CUBIT: TOKEN MIXER WITH KERNEL RIDGE REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Since its introduction in 2017, the Transformer has become one of the most widely adopted architectures in modern deep learning. Despite extensive efforts to improve positional encoding, attention mechanisms, and feed-forward networks, the core token-mixing mechanism in Transformers remains attention. In this work, we show that the attention module in Transformers can be interpreted as performing Nadaraya–Watson regression, where it computes similarities between tokens and aggregates the corresponding values accordingly. Motivated by this perspective, we propose **Cubit, a potential next-generation architecture that leverages Kernel Ridge Regression (KRR), while the vanilla Transformer relies on Nadaraya-Watson regression.** Specifically, Cubit modifies the classical attention computation by incorporating the closed-form solution of KRR, combining value aggregation through kernel similarities with normalization via the inverse of the kernel matrix. To improve the training stability, we further propose the **Limited-Range Rescale (LRR)**, which rescales the value layer within a controlled range. We argue that Cubit, as a KRR-based architecture, provides a stronger mathematical foundation than the vanilla Transformer, whose attention mechanism corresponds to Nadaraya–Watson regression. We validate this claim through comprehensive experiments. The experimental results suggest that Cubit may exhibit stronger long-sequence modeling capability. In particular, its performance gain over the Transformer appears to increase as the training sequence length grows.

1 INTRODUCTION

Recurrent Neural Networks (RNNs), introduced in the 1980s (Hopfield, 1982; Jordan, 1986; Elman, 1991; Graves, 2012), process sequences by recurrently updating hidden states across tokens, incurring linear computational complexity with respect to sequence length. In 2017, the Transformer architecture (Vaswani et al., 2017) was introduced, proposing self-attention mechanisms that capture global dependencies but at the quadratic computational cost. Despite this scalability limitation, Transformers have become the dominant architecture across natural language processing, computer vision, and multimodal learning, achieving remarkable empirical success. Using the powerful Transformer architecture, significant progress has been made in language modeling (Fedus et al., 2022; Puigcerver et al., 2024; Jiang et al., 2024; Meta; Liu et al., 2024; Team et al., 2025) and computer vision (Riquelme et al., 2021; Lin et al., 2024). The mixture-of-experts architecture (Jacobs et al., 1991; Shazeer et al., 2017; Roller et al., 2021) has emerged as an efficient alternative that allows parameter scaling while maintaining manageable computational requirements. More recently, combining Transformer backbones with mixture-of-experts designs has enabled the development of extremely large yet computationally efficient language models, demonstrating the effectiveness of sparse parameter scaling (Dai et al., 2024; Jiang et al., 2024; Shen et al., 2024; Wei et al., 2024; Liu et al., 2024).

The Transformer architecture can be theoretically grounded in Nadaraya-Watson Regression (Nadaraya, 1964; Watson, 1964) . The architecture comprises two principal components: feed-forward networks (FFNs) and multi-head self-attention mechanisms. The FFN layers can be interpreted as key-value memory systems (Geva

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.

et al., 2021). The attention mechanism itself operates as Nadaraya-Watson Regression with an exponential kernel (via softmax) and L1 normalization. Consequently, despite numerous architectural modifications were proposed—including sparse attention, linear attention, and various approximation schemes—the fundamental computational paradigm remains rooted in Nadaraya-Watson regression with inherent quadratic complexity.

In this work, we introduce Cubit, which replaces the Nadaraya-Watson regression in the attention module of Transformers with Kernel Ridge Regression (Murphy, 2012; Williams & Rasmussen, 1995). This framework readily extends to alternative regression methodologies, including local linear regression variants (Macaulay, 1931; Cleveland & Loader, 2013; Murray & Bellhouse, 2019). Compared to Transformers based on Nadaraya-Watson Regression, Cubit with Kernel Ridge Regression offers several theoretical benefits, including explicit regularization for bias-variance trade-off, faster convergence rates in RKHS, and greater robustness to noise and boundary effects (Long et al., 2024; Bak & Lee, 2025; Mollenhauer et al., 2025; Wen et al., 2025; Barzilai et al., 2025). We summarize our principal contributions as follows:

- We establish a unified theoretical framework connecting token mixing mechanisms to classical regression methods, systematically analyzing **Nadaraya-Watson Regression and Kernel Ridge Regression**. This perspective helps to understand and develop novel model architectures.
- Compared to Transformer based on Nadaray-Watson Regression), we propose the **Cubit based on Kernel Ridge Regression**, with **Limited-Range Rescale (LRR)** to improve the training stability.
- We validate Cubit across diverse datasets, sequence lengths, and model sizes. The experimental results suggest that Cubit exhibits stronger long-sequence modeling capability than the Transformer. In particular, its performance gain tends to increase as the training sequence length becomes longer.

2 METHOD

2.1 KERNEL RIDGE REGRESSION

We begin by revisiting the kernel ridge regression (KRR) method, which provides key insights into interpreting attention mechanisms as regression procedures. This perspective further motivates the design of a novel attention mechanism grounded in the KRR formulation.

Consider a positive definite kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and its associated reproducing kernel Hilbert space $\mathcal{H}_K$ with norm $\|\cdot\|_{\mathcal{H}_K}$. By the Moore-Aronszajn theorem, such a kernel uniquely defines an RKHS where evaluation functionals are continuous. The representation theorem states that the minimizer of the regularized empirical risk over $\mathcal{H}_K$ admits a finite-dimensional representation in terms of kernel evaluations at training points. More specifically, we consider the problem of learning a vector-valued function $f : \mathcal{X} \to \mathbb{R}^m$ from a given dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ and $\boldsymbol{y}_i \in \mathbb{R}^m$. KRR solves the following optimization problem:

$$\min_{f \in \mathcal{H}_K} \sum_{i=1}^{N} \|\boldsymbol{y}_i - f(\boldsymbol{x}_i)\|_2^2 + \lambda \|f\|_{\mathcal{H}_K}^2, \tag{1}$$

where $\lambda > 0$ is the regularization parameter controlling the bias-variance trade-off. The representation theorem guarantees that the optimal solution takes the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{c}_i K(\boldsymbol{x}, \boldsymbol{x}_i), \tag{2}$$

which reduces the infinite-dimensional optimization in the functional space to finding coefficients $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N]^\top \in \mathbb{R}^{N \times m}$ within a finite-dimensional vector space.

Substituting the kernel expansion equation 2 into the objective equation 1 yields the finite-dimensional optimization:

$$\min_{\boldsymbol{C}} \|\boldsymbol{Y} - \boldsymbol{K}\boldsymbol{C}\|_{\mathrm{F}}^2 + \lambda \operatorname{tr}(\boldsymbol{C}^\top \boldsymbol{K}\boldsymbol{C}), \tag{3}$$

where $\boldsymbol{Y} = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N]^\top \in \mathbb{R}^{N\times m}$ denotes the collection of label vectors over all samples, and $\boldsymbol{K} \in \mathbb{R}^{N\times N}$ is the Gram (kernel) matrix with entries $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. It is worthnoting that equation 3 is convex problem in $\boldsymbol{C}$, the global solution admits the following linear system due to the stationary condition:

$$(\boldsymbol{K} + \lambda \boldsymbol{I})\boldsymbol{C} = \boldsymbol{Y}, \tag{4}$$

where $\boldsymbol{I} \in \mathbb{R}^{N\times N}$ is the identity matrix. Therefore, the optimal coefficient matrix is given by

$$\boldsymbol{C} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{Y}.$$

For any new input $\boldsymbol{x} \in \mathcal{X}$, define

$$\boldsymbol{k}(\boldsymbol{x}) = \begin{bmatrix} K(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ K(\boldsymbol{x}, \boldsymbol{x}_N) \end{bmatrix} \in \mathbb{R}^N.$$

The prediction is then given by

$$f(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{C} = \boldsymbol{k}(\boldsymbol{x})^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{Y}. \tag{5}$$

Here, $\boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{Y}$ in equation 5 can be interpreted as Nadaraya-Watson regression estimator, while $(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}$ acts as a normalization operator that couples the information across all data samples.

2.2 Interpreting the Attention Mechanism as Regression

We interpret the token mixing operation in the attention block of the Transformer as a form of similarity-based regression (Nadaraya-Watson Regression (Nadaraya, 1964)), where each token aggregates information from other tokens according to learned similarity scores.

Given N token embeddings $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, we compute the query, key, and value representations for each head $h \in \{1, 2, \cdots, H\}$:

$$\boldsymbol{Q}^{(h)} = \boldsymbol{X}\boldsymbol{W}_Q^{(h)}, \quad \boldsymbol{K}^{(h)} = \boldsymbol{X}\boldsymbol{W}_K^{(h)}, \quad \boldsymbol{V}^{(h)} = \boldsymbol{X}\boldsymbol{W}_V^{(h)},$$

where $\boldsymbol{W}_Q^{(h)}, \boldsymbol{W}_K^{(h)}, \boldsymbol{W}_V^{(h)} \in \mathbb{R}^{D\times d_h}$ are learnable projection matrices with $d_h = D/H$. The attention weights are computed via scaled dot-product similarity with a learnable temperature parameter $w^{(h)} > 0$:

$$\boldsymbol{A}^{(h)} = \text{Softmax}\left(w^{(h)} \cdot \boldsymbol{Q}^{(h)}\boldsymbol{K}^{(h)\top}\right), \tag{6}$$

where $\text{Softmax}(\cdot)$ is applied row-wise to ensure normalization. Then, the output of each head is given by

$$\boldsymbol{Z}^{(h)} = \boldsymbol{A}^{(h)}\boldsymbol{V}^{(h)} = \text{Softmax}\left(w^{(h)} \cdot \boldsymbol{Q}^{(h)}\boldsymbol{K}^{(h)\top}\right)\boldsymbol{V}^{(h)}. \tag{7}$$

From equation 7, we observe that each output token is computed as a weighted aggregation of value vectors, where the weights are determined by query-key similarity scores in equation 6. In particular, the i-th output token can be written as

$$\boldsymbol{z}_i^{(h)} = \sum_{j=1}^{N} A_{ij}^{(h)} \boldsymbol{v}_j^{(h)},$$

where $A_{ij}^{(h)}$ denotes the attention weight between token i and token j. Therefore, the attention mechanism can be interpreted as a form of data-dependent kernel regression, where both the similarity function and the regression targets are learned from data.

Summary. In summary, the attention mechanism can be viewed as a form of similarity-based regression, closely related to kernel smoothing methods such as Nadaraya–Watson regression, but with learned similarity functions.

2.3 Designing the Attention Mechanism via Kernel Ridge Regression

Motivated by the kernel regression interpretation of the standard attention mechanism, we propose a new attention formulation based on KRR, where both similarity evaluation and inverse normalization are explicitly incorporated into the attention scores.

Kernel Ridge Regression Formulation. For head h, let $\phi(\cdot)$ denote the feature map induced by a kernel function $K(\cdot,\cdot)$. By Mercer's theorem, the kernel matrix must be constructed consistently from the same RKHS in order to ensure positive definiteness and the validity of the Representer theorem. Accordingly, the kernel function can be written as

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \psi(\boldsymbol{x}_i)\psi(\boldsymbol{x}_j)^\top,$$

where $\psi : \mathbb{R}^D \to \mathbb{R}^{d_h}$ defines the feature transformation associated with the kernel.

To adapt this framework to attention, we parameterize $\psi(\boldsymbol{x}) = \phi(\boldsymbol{x}\boldsymbol{W}_K^{(h)})$, where ϕ is a prescribed activation function and $\boldsymbol{W}_K^{(h)}$ is a trainable projection matrix. The corresponding normalization term in our kernel regression attention is defined as

$$\boldsymbol{\Sigma}^{(h)} = \left(\mathbf{K}^{(h)}\mathbf{K}^{(h)\top} + \lambda\mathbf{I}\right)^{-1},$$

where $\mathbf{K}^{(h)} = \phi(\boldsymbol{X}\boldsymbol{W}_K^{(h)})$. In practice, the kernel matrix admits various design choices. The key transformation ϕ can range from identity mapping to normalization operators (e.g., ℓ_2 normalization), while the kernel activation $g(\cdot)$ can be flexibly configured. This leads to a general formulation:

$$\boldsymbol{\Sigma}^{(h)} = \left(g\left(\mathbf{K}^{(h)}\mathbf{K}^{(h)\top}\right) + \lambda\mathbf{I}\right)^{-1}, \tag{8}$$

where different combinations of ϕ and g yield various kernel instantiations. where different combinations of ϕ and g yield various kernel instantiations, while the g could be Softmax for both $\boldsymbol{A}^{(h)}$ and $\boldsymbol{\Sigma}^{(h)}$. It remains to model

$$\boldsymbol{k}(\boldsymbol{x}_i) = \begin{bmatrix} K(\boldsymbol{x}_i, \boldsymbol{x}_1) \\ \vdots \\ K(\boldsymbol{x}_i, \boldsymbol{x}_i) \end{bmatrix} \in \mathbb{R}^i.$$

This autoregressive construction reflects the causal constraint that the i-th token can only attend to itself and previous tokens, so only the first i samples are available in the regression. Under the standard kernel formulation, we would have

$$\boldsymbol{k}(\boldsymbol{x}_i) = \begin{bmatrix} \phi(\boldsymbol{x}_i\boldsymbol{W}_K^{(h)})\phi(\boldsymbol{x}_1\boldsymbol{W}_K^{(h)})^\top \\ \vdots \\ \phi(\boldsymbol{x}_i\boldsymbol{W}_K^{(h)})\phi(\boldsymbol{x}_i\boldsymbol{W}_K^{(h)})^\top \end{bmatrix}.$$

Although this formulation closely follows classical kernel theory, it is overly restrictive for attention modeling in practice. To improve flexibility, we adopt the query-key parameterization from standard attention and instead define

$$\boldsymbol{k}(\boldsymbol{x}_i) = \begin{bmatrix} \phi(\boldsymbol{x}_i\boldsymbol{W}_Q^{(h)})\phi(\boldsymbol{x}_1\boldsymbol{W}_K^{(h)})^\top \\ \vdots \\ \phi(\boldsymbol{x}_i\boldsymbol{W}_Q^{(h)})\phi(\boldsymbol{x}_i\boldsymbol{W}_K^{(h)})^\top \end{bmatrix}.$$

This relaxation introduces greater expressive power and improves numerical stability. In particular, when ϕ is the identity map and softmax normalization is applied to $[\boldsymbol{k}(\boldsymbol{x}_i); \cdots; \boldsymbol{k}(\boldsymbol{x}_N)]$, the resulting formulation reduces to the standard attention mechanism in equation 6.

Therefore, the final output in equation 7 is given by

$$\mathbf{Z}^{(h)} = \mathbf{A}^{(h)}\mathbf{\Sigma}^{(h)}\mathbf{V}^{(h)}, \tag{9}$$

where $\mathbf{A}^{(h)}$ denotes the kernel similarity matrix between each token and its accessible context tokens, and $\mathbf{\Sigma}^{(h)}$ is the KRR-inspired normalization matrix.

Limited-Range Rescale. We introduce a learnable scaling vector $\mathbf{s}^{(h)} \in \mathbb{R}^N$ to adaptively adjust the kernel precision, where each element is computed as:

$$\hat{s}_i^{(h)} = \alpha \cdot \sigma\left([\mathbf{W}_s\mathbf{x}]_i\right) + \beta, \quad \forall i \in \{1, \ldots, N\}, \tag{10}$$

with $\sigma(\cdot)$ denoting the sigmoid function, and $\alpha, \beta > 0$ ensuring $\hat{s}_i^{(h)} \in (\beta, \alpha + \beta)$. This guarantees the invertibility of $\hat{\mathbf{S}}^{(h)} = \mathrm{diag}(\hat{\mathbf{s}}^{(h)})$, with bounded inverse elements $\frac{1}{\hat{s}_i^{(h)}} \in \left(\frac{1}{\alpha+\beta}, \frac{1}{\beta}\right)$. The scaling mechanism modifies the kernel ridge regression solution. Specifically, we solve:

$$\hat{\mathbf{S}}^{(h)}(\mathbf{\Sigma}^{(h)})^{-1}\mathbf{o} = \mathbf{V}, \tag{11}$$

$$\mathbf{o} = (\mathbf{\Sigma}^{(h)})(\hat{\mathbf{S}}^{(h)})^{-1}V \tag{12}$$

We could have $S^{(h)} = (\hat{S}^h)^{-1}$. Finally, the KRR-based token mixer produces the output via the following:

$$\mathbf{Z}^{(h)} = \mathbf{A}^{(h)}O = \mathbf{A}^{(h)}\mathbf{\Sigma}^{(h)}\mathbf{S}^{(h)}\mathbf{V}^{(h)}. \tag{13}$$

Local Linear Regression and Kernel Ridge Regression. We further discuss the Cubit with Local Linear Regression Appendix H.

3 Experiment

Baseline. We compare the proposed Cubit with the Transformer (Vaswani et al., 2017) and DeltaFormer (Xu et al., 2025). The Transformer is the foundation of the recent Large Language Model. The DeltaFormer combines the DeltaNet and Transformer, suggesting that it may have higher expressiveness than the Transformer.

Datasets. Our analysis involves training language models on the Arxiv and Books3 datasets, which are frequently used for evaluating model performance (Press et al., 2022). Moreover, we train the model on large-scale dataset FinWeb-Edu (Penedo et al., 2024; Lozhkov et al., 2024) and evaluate on downstream datasets, including ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and WinoGrade (Sakaguchi et al., 2021), SocialIQA (Sap et al., 2019), and RACE (Lai et al., 2017).

Experiment settings. Initially, we compare Cubit with other baselines at training lengths 512 and 1024, using decoder-only Transformers (Brown et al., 2020) with model size 125M, whose configuration is shown in Appendix G. Subsequently, we evaluate the performance of larger model sizes, specifically 350M and 1.3 B. Finally, we suggest that the Cubit may presents advantages for long sequences.

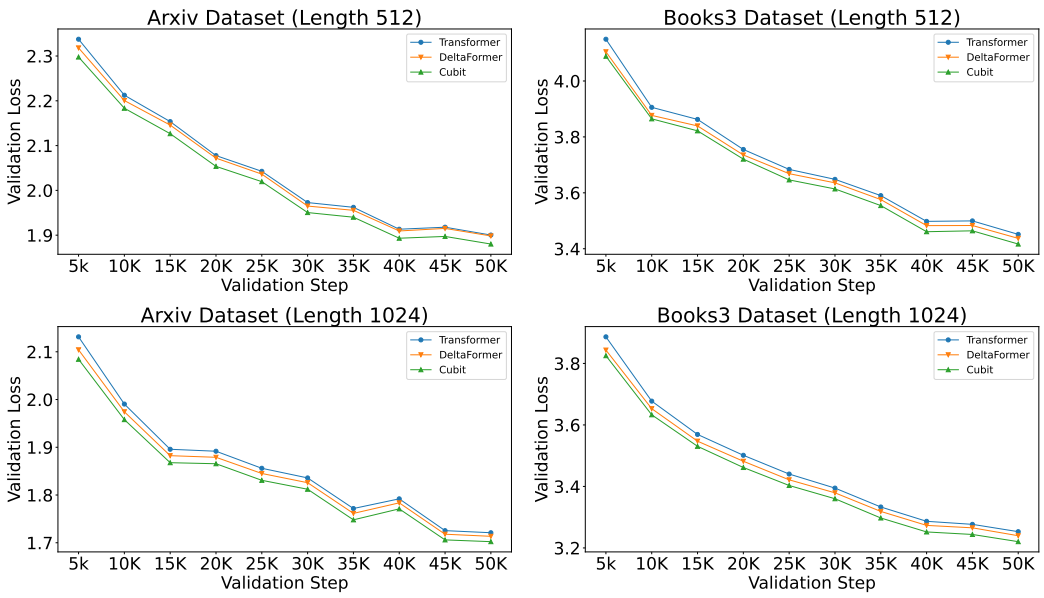


Figure 1: The performance of different methods on the Arxiv and Books3 dataset, with model parameter 125M, training lengths of 512 and 1024.

3.1 Compare with Baseline

The Cubit consistently outperforms baselines across diverse datasets. As shown in Figure 1, our method achieves lower perplexity than both standard Transformer and DeltaFormer on multiple benchmarks. On Arxiv (trained with 512 tokens), Cubit attains a loss of 1.8802, improving over Transformer (1.9003) and DeltaFormer (1.8983). Similarly, on Books3, it achieves 3.4168, surpassing Transformer (3.4514) and DeltaFormer (3.4371). These results demonstrate Cubit's robust generalization across different text domains.

The Cubit scales effectively across sequence lengths. As shown in Figure 1, our method consistently outperforms the Transformer baseline on both Arxiv and Books3 datasets across 512 and 1024 token training lengths. On Arxiv, Cubit achieves losses of 1.8802 and 1.7023 versus Transformer's 1.9003 and 1.7210; on Books3, 3.4168 and 3.2204 versus 3.4514 and 3.2529. Notably, the performance gap widens at longer contexts, indicating Cubit's superior capability in capturing long-range dependencies.

3.2 The Effect of Multi-Domain Datasets

The Cubit achieves better performance on a multi-domain dataset. As present in Figure 2, we evaluate Cubit against baselines across varying sequence lengths. At 512 tokens, it achieves a validation loss of 3.7110, improving over the Transformer (3.7443) and DeltaFormer (3.7242), respectively. At 1024 tokens, Cubit maintains its lead with 3.4597 versus 3.4960 (Transformer) and 3.4705 (DeltaFormer). Notably, the performance gap widens with longer sequences, suggesting compounding benefits from its architectural innovations. These consistent improvements across both lengths on a diverse multi-domain dataset establish Cubit as a potential next-generation architecture for sequence modeling.

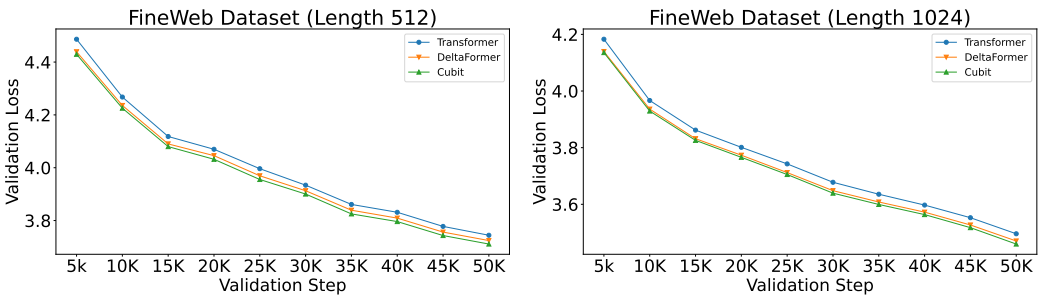


Figure 2: The performance of different methods on the FineWeb dataset, with model parameter 125M.

3.3 The Effect of Longer Training Length

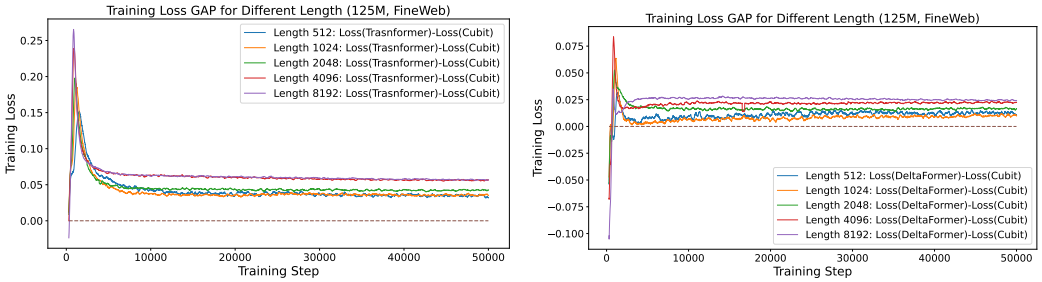


Figure 3: The performance of long training length on the FineWeb dataset, with model parameter 125M.

The Cubit performance gain scales with sequence length. As presented in Figure 3, training dynamics reveal that Cubit's advantage over Transformer increases monotonically with sequence length: the loss gap stabilizes at 0.05–0.06 for 8192 tokens, exceeding margins at 512, 1024 and 4096 tokens. With the training length increase, the loss gap between Cubit and DeltaFormer also gradually increases. This suggests Cubit's kernel-based mechanism more effectively captures long-range dependencies, while standard attention suffers at longer contexts.

The gain stems from long-sequence processing, not more tokens. To disentangle sequence length from total training tokens, we compare Transformer and Cubit at 1024 tokens with doubled batch size in Figure 6. The smaller performance gap under increased batch size confirms that improvements derive from architectural efficacy on long sequences rather than simply more training tokens.

3.4 The Effect of Large Model

Cubit demonstrates favorable scaling properties, whereas DeltaFormer fails to maintain its advantage. As illustrated in Figure 4, at 350M parameters, Cubit achieves a loss of 3.3166, substantially outperforming both the standard Transformer (3.3438) and DeltaFormer (3.3284). Upon scaling to 1.3B parameters, Cubit continues to exhibit strong scaling behavior with a loss of 3.1849, while the performance gap between Transformer (3.2100) and DeltaFormer (3.2057) diminishes considerably. These results suggest that Cubit benefits consistently from increased model capacity, whereas DeltaFormer's initial improvements over the Transformer baseline erode as model size grows.

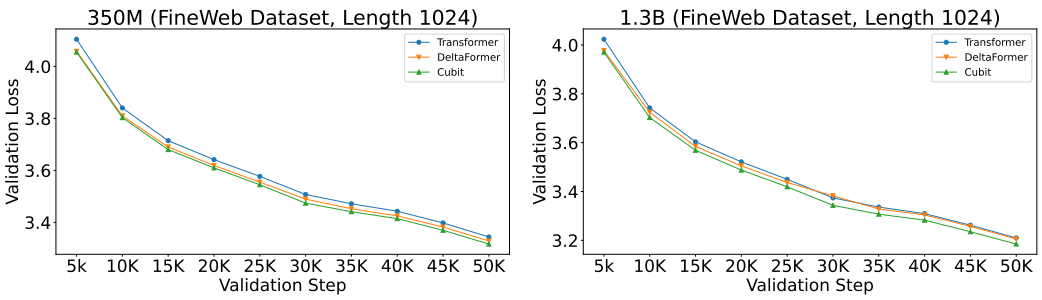


Figure 4: The performance of larger model size on the FineWeb dataset.

3.5 Ablation Study

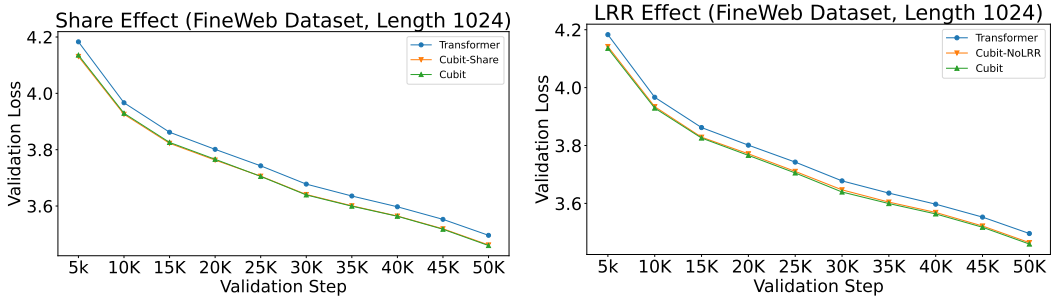


Figure 5: The performance of the share key embedding and no Limited-Range Rescale, with model size 125M and training length 1024 on the FineWeb dataset.

Architectural innovation, not parameter expansion, drives performance improvement. Figure 5 presents an ablation comparing two Reference Embedding configurations: direct Key embedding sharing versus an independent projection matrix. Cubit equipped with a dedicated projection matrix outperforms both its shared-embedding variant (Cubit-Share) and the standard Transformer. Critically, even when Reference embeddings are shared, Cubit maintains consistent superiority over the Transformer across all training stages from initialization through convergence. This establishes that performance gains originate from the structural design rather than increased model capacity.

LRR provides complementary benefits, and the Cubit-NoLRR is better than Transformer. Incorporating the LRR mechanism yields a validation loss of 3.4597 for Cubit, representing a measurable improvement over both the Cubit-Cubit and the Transformer. Cubit-NoLRR—which operates without the LRR component—achieves a loss of 3.4645, while the conventional Transformer architecture reaches 3.4960 under identical experimental conditions. These results demonstrate that Cubit is the strongest, followed by Cubit-NoLRR, and Transformer as the baseline.

3.6 Performance on Downstream Tasks

Downstream Evaluation. We evaluate performance on standard benchmarks, including ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and Wino-

Table 1: Main language modeling results against different methods. All models are trained on the same subset of the FineWeb-Edu dataset (Penedo et al., 2024; Lozhkov et al., 2024) with the GPT-2 tokenizer.

Model	ARC-C	Hellaswag	PIQA	SciQ	Winograde	SocialIQA	RACE	Avg
350M params/50B Tokens								
Transformer	45.20	44.93	69.00	69.97	53.35	32.91	31.00	49.48
DeltaFormer	44.82	45.52	69.80	70.57	52.57	32.91	31.20	49.63
Cubit	45.50	45.68	68.90	71.06	54.46	32.80	31.67	**50.01**
350M params/10B Tokens								
Transformer	43.10	40.82	69.60	69.42	50.28	33.06	30.62	48.13
DeltaFormer	43.56	40.82	66.60	68.88	52.64	33.42	30.72	48.09
Cubit	44.02	41.54	68.20	68.44	51.14	33.73	31.10	**48.31**
1.3B params/10B Tokens								
Transformer	48.70	48.40	70.10	71.71	55.25	33.42	33.11	51.53
DeltaFormer	47.10	48.96	70.20	72.52	54.85	32.96	32.73	51.33
Cubit	47.10	49.79	72.60	71.65	54.22	33.67	33.40	**51.78**

Grade (Sakaguchi et al., 2021), SocialIQA (Sap et al., 2019) and RACE (Lai et al., 2017), using the `lm-evaluation-harness` (Gao et al., 2024) codebase. We train the model with 50K steps with training length 1024 and training tokens 50B. The model sizes are 350M, and 1.3B. We display the zero-shot evaluation results of models here in Tables 1.

With the same model size, the Cubit achieves better performance, from small data size (e.g., 10B) to large data size (e.g., 50B). With the smallest data size 10B, the Cubit achieves the best average performance 48.31, which is better than the Transformer with 48.13 average performance and DeltaFormer with 48.09 average performance. With the data size increasing to 50B, the Cubit achieves the best performance 50.01 average performance, which is better than Transformer with 49.48 average performance and DeltaFormer with 49.63 performance. Therefore, the Cubit consistently achieves better performance, from smaller data size to large data size.

With the same training data size, the Cubit is always better than the routers, from small model size (e.g., 350M) to large model size (e.g., 1.3B). With the model size 350M model size, the Cubit achieves 48.31 average performance, which is better than Transformer with 48.13 average performance and DeltaFormer with 48.09 average performance. With the model size increase from 350M to 1.3B, the Cubit achieves 51.78 average performance, which is better than Transformer with 51.53 average performance. And Cubit is better than DeltaFormer with 51.33 average performance.e Therefore, with the same training data size from small model size to large model size, Cubit is better than Transformer and DeltaFormer.

4 Conclusion

In this work, we propose Cubit, a novel architecture based on Kernel Ridge Regression that replaces the Nadaraya-Watson estimator underlying Transformers. We conduct extensive evaluations across diverse datasets, sequence lengths, and model scales. Cubit consistently outperforms the Transformer, with performance gains that become increasingly pronounced at longer training lengths. We believe Cubit has the potential to be next-generation foundation model architecture.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Kwan-Young Bak and Woojoo Lee. Effect of dimensionality on convergence rates of kernel ridge regression estimator. *Journal of Statistical Planning and Inference*, 236:106228, 2025.

Daniel Barzilai, Guy Kornowski, and Ohad Shamir. Overfitting regimes of nadaraya-watson interpolators. *arXiv e-prints*, pp. arXiv–2502, 2025.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

Richard A Berk. *Regression analysis: A constructive critique*, volume 11. Sage, 2004.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Langat Reuben Cheruiyot. Local linear regression estimator on the boundary correction in nonparametric regression estimation. *Journal of Statistical Theory and Applications*, 19(3):460–471, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

William S Cleveland. Lowess: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35(1):54, 1981.

William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610, 1988.

William S Cleveland and Catherine Loader. Smoothing by local regression: Principles and methods. In *Statistical theory and computational aspects of smoothing: Proceedings of the COMPSTAT'94 Satellite Meeting held in Semmering, Austria, 27–28 August 1994*, pp. 10–49. Springer, 2013.

Damai Dai, Chengqi Deng, Chenggang Zhao, R.x. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y.k. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70/.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. Cogview: Mastering text-to-image generation via transformers. *Advances in neural information processing systems*, 34:19822–19835, 2021.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Jeffrey L Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7(2):195–225, 1991.

Robert F Engle, Clive WJ Granger, John Rice, and Andrew Weiss. Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American statistical Association*, 81(394):310–320, 1986.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Evelyn Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, 2021.

Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in neural information processing systems*, 35:35971–35983, 2022.

Mingxuan Han, Chenglong Ye, and Jeff Phillips. Local kernel ridge regression for scalable, interpolating, continuous regression. 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Daniel Horbunov and Rostyslav Maiboroda. Consistency of local linear regression estimator for mixtures with varying concentrations. *Modern Stochastics: Theory and Applications*, 11(3):359–372, 2024.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Jae-Keun Jo. Improvement of boundary bias in nonparametric regression via twicing technique. *Communications for Statistical Applications and Methods*, 4(2):445–452, 1997.

Michael I Jordan. Serial order: A parallel distributed processing approach. 1986.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. Finetuning pretrained transformers into rnns. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 10630–10643, 2021.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959*, 2016.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 785–794, 2017.

Baisheng Li, Banggu Wu, Bole Ma, Bowen Xiao, Chaoyi Zhang, Cheng Li, Chengyi Wang, Chengyin Xu, Chi Zhang, Chong Hu, et al. Virtual width networks. *arXiv preprint arXiv:2511.11238*, 2025.

Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. Video-LLaVA: Learning united visual representation by alignment before projection. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5971–5984, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.342. URL https://aclanthology.org/2024.emnlp-main.342/.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jihao Long, Xiaojun Peng, and Lei Wu. Optimal rates and saturation for noiseless kernel ridge regression. *arXiv preprint arXiv:2402.15718*, 2024.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *Advances in Neural Information Processing Systems*, 37:71831–71854, 2024.

Frederick R Macaulay. Introduction to" the smoothing of time series". In *The Smoothing of Time Series*, pp. 17–30. NBER, 1931.

Nicole MÃžcke and Gilles Blanchard. Parallelizing spectrally regularized kernel algorithms. *Journal of Machine Learning Research*, 19(30):1–29, 2018.

AI Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 4-7-2025.

Mattes Mollenhauer, Nicole MÃžcke, Dimitri Meunier, and Arthur Gretton. Regularized least squares learning with heavy-tailed noise is minimax optimal. *arXiv preprint arXiv:2505.14214*, 2025.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Lori Murray and David Bellhouse. Wf sheppard's smoothing method: A precursor to local polynomial regression. *International Statistical Review*, 87(3):604–612, 2019.

Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 135(3):370–384, 1972.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Joan Puigcerver, Carlos Riquelme Ruiz, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jxpsAj7ltE.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, et al. Theory, analysis, and best practices for sigmoid self-attention. *arXiv preprint arXiv:2409.04431*, 2024.

Liliang Ren, Congcong Chen, Haoran Xu, Young Jin Kim, Adam Atkinson, Zheng Zhan, Jiankai Sun, Baolin Peng, Liyuan Liu, Shuohang Wang, et al. Decoder-hybrid-decoder architecture for efficient reasoning with long generation. *arXiv preprint arXiv:2507.06607*, 2025.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *advances in neural information processing systems*, 34:17555–17566, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. Jetmoe: Reaching llama2 performance with 0.1 m dollars. *arXiv preprint arXiv:2404.07413*, 2024.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

Henri Theil. A rank-invariant method of linear and polynomial regression analysis. *Indagationes mathematicae*, 12(85):173, 1950.

Juliana Tolles and William J Meurer. Logistic regression: relating patient characteristics to outcomes. *Jama*, 316(5):533–534, 2016.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10): 6761–6774, 2024.

Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 359–372, 1964.

Tianwen Wei, Bo Zhu, Liang Zhao, Cheng Cheng, Biye Li, Weiwei Lü, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Liang Zeng, et al. Skywork-moe: A deep dive into training techniques for mixture-of-experts language models. *arXiv preprint arXiv:2406.06563*, 2024.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Hongwei Wen, Annika Betken, and Wouter Koolen. On the robustness of kernel ridge regression using the cauchy loss function. *arXiv preprint arXiv:2503.20120*, 2025.

Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers. *arXiv preprint arXiv:2309.08586*, 2023.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *Advances in neural information processing systems*, 32, 2019.

Mingyu Xu, Tenglong Ao, Jiaao He, Jianqiao Lu, Guang Shi, and Shu Zhong. Deltaformer: Unlock the state space of transformer. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

Scott L Zeger and Peter J Diggle. Semiparametric models for longitudinal data with application to cd4 cell numbers in hiv seroconverters. *Biometrics*, pp. 689–699, 1994.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor product attention is all you need. *arXiv preprint arXiv:2501.06425*, 2025.

Chuanyang Zheng, Jiankai Sun, Yihang Gao, Enze Xie, Yuehao Wang, Peihao Wang, Ting Xu, Matthew Chang, Liliang Ren, Jingyao Li, Jing Xiong, Kashif Rasul, Mac Schwager, Anderson Schneider, Zhangyang Wang, and Yuriy Nevmyvaka. Understanding the mixture-of-experts with nadaraya-watson kernel. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=NdDlqHV1md.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.

A LIMITATION

We build Cubit to replace Transformer, which is validated with sufficient experiments. As Cubit is more powerful, it is not clear whether Cubit will be abused.

B BROADER IMPACTS

The Cubit provides a more powerful model, which may benefit society. However, it is not clear whether the Cubit will be abused.

C AI USE STATEMENT

In this work, the LLM is used to polish the paper to improve the writing, including the grammatical structures, spelling, punctuation and clarity.

D ETHICS STATEMENT

This work has been conducted in full accordance with the established Code of Ethics. All procedures, methods, and interactions associated with this project were carried out with strict adherence to the principles and standards set out in the Code, including those relating to integrity, transparency, accountability, and respect for all parties involved. The author affirms that no aspect of this work deviates from the ethical guidelines stipulated therein.

E REPRODUCIBILITY STATEMENT

This work provides the complete model configuration in Appendix **??**, together with the full implementation code in Appendix **??**. These materials are included so that the experimental results presented in this paper can be fully reproduced by other researchers.

F RELATED WORK

Transformer Architecture Transformer architecture (Vaswani et al., 2017) was proposed in 2017, with Feed-Forward Network and Attention. In the following years, there are modifications of Transformer. From the view of FFN, there are mixture-of-expert and different activation functions. From the view of attention, there are GQA (Ainslie et al., 2023), MQA (Shazeer, 2019), MLA (Liu et al., 2024), TPA (Zhang et al., 2025) and so on. There is also gated attention (Qiu et al., 2025) to improve the performance. Also, there are works that are trying to replace the softmax attention with ReLU attention (Wortsman et al., 2023) and sigmoid attention (Ramapuram et al., 2024). The skip connection (He et al., 2016) is also discussed, such as hyper-connection (Zhu et al., 2024), attention residual (Team et al., 2026), deepnorm (Wang et al., 2024), and sandwitchnorm (Ding et al., 2021). However, these modifications do not modify the modeling of the attention block, which is actually the Nadaraya-Watson Regression.

Token Mixer To address the scalability limitations of quadratic attention, numerous linear-complexity linear token mixers have been developed. Early recurrent approaches include LSTM (Graves, 2012) and its extension mLSTM (Beck et al., 2024) with matrix-valued memory states. Linear attention mechanisms (Kasai et al., 2021) approximate softmax attention via kernel feature mappings to achieve linear complexity.

State space models (SSMs) represent another major family, beginning with S4 (Gu et al., 2021) and its variants (Gu et al., 2022). Mamba (Gu & Dao, 2024) introduced input-dependent selective state updates with hardware-aware parallel scans, later unified with attention via SSD in Mamba-2 (Dao & Gu, 2024). Linear RNN variants include RetNet (Sun et al., 2023), RWKV (Peng et al., 2025), HGRN-2 (Qin et al., 2024), and Gated DeltaNet (Yang et al., 2024). MegaLodon (Ma et al., 2024) combines exponential moving average with chunked attention. MLP-Mixer (Tolstikhin et al., 2021) demonstrates token mixing without recurrence or attention via channel-wise MLPs. Transformer (Vaswani et al., 2017; Xu et al., 2025) employs self-attention with quadratic complexity, enabling strong relational reasoning and in-context learning at the cost of computational scalability for long sequences.

Regression Method Regression methods fall into three categories by learning approach. Non-parametric methods make minimal functional assumptions, letting data determine model structure. Key examples include k -nearest neighbor regression (Fix, 1985), kernel-based approaches (Nadaraya-Watson (Nadaraya, 1964), ridge kernel (Murphy, 2012), and local linear/polynomial variants (Cleveland, 1981; Cleveland & Devlin, 1988)), and Kernel Ridge Regression (Williams & Rasmussen, 1995)—which defines a function-space prior via covariance kernels for predictive distributions with built-in uncertainty. Parametric methods assume fixed functional forms with finite parameters: linear regression (Freedman, 2009; Berk, 2004) for interpretable relationships, polynomial regression (Theil, 1950) for non-linear patterns, and logistic regression (Tolles & Meurer, 2016) for categorical responses via log-odds modeling. Semi-parametric methods balance both worlds: partial linear models (Engle et al., 1986; Zeger & Diggle, 1994) combine linear parametric terms with non-parametric components, while Generalized Additive Models (Nelder & Wedderburn, 1972) represent responses as sums of smooth univariate functions, preserving additive structure while capturing complex patterns.

G Model Configuration

Table 2: **Model Configurations.**

	125M	**350M**
Training sequence length	512	512
Batch size	32	32
Number of iterations	50k	50k
Dropout prob.	0.0	0.0
Attention dropout prob.	0.0	0.0
Attention head	12	16
Feature dimension	768	1024
Layer number	12	24
Optimizer	Adam	Adam
Optimizer parameter betas	[0.9, 0.95]	[0.9, 0.95]
Learning rate	$6e-4$	$3e-4$
Precision	float32	float32

Table 3: The training loss with different methods, with training length 512 and Books3 dataset

Model	Regression	5K	10K	15K	20K	25K	30K	35K	40K	45K	50K
Transformer	Nadaraya-Watson Regression	4.1948	3.9584	3.8535	3.7920	3.7129	3.6492	3.6007	3.5323	3.4990	3.4701
Cubit	Local Linear Regression	4.1647	3.9389	3.8342	3.7716	3.6959	3.6318	3.5832	3.5155	3.4824	3.4543
Cubit	Kernel Ridge Regression	4.1338	3.9173	3.8141	3.7515	3.6760	3.6118	3.5639	3.4966	3.4643	3.4350

H LOCAL LINEAR REGRESSION

The boundary effect constitutes a fundamental challenge in non-parametric regression, where kernel-based estimators exhibit inflated bias near the domain boundaries. Local Linear Regression (LLR) mitigates this phenomenon, revealing complementary perspectives on adaptive smoothing. Consider the standard NW estimator as a local constant approximation. At a boundary point $\boldsymbol{x}$, the asymmetric neighborhood induced by the kernel truncation yields biased estimates because the conditional expectation varies significantly across the effective support. Mathematically, the NW bias scales as $\mathcal{O}(h^2)$ in the interior but deteriorates to $\mathcal{O}(h)$ near boundaries, where h denotes the effective bandwidth Horbunov & Maiboroda (2024); Cheruiyot (2020); Jo (1997).

Local Linear Regression addresses this deficiency by fitting a local affine model rather than a constant. The LLR estimator solves:

$$\min_{\boldsymbol{\beta}_0,\boldsymbol{\beta}_1} \sum_{j=1}^{N} K_{ij}^{(h)} \left\| \boldsymbol{v}_j^{(h)} - \boldsymbol{\beta}_0 - \boldsymbol{\beta}_1^\top (\boldsymbol{x}_j - \boldsymbol{x}_i) \right\|_2^2, \tag{14}$$

yielding the prediction $\boldsymbol{z}_i^{\text{LLR}} = \boldsymbol{\beta}_0^*$. The inclusion of the linear term $\boldsymbol{\beta}_1$ enables automatic bias correction: the estimator adapts to local trends, reducing boundary bias to $\mathcal{O}(h^2)$ throughout the domain. In matrix form:

$$\boldsymbol{Z}^{\text{LLR},(h)} = \boldsymbol{S}^{(h)} \boldsymbol{V}^{(h)}, \quad \text{where} \quad \boldsymbol{S}^{(h)} = \boldsymbol{e}_1^\top \left(\boldsymbol{X}_i^\top \boldsymbol{W}_i \boldsymbol{X}_i \right)^{-1} \boldsymbol{X}_i^\top \boldsymbol{W}_i, \tag{15}$$

with $\boldsymbol{X}_i = [\mathbf{1}, (\boldsymbol{x}_j - \boldsymbol{x}_i)_{j=1}^N]$ the design matrix and $\boldsymbol{W}_i = \text{diag}(K_{i1}^{(h)}, \ldots, K_{iN}^{(h)})$ the kernel weights. For the naive implementation without a kernel, the local linear regression is very slow, though the performance may be better because of reducing the boundary effect. For future work, we may use better regression method, such as Local Kernel Ridge Regression Han et al. (2022). The result is presented in Table 3.

Future Work. For this work, we rethink the token mixer with regression, so that we propose Cubit with Kernel Ridge Regression to replace Transformer with Local Linear Regression. And we also analyze the performance of Cubit with Local Linear Regression. In the future, we may propose better regression methods to improve the performance, such as Local Kernel Ridge Regression Han et al. (2022). And we should also think about whether there is any other theoretical framework that we could use to explain the token mixer. Also, the Local Linear Regression is relatively slow, compared to Cubit with Kernel Ridge Regression. Therefore, in the future, we may also consider how to speed up the Cubit with Local Linear Regression and Kernel Ridge Regression.

I THE EFFECT OF TRAINING LENGTH AND TRAINING BATCH SIZE

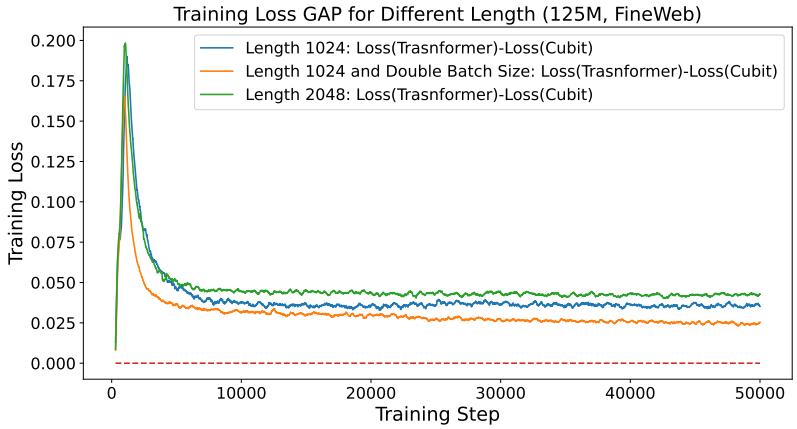


Figure 6: The performance of long training length on the FineWeb dataset, with model parameter 125M. The double batch suggests that the training with double batch size, compared to others.

J EXPERIMENT STATISTICAL SIGNIFICANCE

Table 4: The validation loss with three random seeds, with training length 1024 and FinWeb dataset

Model		5K	10K	15K	20K	25K	30K	35K	40K	45K	50K
Transformer	Mean	4.1768	3.9574	3.8677	3.8142	3.7396	3.6882	3.6409	3.5954	3.5275	3.5185
	variance	0.0173	0.0077	0.0074	0.0120	0.0070	0.0111	0.0038	0.0116	0.0318	0.0203
DeltaFormer	Mean	4.1429	3.9340	3.8471	3.7943	3.7174	3.6674	3.6228	3.5797	3.5114	3.5024
	variance	0.0106	0.0017	0.0111	0.0198	0.0126	0.0137	0.0112	0.0086	0.0248	0.0239
Cubit	Mean	4.1279	3.9159	3.8293	3.7759	3.7000	3.6493	3.6037	3.5596	3.4913	3.4811
	variance	0.0122	0.0096	0.0051	0.0123	0.0080	0.0091	0.0030	0.0110	0.0319	0.0195

K IMPLEMENTATION DETAILS

In this section, we present the implementation of the proposed Cubit module in `PyTorch` Paszke et al. (2019).

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math

### This is the Cubit with Kernel Ridge Regression, which is the default choice of Cubit.
class Cubit(nn.Module):
    def __init__(
        self,
        hidden_size: int,
        attention_head: int,
        eps: float = 1e-10,
```

```
        upper: float = 2.0,
        lower: float = 0.5,
        share: bool = False,
        causal_mask: bool = True
    ):
        super().__init__()

        self.hidden_size = hidden_size
        self.attention_head = attention_head
        self.hidden_size_per_head = hidden_size // attention_head
        self.share = share
        self.causal_mask = causal_mask
        self.eps = eps

        # Linear projections for Q, K, V
        self.q = nn.Linear(hidden_size, hidden_size)
        self.k = nn.Linear(hidden_size, hidden_size)
        self.v = nn.Linear(hidden_size, hidden_size)

        # Optional separate R projection
        if not share:
            self.r = nn.Linear(hidden_size, hidden_size)


        # Learnable bounds for LR scale - shape: (1, attention_head, 1, 1)
        self.lower = nn.Parameter(torch.full((1, attention_head, 1, 1), lower))
        self.upper_scale = nn.Parameter(torch.full((1, attention_head, 1, 1), upper - lower))

        # LR scaling factor per head
        self.LRR = nn.Linear(hidden_size, attention_head)

        # Learnable scale for normalized R - shape: (1, attention_head, 1, 1)
        self.scale = nn.Parameter(torch.ones(1, attention_head, 1, 1))

        # Regularization strength lambda (log space for stability) - shape: (1, attention_head,
            1, 1)
        self.log_lambda = nn.Parameter(
            torch.full((1, attention_head, 1, 1), math.log(eps)),
            requires_grad=True
        )



    def forward(self, x, pos_encoding_function,softmax, mask) -> torch.Tensor:
        b, t, d = x.shape
        device = x.device

        # Q, K, V, R projections
        q=self.q(x).reshape(b,t,self.attention_head,self.hidden_size_per_head).permute(0,2,1,3)
        k=self.k(x).reshape(b,t,self.attention_head,self.hidden_size_per_head).permute(0,2,1,3)
        v=self.v(x).reshape(b,t,self.attention_head,self.hidden_size_per_head).permute(0,2,1,3)
        # Determine R: separate if share=False, else reuse K
        if self.share:
             r=k
        else:
            r=self.r(x).reshape(b,t,self.attention_head,self.hidden_size_per_head).permute
                (0,2,1,3)
        # LR factor: sigmoid-scaled between lower and upper
        # lrr_logits shape: (B, n_head, T, 1)
        lrr_logits = self.lrr(x).reshape(b, t, self.attention_head, 1).permute(0, 2, 1, 3)
        # lrr shape: (B, n_head, T, 1)
        lrr = self.lower + self.upper_scale * torch.sigmoid(lrr_logits)




        # Normalize R with learnable scale
        # norm_r shape: (B, n_head, T, head_dim)
        norm_r = r / torch.norm(r, dim=-1, p=2,
```

```
        keepdim=True) * self.scale

        # add position encoding
        q,k,r,norm_r=pos_encoding_function(q,k,r,norm_r)

        # Sigma inverse: softmax similarity + lambda * I
        # r @ norm_r.transpose(-2, -1) shape: (B, n_head, T, T)
        sigma_inv = softmax(r @ norm_r.transpose(-2, -1), mask)

        # I shape: (1, 1, T, T) -> broadcast to (B, n_head, T, T)
        I = torch.eye(t, device=device).unsqueeze(0).unsqueeze(0)
        lambda_reg = torch.exp(self.log_lambda)  # shape: (1, n_head, 1, 1)
        sigma_inv = sigma_inv + lambda_reg * I

        # Solve linear system for each head
        # rhs shape: (B, n_head, T, head_dim)
        rhs = lrr * v


        # solution shape: (B, n_head, T, head_dim)

        if self.casual_mask:
            solution = torch.linalg.solve_triangular(sigma_inv, rhs,upper=False)
        else:
            solution = torch.linalg.solve(sigma_inv, rhs)



        # Standard attention with Q, K
        A_weights = softmax(
            q @ k.transpose(-2, -1) / math.sqrt(self.hidden_size_per_head),
            mask
        )  # shape: (B, n_head, T, T)

        # Final output
        output = A_weights @ solution  # shape: (B, n_head, T, head_dim)

        # Reshape back to (B, T, hidden_size)
        output = output.permute(0, 2, 1, 3).reshape(b, t, self.hidden_size)

        return output

### This is the Cubit with Local Linear Regression
def llr(
    Q: torch.Tensor,
    K: torch.Tensor,
    V: torch.Tensor,
    attention_mask: torch.Tensor,
    weight_func: callable,
    regularization_eps: float = 1.0,
) -> torch.Tensor:
    """
    Local Linear Regression (LLR): we use the non-centered formulation to avoid the M with shape
         (B,T,T,dim+1)

    Args:
        Q: Query tensor of shape (batch_size, seq_len, dim)
        K: Key tensor of shape (batch_size, seq_len, dim)
        V: Value tensor of shape (batch_size, seq_len, dim)
        attention_mask: Attention mask tensor
        weight_func: Function to compute attention weights from
            similarity scores, such as softmax
        regularization_eps: Ridge regression regularization coefficient. If regularization_eps
             becomes positive infinity, LLR degrades to Nadaraya-Watson Regression, which is
             Transformer.

    Returns:
        Output tensor of shape (batch_size, seq_len, dim)
    """
```

```
    batch_size, seq_len, dim = Q.shape
    scaling_factor = dim ** 0.5  # Override with standard scaling

    # Step 1: Compute similarity scores
    # Standard scaled dot-product attention scores
    similarity = torch.bmm(Q / scaling_factor, K.transpose(-2, -1))  # (B, T, T)
    similarity = similarity.unsqueeze(1)
    W = weight_func(similarity, attention_mask)  # (B, 1, T, T)
    W = W.squeeze(1)  # (B, T, T)

    # Step 2: Construct augmented feature matrix M = [1, K]
    # Append constant term for intercept estimation in linear regression
    ones = torch.ones(batch_size, seq_len, 1, device=Q.device, dtype=Q.dtype)
    M = torch.cat([ones, K], dim=-1)  # (B, T, 1+dim)

    # Step 3: Compute weighted Gram matrix H = M^T @ diag(W)  @ M
    # This corresponds to X^T X in ordinary least squares
    H = torch.einsum('bij,bjk,bjl->bikl', W, M, M)

    # Step 4: Compute weighted moment matrix G = M^T @ diag(W) @ V
    # G[b, i, :, :] = M[b, i, :]^T @ diag(W[b, i, :]) @ V[b, :, :]
    # This corresponds to X^T y in ordinary least squares
    G = torch.einsum('bij,bjk,bjl->bikl', W, M, V)  # (B, T, 1+dim, dim)

    # Step 5: Apply ridge regularization (slope terms only, preserve intercept)
    # Regularization matrix: zero out intercept regularization
    reg_matrix = torch.eye(1 + dim, device=H.device, dtype=H.dtype)
    # Do not regularize the intercept term so that the performance degrades to Nadarray-Watson
        Regression for regularization_eps becomes positive infinity.
    reg_matrix[0, 0] = 0.0
    H = H + reg_matrix * regularization_eps

    # Step 6: Solve Regression
    # theta = (M^T W M + lambda*I)^{-1} M^T W V
    regression_coeffs = torch.linalg.solve(H, G)  # (B, T, 1+dim, dim)

    # Step 7: Evaluate local linear model at query positions
    intercept = regression_coeffs[:, :, 0, :]          # (B, T, dim) - theta_0
    slope = regression_coeffs[:, :, 1:, :]             # (B, T, dim, dim) - theta_{1:dim}

    # f(q) = theta_0 + theta^T * q
    output = intercept + torch.einsum('btdo,btd->bto', slope, Q)

    return output
```