MOSAIC:

Multiscale Oscillator State + Associative Indexed Cache for No-Attention Language Modeling

Daniel Owen van Dommelen
Independent Research
`theapemachine@gmail.com`

December 2025

Abstract

Attention with a per-token KV cache implements one extremely general trick: *store a vector for every past token, then do content-based lookup over all of them.* This yields perfect copying, avoids lossy compression of long contexts, and provides a substrate for in-context learning—but it also incurs $O(T)$ memory growth and pairwise interactions.

We propose **MOSAIC** (**M**ultiscale **O**scillator **S**tate + **A**ssociative **I**ndexed **C**ache), a streaming causal language model that is genuinely *no attention, no KV cache.* MOSAIC decomposes "memory" into fixed-size explicit data structures controlled by a small neural controller: (i) a cheap local causal mixer for syntax/short patterns, (ii) a multiscale continuous state bank that preserves long-range intent at constant memory, and (iii) a hard-addressed associative cache that enables fast exact-ish recall without scanning the past. An optional n-gram continuation cache provides verbatim copying/continuation behavior with $O(1)$ table access.

This paper is a *production-first* report: MOSAIC is implemented as first-class manifest-addressable components in Caramba (`config/presets/mosaic.yml`), enabling systematic ablations (cache sizes, timescales, write sparsity) on consumer hardware. We focus on the architectural spec, its streaming inference loop, and the evaluation plan for measuring copying fidelity, long-range dependency retention, and laptop-feasible latency/memory.

Keywords: language modeling, no-attention, external memory, associative cache, n-gram cache, state space, long context, streaming inference, continual learning

1 Introduction

Transformer attention [12] is often described as a token-mixing operator, but in practice it is also a memory system: it stores a vector per past token and performs content-based lookup against the entire history. The KV cache makes this explicit by persisting those vectors across decode steps.

1.1 Attention as a "store everything" memory

The attention+KV-cache pattern provides three high-value capabilities:

1. **Copying from context:** verbatim continuation and exact recall (names, brackets, numbers).
2. **Long-range dependencies without forced compression:** past tokens remain individually addressable.
3. **A substrate for in-context learning:** rapid pattern acquisition via retrieval-like behavior.

However, it also brings two costs that dominate on consumer hardware: memory that grows as $O(T)$ with context length and dense interactions that scale with history.

1.2 The MOSAIC hypothesis

MOSAIC starts from a different decomposition: *stop trying to make the network itself be the memory.* Instead, make the network a controller for a small number of explicit, fixed-size data structures. The network decides what to keep, how to compress it, and how to retrieve it; the memory is sublinear, lossy, and constant-size.

This paper focuses on a concrete, implementable design that is streaming causal, attention-free, and laptop-feasible.

1.3 Contributions

1. We specify **MOSAIC**, a no-attention/no-KV-cache streaming LM with constant memory w.r.t. context length.
2. We introduce a **multiscale continuous state bank** (leaky integrators) that preserves long-range intent at $O(Kd)$ memory/compute.
3. We introduce a **hard-addressed associative cache** (fixed hash table) that provides $O(1)$ lookup/update as a replacement for "store all KV pairs".
4. We implement MOSAIC as **manifest-addressable components** in Caramba (`config/presets/mosaic.yml`) to enable systematic ablations and laptop experiments.

2 Related Work

Attention reductions. Most "attention-efficient" work retains the core primitive (softmax over past tokens) while changing how it is computed. Low-rank or projected variants reduce interaction cost (e.g., Linformer [13]); kernel and hashing methods approximate attention without enumerating all pairs (e.g., Performer [3], Reformer [8]); sparse/local patterns reduce compute while preserving some long-range access (e.g., Longformer [2], BigBird [14]). These methods can reduce compute, but autoregressive decoding typically still benefits from (or requires) history-dependent state that grows with context length.

KV-cache optimization. Orthogonal work reduces the *storage format* of KV caches via head sharing (MQA/GQA) [11, 1], latent caching [5], or quantization [7, 9]. MOSAIC targets a different axis: removing the "store one vector per token" mechanism entirely, replacing it with fixed-size explicit memory structures.

Explicit memory and cache language models. Classical compression/prediction schemes and cache language models motivate the idea that exact copying and repetition can be handled by algorithmic structures at constant-time access, while a neural model provides generalization. MOSAIC adopts this hybrid view: continuous state for general context plus discrete cache structures for fast recall/continuation.

State-space models and external memory. Modern recurrent/state-space designs can provide strong long-range *influence* with constant memory (e.g., Mamba [6], RWKV [10], Griffin [4]), but they are not designed for precise retrieval of discrete facts. Conversely, explicit read/write memories have long promised algorithmic recall, but training stable addressing policies is historically challenging. MOSAIC treats these as complementary: continuous state for "vibe/intent" and explicit data structures for copying and discrete recall.

3 Methodology

3.1 Overview

MOSAIC is a streaming causal language model that replaces attention+KV-cache with explicit constant-size state:

1. **Local mixer**: short-range token interactions (depthwise causal conv + gated MLP).
2. **Multiscale continuous state**: long-range intent via a bank of leaky integrators.
3. **Associative indexed cache**: hard-addressed fixed-size hash memory for fast recall.

An optional n-gram continuation cache provides cheap verbatim continuation.

3.2 Local mixer: causal convolutional mixer

Let $x_t \in \mathbb{R}^d$ be the residual stream and $u_t = \mathrm{RMSNorm}(x_t)$. The local mixer applies a depthwise causal convolution over a window of k activations:

$$\tilde{u}_t = \mathrm{DWConv}_k(u_{\le t}), \qquad m_t = \sigma(W_g \tilde{u}_t) \odot \tilde{u}_t, \qquad \Delta_t^{\mathrm{local}} = W_2\, \phi(W_1 m_t).$$

We update $x_t \leftarrow x_t + \Delta_t^{\mathrm{local}}$.

3.3 Multiscale continuous state

Maintain K state vectors $s_{k,t} \in \mathbb{R}^d$ with learnable decays $\lambda_k \in (0,1)$:

$$s_{k,t+1} = \lambda_k \odot s_{k,t} + W_k^{\mathrm{in}} u_t, \qquad g_t = W^{\mathrm{out}}\, [s_{1,t}; \ldots; s_{K,t}].$$

3.4 Associative indexed cache (hard-addressed)

Maintain a fixed table $M \in \mathbb{R}^{B \times D_m}$ (optionally H independent routes). At each step, route to one (or a small constant number of) bucket indices b_t and read:

$$r_t = W_r\, M[b_t].$$

With a saliency gate $p_t = \sigma(w^\top u_t)$, write sparsely:

$$M[b_t] \leftarrow (1 - \eta p_t)\, M[b_t] + (\eta p_t)\, W_v u_t.$$

This replaces "store one KV per past token" with $O(1)$ lookup/update into fixed memory.

3.5 The hidden bottleneck: addressing

The most failure-prone part of MOSAIC is not the memory size; it is the *addressing problem.* Attention performs content-based retrieval by directly comparing Q against all K. A hard-addressed table requires the controller to generate the correct address from the current state. If relevant information has decayed from the continuous state, the model may be unable to produce the address that would allow it to retrieve the missing information.

Mitigation: learnable discretized routing (product-quantized VQ). Instead of fixed hashing, we use a learned router that remains $O(1)$ at inference. A practical choice is product-quantized VQ routing: project u_t to a small vector, split into G groups, and assign each group to one of K codes via nearest-neighbor lookup. The resulting tuple defines a bucket address in a K^G space (e.g., K=128, G=2 $\Rightarrow$ 16,384 buckets). Straight-through estimators allow end-to-end training while preserving discrete routing.

Neighbor reads for drift tolerance (constant factor). To improve recall when embeddings shift, we read a small constant neighborhood: top-2 codes per group yields at most 2^G candidate buckets (e.g., 4 when $G = 2$). This preserves constant-time access while significantly improving robustness under drift.

Mitigation: set-associative buckets (tags + slots). A fixed table can be made more robust by storing multiple entries per bucket (small associativity) and attaching a lightweight tag/key to each entry. Reads then compare only within the bucket. This preserves $O(1)$ access while reducing destructive overwrites.

3.6 Training: making the memory get used

Hard addressing and sparse writes create an optimization hazard: the model can learn to rely only on smooth-gradient paths (local mixer + state bank) and ignore the associative cache. Practical training therefore benefits from a curriculum and auxiliary objectives:

- **Write curriculum:** begin with heuristic writes (e.g., punctuation, rare tokens, entity-like tokens) and gradually hand control to a learned saliency gate.
- **Write sparsity:** regularize expected write rate to keep memory efficient and avoid thrashing.
- **Utility prediction:** train a head to predict whether a write will be useful k steps later (self-supervised credit assignment).
- **Collision pressure:** discourage mapping dissimilar contexts to the same bucket (contrastive loss on address codes).

Forced-read dropout. To prevent the model from ignoring memory, we explicitly drop the local mixer contribution on a small fraction of tokens/spans during training, forcing prediction to depend on the state bank and retrieved memory.

Utility prediction and contrastive recall. We add a utility head that predicts whether a write will be queried in the near future, and an InfoNCE-style auxiliary that makes retrieved vectors predictive of future hidden state. These losses provide direct gradients to make the memory pathway carry useful information.

Stage D2: scheduled sampling for student-controlled memory. After teacher-forced memory (D1), we transition to student-controlled routing and gating using scheduled sampling: with probability p_t we apply teacher actions (write gate/address, read address), otherwise we use the model's own router outputs. We anneal p_t from 1.0 to 0.0 over training. For discretized routers (e.g., VQ routing), we additionally supervise router decisions with per-group cross-entropy over code assignments.

3.7 Fusion

We combine streams with learned gates:

$$x_t \leftarrow x_t + \Delta_t^{\text{local}} + \sigma(a^\top u_t)\, g_t + \sigma(b^\top u_t)\, r_t.$$

3.8 Optional n-gram continuation cache

We maintain a fixed-size N-gram table over token IDs that yields a sparse next-token distribution and add it as a logit bias:

$$\ell_t \leftarrow \ell_t + \alpha \log(p_{\text{ng}} + \varepsilon).$$

3.9 Streaming inference

Per token, MOSAIC updates only fixed-size buffers and tables:

Algorithm 1 MOSAIC decode step (no attention, no KV cache)

```
Input $x_t$, states $\{s_k\}$, conv buffer, memory $M$
$u_t \leftarrow \mathrm{RMSNorm}(x_t)$
$\Delta_t^{\mathrm{local}} \leftarrow \mathrm{LocalMixer}(u_t, \mathrm{buf})$
$g_t \leftarrow \mathrm{StateBank}(u_t, \{s_k\})$
$r_t \leftarrow \mathrm{HashRead}(u_t, M)$
$\ell_t \leftarrow \mathrm{LMHead}(x_t + \Delta_t^{\mathrm{local}} + \mathrm{gate}(g_t, r_t))$
$\ell_t \leftarrow \ell_t + \mathrm{NGramBias}(\cdot)$ (optional)
Update $M$ on sparse write events
Sample $x_{t+1} \sim \mathrm{softmax}(\ell_t)$
```

3.10 Implementation (Caramba)

MOSAIC is implemented as manifest-addressable layers: `MosaicBlockLayer` (`layer/mosaic/block.py`) and `MosaicNGramCacheLogitsLayer` (`layer/mosaic/ngram_cache.py`).

4 Experiments

4.1 Setup

We define a laptop-feasible MOSAIC baseline in `config/presets/mosaic.yml` and evaluate:

- **Language modeling quality**: held-out perplexity on token shards.
- **Copying**: targeted synthetic tests (repeated spans, bracket closure, identifier reuse).
- **Long-range constraints**: instruction retention with long distractor spans.
- **Efficiency**: tokens/sec and peak resident memory during streaming decode.

4.2 Stress tests (what should break first)

To make MOSAIC falsifiable, we include targeted adversarial evaluations:

- **Hash collision stress:** long contexts with many distinct entities/facts to quantify interference.
- **Few-shot in-context learning probes:** measure whether MOSAIC can acquire and apply a new pattern from a handful of examples without gradient updates.
- **Non-verbatim manipulation:** tasks like "repeat this list sorted" to distinguish mere copying from compositional reuse.

4.3 Ablations

We ablate memory mechanisms to attribute behaviors:

- **-hash memory**: disable associative cache reads/writes.
- **-state bank**: disable multiscale long state.
- **+n-gram cache**: enable continuation cache logit bias.

Table 1: Planned MOSAIC results (placeholders).

Variant	PPL	Copy score	tok/s	Peak MB
MOSAIC	–	–	–	–
MOSAIC (-hash)	–	–	–	–
MOSAIC (-state)	–	–	–	–
MOSAIC (+n-gram)	–	–	–	–

5 Discussion

5.1 Trade-offs

Hash memories offer $O(1)$ access but introduce collisions and interference. Multiscale state provides stable long-range influence but cannot perform exact recall. The n-gram cache provides near-perfect continuation for repeated strings but is not a substitute for semantic retrieval.

5.2 Why the n-gram cache is disproportionately high-yield

Many "hard" failures in small on-device models are not deep reasoning errors; they are failures of verbatim continuation (identifiers, brackets, repeated substrings). A classical n-gram cache can supply this cheaply as an additive logit bias, freeing neural capacity for generalization.

5.3 Continual learning as a first-class design axis

MOSAIC naturally separates learning timescales:

- **Fast (no gradients):** memory writes during inference (session adaptation).
- **Medium (tiny gradients):** consolidate frequently retrieved behaviors into small adapters (e.g., low-rank side modules), using replay buffers and regularization toward the base model.
- **Slow (offline):** full training runs informed by logged memory misses, useful writes, and tool traces.

5.4 Structured internal control (a controller DSL)

If the model is a controller over memory and tools, internal reasoning need not be natural language. A small typed action DSL (memory ops, tool calls, state updates) can reduce failure modes and enable constrained decoding and verification. Natural language becomes a rendering layer rather than the core compute substrate.

6 Conclusion

MOSAIC operationalizes a simple principle: dense learned computation is expensive; explicit algorithmic memory is cheap. By turning attention's implicit memory into explicit fixed-size data structures, we obtain a streaming LM architecture whose memory does not grow with context length.

Statements and Declarations

Conflict of Interest. The author declares no competing interests.

Data Availability. All datasets used in this study are publicly available.

Funding. This research was conducted without external funding.

A Manifest snippet

The reference preset for this paper is `config/presets/mosaic.yml`.

B Implementation notes

Core implementation: `layer/mosaic/block.py` and `layer/mosaic/ngram_cache.py`.

References

[1] 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. *EMNLP*, 2023.

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

[3] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*, 2021. arXiv:2009.14794.

[4] Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024.

[5] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[6] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[7] Coleman Hooper et al. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

[8] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020. arXiv:2001.04451.

[9] Junyan Li, Yang Zhang, Muhammad Yusuf Hasan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression. *arXiv preprint arXiv:2506.18879*, 2025. ICML 2025 poster.

[10] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing RNNs for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

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

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

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

[14] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.