Article

Not peer-reviewed version

# A Unified Perspective on Efficient Attention: Generalized Memory and Kernel Function Selection in Transformers

Jiyong Ma *

Posted Date: 7 November 2025

doi: 10.20944/preprints202511.0515.v1

Keywords: scaled-dot-product attention (SDPA); kernel function; memory model

 doi:10.20944/preprints202511.0515.v1



*Article*

# A Unified Perspective on Efficient Attention: Generalized Memory and Kernel Function Selection in Transformers

**Jiyong Ma**

Oracle Corporation, United States Minor Outlying Islands; jiyongma06@gmail.com

**Abstract**

This article introduces a new theoretical perspective on the Scaled-Dot-Product Attention (SDPA) in transformers by connecting it to distributed memory theory. We propose that SDPA is an extension of distributed memory and reframe it as a Generalized Memory Model, a mechanism for learning multi-way associations that expands upon classical frameworks. Our experimental findings validate this perspective and yield practical methods for building more efficient attention mechanisms. We demonstrate that model convergence is maintained even when query and key vectors are identical, a modification that halves their memory requirement and offers a significant efficiency gain for long sequences. Furthermore, our comprehensive kernel analysis shows that all tested kernels, including a simple linear kernel, provide a path to convergence, establishing a strong baseline for attention approximation. While the Radial Basis Function (RBF) kernel offers marginal improvements, we also introduce several kernels that successfully converge with normalized weights. Collectively, this work provides both a unifying theoretical lens for transformer attention and a practical guide to kernel selection for developing more robust and efficient models.

**Keywords:** scaled-dot-product attention (SDPA); kernel function; memory model

## 1. Introduction

Early research by James A. Anderson [1] and Leon N. Cooper [2] established foundational concepts for neural network-based memory models. They proposed that memories are not stored in specific locations but are distributed across a network of interconnected neurons. In his 1985 paper "Distributed Memory" [3] Leon N. Cooper proposed a mathematical model explaining how memory remains stable even as brain cells die. He argued that memories are not stored in individual neurons but are distributed across the network of connections (synapses) between them, much like a hologram encodes an image across its entire surface. This distributed system acts as a "content-addressable" memory, allowing a complete memory to be recalled from a partial cue. The model uses a matrix of synaptic strengths that are modified based on neural activity. Cooper also suggested this framework could explain both short-term memory (via temporary synaptic changes) and long-term memory (via permanent ones) within a single, unified system.

While James Anderson and Leon Cooper's early models established that memory could be distributed across a network, John Hopfield's 1982 work introduced a crucial difference: dynamics and error correction. Hopfield introduced a recurrent, nonlinear network where stored memories are stable states in an "energy landscape" [4] When presented with a partial or noisy cue, the network dynamically evolves, like a ball rolling downhill, until it settles into the nearest complete, stored memory. This iterative process makes Hopfield networks a true content-addressable memory system, capable of robust pattern completion and error correction, a significant advance over the earlier linear models.

The Transformer architecture, introduced in the groundbreaking 2017 paper [5] "Attention Is All You Need," revolutionized natural language processing. Its core innovation was replacing traditional

 doi:10.20944/preprints202511.0515.v1



recurrent and convolutional layers with a mechanism based on self-attention. This design choice was pivotal: by removing the inherently sequential computations of recurrent models, the Transformer could be massively parallelized, dramatically reducing training times on modern hardware. This parallel self-attention mechanism enables the model to directly assess the importance of all words in an input sequence relative to each other, making it exceptionally effective at capturing complex context and long-range dependencies.

## 2. Scaled-Dot-Product Attention in Transformer

In this discussion, we focus on the self-attention mechanism [5], some mathematical symbols are adopted from [6].

We assume that $q$, $\boldsymbol{v}$, $k$ are d-dimensional random vectors respectively, where $q$ is a query vector, $\boldsymbol{v}$ is a value vector, and $k$ is a key vector. The vectors $q$ and $k$ reside in the same d-dimensional space $\boldsymbol{R}^d$ respectively, while the vector $\boldsymbol{v}$ may reside in a different d-dimensional space.

The vectors $q$ and $\boldsymbol{v}$ are projections of a high-dimensional input vector $\boldsymbol{x}$, which lies in m-dimensional space $\boldsymbol{R}^m$. The dimension $m$ is much larger than the dimension $d$. The space $\boldsymbol{R}^d$ is a d-dimensional subspace within the larger space $\boldsymbol{R}^m$.

For simplicity, we will drop the dependent variable $\boldsymbol{x}$ in vectors $\boldsymbol{v}$ and $q$ in the following discussion.

Let $T$ denote the sequence length. We denote a sequence of query vectors as $\{\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3}, \ldots, \boldsymbol{q_T}\}$. Similarly, we denote a sequence of value vectors as $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \ldots, \boldsymbol{v_T}\}$.

The vectors $\boldsymbol{q_i}$ and $\boldsymbol{v_i}$ are two low-dimensional projections of an input vector $\boldsymbol{x_i}$. The vector $\boldsymbol{x_i}$ lives in m-dimensional space $\boldsymbol{R}^m$.

We also have a sequence of key vectors denoted as $\{\boldsymbol{k_1}, \boldsymbol{k_2}, \boldsymbol{k_3}, \ldots, \boldsymbol{k_T}\}$. Each key vector $\boldsymbol{k_i}$ represents an address memory unit, which is also a low-dimensional projection of the input vector $\boldsymbol{x_i}$. Each key vector $\boldsymbol{k_i}$ can also be viewed as a hidden state associated a value vector $\boldsymbol{v_i}$ which represents a content or feature memory unit.

A query vector $q$ will match each address memory unit $\boldsymbol{k_i}$ to compute a similarity score. The similarity score can be represented as the cross-correlation value, which is the inner product of the two vectors $q$ and $\boldsymbol{k_i}$ denoted as $\boldsymbol{q^t k_i}$. Note that the vector $q$ is one of vectors in the sequence $\{\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3}, \ldots, \boldsymbol{q_T}\}$.

This self-attention mechanism allows the model to dynamically attend to different parts of the input sequence when computing the output value vector.

The definition of the softmax weighting function is described in [6]:

$$w_i(\boldsymbol{q}) = \frac{exp(\alpha \boldsymbol{q^t k}_i)}{\sum_{j=1}^{T} exp(\alpha \boldsymbol{q^t k}_j)} \tag{1}$$

Where:

- $\alpha$ is a positive scaling factor.
- $\boldsymbol{t}$ is a transpose operator, which transforms a column vector to a row vector.
- $\boldsymbol{q^t k_i}$ or the dot product denoted as $<q, \boldsymbol{k_i}>$ is the inner product between the query vector $\boldsymbol{q}$ and i-th key vector $\boldsymbol{k_i}$.
- The denominator $\sum_{j=1}^{T} exp(\alpha \boldsymbol{q^t k}_j)$ is the sum of the exponentials of all the inner products between the query $q$ and the set of key vectors $\{\boldsymbol{k_1}, \boldsymbol{k_2}, \boldsymbol{k_3}, \ldots, \boldsymbol{k_T}\}$.
- $w_i(\boldsymbol{q})$ is the softmax weight corresponding to the i-th key vector $\boldsymbol{k_i}$

The context vector $\hat{\boldsymbol{v}}$ is estimated by weighted average of all vectors $\boldsymbol{v_i}$:

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{T} w_{\mathrm{i}}(\boldsymbol{q}) \boldsymbol{v}_{\mathbf{i}} \tag{2}$$

 doi:10.20944/preprints202511.0515.v1



## 3. Tensor Representation of Distributed Memory

Leon N. Cooper's distributed memory model [3] provided a framework, demonstrating how associative memory could be encoded in a synaptic matrix ($\boldsymbol{A}$). This 2nd-order tensor (a matrix) was mathematically elegant and biologically plausible for storing pairwise associations: a specific input pattern ($\boldsymbol{f}$) is linked to a corresponding output pattern ($\boldsymbol{g}$). The memory is built by summing the outer products of these vector pairs. Retrieval is then a simple matrix-vector multiplication, where an input cue probes the memory to recall its associated output.

### *3.1. Pairwise Associations*

Cooper's Model: Stores associations between two vectors, ($\boldsymbol{f}$, $\boldsymbol{g}$). The memory is a 2nd-order tensor (matrix $\boldsymbol{A}$).

Cooper's Model: The memory matrix is constructed by summing outer products: $\boldsymbol{A} = \Sigma\, \boldsymbol{g} \otimes \boldsymbol{f}$.

The outer product $\boldsymbol{g} \otimes \boldsymbol{f}$ multiplies two vectors to create a matrix. Each element ($i$, $j$) of the resulting matrix is the product of the $i$-th element of column vector $\boldsymbol{g}$ and the j-th element of row vector $\boldsymbol{f}$

Cooper's Model: Retrieval of $\boldsymbol{h}$ is done by multiplying the matrix $\boldsymbol{A}$ with an input vector $\boldsymbol{q}$: $\boldsymbol{h} = \boldsymbol{A} * \boldsymbol{q}$.

### *3.2. Extending Cooper's Distributed Memory for Multi-Way Associations*

We begin by defining the total memory, $\boldsymbol{M}$, analogous to $\boldsymbol{A}$, as a sequence of memory states or operators over time:

$$\boldsymbol{M} = (\boldsymbol{M}(1), \boldsymbol{M}(2), \ldots, \boldsymbol{M}(T)) \tag{3}$$

Here, each $\boldsymbol{M}(i)$ represents the complete memory operator at a specific time step $i$. This formulation introduces a dynamic aspect, allowing the memory to evolve.

Following Cooper's representation, each memory operator can be composed of two distinct components:

$$\boldsymbol{M}(i) = (\boldsymbol{L}(i), \mathbf{S}(i)) \tag{4}$$

Where $\boldsymbol{L}(i)$ is a long-term memory, which represents the stable, context-independent, and foundational knowledge of the unit $i$. It is the a priori semantic identity or pattern that the system has learned and stored over time. This memory may be static and may not change based on the immediate context. It answers the question: "What is this concept in isolation?"

Where $\mathbf{S}(i)$ is a short-term memory, which represents the transient, context-dependent understanding of the item, which emerges from its interaction with its neighbors and its position in the sequence.

### *3.3. The Key-Value Representation of the Memory Operator*

To make this model computationally concrete, we adopt the key-value structure from the attention mechanism. The memory operator $\boldsymbol{M}(i)$ at any given time is represented as a pair:

$$\boldsymbol{M}(i) = (\boldsymbol{v_i}, \boldsymbol{k_i}) \tag{5}$$

The key vector $\boldsymbol{k_i}$ serves as the content-addressable index of the memory. It is the pattern to be matched against. In the context of our dual-memory model, the key can be a function of both long-term knowledge $\boldsymbol{L}(i)$ and short-term context $\mathbf{S}(i)$. It answers the question, "What information do I represent?"

 doi:10.20944/preprints202511.0515.v1



The value vector $\boldsymbol{v_i}$ is the content to be retrieved. It holds the actual information or signal associated with its corresponding key. It answers the question, "What information should I provide if I am selected?"

This key-value pairing is a powerful evolution of Cooper's model. Instead of a monolithic matrix $\boldsymbol{A}$ storing all associations implicitly, the memory is now a structured set of (content, address) pairs, making the retrieval process more explicit and flexible.

*3.4. Associative Retrieval via Query*

Memory retrieval is no longer a simple matrix multiplication but an interactive, query-based process. A query vector $\boldsymbol{q}$ represents the current focus of attention or the information need of the system. The interaction between the memory and the query $\boldsymbol{q}$ is defined by the triplet:

$$(\boldsymbol{M}(i), \boldsymbol{q}) = (\boldsymbol{v_i}, \boldsymbol{k_i}, \boldsymbol{q}) \tag{6}$$

This interaction is governed by an operator $\boldsymbol{F}$, which takes the full memory $\boldsymbol{M}$ and the query $\boldsymbol{q}$ to produce a final output context vector $\hat{\boldsymbol{v}}$.

The retrieval operator $\boldsymbol{R}$ acts $(\boldsymbol{M}, \boldsymbol{q})$ is to estimate the context vector $\hat{\boldsymbol{v}}$:

$$\hat{\boldsymbol{v}} = \boldsymbol{R}(\boldsymbol{M}, \boldsymbol{q}) \tag{7}$$

The standard and most powerful way to define this operator is through a weighted sum, where the weights $G(\boldsymbol{k_j}, \boldsymbol{q})$ are determined by the similarity between the query and each key:

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{T} \boldsymbol{v_i}\, G(\boldsymbol{k_i}, \boldsymbol{q}) \tag{8a}$$

Or with normalized term as:

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{T} \boldsymbol{v_i}\, G(\boldsymbol{k_i}, \boldsymbol{q}) / \sum_{j=1}^{T} G(\boldsymbol{k_j}, \boldsymbol{q}) \tag{8b}$$

This formula is the essence of the attention mechanism. The function $G(\boldsymbol{k_j}, \boldsymbol{q})$ is a kernel that measures the similarity. One form kernel is the exponential function, which makes it a universal kernel, capable of learning any continuous similarity function: $G(\boldsymbol{k_j}, \boldsymbol{q}) = exp(\alpha \boldsymbol{q^t k_i}) = exp(\alpha \boldsymbol{k_i^t q})$

This triplet forms the fundamental computational unit of the associative retrieval process, mirroring the Scaled Dot-Product Attention mechanism:

**Matching**: The query $\boldsymbol{q}$ is compared against every $\boldsymbol{k_i}$ in the memory sequence $\boldsymbol{M}$. This is typically performed using a dot product, the inner product value $(\boldsymbol{k_i^t}\, \boldsymbol{q})$ measures the similarity or "resonance" between the current need and the memory's address.

**Weighting**: The raw similarity scores are normalized (e.g., via a softmax function) to produce an attention distribution, $exp(\alpha \boldsymbol{q^t k_i})$. These weights represent the relevance of each memory item $\boldsymbol{M}(i)$ to the query.

**Retrieval**: The final output is a context vector of weighted sum of all value vectors defined as Eq. 1 or Eq. 8a.

While powerful, this exponential kernel is computationally expensive compared with pure linear kernel. We can explore different simplification of kernels.

 doi:10.20944/preprints202511.0515.v1



## 4. The Linear Kernel: A Connection to RNNs

In the linear case, the retrieval operator $\boldsymbol{R}$ simplifies significantly. Instead of a complex, query-dependent weighting, the operation becomes a direct matrix-vector multiplication.

$$\boldsymbol{R}(\boldsymbol{M}, \boldsymbol{q}) = \boldsymbol{H}(\boldsymbol{M})\boldsymbol{q} \tag{9}$$

Here, the term $\boldsymbol{H}(\boldsymbol{M})$ represents a single, collective-hidden memory matrix that is constructed from the entire sequence:

$$\boldsymbol{H}(\boldsymbol{M}) = \sum_{i=1}^{T} \boldsymbol{v}_{\mathbf{i}}\, \boldsymbol{k}_i^{\boldsymbol{t}} \tag{10}$$

This formulation is a direct echo of Cooper's distributed memory theory, defined as the equation: $A = \Sigma\, g \otimes f$, where $g$ is value vector, $f$ is key vector. It forges a single memory matrix by summing the outer products of all individual memories (key-value pairs). The system stores a superimposed representation of all information in this one matrix. Retrieval is then performed by probing this matrix with the query. Substitute the Eq. 10 into Eq. 9, we have the following equation:

$$\boldsymbol{R}(\boldsymbol{M}, \boldsymbol{q}) = \boldsymbol{H}(\boldsymbol{M})\boldsymbol{q} = \sum_{i=1}^{T} \boldsymbol{v}_{\mathbf{i}}\, (\boldsymbol{k}_i^{\boldsymbol{t}}\, \boldsymbol{q}) \tag{11}$$

The final expression shows that this is equivalent to a weighted sum of the value vectors, where the weight is simply the raw dot product (or inner product) $<q, \boldsymbol{k_i}>$ between the query and each key denoted as $\boldsymbol{q^t k_i}$. For linear kernel $G(\boldsymbol{k_i}, \boldsymbol{q}) = \boldsymbol{q^t k}_i$, normalizing the output yields:

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{T} \boldsymbol{v}_{\mathbf{i}}\, (\boldsymbol{k}_i^{\boldsymbol{t}}\, \boldsymbol{q}) / \sum_{j=1}^{T} (\boldsymbol{k}_i^{\boldsymbol{t}}\, \boldsymbol{q}) \tag{12a}$$

It is important to note that the denominator, $\sum_{j=1}^{T} (\boldsymbol{k}_i^{\boldsymbol{t}}\, \boldsymbol{q})$ being a simple sum of dot products, can be close to zero, which could lead to numerical instability [7]. The unnormalized estimation of the context vector $\hat{\boldsymbol{v}}$ is given by:

$$\hat{\boldsymbol{v}} = \sum_{i=1}^{T} \boldsymbol{v}_{\mathbf{i}}\, (\boldsymbol{k}_i^{\boldsymbol{t}}\, \boldsymbol{q}) \tag{12b}$$

## 5. The Recursive Nature of Linear Memory

This linear formulation reveals a powerful insight: the memory matrix $\boldsymbol{H}(\boldsymbol{M})$, defined in the Eq. 10, which we will now call $\mathbf{A}_T = \boldsymbol{H}(\boldsymbol{M})$ given by:

$$\mathbf{A}_T = \sum_{i=1}^{T} \boldsymbol{v}_{\mathbf{i}}\, \boldsymbol{k}_i^{\boldsymbol{t}} \tag{13}$$

 doi:10.20944/preprints202511.0515.v1



Let $\boldsymbol{b_T^t}$ be the sum of all row key vectors, used for normalization:

$$\boldsymbol{b_T^t} = \sum_{i=1}^{T} \boldsymbol{k_i^t} \tag{14}$$

When a new item $\boldsymbol{v_{T+1}}, \boldsymbol{k_{T+1}}$ arrives, we don't need to recompute the entire memory from scratch. We can simply update the previous state:

$$\mathbf{A}_{T+1} = \mathbf{A}_T + \boldsymbol{v_{T+1}} \boldsymbol{k_{T+1}^t} \tag{15}$$

$\boldsymbol{b_{T+1}^t}$ can be updated recursively the following equation:

$$\boldsymbol{b_{T+1}^t} = \boldsymbol{b_T^t} + \boldsymbol{k_{T+1}^t} \tag{16}$$

This incremental update is conceptually identical to the state transition in a Recurrent Neural Network (RNN). An RNN updates its hidden state by combining the previous state with the new input, here, we update the memory matrix $\mathbf{A}_{T+1}$ by incorporating the new key-value pair, we update $\boldsymbol{b_{T+1}^t}$ by incorporating the new key. This reveals that linear attention is a form of recurrent state-space model.

For context vector estimation, let's break down the retrieval at step $T$ and $T$+1.

The unnormalized output $\widehat{\boldsymbol{v}}_T$ is the memory matrix $\mathbf{A}_T$ probed by the query $\boldsymbol{q_T}$:

$$\widehat{\boldsymbol{v}}_T = \mathbf{A}_T \boldsymbol{q_T} = \sum_{i=1}^{T} \boldsymbol{v_i} \, \boldsymbol{k_i^t} \boldsymbol{q_T} \tag{17}$$

By defining $\boldsymbol{d_{i,T}} = \boldsymbol{k_i^t} \boldsymbol{q_T}$, the above equation becomes:

$$\mathbf{A}_T \boldsymbol{q_T} = \sum_{i=1}^{T} \boldsymbol{d_{i,T}} \boldsymbol{v_i} \tag{18}$$

The normalization factor $z_T$ is defined as the sum of all similarities:

$$\boldsymbol{z_T} = \boldsymbol{b_T^t} \boldsymbol{q_T} = \sum_{i=1}^{T} \boldsymbol{k_i^t} \boldsymbol{q_T} = \sum_{i=1}^{T} \boldsymbol{d_{i,T}} \tag{19}$$

The normalized vector $\widehat{\boldsymbol{v}}_T$ is their ratio:

$$\widehat{\boldsymbol{v}}_T = \boldsymbol{u}_T / z_T = \mathbf{A}_T \boldsymbol{q_T} / z_T = \sum_{i=1}^{T} \boldsymbol{d_{i,T}} \boldsymbol{v_i} \, / \sum_{i=1}^{T} \boldsymbol{d_{i,T}} \tag{20}$$

Note that $\widehat{\boldsymbol{v}}_T$ can be derived by $\mathbf{A}_T \boldsymbol{q_T}$ can be computed with linear time complexity. Likewise, the normalization factor $z_T$ can also be computed in linear time.

 doi:10.20944/preprints202511.0515.v1



For the next time step, T+1, the context vector is similarly estimated with normalized weights as follows:

$$\hat{\boldsymbol{v}}_{T+1\square} = \boldsymbol{u}_{T+1}/z_{T+1} = \mathbf{A}_{T+1}\boldsymbol{q}_{\mathbf{T+1}}/z_{T+1} \tag{21}$$

Where $\boldsymbol{u}_{T+1}$ and $z_{\mathrm{T+1}}$ are defined as follows:

$$\boldsymbol{u}_{T+1} = \mathbf{A}_{T+1}\boldsymbol{q}_{\mathbf{T+1}} = (\mathbf{A}_T + \boldsymbol{v}_{T+1}\boldsymbol{k}_{T+1}^{t})\boldsymbol{q}_{\mathbf{T+1}}$$

$$\boldsymbol{u}_{T+1} = \mathbf{A}_{T+1}\boldsymbol{q}_{\mathbf{T+1}} = (\mathbf{A}_T\boldsymbol{q}_{\mathbf{T+1}} + \boldsymbol{v}_{T+1}\boldsymbol{k}_{T+1}^{t}\boldsymbol{q}_{\mathbf{T+1}})$$

$$z_{\mathrm{T+1}} = \boldsymbol{b}_{T+1}^{t}\boldsymbol{q}_{\mathbf{T+1}}$$

The unnormalized context vector estimation is defined as:

$$\hat{\boldsymbol{v}}_{T+1\square} = \boldsymbol{u}_{T+1} \tag{22}$$

## 6. Kernel Function Selection

We discuss how to select kernels of $G(\boldsymbol{k}_j, \boldsymbol{q})$ as defined in Eq. 8a, and Eq. 8b to reduce computation in attention model.

In the standard Transformer, the core of the associative retrieval mechanism is the dot-product attention, which can be viewed as applying a universal kernel function, $G(\boldsymbol{k}_j, \boldsymbol{q}) = exp(\alpha \boldsymbol{k}_j^{t}\boldsymbol{q})$, followed by normalization. This is standard Scaled-Dot-Product function or softmax function. In the following discussion, $\boldsymbol{x}$ and $\boldsymbol{y}$ refer to $\boldsymbol{k}_j$ and $\boldsymbol{q}$, respectively.

Consider a kernel of the form k($\boldsymbol{x}$, $\boldsymbol{y}$) = ψ(<$\boldsymbol{x}$, $\boldsymbol{y}$ >), where < $\boldsymbol{x}$, $\boldsymbol{y}$ > is the inner product of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and ψ is a real-valued function. If we let z = < $\boldsymbol{x}$, $\boldsymbol{y}$ >, the function ψ(z) = exp(αz) is a universal kernel [11] when $\alpha > 0$.

The most direct simplification of the kernel function is to replace exp(αz) with the linear kernel, ψ(z) = αz.

Another way to reduce the kernel's computation is to use a feature map, φ, which maps each component of a vector to a non-negative value. For instance, φ(r) could be elu(r) + 1 or relu(r) [8], where r is a real value. In this way, the kernel G(x, y) = k($\boldsymbol{x}$, $\boldsymbol{y}$)=G($\boldsymbol{x}$, $\boldsymbol{y}$) = $\varphi(\boldsymbol{x})^{t}\varphi(\boldsymbol{y})$ is a valid kernel. However, it is not a universal kernel because its feature map is not sufficiently expressive. Because The mapping, such as elu(r) + 1 or relu(r), compresses or loses information from the negative components of a vector because it either compresses these values or discards them entirely.

Note that when computing the normalized weights in Eq. 8b with the RBF kernel listed in Table 1, where k($\boldsymbol{x}$, $\boldsymbol{y}$) = $G(\boldsymbol{k}_j, \boldsymbol{q})$, $\boldsymbol{x} = \boldsymbol{k}_j$, and $\boldsymbol{y} = \boldsymbol{q}$, $G(\boldsymbol{k}_j, \boldsymbol{q}) = exp(-\|\boldsymbol{q}\|^2/\gamma)exp(\boldsymbol{k}_j^{t}(2\boldsymbol{q} - \boldsymbol{k}_j)/\gamma)$, the factor of $exp(-\|\boldsymbol{q}\|^2/\gamma)$, appears in both the numerator and the denominator. This factor cancels out, which results in the normalized weights being computed using the softmax function.

**Table 1. Kernels.**

| Name | Expression | Type |
|---|---|---|
| Exponential | ψ (z)=exp(z) | Universal |
| Linear | ψ (z)=z | Valid |
| Geometric Series | ψ (z)=1/(1-z) | Conditionally universal if its input, z, is within in range of (-1,1). This condition is met when |

 doi:10.20944/preprints202511.0515.v1



| | | |
|---|---|---|
| | | z is computed from cosine kernel |
| Cosine | $k(\boldsymbol{x}, \boldsymbol{y}) = <\boldsymbol{x}, \boldsymbol{y}> / \sqrt{(\alpha + \|\boldsymbol{x}\|^2)(\alpha + \|\boldsymbol{y}\|^2)}$ | Valid when $\alpha \geq 0$ such as $\alpha = 1.e-3$ |
| Shift Cosine | $k(\boldsymbol{x}, \boldsymbol{y}) = (1+z)$ | Valid when z is computed from cosine kernel |
| RBF | $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/\gamma)$ | Universal when $\gamma > 0$ |
| Clipped Geometric Series | $k(\boldsymbol{x}, \boldsymbol{y}) = 1/(1+\text{eps}-c(<\boldsymbol{x}, \boldsymbol{y}>)/M)$ | Conditionally universal, $c(<\boldsymbol{x}, \boldsymbol{y}>) = M$ when $<\boldsymbol{x}, \boldsymbol{y}> > M$, where M>0, such as M=300, eps is small positive number such as eps=1.e-4 |
| Relu | $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})^t \varphi(\boldsymbol{y}) + \text{eps}$ | Valid, where $\varphi = \text{Relu}$, and eps is small positive number such as eps=1.e-4 |
| Relu2 | $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})^t \varphi(\boldsymbol{y}) + \text{eps}$ | Valid, where $\varphi = \text{Relu}^2$, and eps is small positive number such as eps=1.e-4 |
| Elu | $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})^t \varphi(\boldsymbol{y})$ | Valid, where $\varphi = \text{Elu} + 1$ |

There are other kernels, but a comprehensive list is beyond the scope of this paper.

## 7. Experiments

Experiments were conducted on the Shakespeare character-level dataset [12] using a 6-layer GPT-style Transformer (6 attention heads, 256 context length, 384 embedding dimension). This architecture provides a resource-efficient testbed for our analysis.

Our findings demonstrate that all tested kernels defined in the Table 1, as described in the 'Kernel Function Selection' section, enable the model to converge successfully, confirmed by a steady reduction in the training loss. Significantly, even the pure linear kernel proved sufficient for model convergence, establishing a strong and efficient baseline for attention approximation.

In our experiment, we also observed that the model successfully converges even when the query and key vectors are constrained to be identical. This modification implies that a single source of feature vectors is utilized for both functions. A direct consequence of this approach is a 50% reduction in the memory required for storing the key and query vectors, a significant efficiency gain, particularly for models with large sequence lengths where memory complexity can be a bottleneck.

Furthermore, a comparative analysis of kernel functions was conducted. The results indicate that the Radial Basis Function (RBF) kernel, yields a marginal performance improvement over the original exponential kernel.

Specifically, with the Clipped Geometric Series, as defined in Table 1, the model also demonstrates convergence with normalized weights.

## 8. Conclusion

In this work, we have established a new theoretical foundation for Scaled-Dot-Product Attention (SDPA) by reframing it as a Generalized Memory Model. This perspective, which extends classical distributed memory theory, provides a new lens to understand the core associative properties of attention. Our empirical findings strongly support this model. We demonstrated the robustness of a simple linear kernel, which achieves convergence even without normalization, establishing it as a

 doi:10.20944/preprints202511.0515.v1



powerful and efficient baseline for attention approximation. Furthermore, we introduced a novel class of simplified, exponential-free kernels designed to reduce computational demands on hardware lacking specific acceleration.

While our experiments on the Shakespeare dataset provide a strong proof-of-concept, scaling these findings to larger corpora is a crucial next step. Future research should focus on characterizing these kernels' performance profiles on dedicated hardware accelerators. Ultimately, by connecting modern attention mechanisms to classical memory theory, this research offers both a practical roadmap for developing more efficient sequence models and a deeper theoretical understanding of how they work.

## References

1. James A. Anderson. A simple neural network generating an interactive memory. Mathematical Biosciences Volume 14, Issues 3–4, August 1972, Pages 197-220.
2. L. N. Cooper. A possible organization of animal memory and learning. Proceedings of the Nobel Symposium on Collective Properties of Physical Systems, B. Lundquist and S. Lundquist (Eds.), New York: Academic Press, pp. 252-264, 1973.
3. Leon N Cooper. Distributed memory, 1985. https://apps.dtic.mil/sti/tr/pdf/ADA153364.pdf
4. JJ Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci USA. 1982.
5. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017.
6. Jiyong Ma. Deriving the Scaled-Dot-Function via Maximum Likelihood Estimation and Maximum Entropy Approach, 2025, arXiv:2509.12285
7. Zhen Qin, XiaoDong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, Yiran Zhong. The Devil in Linear Transformer.2022, arXiv:2210.10340
8. Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. Proceedings of the 37th International Conference on Machine Learning, PMLR 119:5156-5165, 2020.
9. Weizhe Hua, Zihang Dai, Hanxiao Liu, Quoc Le. Transformer Quality in Linear Time. Proceedings of the 39th International Conference on Machine Learning, PMLR 162:9099-9117, 2022.
10. Ingo Steinwart. On the Influence of the Kernel on the Consistency of Support Vector Machines, Journal of Machine Learning Research 2001.
11. I. J. Schoenberg. Positive definite functions on spheres. Duke Mathematical Journal, 9(1):96–108, 1942.
12. Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT