi) \leq \dim(\mathcal{H}_\Phi)$. We summarize this result in the following theorem.

Theorem 3. *Let Φ be an spd kernel, then it holds* $\mathrm{Cap}(\Phi) \leq \dim(\mathcal{H}_\Phi)$.

The reverse statement is not true. More precisely, we show in Proposition 8 that there exist kernels Φ such that $\dim(\mathcal{H}_\Phi) = \infty$, but $\mathrm{Cap}(\Phi) < \infty$.

3 Capacity of quasi linear Attention

In Theorem 1, we have derived a large class of kernels which admits infinite capacity, indicating a high expressivity. However, the computation of the attention mechanism requires generally $\mathcal{O}(NM)$ operations, which is a major computational bottleneck. Therefore, we now focus on kernels for which the attention mechanism can be computed exactly in quasi linear time, i.e., in $\mathcal{O}((N+M)\log^\nu(N+M))$ operations for some $\nu > 0$. We will mainly analyze the complexity of computing the kernel sums

$$z_m = \sum_{n=1}^{N} \Phi(q_m, k_n) v_n, \qquad m = 1,\dots,M, \tag{7}$$

for queries $\boldsymbol{q} = (q_m)_{m=1}^M$, keys $\boldsymbol{k} = (k_n)_{n=1}^N$ and values $\boldsymbol{v} = (v_n)_{n=1}^N$. In fact, from a computational viewpoint, considering the kernel sums is equivalent to considering the attention mechanism, as the normalization can be computed in $\mathcal{O}(MC)$ operations. In practice, the time of computing this normalization is dominated by the $\mathcal{O}((N+M)(D+C))$ cost of reading the data.

Laplace Langrené & Warin (2021) propose an algorithm based on (Bentley, 1980) with quasi linear complexity to compute kernel sums for the Laplace kernel $\Phi(q,k) = \exp(-\|q-k\|_1)$, which fulfills $\mathrm{Cap}(\Phi) = \infty$ by Theorem 1. However, the algorithm relies on a decomposition of the Laplace kernel sum into 2^D generalized empirical cumulative distribution functions. In particular, the complexity depends exponentially on the dimension. Thus, for an exemplary head dimension of $D = 64$, the quasi linear implementation is only expected to be faster than brute-force for sequence lengths larger than $2^{64} > 10^{19}$, which makes it a theoretical advance rather than a practical method.

3.1 FINITE FEATURE MAPS

Following the idea of random Fourier features (Rahimi & Recht, 2007), several works (Peng et al., 2021; Choromanski et al., 2021; Katharopoulos et al., 2020; Schlag et al., 2021) proposed linear attention algorithms by using finite feature maps (FFMs) $\varphi\colon \mathbb{R}^D \to \mathbb{R}^L$. The main idea is to choose $\Phi(q,k) = \varphi(q)^\top \varphi(k)$, for which the kernel sum can be computed as

$$z_m = \sum_{n=1}^{N} \Phi(q_m, k_n) v_n = \sum_{n=1}^{N} \varphi(q_m)^\top \varphi(k_n) v_n = \Big(\sum_{n=1}^{N} v_n \varphi(k_n)^\top \Big) \varphi(q_m). \tag{8}$$

The inner sum $\sum_{n=1}^{N} v_n \varphi(k_n)^\top$ no longer depends on m such that all z_m together can be computed in linear complexity $\mathcal{O}(LC(N+M))$. Schlag et al. (2021) uses the feature dimension as a measure of capacity. This is reasonable because

$$H_\Phi^0 = \operatorname{span}\{\Phi(\cdot, k) \mid k \in \mathbb{R}^D\} = \operatorname{span}\{\varphi(\cdot)^\top \varphi(k) \mid k \in \mathbb{R}^D\} \subseteq \operatorname{span}\{\varphi_1, \dots, \varphi_L\}. \tag{9}$$

Since the completion of a finite dimensional space is the space itself, we also obtain $\mathcal{H}_\Phi \subseteq \operatorname{span}\{\varphi_1, ..., \varphi_L\}$, which has at most dimension L. Corollary 4 gives the connection to Schlag et al. (2021) as an immediate consequence of Theorem 3.

Corollary 4. *Let $\Phi(q,k) = \varphi(q)^\top \varphi(k)$ with FFM $\varphi\colon \mathbb{R}^D \to \mathbb{R}^L$, then $\operatorname{Cap}(\Phi) \leq \dim(\mathcal{H}_\Phi) \leq L$.*

By Corollary 4 the capacity is bounded through the feature dimension L. At the same time, the computational complexity scales linearly with L such that FFMs with more features are slower to evaluate. The next proposition verifies that the assumptions of Theorem 1 are indeed violated for FFMs in the sense that most FFMs are not stationary and the only stationary FFMs are not $\mathcal{C}_0$.

Proposition 5. *Let $\Phi(q,k) = \varphi(q)^\top \varphi(k)$ be a continuous and stationary kernel. Then, it holds $\Phi(q,k) = \sum_{l=1}^{L} \alpha_l \cos(\omega_l^\top (q-k))$, where $\alpha_l \geq 0$ and $\omega_l \in \mathbb{R}^D$ for $l = 1, \dots, L$. In particular, we have that Φ is a stationary $\mathcal{C}_0$ kernel if and only if $\Phi = 0$.*

3.2 QUASI LINEAR ATTENTION WITH INFINITE CAPACITY IN ONE DIMENSION

There exist several univariate kernels $\phi\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, for which kernel sums can be computed in quasi linear time via sorting. This involves the Laplace kernel $\phi(s,t) = \exp(-|s-t|)$ (Hofmeyr, 2021; Langrené & Warin, 2021), the distance kernel $\phi(s,t) = |s-t|$ (Hertrich, 2024; Hertrich et al., 2024; Teuber et al., 2011) and polynomial exponential kernels $\phi(s,t) = p(|s-t|)\exp(-|s-t|)$ for some polynomial p (Hofmeyr, 2021). Using that any continuous piecewise linear function $f\colon \mathbb{R} \to \mathbb{R}$ can be represented as the composition of constants, linear terms and absolute values (Streubel et al., 2014), the sorting algorithm for the distance kernel can be deployed for any kernel $\phi(s,t) = f(s-t)$.

Proposition 6. *Let $f\colon \mathbb{R} \to \mathbb{R}$ be piecewise linear with finitely many nodes, then $\phi(s,t) = f(s-t)$ is quasi linear.*

This includes the bump kernel $f(u) = \max(0, 1-|u|)$ with alternative expressivity results by Vialard & Boufadène (2025, Thm. 3.2). Both bump and Laplace have infinite capacity, see Theorem 1.

Corollary 7. *Both the bump kernel $\phi(s,t) = \max\{0, 1-|s-t|\}$ and the Laplace kernel $\phi(s,t) = \exp(-|s-t|)$ are quasi linear and have infinite capacity.*

On the other hand, the Riesz kernel $r(s,t) = |s| + |t| - |s-t|$ is an example for a kernel with infinite dimensional RKHS and finite capacity. Since the RKHS of r can be identified with the infinite dimensional homogeneous Beppo Levi space (Wendland, 2004, Prop. 10.39), its attention matrix has no bound on its rank. Yet, its global structure does not allow to approximate the identity.

Proposition 8. *For the Riesz kernel $\phi(s,t) = |s|+|t|-|s-t|+\varepsilon$ with $\varepsilon > 0$, we have $\operatorname{Cap}(\phi) = 2$.*

Weighted Absolute Value Sums For the Riesz and spline kernels, the difficult part is to compute $\sum_{n=1}^{N} |s_m - t_n| v_n$, where $s_m \in \mathbb{R}$ and $t_n \in \mathbb{R}$. This sum can be computed in $\mathcal{O}((N+M)(\log_2 N + C))$ by Algorithm 1. A similar algorithm can be applied for the one dimensional Laplace kernel. The core idea is that after sorting, the keys below and above s_m separate:

Algorithm 1 Weighted absolute value sum

#	Step	Work	Memory
1	**Input** $\boldsymbol{s} \in \mathbb{R}^M$, $\boldsymbol{t} \in \mathbb{R}^N$ and $\boldsymbol{v} \in \mathbb{R}^{N \times C}$	-	$(C{+}1)N{+}M$
2	**Output** $z_m \coloneqq \sum_{n=1}^N \lvert s_m - t_n \rvert v_n \in \mathbb{R}^C \quad m = 1, \dots, M$	-	MC
3	$\sigma \coloneqq \texttt{argsort}(\boldsymbol{t})$ s.t. $t_{\sigma(1)} \leq \ldots \leq t_{\sigma(N)}$	$N \log_2 N$	N
4	$p_m \coloneqq \max\big(\{n \colon t_{\sigma(n)} \leq s_m\} \cup \{0\}\big)$ for $m = 1, \dots, M$	$M \log_2 N$	M
5	$\text{pref_t}_n \coloneqq \sum_{j=1}^n t_{\sigma(j)} v_{\sigma(j)} \in \mathbb{R}^C$ for $n = 0, \dots, N$	$(N{+}1)C$	$(N{+}1)C$
6	$\text{pref_v}_n \coloneqq \sum_{j=1}^n v_{\sigma(j)} \in \mathbb{R}^C$ for $n = 0, \dots, N$	$(N{+}1)C$	$(N{+}1)C$
7	$a_m \coloneqq 2\,\text{pref_v}_{p_m} - \text{pref_v}_N \in \mathbb{R}^C$ for $m = 1, \dots, M$	MC	MC
8	$z_m \coloneqq \text{pref_t}_N - 2\,\text{pref_t}_{p_m} + s_m a_m \in \mathbb{R}^C$ for $m = 1, \dots, M$	MC	MC
9	**return** $\boldsymbol{z}$		

$$
\begin{aligned}
&\sum_{n=1}^N |s_m - t_n| v_n = \sum_{n=1}^N |s_m - t_{\sigma(n)}| v_{\sigma(n)} \\
=&\sum_{n=1}^{p_m} (s_m - t_{\sigma(n)}) v_{\sigma(n)} + \sum_{n=p_m+1}^N (t_{\sigma(n)} - s_m) v_{\sigma(n)} \\
=&s_m \sum_{n=1}^{p_m} v_{\sigma(n)} - \sum_{n=1}^{p_m} t_{\sigma(n)} v_{\sigma(n)} + \sum_{n=p_m+1}^N t_{\sigma(n)} v_{\sigma(n)} - s_m \sum_{n=p_m+1}^N v_{\sigma(n)} \\
=&s_m (2\,\text{pref_v}_{p_m} - \text{pref_v}_N) + \text{pref_t}_N - 2\,\text{pref_t}_{p_m} = z_m. \qquad (10)
\end{aligned}
$$

The last line (10) depends on the prefix sums over $\boldsymbol{v}_\sigma$ and $\boldsymbol{t}_\sigma \boldsymbol{v}_\sigma$, which can be computed with $\mathcal{O}(NC)$ work. Once the prefix sums have been computed, the entry at position p_m needs to be extracted, resulting in another $\mathcal{O}(MC)$ operations. The total work of the one dimensional absolute value sum reduces to $\mathcal{O}((N{+}M)(\log_2(N){+}C))$ operations. In short, we pay $\mathcal{O}((N{+}M)\log_2 N)$ to then be able to compute the rest in $\mathcal{O}(C(N{+}M))$ instead of $\mathcal{O}(CNM)$.

Appendix C explains how we implement Algorithm 1 efficiently on the GPU. For the multivariate case, a naive PyTorch implementation needs to build huge temporary tensors in steps 5 to 8. Meanwhile, we compute steps 5 to 8 in a single CUDA kernel and allocate the intermediate tensors pref_v and pref_t on the fly. This ensures a low memory profile with good parallelism.

3.3 Additive Kernels for Quasi Linear Attention with Infinite Capacity

In order to carry over the one dimensional algorithms from the previous subsection to higher dimensions, we propose to use additive kernels of the form

$$
\Phi \colon \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}, \qquad \Phi(q, k) = \sum_{d=1}^D \phi(q_d, k_d) \qquad (11)
$$

for some univariate kernel $\phi \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Similar kernels were considered in Hertrich et al. (2025) as quasi-Monte Carlo design for the kernel slicing algorithm; more details are given in Appendix B. By definition, the kernel sums for such kernels can be computed as

$$
z_m = \sum_{n=1}^N \Phi(q_m, k_n) v_n = \sum_{d=1}^D \sum_{n=1}^N \phi(q_{m,d}, k_{n,d}) v_n, \qquad (12)
$$

which has quasi linear complexity if and only if the kernel sum with respect to the univariate kernel ϕ has quasi linear complexity. Even though additive kernels Φ will never be $\mathcal{C}_0$ (unless $\Phi = 0$), we prove in the next theorem that $\operatorname{Cap}(\phi) = \infty$ implies that $\operatorname{Cap}(\Phi) = \infty$.

Theorem 9. *Let Φ be an additive kernel corresponding to some ϕ. Then, we have that* $\operatorname{Cap}(\Phi) \geq \operatorname{Cap}(\phi)$. *Moreover, the properties spd, quasi linear and stationary carry over from ϕ to Φ. For the Riesz kernel $\phi = r_\varepsilon$, we have* $\operatorname{Cap}(\Phi) \geq 2D$.

We now combine Corollary 7 with the arguments in this section to get our main result.

Table 1: For additive kernels, we state the univariate $\phi(s,t)$. `dpfp` is described in Appendix A and the remaining kernels are defined for $q, k \in \mathbb{R}^D$. All kernels except `tri` are nonnegative. `riesz` and `add_riesz` use $\varepsilon = 10^{-3}$ and are *partially* stationary, i.e., stationary up to terms depending only on q or only on k. The last column is the best known kernel sum complexity in $N+M$.

Name	Formula	dim	stationary	additive	capacity	complexity
`softmax`	$\mathrm{e}^{q^\top k}$	∞	no	no	∞	quadratic
`gauss`	$\mathrm{e}^{-\frac{1}{2}\|q-k\|_2^2}$	∞	yes	no	∞	quadratic
`laplace`	$\mathrm{e}^{-\|q-k\|_1}$	∞	yes	no	∞	quasi linear
`riesz`	$\|q\|_2 + \|k\|_2 - \|q-k\|_2 + \varepsilon$	∞	partially	no	$-$	quadratic
`add_riesz`	$\|s\| + \|t\| - \|s-t\| + \varepsilon$	∞	partially	yes	$\geq 2D$	quasi linear
`add_laplace`	$\mathrm{e}^{-\|s-t\|}$	∞	yes	yes	∞	quasi linear
`add_bump`	$\max\{0, 1-\|s-t\|\}$	∞	yes	yes	∞	quasi linear
`tri`	$\cos(s-t)$	$2D$	yes	yes	$\leq 2D$	linear
`relu`	$\mathrm{ReLU}(s)\mathrm{ReLU}(t)$	D	no	yes	$\leq D$	linear
`elu`	$(\mathrm{elu}+1)(s)(\mathrm{elu}+1)(t)$	D	no	yes	$\leq D$	linear
`dpfp`	Appendix A	$6D$	no	no	$\leq 6D$	linear

Corollary 10. *The additive- Laplace and bump kernel defined by* $\Phi(q,k) = \sum_{d=1}^{D} \phi(q_d, k_d)$ *with* $\phi(s,t) = \exp(-|s-t|)$ *and* $\phi(s,t) = \max(0, 1-|s-t|)$ *fulfill* $\mathrm{Cap}(\Phi) = \infty$ *and corresponding kernel sums can be computed in quasi linear complexity.*

4 Implementation and Experiments

Next, we investigate our findings numerically by three experiments. The first one in Section 4.1 aims to verify theoretical claims from the previous sections. To this end, we perform an associative-recall experiment, which numerically computes the capacity of the considered kernels. The second experiment in Section 4.2 investigates the computation time. To complement theoretical complexity analysis of the proposed quasi linear kernels, we investigate the "break-even" point. This describes the minimal sequence length N such that our attention algorithm is faster than softmax attention. In this part, we also comment on possible implementations of attention. Finally, the third experiment in Section 4.3 evaluates the performance of the kernels based on the text embedding transformer nomic-embed-text-v1 by Nussbaum et al. (2025). We include a similar experiment for a vision transformer in Appendix E. Table 1 gives an overview of the kernels evaluated in this section.

All experiments use PyTorch 2.13.0+cu129 and KeOps 2.3. While training runs on a multi GPU cluster, all time measurements (Table 2 & 5 and Figure 2) were taken on a single NVIDIA GeForce RTX 5090. The code corresponding to our experiments is available as supplementary material.

Bandwidth The bandwidth is set via $\Phi_\tau(q,k) \coloneqq \Phi(q/\tau, k/\tau)$, with $\tau = D^{1/4}$ as default for `softmax`, `gauss` and `tri`. Since both text and vision transformer use head dimension $D = 64$, we use $\tau \approx 2.828$. We also tested other bandwidths but found $\tau = D^{1/4}$ to be near optimal in both the text and vision transformer. Similarly, $\tau = 1.5$ for `add_bump`, $\tau = 0.5$ for `add_laplace` and $\tau = 6$ for `laplace` work well on both tasks. For $\varepsilon = 0$, the kernels `riesz`, `add_riesz`, `relu` and `dpfp` are scale invariant, i.e., $\Phi_\tau = \tau^\alpha \Phi_1$ for all $\tau > 0$ and some $\alpha > 0$. Therefore, the scale cancels out within attention and for $\varepsilon > 0$ it merely rescales ε, so no tuning is necessary. As the only not nonnegative kernel, `tri` causes numerical instability during training. To stabilize this, we clamp the attention normalization at 10^{-6} and use gradient clipping to avoid overshooting.

4.1 Associative recall capacity

Here, we assess the capacity of various kernels by an associative-recall experiment similar to Schlag et al. (2021, Sec. 6.1). More precisely, we start with learnable tokens $\boldsymbol{u} = (u_n)_{n=1}^N \subseteq \mathbb{R}^D$ and define keys and queries as $k_n = W_K u_n$ and $q_n = W_Q u_n$ for learnable matrices $W_Q, W_K \in \mathbb{R}^{D\times D}$. Then, for values $v_1, \dots, v_N \in \mathbb{R}^N$ chosen as the unit vectors, we minimize the loss function

$$\mathcal{L}(\boldsymbol{u}, W_Q, W_K) = \mathbb{E}_{\sigma,\xi}\left[\sum_{n=1}^{N} \|v_{\xi(n)} - y_{\sigma(n)}\|^2\right], \tag{13}$$

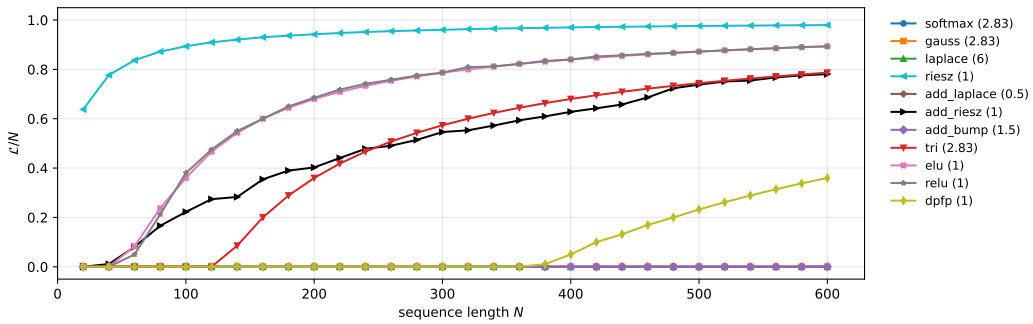


Figure 1: Associative-recall: Trained full-batch with Adam (lr 0.03) for at most $120k$ steps, stopping early once the loss falls below 10^{-4}, $D = 64$, where the tokens are initialized as standard normal and $W_Q = W_K = I$. Curves show the best loss over 3 seeds. Scale τ in brackets behind kernel name.

where $y_m = A(\boldsymbol{q}, \boldsymbol{k}_\sigma)\boldsymbol{v}_\xi = \sum_{l=1}^N A_{m,l}(\boldsymbol{q}, \boldsymbol{k}_\sigma)v_{\xi(l)}$ with $\boldsymbol{k}_\sigma = (W_K u_{\sigma(n)})_{n=1}^N$ is the attention vector as defined in (1) and σ and ξ are uniformly randomly drawn permutations. The loss does not depend on σ and ξ as the following lemma shows.

Lemma 11. *For all permutations σ and ξ it holds*

$$\sum_{n=1}^N \|v_{\xi(n)} - y_{\sigma(n)}\|_2^2 = \|A(\boldsymbol{q}, \boldsymbol{k}) - \mathrm{Id}_N\|_F^2, \qquad \boldsymbol{k} \coloneqq (W_K u_n)_{n=1}^N. \tag{14}$$

While Schlag et al. (2021) phrase this as a retrieval task, we show in Appendix F that it can be related to our capacity notion in the sense that

$$\inf_{\boldsymbol{u}, W_Q, W_K} \mathcal{L}(\boldsymbol{u}, W_Q, W_K) \geq \inf_{\boldsymbol{q}, \boldsymbol{k}} \|A(\boldsymbol{q}, \boldsymbol{k}) - \mathrm{Id}_N\|_F^2. \tag{15}$$

In particular, we have that $\inf_{\boldsymbol{u}, W_Q, W_K} \mathcal{L}(\boldsymbol{u}, W_Q, W_K) = 0$ only if $\mathrm{Cap}(\Phi) \geq N$.

In Figure 1, we plot the minimal value of $\mathcal{L}(\boldsymbol{u}, W_Q, W_K)$ after 120k optimization steps on W_Q, W_K and $\boldsymbol{u}$ for the kernels of Table 1. We observe that all FFM kernels roughly achieve a loss of zero for sequence lengths $N < L$ and end up with a strictly positive loss for $N > L$. This matches Corollary 4 stating that $\mathrm{Cap}(\Phi) \leq L$. On the other hand, the stationary $\mathcal{C}_0$ kernels `gauss` and `laplace` succeed in this task as predicted by Theorem 1. In alignment with Corollary 10, `add_laplace` and `add_bump` also pass the task up to $N = 600$. While Theorem 9 guarantees $\mathrm{Cap}(\Phi) \geq 128$ for `add_riesz`, the optimization already fails around $N = 40$, and `riesz` performs worst throughout.

4.2 Implementations and Speed Benchmarks

Now, we compare the runtime of the different backends of the attention algorithm. We already know that for $M = N$ softmax attention has complexity $\mathcal{O}(N^2)$, while our kernels have complexity $\mathcal{O}(N \log N)$. Hence, for long enough sequence lengths, quasi linear kernels should eventually become faster. Thus, we are interested in the "break-even" point, i.e., the minimal number N such that the quasi linear algorithms are faster than the standard softmax attention. For the quasi linear algorithms, we provide backends for `add_laplace` and the distance kernel. The latter can be used to compute all spline kernels and we choose `add_riesz` as a simple representative. Here, we consider the following backends.

- `GLOBAL`: Sorting $\boldsymbol{q}, \boldsymbol{k}$ and computing the prefix sums involves two separate CUDA kernels. Same structure is implemented for both `add_riesz` and `add_laplace`.
- `FUSED`: all operations are in a single kernel; this is limited by memory to $N \leq 4096$.
- `KEOPS`: a GPU-friendly $\mathcal{O}(N^2)$ reference (Charlier et al., 2021).

We provide implementation details in Appendix C. All backends of the quasi linear attention algorithm run with `fp32` precision. While an `fp16` implementation is algorithmically possible and can

Table 2: Forward-pass of kernel attention (1) with runtime in milliseconds, mean over 10 runs after 5 warm-up iterations. Relative standard deviations are below 18% for $N \leq 512$, below 4% at $N = 1024$ and below 3% for $N \geq 2048$. Bold marks the fastest method overall; underline marks the fastest among the five `fp32` methods. Shape: $B = 4$, $H = 12$, $D = 64$, $C = 64$, $M = N$.

	add_laplace fp32	add_riesz fp32			softmax			
N	GLOBAL	GLOBAL	FUSED	KEOPS	MEM_EFF32	MEM_EFF16	FLASH16	CUDNN16
128	0.442	0.455	0.305	0.381	0.029	**0.021**	**0.021**	0.023
256	0.575	0.539	0.353	0.425	0.058	0.031	**0.022**	0.030
512	0.824	0.694	0.478	0.688	0.144	0.058	**0.041**	0.043
1024	1.305	1.014	0.719	1.648	0.431	0.152	**0.093**	0.119
2048	2.260	1.690	1.278	4.754	1.645	0.537	0.266	**0.265**
4096	4.472	3.583	2.991	17.14	6.219	1.948	1.016	**1.001**
8192	9.109	7.336	–	63.94	24.12	7.424	3.780	**3.726**
16384	20.31	16.86	–	249.1	94.95	29.57	14.99	**14.90**
32768	42.07	**34.88**	–	985.6	379.4	118.8	59.79	60.11
65536	86.94	**73.76**	–	3877	1523	478.1	239.2	241.3
131072	**190.1**	200.0	–	15608	6118	1930	968.8	972.1

lead to further speedups, it requires substantial code engineering, which goes beyond the scope of this paper. We comment in Appendix C.4 on this issue. We compare the evaluation times of this kernel attention with the standard softmax attention. Here, we consider the softmax backends:

- `MEM_EFF32` and `MEM_EFF16` (Lefaudeux et al., 2022) in `fp32` and `fp16` precision.
- `FLASH16` and `CUDNN16` (Dao, 2024) using `fp16` precision.

Our `weighted_abs_sum` supports gradients and padding, such that all spline kernels can be used in both training and inference. Quasi linear causal attention is possible and explained in Appendix C.4 but not implemented. The `add_laplace` is forward only.

We report the runtime of a forward pass of (1) in Table 2. Appendix C covers the backward-pass with similar conclusions. At equal precision `FUSED` becomes faster than `MEM_EFF32` at $N \approx 2048$. For $N = 131072$, `GLOBAL` is roughly $31\times$ faster. The `fp16` backends are 2–$6\times$ faster than their `fp32` counterparts, so the break-even against `FLASH16` occurs slightly beyond $N = 16384$. We view this as an engineering rather than a conceptual gap: the `fp16` backends exploit tensor cores and years of kernel tuning, whereas a flash-style `fp16` tiling of the quadratic regime combined with our quasi linear kernel is possible in principle and would move the crossover accordingly. Regarding memory, our backend allocates roughly $3\times$ the forward peak memory of `MEM_EFF32`, as it must hold sorted copies of the queries, keys and their sorting permutations. Unlike a naive PyTorch implementation of Algorithm 1, which must materialize the prefix sums of size $\mathcal{O}(PDNC)$, we compute the prefix sums on the fly, such that the overhead reduces to $\mathcal{O}(PDN)$.

4.3 Text embedding transformer

Finally, we evaluate the performance of the quasi linear kernels on a text embedding transformer. To this end, we consider the sentence embedder nomic-embed-text-v1 (Nussbaum et al., 2025) (12 layers, 12 heads, $D = 64$, rotary positional embeddings (RoPE)).

Setup and Training To avoid the computationally expensive and hyperparameter-sensitive training procedure, we start with a pretrained teacher model using softmax attention and distill student models using other kernels from this teacher model. Here, we proceed in three phases:

1. **Initialization:** We initialize the student model by copying the weights of the teacher model.
2. **Layer-by-layer distillation:** For each attention layer, we match the MSE between the softmax teacher and the student.
3. **End-to-end distillation:** We minimize the MSE between the end-to-end application of the teacher and student model.

For both distillations, we use `nomic-ai/nomic-embed-unsupervised-data`, specifically the `reddit_title_body` split. Texts are truncated to 512 tokens, batched in groups of 16, and decorated with one of the task prefixes `classification`, `clustering`, `search_document` and `search_query`, sampled with probabilities 0.45, 0.1, 0.35 and 0.1. In the first distillation, each layer is trained for 10,000 steps with Adam, with a learning rate cosine-annealed from $5 \cdot 10^{-4}$

Table 3: Results after full two-stage distillation for MTEB and LoCo. Best performance among quasi linear kernels is underlined.

Kernel	scale τ	MTEB ↑ ≤ 512	LoCo (nDCG@10) ↑ 2048	4096	8192
`softmax` (teacher)	2.828	64.41	0.873	0.882	0.883
`gauss`	2.828	64.41	0.875	0.884	0.886
`laplace`	6.0	64.38	0.885	0.891	0.887
`riesz`	1.0	61.69	0.805	0.839	0.858
`add_riesz`	1.0	61.72	0.769	0.813	0.832
`add_laplace`	0.5	62.12	<u>0.830</u>	<u>0.845</u>	0.858
`add_bump`	1.5	62.12	0.829	0.843	<u>0.859</u>
`tri`	2.828	61.53	0.829	0.838	0.851
`relu`	1.0	62.29	0.472	0.516	0.556
`elu`	1.0	61.55	0.699	0.773	0.794
`dpfp`	1.0	<u>63.42</u>	0.681	0.811	0.823

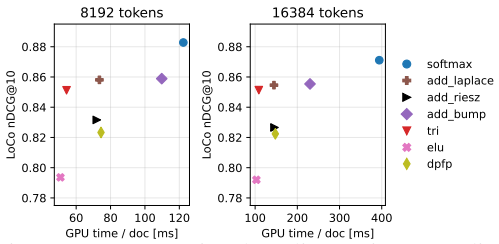


Figure 2: LoCo retrieval quality against encoding cost at 8192 and 16384 tokens. Each point is one kernel: y is the mean nDCG@10 over five LoCo tasks, x is GPU time per document.

to $5 \cdot 10^{-5}$. In the second distillation, all parameters are trained for 10,000 steps with Adam, with a linear warmup over the first 10% of steps to $5 \cdot 10^{-5}$, followed by cosine annealing to 10^{-5}.

Evaluation and Results We evaluate on two regimes: the MTEB(eng, v2) benchmark (Muennighoff et al., 2023), an average over 41 datasets whose documents are short ($N \leq 512$), and LoCo long-context retrieval (Saad-Falcon et al., 2024) (nDCG@10, averaged over 5 subsets) at lengths 2048, 4096 and 8192, well beyond the training length of 512.

We observe that `gauss` and `laplace` fully recover the teacher. The Riesz kernel and the quasi linear kernels moderately lose performance. The FFM kernels work well for the short sequences from the MTEB benchmark, but mostly degrade in performance for the LoCo. Only the `tri` kernel achieves reasonable results. On the other hand, the infinite capacity kernels `add_bump` and `add_laplace` remain close to the `softmax` teacher model. Moreover, on LoCo, stationary kernels tend to extrapolate better than non-stationary ones. This may be due to RoPE (with NTK scaling), which is designed for rotation-invariant kernels such as `softmax`. While `laplace`, `tri` and our additive kernels are not rotation invariant, they remain stationary, whereas the FFM kernels `relu`, `elu` and `dpfp` are neither. For this retrieval task, which relies on few relevant tokens and mean pooling, stationarity appears to matter more for extrapolation than capacity.

Further, we evaluate the time-performance trade-off. More precisely, we plot in Figure 2 the performance on LoCo versus the required evaluation time for each kernel. We consider both sequence lengths of 8192 and 16384, where longer sequences are truncated and shorter sequences are padded. We can see that for long sequences the `tri` and `add_laplace` kernels both significantly improve the computation time compared to the softmax kernel with a moderate loss of performance. Here, the `tri` kernel is slightly faster while the `add_laplace` kernel has a slightly better performance.

5 Conclusions, Limitations and Future Work

We analyzed kernel attention to identify kernels that are both expressive and enable quasi linear computation. Expressivity is measured through the maximum sequence length for which the attention matrix can approximate the identity (capacity). We showed that expressive kernels such as softmax, Gauss and Laplace have infinite capacity while finite feature map kernels saturate. By leveraging additive kernels and one dimensional sorting, we identified a class of kernels with infinite capacity and quasi linear complexity. Experiments confirm that our CUDA-based kernel attention is faster than softmax attention for sequence lengths exceeding 2048 elements in `fp32`.

While our capacity metric provides a rigorous theoretical framework for comparing kernel attention, it captures only one aspect of expressiveness. Theoretically, kernel attention with higher capacity can represent classes of functions that cannot be represented with lower capacity. However, the full expressive power of transformers is influenced by additional properties beyond capacity, such as stationarity and locality. In the long term, an `fp16` implementation, leveraging tensor cores and optimized memory access patterns, could yield significant additional speedups, particularly for long sequences. Further, we intend to extend our implementation to causal attention and to evaluate our method on larger-scale benchmarks to validate its scalability and performance.

AI USE STATEMENT

We did not use generative AI for any tasks with required disclosure: all research ideas, methodology, experimental design, and interpretation of results are the authors' own work, as is the design of the GPU implementation, including the parallelization strategy, the memory and thread layout and the kernel fusion; synthetic data generation, translation, and dataset cleaning are not applicable. Starting from a complete, author-written implementation of this design, we used generative AI tools to extend and clean up the code and repository, to unify naming and structure, and to apply hardware-specific low-level optimizations; these changes affect how the computation is executed on the device rather than the algorithm itself, they did improve performance, and the runtimes we report are those of the resulting code. We also used generative AI tools to identify related work and candidate citations. All AI-modified code was reviewed by the authors and verified for numerical agreement with the author-written reference implementation before being re-tested and re-benchmarked, and all AI-suggested references were verified against the original sources. The paper was written by the authors, with generative AI used to polish wording. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The source code for our experiments is available as supplementary material. All required datasets and models are openly available from the literature.

REFERENCES

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *Int. Conf. Learn. Represent.*, 2024a. URL https://openreview.net/forum?id=LY3ukUANko.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Int. Conf. Mach. Learn.*, pp. 1763–1840, 2024b. URL https://proceedings.mlr.press/v235/arora24a.html.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980. doi: 10.1145/358841.358850.

Siwan Boufadene, François-Xavier Vialard, and Jean Feydy. Fast large deformation matching with the energy distance kernel. *arXiv preprint arXiv:2505.03342*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Adv. Neural Inf. Process. Syst.*, pp. 1877–1901, 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the GPU, with autodiff, without memory overflows. *J. Mach. Learn. Res.*, 22(74):1–6, 2021. URL http://jmlr.org/papers/v22/20-275.html.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Łukasz Kaiser,

David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with Performers. In *Int. Conf. Learn. Represent.*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Int. Conf. Learn. Represent.*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Adv. Neural Inf. Process. Syst.*, pp. 16344–16359, 2022. doi: 10.52202/068431-1189.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. Learn. Represent.*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. doi: 10.1007/BF02288367.

Zhe Feng, Sen Lian, Changwei Wang, Muyang Zhang, Tianlong Tan, Rongtao Xu, Weiliang Meng, and Xiaopeng Zhang. LaplacianFormer: Rethinking linear attention with Laplacian kernel. In *Int. Conf. Learn. Represent.*, 2026. URL https://openreview.net/forum?id=bJZExGYWqx.

Takashi Furuya, Maarten V. de Hoop, and Gabriel Peyré. Transformers are universal in-context learners. In *Int. Conf. Learn. Represent.*, 2025. URL https://openreview.net/forum?id=6S4WQD1LZR.

Johannes Hertrich. Fast kernel summation in high dimensions via slicing and Fourier transforms. *SIAM J. Math. Data Sci.*, 6(4):1109–1137, 2024. doi: 10.1137/24M1632085.

Johannes Hertrich, Christian Wald, Fabian Altekrüger, and Paul Hagemann. Generative sliced MMD flows with Riesz kernels. In *Int. Conf. Learn. Represent.*, 2024. URL https://openreview.net/forum?id=VdkGRV1vcf.

Johannes Hertrich, Tim Jahn, and Michael Quellmalz. Fast summation of radial kernels via QMC slicing. In *Int. Conf. Learn. Represent.*, 2025. URL https://openreview.net/forum?id=iNmVX9lx9l.

David P. Hofmeyr. Fast exact evaluation of univariate kernel sums. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(2):447–458, 2021. doi: 10.1109/TPAMI.2019.2930501.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh (eds.), *Int. Conf. Mach. Learn.*, pp. 5156–5165, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Int. Conf. Learn. Represent.*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Nicolas Langrené and Xavier Warin. Fast multivariate empirical cumulative distribution function with connection to kernel density estimation. *Comput. Stat. Data Anal.*, 162:107267, 2021. doi: 10.1016/j.csda.2021.107267.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xFormers: A modular and hackable Transformer modelling library. https://github.com/facebookresearch/xformers, 2022.

Llama Team, AI @ Meta. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein (eds.), *Conf. Eur. Chapter Assoc. Comput. Linguist.*, pp. 2014–2037, Dubrovnik, Croatia, 2023. doi: 10.18653/v1/2023.eacl-main.148.

Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. Nomic Embed: Training a reproducible long context text embedder. *Trans. Mach. Learn. Res.*, 2025. URL https://openreview.net/forum?id=IPmzyQSiQE.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. Random feature attention. In *Int. Conf. Learn. Represent.*, 2021. URL https://openreview.net/forum?id=QtTKTdVrFBB.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosFormer: Rethinking softmax in attention. In *Int. Conf. Learn. Represent.*, 2022. URL https://openreview.net/forum?id=Bl8CQrx2Up4.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Adv. Neural Inf. Process. Syst.*, pp. 1177–1184, 2007. URL https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

Nicolaj Rux, Johannes Hertrich, and Sebastian Neumayer. Numerical methods for kernel slicing. *arXiv preprint arXiv:2510.11478*, 2025.

Jon Saad-Falcon, Daniel Y. Fu, Simran Arora, Neel Guha, and Christopher Ré. Benchmarking and building long-context retrieval models with LoCo and M2-BERT. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Int. Conf. Mach. Learn.*, pp. 42918–42946, 2024. URL https://proceedings.mlr.press/v235/saad-falcon24a.html.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In Marina Meila and Tong Zhang (eds.), *Int. Conf. Mach. Learn.*, pp. 9355–9366, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

M. A. Shubin. Almost periodic functions and partial differential operators. *Russian Math. Surveys*, 33(2):1–52, 1978. doi: 10.1070/RM1978v033n02ABEH002303.

Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Information Science and Statistics. Springer, New York, 2008. doi: 10.1007/978-0-387-77242-4.

Tom Streubel, Andreas Griewank, Manuel Radons, and Jens-Uwe Bernt. Representation and analysis of piecewise linear functions in abs-normal form. In Christian Pötzsche, Clemens Heuberger, Barbara Kaltenbacher, and Franz Rendl (eds.), *Syst. Model. Optim.*, pp. 327–336, Klagenfurt, Austria, 2014. doi: 10.1007/978-3-662-45504-3_32.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6):109:1–109:28, 2022. doi: 10.1145/3530811.

Tanja Teuber, Gabriele Steidl, Pascal Gwosdek, Christian Schmaltz, and Joachim Weickert. Dithering by differences of convex functions. *SIAM J. Imaging Sci.*, 4(1):79–108, 2011. doi: 10.1137/100790197.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Conf. Empir. Methods Nat. Lang. Process. and Int. Jt. Conf. Nat. Lang. Process.*, pp. 4344–4353, Hong Kong, China, 2019. doi: 10.18653/v1/D19-1443.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inf. Process. Syst.*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

François-Xavier Vialard and Siwan Boufadène. Sliced ReLU attention: Quasi-linear contextual expressivity via sorting. *arXiv preprint arXiv:2512.11411*, 2025.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Holger Wendland. *Scattered Data Approximation*, volume 17 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2004. doi: 10.1017/CBO9780511617539.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. *Proc. AAAI Conf. Artif. Intell.*, 35(16):14138–14148, 2021. doi: 10.1609/aaai.v35i16.17664.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Int. Conf. Mach. Learn.*, pp. 56501–56523, 2024a. URL https://proceedings.mlr.press/v235/yang24ab.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Adv. Neural Inf. Process. Syst.*, pp. 115491–115522, 2024b. doi: 10.52202/079017-3668.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *Int. Conf. Learn. Represent.*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for longer sequences. In *Adv. Neural Inf. Process. Syst.*, pp. 17283–17297, 2020. URL https://papers.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

A Background on FFMs

Here, we discuss the feature maps proposed by Katharopoulos et al. (2020); Choromanski et al. (2021); Peng et al. (2021); Schlag et al. (2021). In Katharopoulos et al. (2020), the feature map $\varphi\colon \mathbb{R}^D \to \mathbb{R}^D$ with $\varphi(x) \coloneqq [\mathrm{elu}(x_1)+1, \dots, \mathrm{elu}(x_D)+1]$ is used. Later, Peng et al. (2021) used Bochner's theorem to construct a feature map that approximates the Gauss kernel via

$$\varphi_{\cos}\colon \mathbb{R}^D \to \mathbb{R}^{2L}, \varphi_{\cos}(x) \coloneqq {}^1\!/\!\sqrt{L}\big[\sin(\omega_1^\top x), \dots, \sin(\omega_L^\top x), \cos(\omega_1^\top x), \dots, \cos(\omega_L^\top x)\big], \quad (16)$$

where $\omega_1, \dots, \omega_L \sim \mathcal{N}(0, \tau I_D)$ are sampled i.i.d. The associated kernel is then given as

$$\begin{aligned}\varphi_{\cos}(q)^\top \varphi_{\cos}(k) &= \frac{1}{L}\sum_{l=1}^{L}\big(\sin(\omega_l^\top q)\sin(\omega_l^\top k) + \cos(\omega_l^\top q)\cos(\omega_l^\top k)\big)\\ &= \frac{1}{L}\sum_{l=1}^{L}\cos\big(\omega_l^\top (q-k)\big). \end{aligned} \quad (17)$$

Now, Bochner's theorem implies for $z = q - k$ the relation

$$G(q,k) = (2\pi\tau)^{-\frac{D}{2}}\int_{\mathbb{R}^D}\cos(\omega^\top z)\mathrm{e}^{-\frac{\|\omega\|^2}{2\tau}}\,\mathrm{d}\omega = \mathop{\mathbb{E}}_{\omega\sim\mathcal{N}(0,\tau I_D)}\big[\cos(\omega^\top z)\big] \approx \frac{1}{L}\sum_{l=1}^{L}\cos(\omega_l^\top z). \quad (18)$$

This idea is due to Rahimi & Recht (2007), who also establish error estimates. The kernel induced by this feature map is not nonnegative anymore, which can lead to numerical instability.

In Peng et al. (2021), they resample ω_l during training, but keep them fixed at test time, as they did not notice any change of behavior when resampling. Moreover, they use exactly $L = D$ directions. Attention generates the queries q, keys k and values v by multiplying a token $u \in \mathbb{R}^D$ with three separate matrices $W_Q, W_K, W_V \in \mathbb{R}^{D\times D}$. The linear projections $\omega_1, \dots, \omega_D$ can be fused with the matrices W_Q and W_K to ωW_K and ωW_Q, and we can assume instead the form $1/L \sum_{l=1}^{L} \cos(q_l - k_l)$ with $L = D$. Therefore, we interpret the resampling during training as a training mechanism rather than a property of the kernel or attention.

Choromanski et al. (2021) use a similar argument to directly approximate the softmax kernel with the feature map $\varphi_{\exp}(x) = \exp(-\|x\|_2^2/2)[\exp(x_1), \dots, \exp(x_D)]$. With the same argument for $\varphi_{\cos}$ from the previous paragraph we omit the random projections. In contrast to $\varphi_{\cos}$, their FFM $\varphi_{\exp}$ is nonnegative, but not stationary, see Proposition 5. Among the FFMs introduced by Choromanski et al. (2021), they found $\varphi_{\mathrm{ReLU}}(x) = [\mathrm{ReLU}(x_1), \dots, \mathrm{ReLU}(x_D)]$ to generally perform best, which is why we use φ_{ReLU} during the numerical comparison.

Table 4 summarizes the previous FFMs $\varphi\colon \mathbb{R}^D \to \mathbb{R}^{JD}$ of Katharopoulos et al. (2020); Choromanski et al. (2021); Peng et al. (2021), which all are of the form

$$\varphi(x) = [f_1(x_1), \dots, f_1(x_D), \dots, f_J(x_1), \dots, f_J(x_D)]^\top \in \mathbb{R}^{JD}. \tag{19}$$

Table 4: Examples of FFMs of the form (19).

Paper	φ	J	f_j
Katharopoulos et al. (2020)	φ_{elu}	1	$f_1(x) = \mathrm{elu}(x) + 1$
Peng et al. (2021)	φ_{tri}	2	$f_1(x) = \sin(x), f_2(x) = \cos(x)$
Choromanski et al. (2021)	φ_{ReLU}	1	$f_1(x) = \mathrm{ReLU}(x)$

Note that all FFM of the form (19) are additive kernels over $F(s,t) = f(s)^\top f(t)$, because

$$\Phi(q,k) = \varphi(q)^\top \varphi(k) = \sum_{d=1}^{D} F(q_d, k_d), \quad \text{with} \quad F(s,t) = f(s)^\top f(t). \tag{20}$$

However, their univariate kernel F is a FFM with $\dim \mathcal{H}_F = J$ of dimension 1 or 2, while the Laplace, Riesz and bump kernel have infinite dimension.

Schlag et al. (2021) replaces the random features proposed by Choromanski et al. (2021); Peng et al. (2021) with *deterministic parameter-free projection* (DPFP): Let $r\colon \mathbb{R}^D \to \mathbb{R}^{2D}$ with $r(x) = [\mathrm{ReLU}(x), \mathrm{ReLU}(-x)] \in \mathbb{R}^{2D}$ and $\nu \in \{1, \dots, D-1\}$. The DPFP feature map is $\varphi\colon \mathbb{R}^D \to \mathbb{R}^{2D\nu}$ with $\varphi_{2Dj+k}(x) = r(x)_k \, r(x)_{k+j}$ for $j = 0, \dots, \nu - 1$ and $k = 1, \dots, 2D$, where the indices are taken modulo $2D$. This differs from the general form (19). Here, we require $\nu < D$, so that the components of φ are linearly independent. In fact, $\nu \geq D$ is wasteful, as it does not increase the dimension of the RKHS anymore. In practice, we follow Schlag et al. (2021) and use $\nu = 3$ with $D = 64$, so that $\dim \mathcal{H} = 2\nu D = 384$.

B Connection to slicing

Additive kernels can be seen as a deterministic slicing of radial kernels. Given a univariate kernel $f(|s-t|)$, the sliced kernel identity by Hertrich (2024) expresses a radial kernel on $\mathbb{R}^D$ as an expectation over uniformly random directions,

$$F(\|q-k\|_2) = \mathbb{E}_{\xi \sim \mathcal{U}(\mathbb{S}^{D-1})}\big[f(|\langle q-k, \xi\rangle|)\big] \approx \frac{1}{L}\sum_{l=1}^{L} f(|\langle q-k, \xi_l\rangle|), \tag{21}$$

where F is obtained from f by the Riemann Liouville transform $\mathcal{S}_d$, see also Rux et al. (2025, Eq. (10)). The fast summation of Hertrich (2024) estimates (21) by drawing L directions $\xi_1, \dots, \xi_L$ and applying a one dimensional fast summation along each of them, which costs

$\mathcal{O}(L(N+M)\log N)$ and converges at the Monte Carlo rate $\mathcal{O}(L^{-1/2})$. Replacing the random directions by the fixed coordinate axes $e_1, \dots, e_D$ yields, up to the factor $1/D$, exactly the additive kernel,

$$\frac{1}{L}\sum_{l=1}^{L} f(|\langle q-k, \xi_l\rangle|) = \frac{1}{D}\sum_{l=1}^{D} f(|\langle q-k, e_l\rangle|) = \frac{1}{D}\sum_{d=1}^{D} f(|q_d - k_d|) \tag{22}$$

The two constructions therefore share the same one dimensional primitive. Choosing for $L = D$ the directions $\xi_l = e_l$ can further be interpreted as quasi-Monte Carlo design for the integration of the sphere, which sometimes achieves better error rates than the previously mentioned standard estimate for Monte Carlo methods. We refer to Hertrich et al. (2025) for details.

In the special case of Riesz, the sliced multivariate Riesz kernel yields the univariate Riesz kernel. Therefore, additive Riesz $\|q\|_1 + \|k\|_1 - \|q-k\|_1$ can be seen as a Monte Carlo approximation of the multivariate Riesz kernel $\|q\|_2 + \|k\|_2 - \|q-k\|_2$. However, the additive Laplace kernel does not appear as Monte Carlo approximation neither of ℓ_1 Laplace nor of ℓ_2 Laplace. Instead additive Laplace is the Monte Carlo approximation of a power series given in Hertrich (2024, Table. 1). For more background on slicing see Hertrich (2024); Rux et al. (2025). Applications of the slicing algorithm were also presented in Hertrich et al. (2024); Boufadene et al. (2025).

C Fast Computation of Spline-Kernel Attention

In most transformer architectures, the head dimension D is either $32, 64$ or 128, and the number of channels C usually equals D, or $C = D+1$ if the values are extended by a row of constant ones for normalization. We are interested in the case where the sequence lengths satisfy $N \approx M$ and $N, M \gg D, C$. Let $\boldsymbol{q} \in \mathbb{R}^{M\times D}$, $\boldsymbol{k} \in \mathbb{R}^{N\times D}$ and $\boldsymbol{v} \in \mathbb{R}^{N\times C}$. We focus on the efficient computation of *absolute value sums*

$$z_m = \sum_{n=1}^{N} \|q_m - k_n\|_1 v_n, \qquad m = 1, \dots, M. \tag{23}$$

The naive computation of the absolute value sum requires $\mathcal{O}((D+C)NM)$ operations and would be the computational bottleneck if N and M are both large. Instead, it is possible to reduce the complexity to $\mathcal{O}(D(N+M)(\log_2 N+C))$. Simply, rewrite

$$\sum_{n=1}^{N} \|q_m - k_n\|_1 v_n = \sum_{d=1}^{D}\sum_{n=1}^{N} |q_{m,d} - k_{n,d}| v_n. \tag{24}$$

Each inner sum is of the type $\sum_{n=1}^{N} |s_m - t_n| v_n$, where $s_m = q_{m,d} \in \mathbb{R}$ and $t_n = k_{n,d} \in \mathbb{R}$. The absolute value sum in $\mathbb{R}$ can be computed in $\mathcal{O}((N+M)(\log_2 N+C))$ by Algorithm 1. By Proposition 6 all spline kernels with finitely many nodes, can be computed in quasi linear time, by calling Algorithm 1 on each node. More details are given inside the proof of Proposition 6

C.1 Parallelization and Memory overhead

In practice, an efficient scalable implementation is often worth more than theoretical runtime asymptotics. Although flash attention (Dao et al., 2022) did not change the asymptotic work, it had significant impact. While (23) can be computed asymptotically fast, two questions must be answered simultaneously for both the forward and the backward pass: (i) Can the implementation be *parallelized*? and (ii) Can we handle *memory efficiently*? Without a parallel implementation, this method cannot compete against massively parallel algorithms such as flash attention. Even a parallel implementation will not convince in practice if it is not memory efficient.

In practice, each tensor has an additional batch dimension P, namely $\boldsymbol{q} \in \mathbb{R}^{P\times D\times M}$, $\boldsymbol{k} \in \mathbb{R}^{P\times D\times N}$ and $\boldsymbol{v} \in \mathbb{R}^{P\times N\times C}$. If we implement Algorithm 1 via PyTorch primitives, then the prefix sums are materialized for each D and each batch P, resulting in $\mathcal{O}((N+M)DCP)$ memory costs. This is a known issue that has been attacked already by Dao et al. (2022); Charlier et al. (2021) for certain expand-reduce computations. Of course, we could instead serialize over D and compute each part independently or in a smaller batch, but then again we pay with parallelism for memory. Yet, there is a way to parallelize completely over P, D and C, staying serial only in N/M, without materializing

any huge intermediates. The idea is to sort $\boldsymbol{q}$ as well and never materialize the entire prefix sums. Instead, for each d, we scan through the sorted keys $k_{\sigma(n),d}$ and values $v_{\sigma(n)}$ for $n = 1, \dots, N$, compute the prefix sums on the fly and write the result for query m as soon as the scan reaches position p_m. Since we also sort $\boldsymbol{q}$, the positions p_m are monotonically increasing, which ensures that no previous values are needed again. In the end, we correct the result with the total sum. In practice, we use a grid of size (D, P) and block size C. Since a warp always works in 32 threads, our implementation favors C being a multiple of 32 and $C \leq 1024$, which is usually satisfied. The fused scan kernel has no thread divergence and data independent runtime. One disadvantage is that the final reduction over D cannot be performed within a block, so atomic stores are needed. This is the main bottleneck of our implementation.

C.2 Gradients and Autodiff

Since our implementation is in CUDA, we need to implement the gradient for autograd ourselves. All gradients of our method can be computed in $\mathcal{O}((N{+}M)\log(N{+}M))$, such that both forward and backward run subquadratic in $N{+}M$. The subgradient of the absolute value is sign, and within the gradients it is necessary to compute sums of the form

$$\begin{aligned}\sum_{n=1}^{N} \operatorname{sign}(s_m - t_n) v_n &= \sum_{n=1}^{N} \operatorname{sign}(s_m - t_{\sigma(n)}) v_{\sigma(n)} = \sum_{n=1}^{p_m} v_{\sigma(n)} - \sum_{n=p_m+1}^{N} v_{\sigma(n)} \\ &= 2\text{pref_v}_{p_m} - \text{pref_v}_N. \end{aligned} \tag{25}$$

Again, these sums can be computed with $\mathcal{O}((N{+}M)(\log_2 N{+}C))$ work.

Let $\boldsymbol{z} \in \mathbb{R}^{M \times C}$ be the output of (23) and $\boldsymbol{g} \in \mathbb{R}^{M \times C}$ the upstream gradient. In the following, we compute the vector Jacobian products (VJP) of $\mathcal{J} = \langle \boldsymbol{g}, \boldsymbol{z} \rangle$. The VJP w.r.t. $\boldsymbol{v} \in \mathbb{R}^{N \times C}$ is

$$\frac{\partial \mathcal{J}}{\partial v_{n,c}} = \sum_{d=1}^{D} \sum_{m=1}^{M} |q_{m,d} - k_{n,d}|\, g_{m,c}. \tag{26}$$

This gradient can be computed just like the forward pass by swapping $\boldsymbol{q} \leftrightarrow \boldsymbol{k}$ and replacing $\boldsymbol{v}$ by $\boldsymbol{g}$. The VJP w.r.t. $\boldsymbol{q} \in \mathbb{R}^{M \times D}$ reads

$$\frac{\partial \mathcal{J}}{\partial q_{m,d}} = \sum_{c=1}^{C} g_{m,c} \sum_{n=1}^{N} \operatorname{sign}(q_{m,d} - k_{n,d}) v_{n,c}. \tag{27}$$

In theory, the tensor pref_v could be passed from the forward to the backward pass. However, as we compute it on the fly, we need to recompute it during the backward pass. Lastly, the VJP w.r.t. $\boldsymbol{k} \in \mathbb{R}^{N \times D}$ is given via

$$\frac{\partial \mathcal{J}}{\partial k_{n,d}} = \sum_{c=1}^{C} v_{n,c} \sum_{m=1}^{M} \operatorname{sign}(k_{n,d} - q_{m,d})\, g_{m,c}. \tag{28}$$

Both sign sums have a reduction over C, which is our block dimension. Therefore, this reduction can be performed within a block and no atomics are needed anymore.

Table 5 measures the speed of our additive Riesz attention versus softmax backend. For `FUSED`, only the forward pass is fused: the backward walk reuses the sorted tensors, which the fused forward does not materialize, so it runs on the `GLOBAL` backward. The break even points are slightly better than the forward computation in Table 2, as `GLOBAL` even beats `FLASH16` before $N = 16384$. This has a simple reason: We measure the elapsed time for the forward and the $\boldsymbol{q}, \boldsymbol{k}$ and $\boldsymbol{v}$- backwards. Usually, those are 4 equally sized computations. For example, `FLASH16`'s forward+backward at $N = 131072$ is 3442 and about $3.55\times$ larger than its forward 968.8 in Table 2. However, `GLOBAL`'s presort of $\boldsymbol{q}$ and $\boldsymbol{k}$, can be reused during the three backwards leading to a factor of 3 from 200 to 600.6. Similarly, the relative memory overhead between `GLOBAL` and `fp32` softmax backend is around 1.5, while the forward was around 3 times larger.

C.3 Padding

Consider a batch of P samples with sequence lengths N_p, $p = 1, \dots, P$, and maximal length $N = \max_p N_p$. Padding extends the queries $\boldsymbol{q} \in \mathbb{R}^{D \times N_p}$, keys $\boldsymbol{k} \in \mathbb{R}^{D \times N_p}$ and values $\boldsymbol{v} \in \mathbb{R}^{C \times N_p}$

Table 5: Forward+backward runtime in milliseconds, mean over 10 runs after 5 warm-up iterations, with forward and backward timed together in a single region. Relative standard deviations are below 10% for $N \leq 512$, below 1.2% at $N = 1024$ and below 0.7% for $N \geq 2048$. Bold marks the fastest method overall; underline marks the fastest among the four `fp32` methods. Shape: $B = 4$, $H = 12$, $D = 64$, $C = 64$, $M = N$.

	add_riesz fp32			softmax			
N	GLOBAL	FUSED	KEOPS	MEM_EFF32	MEM_EFF16	FLASH16	CUDNN16
128	1.066	0.913	1.598	0.098	0.071	0.059	**0.057**
256	1.329	1.147	2.424	0.233	0.104	**0.067**	0.069
512	1.832	1.620	4.087	0.609	0.231	**0.132**	0.143
1024	2.950	2.664	11.96	1.923	0.634	**0.324**	0.363
2048	5.283	4.927	40.33	7.360	2.262	**0.978**	1.050
4096	11.19	10.66	140.5	27.95	8.363	**3.731**	4.029
8192	23.18	–	520.9	109.0	31.72	**13.90**	14.99
16384	**50.03**	–	2007	433.4	129.8	54.35	57.88
32768	**103.8**	–	7903	1729	503.6	214.9	228.6
65536	**219.1**	–	31550	6905	1990	856.9	913.2
131072	**600.6**	–	125800	27730	8008	3442	3666

of sample p to $\tilde{\boldsymbol{q}}, \tilde{\boldsymbol{k}} \in \mathbb{R}^{D \times N}$ and $\tilde{\boldsymbol{v}} \in \mathbb{R}^{C \times N}$ with

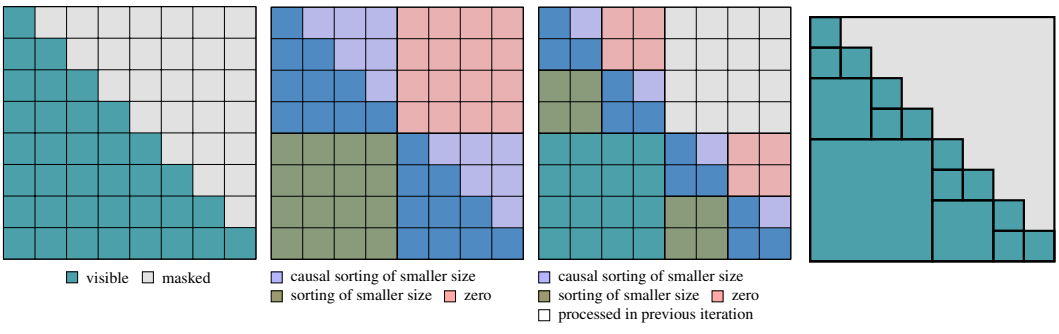
$$\tilde{q}_n = q_n,\ \tilde{k}_n = k_n,\ \tilde{v}_n = v_n \quad \text{for } n \leq N_p, \quad \tilde{q}_n = 0,\ \tilde{k}_n = 0,\ \tilde{v}_n = 0 \quad \text{for } N_p < n \leq N. \tag{29}$$

Since the padded values vanish, the kernel sums (7) are unchanged, $\tilde{z}_m = \sum_{n=1}^{N} \Phi(\tilde{q}_m, \tilde{k}_n)\tilde{v}_n = z_m$ for $m \leq N_p$, so the weighted absolute value sums of Algorithm 1 are applied to the padded tensors without any modification. Only the normalization $\sum_{n=1}^{N} \Phi(\tilde{q}_m, \tilde{k}_n)$ counts every padded key with weight one and hence contains the additional term $(N - N_p)\,\Phi(\tilde{q}_m, 0)$. In order to fix the normalization, we compute this correction for all queries in $\mathcal{O}(ND)$ operations, which is negligible compared to the kernel sums. Then, we obtain for $m = 1, \ldots, N_p$ that

$$\frac{\sum_{n=1}^{N_p} \Phi(q_m, k_n) v_n}{\sum_{n=1}^{N_p} \Phi(q_m, k_n)} = \frac{\sum_{n=1}^{N} \Phi(\tilde{q}_m, \tilde{k}_n)\tilde{v}_n}{\sum_{n=1}^{N} \Phi(\tilde{q}_m, \tilde{k}_n) - (N - N_p)\,\Phi(\tilde{q}_m, 0)}, \tag{30}$$

while the outputs of the padded queries $m > N_p$ are discarded. Thus, we can pad samples of different length to a common length N in order to process inputs of different sizes in one batch, which is necessary, for example, for text models.

C.4 Causal masking



Causal mask | Decomposition in Step 1 | Decomposition in Step 2 | Final decomposition

Figure 3: Illustration of the causal mask and of Algorithm 2 for computing the kernel summation with causal mask in quasi linear time. From left to right: causal mask applied to the kernel matrix; block decomposition in the first step of Algorithm 2; block decomposition in the second step; the resulting block partition after the full recursion.

Many current transformers use causal masking. That is, the kernel sum (7) is replaced by $z_m = \sum_{n=1}^{m} \Phi(q_m, k_n) v_n$ for $m = 1, \ldots, N$. We refer to Figure 3 for an illustration of the mask.

Algorithm 2 Sorting algorithm for causal masked kernel sums

1 **Input** $\boldsymbol{s} \in \mathbb{R}^M$, $\boldsymbol{t} \in \mathbb{R}^N$ and $\boldsymbol{v} \in \mathbb{R}^{N \times C}$
2 **Output** $(z_m)_{m=1}^M = \mathrm{CausalSorting}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{v})$ given by $z_m \coloneqq \sum_{n=1}^m \phi(s_m, t_n) v_n \in \mathbb{R}^C$
3 **Denote** by $\mathrm{Sorting}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{v})$ the application of Algorithm 1
4 **If** $\min(M, N) = 1$**:**
5 compute $(z_m)_{m=1}^M$ naively
6 **Else:**
7 compute $(z_m)_{m=1}^{M/2} = \mathrm{CausalSorting}((s_m)_{m=1}^{M/2}, (t_n)_{n=1}^{N/2}, (v_n)_{n=1}^{N/2})$
8 compute $(z_{m,1})_{m=M/2+1}^{M} = \mathrm{Sorting}((s_m)_{m=M/2+1}^{M}, (t_n)_{n=1}^{N/2}, (v_n)_{n=1}^{N/2})$
9 compute $(z_{m,2})_{m=M/2+1}^{M} = \mathrm{CausalSorting}((s_m)_{m=M/2+1}^{M}, (t_n)_{n=N/2+1}^{N}, (v_n)_{n=N/2+1}^{N})$
10 set $z_m = z_{m,1} + z_{m,2}$ for $m = M/2+1, ..., M$
11 **return** $\boldsymbol{z}$

In this setting, Algorithm 1 is not directly applicable, since every query now sees a different set of keys. Nevertheless, we can overcome this issue by a divide-and-conquer algorithm outlined in Algorithm 2. For simplicity, we assume that the sequence lengths M and N are powers of two. The main idea of the algorithm (for $M = N$) is that we split the masked kernel matrix into four blocks. Each diagonal block corresponds to a kernel sum with causal masking with sequence length $N/2$. The upper right off-diagonal block is zero (since the mask is zero). The lower left off-diagonal block is a standard kernel sum without masking and can be computed by Algorithm 1 (or the equivalent algorithm for the one dimensional Laplace kernel).

To verify the quasi linear complexity (simply assuming that $M = N = 2^K$), we note that for each length 2^k for $k = 1, \ldots, K$, we call 2^{K-k} times Algorithm 1 with sequence length 2^k. In particular, we have for each k the cost $\mathcal{O}(2^{K-k} 2^k \log(2^k)) = \mathcal{O}(2^K \log(2^k)) \leq \mathcal{O}(N \log(N))$. So in total, we obtain the complexity $\mathcal{O}(KN \log(N)) \leq \mathcal{O}(N \log^2(N))$. In effect, Algorithm 2 decomposes the triangular mask into $2N - 1$ blocks, see the final decomposition in Figure 3, which it traverses serially by the recursion. This order is not required, however: all blocks can be computed independently, since synchronization is only needed in line 10, which can be resolved by atomic additions accumulating into the output tensor.

C.5 Autoregressive decoding

During autoregressive decoding, tokens are generated one at a time. With causal attention, the keys and values of previous tokens do not change and are stored in a KV cache of size $\mathcal{O}(N(D+C))$. For each new query, softmax attention reads the full cache at cost $\mathcal{O}(N(D+C))$, so decoding is memory-bound rather than compute-bound. This procedure does not depend on the softmax structure and can be done for any kernel.

D Weighted Laplace Sums

For the Laplace kernel the difficult part is to compute $\sum_{n=1}^N \exp(-|s_m - t_n|) v_n$, where $s_m \in \mathbb{R}$ and $t_n \in \mathbb{R}$. Splitting at s_m as in (10) gives $e^{-s_m} \sum_{t_n \leq s_m} e^{t_n} v_n + e^{s_m} \sum_{t_n > s_m} e^{-t_n} v_n$, so the same prefix-sum idea applies. The individual factors $e^{\pm t_n}$ and $e^{\mp s_m}$, however, overflow in floating point as soon as the inputs leave a small range, even though their product is bounded by one. Algorithm 3 avoids this by expressing the prefix sums relative to the current node instead of the origin, which costs $\mathcal{O}((N+M)(\log_2 N + C))$ as well.

By induction, $L_n = \sum_{j=1}^n e^{-(t_{\sigma(n)} - t_{\sigma(j)})} v_{\sigma(j)}$ and $R_n = \sum_{j=n}^N e^{-(t_{\sigma(j)} - t_{\sigma(n)})} v_{\sigma(j)}$, so that with $t_{\sigma(n)} \leq s_m$ for $n \leq p_m$ and $t_{\sigma(n)} > s_m$ for $n > p_m$ the last line indeed equals z_m, where empty terms are read as zero.

Every exponent occurring in Algorithm 3 is non-positive: $\delta_n \geq 0$ by sorting, $s_m - t_{\sigma(p_m)} \geq 0$ and $t_{\sigma(p_m+1)} - s_m > 0$ by the definition of p_m. Hence, all exponentials lie in $(0, 1]$, and induction gives $|L_n^{(c)}| \leq \sum_{j \leq n} |v_{\sigma(j)}^{(c)}|$ and $|R_n^{(c)}| \leq \sum_{j \geq n} |v_{\sigma(j)}^{(c)}|$ for every channel c, so that $|z_m^{(c)}| \leq \sum_{n=1}^N |v_n^{(c)}|$.

Algorithm 3 Weighted Laplace sum

#	Step	Work	Memory
1	**Input** $\boldsymbol{s} \in \mathbb{R}^M$, $\boldsymbol{t} \in \mathbb{R}^N$ and $\boldsymbol{v} \in \mathbb{R}^{N \times C}$	-	$(C{+}1)N{+}M$
2	**Output** $z_m := \sum_{n=1}^{N} \exp(-\lvert s_m - t_n \rvert) v_n \in \mathbb{R}^C \quad m = 1, \dots, M$	-	MC
3	$\sigma \coloneqq \texttt{argsort}(\boldsymbol{t})$ s.t. $t_{\sigma(1)} \le \ldots \le t_{\sigma(N)}$	$N \log_2 N$	N
4	$p_m \coloneqq \max\big(\{n \colon t_{\sigma(n)} \le s_m\} \cup \{0\}\big)$ for $m = 1, \dots, M$	$M \log_2 N$	M
5	$\delta_n \coloneqq t_{\sigma(n)} - t_{\sigma(n-1)} \ge 0$ for $n = 2, \dots, N$	N	N
6	$L_n \coloneqq e^{-\delta_n} L_{n-1} + v_{\sigma(n)} \in \mathbb{R}^C$ for $n = 1, \dots, N$, $L_0 \coloneqq \mathbf{0}$	$2NC$	$(N{+}1)C$
7	$R_n \coloneqq e^{-\delta_{n+1}} R_{n+1} + v_{\sigma(n)} \in \mathbb{R}^C$ for $n = N, \dots, 1$, $R_{N+1} \coloneqq \mathbf{0}$	$2NC$	$(N{+}1)C$
8	$z_m := e^{-(s_m - t_{\sigma(p_m)})} L_{p_m} + e^{-(t_{\sigma(p_m+1)} - s_m)} R_{p_m+1} \in \mathbb{R}^C$ for $m = 1, \dots, M$	MC	MC
9	**return** $\boldsymbol{z}$		

All intermediate quantities are thus bounded by the same quantity as the exact sum itself, independently of the magnitude of $\boldsymbol{s}$ and $\boldsymbol{t}$.

In `weighted_laplace_sum` we provide a parallel implementation following the structure of the `weighted_abs_sum`. Gradients and masking are out of scope.

E Vision Transformer

We train a small vision transformer (Dosovitskiy et al., 2021) for classification on CIFAR-10: 12 layers, 3 heads of dimension $D = 64$, hidden dimension 192, patch size 4, hence sequence length 64. Since the sequence length does not exceed the head dimension, even a 64-dimensional RKHS suffices to separate all tokens and the capacity mechanism of the previous section cannot bind. This experiment is therefore deliberately a controlled sanity check: any remaining quality differences must come from properties other than dimension.

Each model is trained once from *scratch* with random initialization, as well as being distilled from a softmax teacher model (96.57% accuracy) in three stages: The student shares the architecture of the teacher and differs only in the kernel, so that stage *copy* simply utilizes the teacher weights. *Layer-by-layer* then finetunes one student attention module at a time to minimize the MSE to the teacher's attention output at that layer, and *end-to-end* finally finetunes all parameters against the teacher logits with temperature-scaled KL. This distillation setup is identical to the text transformer with the single difference of using the KL divergence instead of the MSE loss for *end-to-end*.

Training from scratch uses AdamW, lr $10^{-3} \to 10^{-5}$ cosine annealing with 100 warmup epochs, weight decay 0.1, 1000 epochs, batch size 128, RandAugment, MixUp/CutMix and label smoothing 0.1. In both the vision and text transformers, the layer-by-layer stage runs 10k steps per layer. The end-to-end stage runs 50k steps with KL temperature $T = 4$ for the vision transformer, and 10k steps with the MSE loss for the text transformer (Section 4.3).

Table 6: CIFAR-10 test accuracy (%). Training from scratch as well as piecewise distillation.

			distillation from softmax		
Kernel	scale τ	scratch ↑	copy ↑	+layerwise ↑	+end-to-end ↑
`softmax` (teacher)	2.828	96.57	—	—	—
`gauss`	2.828	96.70	96.10	96.56	96.55
`laplace`	6.0	96.71	94.92	96.57	96.64
`riesz`	1.0	94.14	58.12	95.61	95.83
`add_riesz`	1.0	93.68	55.93	95.84	96.21
`add_laplace`	0.5	93.94	43.67	95.96	96.40
`add_bump`	1.5	94.56	43.33	96.13	96.26
`tri`	2.828	94.63	39.47	96.01	96.13
`relu`	1.0	95.08	62.40	96.12	95.91
`elu`	1.0	95.17	43.17	96.17	96.35
`dpfp`	1.0	95.15	75.84	96.39	96.41

Table 6 supports three conclusions. First, as predicted, no capacity gap appears: all kernels train to within 1.0% accuracy of softmax after full distillation. Second, the softmax–Gauss correspondence (32) is not merely formal, as the *copy* stage Gauss student already reaches 96.10%, so the reweighting argument holds numerically, not just algebraically. Interestingly, `laplace` with $\tau = 6$ also reaches 94.92% without any finetuning, while all other kernels lie at least 19% below. Third, distillation matches or improves training from scratch for all kernels, in both accuracy and training time; the layer-by-layer stage does most of the work and end-to-end adds less than 1%.

F Associative recall

Schlag et al. (2021, Sec. 6.1 & 6.2) introduce associative recall in order to verify the capacity limits. They start with a set of tokens $\mathcal{N} \coloneqq \{1, \dots, N\}$ and a learnable embedding $E \colon \mathcal{N} \to \mathbb{R}^D$. Two random permutations σ, ξ of $\mathcal{N}$ are sampled and the sequence of embedded tokens $(u_1, \dots, u_N)$ with $u_n = E(\sigma(n))$ is built. To each token u_n a one-hot value $v_n = e_{\xi(n)} \in \mathbb{R}^N$ is assigned. The keys are computed via $k_n = W_K[u_n; v_n]$ with a learnable $W_K \in \mathbb{R}^{D \times (D+N)}$. To recall the value of token $m \in \mathcal{N}$, a query $q_m = W_Q E(m)$ with learnable $W_Q \in \mathbb{R}^{D \times D}$ is built and the value is retrieved as $\hat{v}^{(m)} = \sum_{n=1}^N A_{m,n}(\boldsymbol{q}, \boldsymbol{k})\, v_n \in \mathbb{R}^N$ with $\boldsymbol{k} = (k_n)_{n=1}^N$, $\boldsymbol{q} = (q_m)_{m=1}^N$ and A from (2). The loss is

$$\ell(\hat{v}^{(m)}, v^{\star(m)}) = \frac{1}{2} \sum_{j=1}^N (\hat{v}_j^{(m)} - v_j^{\star(m)})^2 = \frac{1}{2} \|\hat{v}^{(m)} - v^{\star(m)}\|_2^2, \tag{31}$$

where $v^{\star(m)}$ is the value assigned to token m, and it is evaluated for all $m \in \mathcal{N}$. The loss is (up to a factor of $1/2$) exactly the inner term of (13), so that Lemma 11 also yields the form with the Frobenius norm. We differ only in the key matrix: we use $W_K \in \mathbb{R}^{D \times D}$ and $k_n = W_K u_n$, as in the standard transformer. Our setting is contained in that of Schlag et al. (2021) by extending W_K with a zero matrix $0 \in \mathbb{R}^{D \times N}$.

G Proofs

Proof of Theorem 1. By assumption, there exists some $\mathcal{C}_0$ function F with $\Phi(q, k) = F(q - k)$. Now, for arbitrary $N \in \mathbb{N}$, let $(x_n)_{n=1}^N$ be a collection of points with $\varepsilon \coloneqq \min_{n \neq m} \|x_n - x_m\|_2^2 > 0$. Then, we have by definition for all $r > 0$ with $k_n = q_n = r x_n$ that the Gram matrix G fulfills $G_{n,n} = F(0)$ and $G_{m,n} = F(r(x_m - x_n)) \to 0$ as $r \to \infty$ since $r\|x_m - x_n\| \to \infty$. This yields that $A_{n,n} \to 1$ and $A_{m,n} \to 0$ for all $m \neq n$ such that $\operatorname{Cap}(\Phi) \geq N$ for all $N \in \mathbb{N}$ and thus $\operatorname{Cap}(\Phi) = \infty$. While many transformers normalize the tokens x_n, the keys and queries are obtained via $q_n = W_Q x_n$ and $k_n = W_K x_n$, where $W_Q, W_K \in \mathbb{R}^{D \times D}$ are weight matrices. Choosing $W_Q = W_K = r \operatorname{Id}_D$ recovers $q_n = k_n = r x_n$ from the proof. □

Proof of Proposition 2. For the Gauss kernel $\Psi(q, k) = \exp(-\frac{1}{2}\|q - k\|_2^2)$, we have that $\Phi(q, k) = \Psi(q, k) \exp(\frac{1}{2}(\|q\|_2^2 + \|k\|_2^2))$. In particular, if $\|k_1\|_2 = \dots = \|k_N\|_2$, the attention matrices $A(\boldsymbol{q}, \boldsymbol{k})$ with respect to Φ and Ψ coincide, since

$$A_\Phi(\boldsymbol{q}, \boldsymbol{k}) = \frac{\mathrm{e}^{q_m^\top k_n}}{\sum_{l=1}^N \mathrm{e}^{q_m^\top k_l}} = \frac{\mathrm{e}^{\frac{1}{2}(\|k_1\|_2^2 + \|q_m\|_2^2)} \mathrm{e}^{-\frac{1}{2}\|q_m - k_n\|_2^2}}{\mathrm{e}^{\frac{1}{2}(\|k_1\|_2^2 + \|q_m\|_2^2)} \sum_{l=1}^N \mathrm{e}^{-\frac{1}{2}\|q_m - k_l\|_2^2}} = A_\Psi(\boldsymbol{q}, \boldsymbol{k}). \tag{32}$$

Thus, we obtain by the same proof as for Theorem 1 that $\operatorname{Cap}(\Phi) = \infty$, provided that we choose $\|x_m\| = 1$ for all m in that proof, which is possible because the sphere $\{x \in \mathbb{R}^D \colon \|x\|_2 = 1\}$ for $D \geq 2$ has infinitely many points. □

Proof of Proposition 5. As Φ is continuous and stationary there is continuous $F \colon \mathbb{R}^D \to \mathbb{R}$ with $\Phi(q, k) = F(q - k)$. Since Φ has a FFM expansion $F(q - k) = \Phi(q, k) = \varphi(q)^\top \varphi(k)$ we know that F is spd and has RKHS $\mathcal{H}_\Phi$. By Bochner's theorem, there is a real nonnegative Borel measure $\xi \in \mathcal{M}$ such that $F(x) = \mathcal{F}[\xi](x) = \int_{\mathbb{R}^D} \mathrm{e}^{-\mathrm{i}\omega^\top x} \,\mathrm{d}\xi(\omega)$. Since F is real symmetric ξ, is symmetric too and $F(x) = \int_{\mathbb{R}^D} \cos(\omega^\top x) \,\mathrm{d}\xi(\omega)$. A feature space of $\mathcal{H}_\Phi$ is given by $H_0 \coloneqq E \oplus O$, where E and O denote the even and odd function classes of $L^2(\xi)$. A feature map of Φ is $\Phi_0(x)(\omega) \coloneqq$

$[\cos(\omega^\top x), \sin(\omega^\top x)]$, because

$$\langle \Phi_0(q), \Phi_0(k) \rangle_{H_0} = \int_{\mathbb{R}^D} \cos(\omega^\top (q-k)) \,\mathrm{d}\xi(\omega) = F(q-k) = \Phi(q,k). \tag{33}$$

By Steinwart & Christmann (2008, Thm. 4.21) the map $V\colon H_0 \to \mathcal{H}_\Phi$ given by $V(h) \coloneqq \langle h, \Phi_0 \rangle$ is surjective. For $h = [h_E, h_O]^\top \in H_0$ with $h_E \in E$ and $h_O \in O$, this operator is given by

$$\begin{aligned} V(h)(x) = \langle h, \Phi_0(x) \rangle &= \int_{\mathbb{R}^D} h_E(\omega) \cos(\omega^\top x) + h_O \sin(\omega^\top x) \,\mathrm{d}\xi(\omega) \\ &= \int_{\mathbb{R}^D} \mathrm{e}^{-\mathrm{i}\omega^\top x} (h_E + \mathrm{i} h_O) \,\mathrm{d}\xi(\omega) = \mathcal{F}[(h_E + \mathrm{i}h_O)\xi](x). \end{aligned} \tag{34}$$

As the Fourier transformation is injective $V(h) = 0$ implies $(h_E + \mathrm{i}h_O)\xi = 0$ and thus $h_E\xi = 0$ and $h_O\xi = 0$. Therefore, h_E, h_O vanish ξ-almost everywhere and V is injective. Since Φ has FFM $\varphi\colon \mathbb{R}^D \to \mathbb{R}^{2L}$, it holds $\dim \mathcal{H}_\Phi \leq 2L < \infty$. Since V is a linear bijection, H_0 is finite dimensional, which renders $L^2(\xi) \cong E \oplus O = H_0$ to be finite dimensional. This implies $\operatorname{supp}\xi$ to be a finite set and hence $\xi = \sum_{l=1}^L \alpha_l/2(\delta_{\omega_l} + \delta_{-\omega_l})$ for $\alpha_l \geq 0$ and $\omega_l \in \mathbb{R}^D$. By Bochner's theorem, we thus obtain

$$\Phi(q,k) = F(q-k) = \int_{\mathbb{R}^D} \cos(\omega^\top (q-k)) \,\mathrm{d}\xi(\omega) = \sum_{l=1}^L \alpha_l \cos(\omega_l^\top (q-k)). \tag{35}$$

The function F is a trigonometric polynomial, hence almost periodic. Its mean value satisfies $M\{F^2\} \geq \frac{1}{2}\sum_{l=1}^L \alpha_l^2 > 0$ unless $F = 0$, by the orthogonality of characters (Shubin, 1978, Prop. 1.5). Since $F \in \mathcal{C}_0$ implies $M\{F^2\} = 0$ (Shubin, 1978, Prop. 1.8) we conclude $F = 0$. □

Proof of Proposition 6. Any continuous piecewise linear function $f\colon \mathbb{R} \to \mathbb{R}$ with finitely many nodes can be expressed in terms of absolute values (Streubel et al., 2014, p. 328) as

$$f(x) = a + bx + \sum_{j=1}^J c_j |x - x_j|, \qquad x \in \mathbb{R}. \tag{36}$$

Let $s_1, \ldots, s_M \in \mathbb{R}$, $t_1, \ldots, t_N \in \mathbb{R}$ and $v_1, \ldots, v_N \in \mathbb{R}^C$. Inserting (36) into the kernel sum yields

$$\sum_{n=1}^N f(s_m - t_n) v_n = \underbrace{(a + bs_m) \sum_{n=1}^N v_n - b \sum_{n=1}^N t_n v_n}_{\alpha_m^{(0)}} + \sum_{j=1}^J c_j \underbrace{\sum_{n=1}^N |s_m - (x_j + t_n)| v_n}_{\alpha_m^{(j)}}. \tag{37}$$

The constant and linear part $(\alpha_m^{(0)})_{m=1}^M$ can be computed in $\mathcal{O}(C(N+M))$ operations. For each j, we shift the keys to $t_n^{(j)} \coloneqq t_n + x_j$ and compute $(\alpha_m^{(j)})_{m=1}^M$ by the fast summation of the distance kernel applied to $(s_m)_{m=1}^M$, $(t_n^{(j)})_{n=1}^N$ and $(v_n)_{n=1}^N$. Each call costs $\mathcal{O}(C(N+M)\log N)$ operations, which results in a total of $\mathcal{O}(CJ(N+M)\log N)$. Note that the sorting of $\boldsymbol{t}^{(j)}$ is the same as that of $\boldsymbol{t}$ and does not need to be repeated for each node. Further, $\boldsymbol{s}$ can be sorted once to avoid the repeated bisection. Therefore, the complexity can be improved to $\mathcal{O}((N+M)(\log(N+M) + CJ))$.

Note that the statement extends to the noncontinuous case: every piecewise linear function with finitely many nodes decomposes into a continuous piecewise linear function and a piecewise constant function. The latter only requires the evaluation of $\sum_{n=1}^N \operatorname{sign}(s_m - t_n) v_n$ which can be solved in quasi linear time as explained in (25). □

Proof of Proposition 8. For the Riesz kernel let $N \geq 3$ and $s_n, t_n \in \mathbb{R}$ be arbitrary points with $n \in \mathcal{N} = \{1, \ldots, N\}$. Further, let $\varepsilon > 0$ and $A \coloneqq A_{r_\varepsilon}(\boldsymbol{s}, \boldsymbol{t})$ denote the attention matrix of r_ε given $\boldsymbol{s}$ and $\boldsymbol{t}$. Fix $n_+ \in \arg\max_{n\in\mathcal{N}} t_n$ and $n_- \in \arg\min_{n\in\mathcal{N}} t_n$. The map $t \mapsto r_\varepsilon(s_n, t) = |s_n| + |t| - |s_n - t| + \varepsilon$ is monotone for all $n \in \mathcal{N}$, hence the maximum is attained at $t_{n^\star}$ where $n^\star \in \{n_+, n_-\}$. For all $n \in \mathcal{N} \setminus \{n_+, n_-\}$ it holds $A_{n,n^\star} \geq A_{n,n} \geq 0$ and thus

$$\|A_{n,:} - e_n\|_2^2 \geq (A_{n,n} - 1)^2 + A_{n,n^\star}^2 \geq (1 - A_{n,n})^2 + A_{n,n}^2 \geq \tfrac{1}{2}. \tag{38}$$

Since $\#(\mathcal{N} \setminus \{n_+, n_-\}) \geq N - 2$ we have $\sum_{n\in\mathcal{N}} \|A_{n,:} - e_n\|_2^2 \geq \frac{N-2}{2}$. For $N = 2$ we can choose $s_1 = t_1 = \tau$ and $s_2 = t_2 = -\tau$. Then it holds $A \to \mathrm{Id}_2$ as $\tau \to \infty$, so the capacity is exactly 2. □

Proof of Theorem 9. To distinguish the Gram and attention matrices with respect to Φ and ϕ, we denote them by G_Φ and G_ϕ or A_Φ and A_ϕ, respectively. Now let $N \leq \mathrm{Cap}(\phi)$ and $\varepsilon > 0$. Then, we know that there exist $\boldsymbol{s}, \boldsymbol{t} \in \mathbb{R}^N$ such that $\|A_\phi(\boldsymbol{s}, \boldsymbol{t}) - \mathrm{Id}_N\|_F^2 < \varepsilon$. Define $\boldsymbol{q}, \boldsymbol{k} \in (\mathbb{R}^D)^N$ by $q_{m,d} = s_m$ and $k_{n,d} = t_n$. Then, we obtain by definition that $G_\Phi(\boldsymbol{q}, \boldsymbol{k}) = DG_\phi(\boldsymbol{s}, \boldsymbol{t})$ and therefore $A_\Phi(\boldsymbol{q}, \boldsymbol{k}) = A_\phi(\boldsymbol{s}, \boldsymbol{t})$. In particular, we have $\|A_\Phi(\boldsymbol{q}, \boldsymbol{k}) - \mathrm{Id}_N\|_F^2 = \|A_\phi(\boldsymbol{s}, \boldsymbol{t}) - \mathrm{Id}_N\|_F^2 < \varepsilon$. Since this works for any $\varepsilon > 0$, we obtain that $\inf_{\boldsymbol{q},\boldsymbol{k}} \|A_\Phi(\boldsymbol{q}, \boldsymbol{k}) - \mathrm{Id}_N\|_F^2 = 0$ and $\mathrm{Cap}(\Phi) \geq N$.

Let $\phi = r_\varepsilon$ be the Riesz kernel. For arbitrary D and $N = 2D$, we can choose $q_n = k_n = \tau e_n$ for $n = 1, \ldots, D$ and $q_n = k_n = -\tau e_{n-D}$ for $n = D+1, \ldots, 2D$. Then it holds for $n \neq m$, that $G_{m,n}(\boldsymbol{q}, \boldsymbol{k}) = D\varepsilon$ and for $n = m$, $G_{n,n}(\boldsymbol{q}, \boldsymbol{k}) = D\varepsilon + 2\tau$. Hence, we get $A(\boldsymbol{q}, \boldsymbol{k}) \to \mathrm{Id}_N$ as $\tau \to \infty$, so the capacity is at least $2D$.

Symmetry and translation invariance clearly carry over from ϕ to Φ. If ϕ is positive definite, then for all $\alpha_1, \ldots, \alpha_N \in \mathbb{R}$ and $x_1, \ldots, x_N \in \mathbb{R}^D$ it holds

$$\sum_{n,m=1}^{N} \alpha_n \alpha_m \Phi(x_n, x_m) = \sum_{d=1}^{D} \sum_{n,m=1}^{N} \alpha_n \alpha_m \phi(x_{n,d}, x_{m,d}) \geq 0, \tag{39}$$

so Φ is positive definite. Now, let ϕ be quasi linear and let $q_1, \ldots, q_M \in \mathbb{R}^D$, $k_1, \ldots, k_N \in \mathbb{R}^D$ and $v_1, \ldots, v_N \in \mathbb{R}^C$. Rearranging the kernel sum yields

$$z_m = \sum_{n=1}^{N} \Phi(q_m, k_n) v_n = \sum_{d=1}^{D} \sum_{n=1}^{N} \phi(q_{m,d}, k_{n,d}) v_n. \tag{40}$$

Since ϕ is quasi linear, each of the D inner sums can be computed in quasi linear time for all $m = 1, \ldots, M$ simultaneously. Summing the results over d costs another $\mathcal{O}(MDC)$ operations, hence $(z_m)_{m=1}^M$ is computed in quasi linear time for every fixed D. □

Proof of Lemma 11. Since the normalization in (2) sums over all keys, we have $A_{m,l}(\boldsymbol{q}, \boldsymbol{k}_\sigma) = A_{m,\sigma(l)}(\boldsymbol{q}, \boldsymbol{k})$. With $A \coloneqq A(\boldsymbol{q}, \boldsymbol{k})$ this gives $y_{\sigma(j)} = \sum_l A_{\sigma(j),\sigma(l)}\, e_{\xi(l)}$, and thus

$$\begin{aligned}
\sum_{j=1}^{N} \|e_{\xi(j)} - y_{\sigma(j)}\|_2^2 &= \sum_{j=1}^{N} \Big\| \sum_{l=1}^{N} \big(\delta_{j,l} - A_{\sigma(j),\sigma(l)}\big) e_{\xi(l)} \Big\|_2^2 \\
&= \sum_{j=1}^{N} \sum_{l=1}^{N} \big(\delta_{\sigma(j),\sigma(l)} - A_{\sigma(j),\sigma(l)}\big)^2 \\
&= \sum_{m=1}^{N} \sum_{n=1}^{N} \big(\delta_{m,n} - A_{m,n}\big)^2 = \|A - \mathrm{Id}_N\|_F^2.
\end{aligned} \tag{41}$$

The second equality uses that $(e_{\xi(l)})_{l=1}^N$ is an orthonormal basis, and the third substitutes $n = \sigma(l)$ and $m = \sigma(j)$. □