MOSAIC:

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

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

January 2026

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.

Beyond token-stream language modeling, we also implement an **event-native interface**: external interaction is a JSON `EventEnvelope` encoded as byte tokens ($\{0, \ldots, 255\}$) via an `EventEncoder`/`EventDecoder`. This reframes decoding as **VM time-steps** over an event stream, and enables supervising a differentiable control surface (opcodes, memory gates, registers) with auxiliary losses on synthetic event traces. Phase 1 (event traces + opcode/memory gating) and Phase 2 (commitment lifecycle via `commitment_delta`/`commitment_id`) were both trained successfully, converging to losses of ~ 0.36 and ~ 0.32.

This paper is a *production-first* report: MOSAIC and its event-native runtime are implemented as manifest-addressable components in Caramba, with presets `mosaic.yml`, `mosaic\_event\_native.yml`, and `mosaic\_commitment.yml`. This enables systematic ablations 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, event-native, no-attention, external memory, associative cache, control surfaces, differentiable VM, long context, streaming inference, continual learning

1 Introduction

Transformer attention [?] 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 Event-native interaction and the Differentiable VM view

In Caramba, MOSAIC is operated in an **event-native** regime: the external interface is a JSON envelope (`EventEnvelope`), while the internal compute substrate still operates on discrete steps (tokens)—now interpreted as **VM clock cycles**. Events are encoded into a byte-token stream, processed by MOSAIC, and decoded back into JSON. This enables training and inference to be driven by structured event traces rather than prompt-formatted text.

We also treat MOSAIC as a **differentiable virtual machine (dVM)**: the model emits soft control signals (logits) corresponding to operations (opcodes) and commitment state changes. Crucially, these control surfaces are supervised with auxiliary losses rather than learned via reinforcement learning.

1.4 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 define and implement an **event-native interface** (JSON `EventEnvelope` $\leftrightarrow$ byte tokens) and an in-memory **EventBus** for peer-to-peer event routing.
5. We implement a **dVM control surface** (opcodes, register gates, and commitment deltas) and supervise it with auxiliary losses on synthetic event traces.
6. We implement MOSAIC as **manifest-addressable components** in Caramba (`config/presets/mosaic.yml`, `config/presets/mosaic\_event\_native.yml`, `config/presets/mosaic\_commitment.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 [?]); kernel and hashing methods approximate attention without enumerating all pairs (e.g., Performer [?], Reformer [?]); sparse/local patterns reduce compute while preserving some long-range access (e.g., Longformer [?], BigBird [?]). 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) [?, ?], latent caching [?], or quantization [?, ?]. 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 [?], RWKV [?], Griffin [?]), 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 Event-native interface: JSON events as byte tokens

MOSAIC can be trained and operated on structured event streams. We define an `EventEnvelope` as a minimal JSON-serializable contract (required: `type`, `payload`, `sender`; optional: `priority`, `budget_ms`, `id`, `ts`, plus Phase 2 commitment meta-fields). A deterministic serialization is encoded as UTF-8 bytes, yielding a token stream in $\{0, \dots, 255\}$.

$$\mathbf{b} = \text{utf8}(\text{json}(\text{EventEnvelope})) \in \{0, \dots, 255\}^L.$$

This reframes "tokens" as **VM time-steps** required to process an event. In Caramba, event datasets generate synthetic `EventEnvelope` traces and supervise control signals at aligned byte spans.

3.3 Differentiable VM control surfaces (opcodes, registers, commitments)

In addition to next-token logits, MOSAIC emits auxiliary logits that represent a soft instruction set and related control state:

- **Opcodes:** $\text{logits}_{\text{op}} \in \mathbb{R}^{B\times T\times|\mathcal{O}|}$ (e.g., `READ_MEM`, `WRITE_MEM`), optionally used as a differentiable control surface that modulates compute paths.
- **Registers:** a small non-decaying register file with a write-enable gate and slot selection, supervised with aligned teacher signals.
- **Commitments (Phase 2):** $\text{logits}_{\text{c}} \in \mathbb{R}^{B\times T\times 3}$ over $\{\text{close}, \text{neutral}, \text{open}\}$, mapped to $\{-1, 0, +1\}$.

These heads are trained with auxiliary losses (cross-entropy or BCE), enabling stable supervision of control behavior without RL.

3.4 Local mixer: causal convolutional mixer

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

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

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

3.5 Multiscale continuous state

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

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

3.6 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.

3.7 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.

3.8 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).
- **Opcode and commitment supervision:** train control-surface heads from synthetic traces (event-native datasets) via cross-entropy at aligned spans.
- **State stability:** regularize learned state-bank decay rates to stay within a healthy operating band to prevent early saturation.

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.

3.9 Fusion

We combine streams with learned gates:

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

3.10 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:

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

3.11 Streaming inference

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

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

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

3.11.1 Event-driven runtime (Mode B commitment injection)

At the system level, MOSAIC is run as an event processor. Given an inbound `EventEnvelope`, we encode JSON to byte tokens, run the model autoregressively until a complete JSON object is produced, decode to an `EventEnvelope`, then inject commitment meta-fields from the auxiliary head logits ("Mode B") and update a runtime commitment ledger.

Algorithm 2 Event-native runtime step (JSON bytes + Mode B commitments)

```
1: Input event $e$ (EventEnvelope)
2: Encode $e \rightarrow \mathbf{b} \in \{0, \ldots, 255\}^L$ and append delimiter
3: Feed bytes to MOSAIC with persistent streaming state
4: Autoregress bytes until buffer ends with } and JSON decode succeeds
5: Decode buffer to response event $r$
6: $\delta \leftarrow \arg\max(\text{mosaic\_commitment\_logits}) - 1$
7: Set $r$.commitment_delta $\leftarrow \delta$ (Mode B)
8: Update commitment ledger and publish $r$ to the EventBus
```

3.12 Implementation (Caramba)

MOSAIC is implemented as manifest-addressable layers and core primitives:

- `MosaicBlockLayer` (`layer/mosaic/block.py`) and `MosaicMemory` (`layer/mosaic/memory.py`)
- `MosaicNGramCacheLogitsLayer` (`layer/mosaic/ngram\_cache.py`) (optional inference-time logit bias)
- Event primitives: `EventEnvelope` (`core/event.py`), `EventEncoder`/`EventDecoder` (`core/event\_codec.py`), and `EventBus` (`core/event\_bus.py`)
- Event-native curricula: `data/event\_trace.py` (`dataset.mosaic_event_traces`)

- Phase 2 runtime ledger and handler: `core/commitments.py` and `infer/event\_runtime.py`

4 Experiments

4.1 Setup

We define:

- a token-stream MOSAIC baseline in `config/presets/mosaic.yml`,
- an event-native Phase 1 preset in `config/presets/mosaic\_event\_native.yml`, and
- an event-native Phase 2 commitment preset in `config/presets/mosaic\_commitment.yml`.

We 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.
- **Event-native learning**: final training loss on synthetic event traces (Phase 1/2 curricula).

Table 1: Event-native synthetic curriculum results (Caramba runs).

Phase	**Capability**	**Final loss**
Phase 1	Event traces + opcodes + memory gating	0.36
Phase 2	Commitment lifecycle ($\Delta \in \{-1, 0, +1\}$)	0.32

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 2: Planned downstream MOSAIC results (LM evaluation; pending).

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. In the current MOSAIC implementation, this idea is realized as **soft control surfaces** (opcodes, register gates, and commitment deltas) trained with auxiliary losses and optionally wired into execution as differentiable gates.

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

Reference presets: `config/presets/mosaic.yml` (token-stream), `config/presets/mosaic\_event\_native.yml` (Phase 1), `config/presets/mosaic\_commitment.yml` (Phase 2).

B Implementation notes

Core implementation: `layer/mosaic/block.py`, `layer/mosaic/memory.py`, and event primitives in `core/`.

References