Rotors: Lightweight Geometric Recurrence for Edge Intelligence

Abstract

Applications of AI on edge devices are hampered by hardware limitations. To overcome these limitations, we re-frame sequence modeling through rotations on cooperating $SO(D)$ manifolds. Applied to sequence decoding, this enables stable recurrence over extended contexts that match standard Transformer baselines in perplexity with only $O(1)$ memory complexity at inference. In bypassing the context bottleneck of Transformers, Rotors offer a mathematically robust, deterministic and performant paradigm for real-time edge intelligence.

1 Introduction

Digital sequence processing has traditionally relied on probabilistic state models and, later, recurrent neural architectures. While Hidden Markov Models (HMMs) [1] are constrained by rigid Markovian transition assumptions, Recurrent Neural Networks (RNNs)—such as Long Short-Term Memory (LSTM) networks [3] and Gated Recurrent Units (GRUs) [4]—capture richer temporal dynamics through gated state vectors. However, standard RNNs remain vulnerable to vanishing and exploding gradients, and their unconstrained, dissipative state dynamics degrade long-range stability and expressivity.

The modern Transformer paradigm [5] achieves long-range stability by replacing recurrence with global dot-product self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

Although self-attention enables parallelism during training, autoregressive generation also requires maintaining an expanding Key-Value (KV) cache. The $O(T)$ memory and power requirements of this context scanning pose a major obstacle to real-time edge deployment.

Recent efforts to relieve the quadratic attention bottleneck include linearized attention approximations [6], Key-Value matrix factorizations [7], and State Space Models (SSMs) [8]. While SSMs maintain compact hidden states, their dependence on linear recurrence can induce representation drift over extended contexts, often compensated by hidden dimension expansion and complex input-dependent selection mechanisms.

To address these challenges, we propose **Rotors**, a recurrent sequence model achieving an $O(1)$ inference footprint via isometric geometric updates. Using paired Householder reflections to effect rotations on cooperating manifolds, our system eliminates dynamic history caches and expensive transcendental gating during inference.

2 Recurrent Dynamics

Any recurrent substitute for attention must at each step both *remember* relevant features of new observations and *forget* others. Traditional RNNs achieve this through often complex arrangements of non-linear gates that approximate discrete switching. We instead leverage floating point representational capacity by reframing recurrence as a natively continuous, dynamical state model. Here, remembering and forgetting emerge naturally from a single geometric transformation—eliminating complex gating.

In domains like language modeling the relative sequence of tokens or events is highly consequential. Order agnostic dot-product mechanisms compensate for this with positional encodings [9]. Since rotations on hyperspheres of $D \geq 3$ do not commute, our dynamics instead capture asymmetric dependencies via rotations in $SO(D)$.

However, computing rotations with matrix exponentials would be computationally expensive. They invite numerical drift and risk gradient instability. We adopt an efficient, numerically robust approach. Because $\text{Spin}(D)$ is a double cover of $SO(D)$, orientation-preserving rotations can be achieved through *paired* reflections. Under the Cartan-Dieudonné theorem [10], every orthogonal transformation in D dimensions can be expressed as a finite product of Householder reflec-

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.

tions [11]. By composition, the product of any even number of Householder reflections forms an orientation-preserving orthogonal transformation (a proper rotation) belonging to $SO(D)$, mapping state smoothly across the continuous space without parity inversion.

Accordingly, phase updates of our recurrent heads are driven by a chained sequence of hyperspherical rotations based on paired, input-conditioned Householder reflections:

$$h_t^{(k)} = R_1^{(k)} R_2^{(k)} h_{t-1}^{(k)} \tag{2}$$

where each reflection operator $R_i^{(k)}$ is defined by a unit-normal vector $n_i \in \mathbb{S}^{D-1}$:

$$R_i = I - 2 n_i n_i^T \tag{3}$$

Because Householder matrices are both orthogonal and symmetric ($R^T R = I$), their composition preserves the L_2 norm of $h_{t-1}^{(k)}$ perfectly. Leveraging the inherent isometry of Householder reflections ensures stability over unbounded contexts without radial clamping. Applying these reflections requires only vector dot products and subtractions, bypassing the dissipative gating mechanisms ($\sigma, \tanh$) of traditional RNNs.

3 Cooperating Heads

While the rotational phase $h^{(k)} \in \mathbb{S}^{D-1}$ of a tracking head is stable over indefinitely long contexts, the representational capacity of a single head is nevertheless bounded by its lower-dimensional manifold dimension D. To model more complex phenomena we deploy K parallel heads that cooperate via a higher-dimensional, more expressive *shared state vector* $X \in \mathbb{R}^M$ (where $M > D$).

In each recurrent cycle, prior head phases are extracted from a common, shared state using a learned, head-specific compressive projection or down mapping. Then, token-conditioned rotations of these prior head phases are performed and the shared state is updated by their projection and aggregation. The details of this loop, and the downstream readout, are as follows:

1. **Input:** For discrete state models, all input heads ingest, at each step t, a discrete token index $x_t \in \{0, \ldots, V-1\}$ indexing the learnable embedding matrix $W_{\text{embed}} \in \mathbb{R}^{V \times D}$ or a low-rank approximation thereof.

2. **Prior Phase:** The prior shared state vector $X_{t-1} \in \mathbb{R}^M$ (where the shared dimension spans the concatenated head space, $M = K \cdot D$) is first normalized for numerical stability:

$$X_{t-1}^{\text{bounded}} = \frac{X_{t-1}}{\|X_{t-1}\|_2 + \epsilon} \tag{4}$$

Then, each head extracts its prior phase $h_{t-1}^{(k)} \in \mathbb{R}^D$ via a learned, dimension-reducing down-mapping $W_{\text{down}}^{(k)} \in \mathbb{R}^{D \times M}$ from the global shared state:

$$h_{t-1}^{(k)} = W_{\text{down}}^{(k)} X_{t-1}^{\text{bounded}} \tag{5}$$

3. **Rotation:** To drive the head update, unit-normal vectors $n_1^{(k)}, n_2^{(k)} \in \mathbb{R}^D$ are constructed to span the 2D plane defined by the input embedding vector $u_t = W_{\text{embed}}[x_t]$ (broadcasted across heads) and the head-specific projection vector $v_t^{(k)} = W_v^{(k)} u_t$, where $W_v^{(k)} \in \mathbb{R}^{D \times D}$ is a learned linear transformation matrix:

$$\hat{u}_t^{(k)} = \frac{u_t}{\|u_t\|_2 + \epsilon} \tag{6}$$

Extracting the orthogonal component $v_{\text{perp}}^{(k)} = v_t^{(k)} - \langle v_t^{(k)}, \hat{u}_t^{(k)} \rangle \hat{u}_t^{(k)}$ yields the orthonormal basis component:

$$\hat{v}_t^{(k)} = \frac{v_{\text{perp}}^{(k)}}{\|v_{\text{perp}}^{(k)}\|_2 + \epsilon} \tag{7}$$

Setting $\tau^{(k)} = \|u_t\|_2 \cdot \|v_{\text{perp}}^{(k)}\|_2$, the unit-normal mirror vectors are computed algebraically:

$$n_1^{(k)} = \hat{u}_t^{(k)} \tag{8}$$

$$n_2^{(k)} = \frac{1}{\sqrt{1 + (\tau^{(k)})^2}} \hat{u}_t^{(k)} + \frac{\tau^{(k)}}{\sqrt{1 + (\tau^{(k)})^2}} \hat{v}_t^{(k)} \tag{9}$$

Using these normals, the local head phase $h_{t-1}^{(k)}$ is updated via sequential Householder reflections:

$$h' = h_{t-1}^{(k)} - 2.0 \cdot n_2^{(k)} \langle n_2^{(k)}, h_{t-1}^{(k)} \rangle \tag{10}$$

$$h_t^{(k)} = h' - 2.0 \cdot n_1^{(k)} \langle n_1^{(k)}, h' \rangle \tag{11}$$

This formulation executes an exact rotation without transcendental trigonometric evaluations or state-dependent tracking loops.

4. **Aggregation:** The rotated phases of all K heads $h_t^{(k)} \in \mathbb{R}^D$ are flattened and concatenated directly to form the updated shared state vector $X_t \in \mathbb{R}^M$ (where $M = K \cdot D$):

$$X_t = \text{concat}\left(h_t^{(1)}, h_t^{(2)}, \ldots, h_t^{(K)}\right) \tag{12}$$

The resulting shared state X_t is carried forward directly as the recurrent context for step $t+1$, isolating the recurrent loop from readout to ensure high throughput and prevent feedback pollution.

This concatenation should be understood as a special case of aggregation by projection and summation. The recurrent shared state can alternatively be updated by a dimension expanding up-mapping operator $W_{\text{up}}^{(k)} \in \mathbb{R}^{M \times D}$, additively forming the shared state X_t:

$$X_t = \sum_{k=1}^{K} W_{\text{up}}^{(k)} h_t^{(k)} \tag{13}$$

5. **Readout:** Output logits are generated by evaluating the individual head phases against the global token embedding space $W_{\text{embed}} \in \mathbb{R}^{V \times D}$ completely downstream of the recurrent update loop. Because the shared state X_t is formed by concatenating head phases ($M = K \cdot D$), each head's phase implicitly encapsulates a specialized projection of the shared context toward the token space.

   Letting $\hat{h}_t^{(k)} = h_t^{(k)} / (\|h_t^{(k)}\|_2 + \epsilon)$ denote the normalized phase of head k and $\hat{W}_{\text{embed},v}$ denote the normalized token embedding vectors, each head independently computes token similarities scaled by head-specific variance parameters derived from learned log-apertures $\log \sigma^{(k)}$ (parameterized as $(\sigma^{(k)})^2 = \exp(\log \sigma^{(k)})^2$). The final vocabulary logits are obtained via an averaged consensus across all cooperating heads:

$$\text{logits}_v = \frac{1}{K} \sum_{k=1}^{K} \frac{\alpha \cdot \langle \hat{h}_t^{(k)}, \hat{W}_{\text{embed},v} \rangle - 1.0}{(\sigma^{(k)})^2} \tag{14}$$

   By keeping the readout strictly downstream of the update loop, recurrent dynamics are isolated from prediction feedback and pollution. Learned apertures encourage heads to specialize. Narrow apertures act as sharp high-frequency filters, while broader apertures track slower-moving contextual patterns.

Discussion The shared state X_t serves as an information-preserving and sharing conduit for the heads and the down mapping plays an important role in mixing and dissipation. While the global shared state X_{t-1} is explicitly bounded via normalization rather than isometric flow, the down-mapping $W_{\text{down}}^{(k)}$ acts as a parameterized projection that extracts the relevant subspace components for each head's local manifold.

Whether by concatenation or up-mapping $W_{\text{up}}^{(k)}$ aggregation functions with down-mapping $W_{\text{down}}^{(k)} \in \mathbb{R}^{D \times M}$ as an asymmetric autoencoder pair bridging between the shared state ($\mathbb{R}^M$) and the rotational phases of individual heads ($\mathbb{R}^D$). Parameterizing and learning these operators independently preserves dedicated optimization degrees of freedom and ensures robust gradient flow and superior empirical convergence.

It is important to distinguish between local and global dynamics in Rotors. Exact isometry and L_2-norm preservation apply strictly to the internal phase update of each head via Householder reflections ($\|h_t^{(k)}\|_2 = \|h_{t-1}^{(k)}\|_2$). Conversely, the global aggregation and down/up-mappings intentionally introduce non-isometric mixing to allow information to flow between cooperating heads. Thus, stability is guaranteed locally by the geometry of $SO(D)$, while expressivity is maintained globally through the shared state conduit.

Parameter Complexity Parameters scale linearly with respect to head count and manifold dimensions, maintaining a tight memory footprint ideal for embedded edge deployments. Given the vocabulary size V, manifold dimension D, shared state dimension M (where $M = K \cdot D$), and head count K, the learnable parameters of Rotors are structured as detailed in Table 1.

Table 1: Breakdown of learnable parameters (Head-specific apertures).

Component	**Symbol**	**Shape**
Token Embeddings	W_{embed}	$V \times D$
Item Projections	W_v	$K \times D \times D$
State Down Mappings	W_{down}	$K \times D \times M$
Initial Head Phases	h_0	$K \times D$
Readout Apertures	$\log \sigma$	K

4 Streaming Inference

Rotors, RNNs and Transformers all loop to perform autoregressive generation. For generation on edge devices Rotors' simple recurrence is advantageous. With shared state X_t as the sole recurrent carrier, activation is lightweight and strictly bounded. Rotors can safely process long or growing contexts without padding, chunking or recompilation tricks. In particular, because E_{readout} depends strictly on static base weights, it can be hoisted and reduced

to a single matrix-vector dot product prior to autoregressive generation or training batches. This preserves $O(1)$ latency and ultra-high throughput.

5 Streaming Training

Parallelism affords Transformers an unquestionable training advantage, especially when GPUs are available. Nevertheless, a capability to learn from scratch or even fine-tune directly on resource-constrained edge devices would offer many practical benefits.

So far, however, training directly on edge devices has proven challenging due to unpredictable memory requirements, particularly for extended contexts. Backpropagation Through Time (BPTT) [2] typically requires storing activations and gradients during a forward pass. For recurrent models like Rotors, this entails a linear accumulation of activation memory $\mathcal{O}(T \cdot D)$. For Transformers, the requirement becomes quadratic $\mathcal{O}(T^2)$ in both time and memory.

Curiously, however, Rotors' rotational state transitions can be algebraically reversed by applying inversing Householder reflections. While not implemented in the current work, the potential for time-reversed training with fixed memory has important implications not only for learning on edge devices but for continuous, lifelong and opportunistic learning strategies at all scales.

6 Empirical Evaluation

In this foundational work we explore the potential viability of Rotors as an alternative for edge intelligence. As a preliminary evaluation, we benchmarked CPU mode JAX implementations of Rotors against GPT-2 [12] Transformers on a Raspberry Pi 5.

In this proof-of-concept both models were trained as character-level decoders with the same number of heads ($K = 4$) and embedding dimension ($D = 48$) over a range ($T \in \{16, 32, 64, 128, 256, 512\}$) of context window sizes on the familiar H.G. Wells' *The Time Machine* [15] reference corpus ($V = 70$). The models were trained on 10,000 fixed-size mini-batches ($B = 16$) and best perplexity on a held-out validation sample is reported. Latencies and memory usage were tracked through a shared benchmarking and training harness.

Complete source-code of these benchmarks is provided for reproducibility purposes in the supplementary materials of this submission.

Table 2: Benchmark Configuration.

Metric	**GPT-2**	**Rotors**
Heads (K)	2 x 2	4 (star)
Dimension (D)	48	48
Shared Dimension (M)	-	64
Batch Size (B)	16	16
Vocabulary (V)	70	70
Total Parameters	88,006	**49,636**
Training Steps	10,000	10,000

Table 3: Training Performance Comparison

Window	Model	Best Perplexity (ppl)	Secs/Batch	Train RSS
16	Rotors	4.877	0.017	442.27 MB
16	GPT2	4.491	0.018	526.19 MB
32	Rotors	4.338	0.027	440.64 MB
32	GPT2	4.135	0.025	516.23 MB
64	Rotors	4.135	0.047	433.09 MB
64	GPT2	4.103	0.042	535.19 MB
128	Rotors	4.545	0.086	449.00 MB
128	GPT2	4.343	0.097	591.28 MB
256	Rotors	4.662	0.172	483.45 MB
256	GPT2	4.442	0.289	720.20 MB
512	Rotors	4.890	0.338	533.39 MB
512	GPT2	4.778	0.878	1167.62 MB

Table 4: Inference Performance Comparison

Window	Model	Tokens/Sec	Infer RSS
16	Rotors	4401	220.14 MB
16	GPT2	1494	437.14 MB
32	Rotors	4822	218.14 MB
32	GPT2	1186	431.19 MB
64	Rotors	5019	219.62 MB
64	GPT2	902	431.23 MB
128	Rotors	5047	217.64 MB
128	GPT2	490	433.19 MB
256	Rotors	5024	217.62 MB
256	GPT2	204	444.67 MB
512	Rotors	5067	219.14 MB
512	GPT2	66	475.78 MB

Training Footprint Both models benefit heavily from framework optimizations of BPTT during training. Though dominated by framework overhead, training memory for longer sequences grew more slowly with Rotors than it did for GPT2. As expected, training time per batch for Rotors grew linearly in context length whereas increases were superlinear for GPT2. Advantaged by parallelism even on CPU, GPT2 requires less training time (seconds per training batch) at shorter contexts, but is bested by Rotors on longer contexts.

Model Accuracy Despite operating with 43.6% fewer total parameters (49,636 vs. 88,006) in the benchmark configuration, Rotors achieves only slightly worse perplexities on the test context windows from $T = 16$ to $T = 512$. Alternative hyperparameter configurations occasionally match or outperform GPT2 in perplexity. Qualitatively, both Rotors and GPT2 generate text having similar word and phrase morphology. While these results remain preliminary, they indicate that Rotors may be able to capture complex sequence dependencies nearly as effectively as causal self-attention even at a substantially reduced parameter count.

Inference Footprint Although trained with JAX, Rotors uses numpy at inference for efficiency. Rotors constant-memory recurrence is an important advantage for mission-critical usage on edge devices. As demonstrated in Table 4, Rotors uses approximately half the RSS memory of GPT2 and its peak footprint ($\sim$ 210 MB vs. $\sim$ 435–478 MB) is essentially constant across all context window sizes. This confirms $O(1)$ autoregressive generation.

Generative Throughput While GPT2's token rate degrades severely with sequence length, unencumbered by transcendental gating and framework overhead, Rotors is many times faster. Thanks to avoidance of quadratic self-attention Rotors' autoregressive throughput increases asymptotically with sequence length, presumably as the relative overhead of initialization is reduced – reaching **5,067 tokens/sec** at $T = 512$. This exceptional performance benefits from numpy optimizations but is nevertheless indicative of the performance that edge devices can hope to achieve with ports of this architecture.

7 Future Directions

While this work establishes the foundational mechanics of Rotors, several open avenues remain for future exploration.

- **Reversibility:** Learning strategy implications from potential O(1) state reversal.
- **Adaptations:** Adaptation to deeper models, head topologies, word token vocabularies etc.
- **Telemetry:** Signal processing and telemetric device monitoring
- **Transfer Learning**: Fine-tuning of Transformer embeddings
- **Photonics:** Natural pathway to quantum-optical intelligence from Householder reflections.

Generative AI Disclosure

In accordance with the ICLR policy on the use of generative AI tools, we provide the following disclosure regarding our workflow:

- **Ideation:** Generative AI models were used as a foil for brainstorming, exploring mathematical interpretations, and refining the core conceptual framing of geometric tracking on manifolds. All underlying hypotheses, theoretical frameworks, and structural design choices were critically evaluated, selected, and verified by the author.
- **Implementation:** Generative AI tools assisted in drafting, debugging, and optimizing software implementation and script architectures (including JAX execution pipelines). All generated code blocks, mathematical routines, and performance benchmarks were manually tested, audited, and verified for correctness.
- **Paper Writing:** Generative AI tools were utilized to assist in drafting prose, polishing phrasing, and structuring sections of this manuscript.

Verification and Responsibility: We have thoroughly reviewed, edited, and verified all AI-assisted text, derivations, code, and claims to ensure scientific accuracy and integrity. The author takes full and ultimate responsibility for the entire contents of this work.

References

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[2] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[6] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast autoregressive transformers with linear attention," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020, pp. 5156–5165.

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

[8] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752v2*, 2023–2024.

[9] J. Su, A. Murtadha, Y. Lu, B. Wen, S. Pan, and Y. Liu, "RoFormer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024 (arXiv preprint arXiv:2104.09864, 2021–2023).

[10] Jean Gallier. *Geometric Methods and Applications: For Computer Science and Engineering*. Springer, 2nd edition, 2011.

[11] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the ACM (JACM)*, vol. 5, no. 4, pp. 339–342, 1958.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[13] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[14] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, Y. Katariya, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," *Software available from github.com/google/jax*, 2018.

[15] H. G. Wells, *The Time Machine*, Heinemann, London, 1895.