The Cognitive Control Plane (CCP):

Explicit Memory and Control for Persistent Event-Native AI
with MOSAIC as a no-attention neural control kernel

Daniel Owen van Dommelen

Independent Research

`theapemachine@gmail.com`

January 2026

Abstract

Modern LLMs are strong pattern learners, but weak *systems*: they lack persistent control, explicit memory boundaries, determinism/auditability, and safe tool acquisition. 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*—but this conflates memory, retrieval, and computation, and incurs $O(T)$ memory growth and dense interactions.

We present the **Cognitive Control Plane (CCP)**, a runtime architecture for long-lived, event-driven intelligent systems. CCP separates concerns into (i) an event-native substrate (JSON `EventEnvelope` $\leftrightarrow$ byte tokens), (ii) explicit memory/control surfaces with deterministic trace and replay, and (iii) a test-driven tool lifecycle (definition, sandboxed execution, unit tests, evaluator gates). At its core we use **MOSAIC** (**M**ultiscale **O**scillator **S**tate + **A**ssociative **I**ndexed **C**ache), a streaming, no-attention neural controller that manages fixed-size explicit state: a local causal mixer, a multiscale continuous state bank, and a hard-addressed associative cache (plus an optional n-gram continuation cache).

In local runs, we train supervised control surfaces (opcodes, memory gating, commitments) on synthetic event traces as *smoke-test curricula* (Table 1). We report the implemented architecture, the runtime interfaces that make it testable and auditable, and diagnostics that highlight the central unsolved bottleneck: *hard addressing*.

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 [19] 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   From an architecture to a system substrate

MOSAIC is now only one layer of the overall work. The core claim of this paper is system-level: **persistent intelligent systems require a control plane**. We therefore separate concerns:

- **MOSAIC (kernel):** a no-attention neural controller that maintains internal state and emits control signals.

- **CCP (system):** a runtime architecture that provides event-native I/O, deterministic trace/replay, and safe tool acquisition/verification.

- **Substrate (data plane):** sandboxed tool execution and versioned registries, with objective evaluator gates.

This paper is written as *infrastructure*: we are not claiming AGI; we are defining components and interfaces that can be tested and iterated.

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   The Cognitive Control Plane (CCP)

CCP is a runtime architecture for long-lived, event-driven intelligent systems. It exists to make a learning system *auditable*, *deterministic when needed*, and *safe to extend* with tools.

2.1 Planes and separation of concerns

We use a three-plane decomposition:

- **Data plane:** tool execution (sandbox), I/O, and resource usage.

- **Control plane:** explicit memory, control surfaces, policy gates, and commitments.

- **Deliberative plane (optional):** higher-level planning/reasoning models that can propose actions, tools, and experiments.

This paper focuses on the control plane and its interfaces; MOSAIC is the kernel controller used by the control plane.

2.2 Event-native substrate (JSON events as byte tokens)

CCP uses an event-native boundary: external interaction is a JSON `EventEnvelope` (required: `type`, `payload`, `sender`; optional: `priority`, `budget_ms`, `id`, `ts`, plus commitment meta-fields). A deterministic serialization is encoded as UTF-8 bytes, yielding a token stream in $\{0, \ldots, 255\}$. This reframes decoding as **VM time-steps** over an event stream and enables supervision aligned to structured event traces.

2.3 Determinism, trace, and replay

For systems work, "it ran once" is not evidence. CCP therefore includes deterministic trace logging and replay:

- **Trace:** record event envelopes, tool lifecycle events, and evaluator outcomes as an append-only log.

- **Replay:** deterministically reproduce a run to debug regressions and verify behavior under identical inputs.

This makes training curricula, tool acquisition, and runtime interventions inspectable and testable.

2.4 Tool synthesis and verification as a first-class capability

CCP treats tools as compiled hypotheses rather than free-form code. A minimal lifecycle is:

1. **Define:** propose a tool interface and implementation (`ToolDefinition`).

2. **Test:** auto-generate unit tests from an oracle or dataset (`ToolTestGenerator`).

3. **Sandbox:** execute tests in an isolated environment with explicit capabilities.

4. **Gate:** evaluator enforces policy (capabilities, resource limits) and accepts/rejects a tool.

This allows the system to extend itself while keeping an objective, replayable record of what changed and why.

2.5 Motivating "party trick": Unknown Format Decoder

As a concrete end-to-end driver, we use an "Unknown Format Decoder" demo: given raw bytes, the system must infer a binary format, propose a parser tool, validate it against oracle-generated tests, and adapt when the format shifts. This is implemented as a manifest-runner target and lab dataset with deterministic trace/replay, making failures falsifiable rather than anecdotal.

3 Related Work

No single architecture fully resembles MOSAIC, but MOSAIC draws from and synthesizes several distinct research threads. We organize prior work by the specific capability each addresses.

3.1 Attention-Free State Space Models

The closest relatives to MOSAIC's core approach are SSM-based models that replace attention entirely. **Mamba** [9] uses selective state spaces with input-dependent gating, achieving linear-time sequence modeling. **RWKV** [16] uses linear attention approximations to reinvent RNNs for the Transformer era. **Griffin** [4] mixes gated linear recurrences with local attention for efficient language models.

MOSAIC's multiscale continuous state bank with leaky integrators is similar in spirit but differs by: (i) using multiple timescales (K states with different learned decay rates λ_k), and (ii) being explicitly designed as a "vibe/intent" channel rather than the primary compute path. In MOSAIC, the SSM-like state bank is one of three complementary mechanisms, not the whole architecture.

3.2 Attention Efficiency Improvements

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 [20]); kernel and hashing methods approximate attention without enumerating all pairs (e.g., Performer [3], Reformer [12]); sparse/local patterns reduce compute while preserving some long-range access (e.g., Longformer [2], BigBird [21]). These methods can reduce compute, but autoregressive decoding typically still benefits from (or requires) history-dependent state that grows with context length. MOSAIC takes a different approach: remove attention entirely.

3.3 KV-Cache Optimization

Orthogonal work reduces the *storage format* of KV caches via head sharing (MQA/GQA) [18, 1], latent caching [5], or quantization [10, 14]. MOSAIC targets a different axis: removing the "store one vector per token" mechanism entirely, replacing it with fixed-size explicit memory structures.

3.4 Memory-Augmented Neural Networks

The **Neural Turing Machine** (NTM) [7] and **Differentiable Neural Computer** (DNC) [8] pioneered neural networks with external memory. However, they use soft attention-based addressing over memory (content-based lookup), and DNCs still have $O(N)$ memory growth per sequence.

The **Sparse DNC** [17] introduced LSH-based sparse addressing to reduce complexity. This is closer to MOSAIC's approach, but still differs: MOSAIC uses hard-addressed hash tables with $O(1)$ lookup and no attention whatsoever, making the addressing fundamentally discrete rather than soft.

3.5 Hash-Based Memory Addressing

MOSAIC's product-quantized VQ routing for the associative cache has precedents in Sparse Access Memory (SAM) using k-d trees, LSH-based sparse memory access, and product quantization for nearest-neighbor search [11]. However, these techniques were not previously combined with SSM-style continuous state in a unified language modeling architecture.

3.6 N-gram Cache Models

The n-gram continuation cache is the most "classical" component. This has deep roots: **Kuhn & De Mori** [13] introduced cache-based natural language models for speech recognition. **Grave et al.** [6] proposed improving neural language models with a continuous cache—a very similar concept of using n-gram caching with neural LM interpolation. **Infini-Gram** [15] scaled n-gram models massively, demonstrating their continued value.

Our framing of the n-gram cache as "disproportionately high-yield" for verbatim continuation aligns with this literature: many "hard" failures in small models are not deep reasoning errors but failures of verbatim continuation (identifiers, brackets, repeated substrings).

3.7 What Makes MOSAIC Distinct

The unique contribution is not any single component but the specific combination and framing:

1. **Three-way decomposition:** local mixer + continuous state bank + hard-addressed cache (rather than attention doing all three).

2. **Explicit constant-size memory guarantee:** $O(1)$ rather than $O(T)$.

3. **Training curriculum for memory usage:** write curriculum, forced-read dropout, utility prediction—addressing the "memory getting used" problem that plagued earlier memory-augmented networks.

4. **Event-native + dVM framing:** treating decoding as VM time-steps with supervised control surfaces rather than pure next-token prediction.

None of the closest existing architectures—Griffin, Mamba, or the continuous cache LM—fully overlap with MOSAIC's specific combination, particularly the hard-addressed cache replacing KV-cache entirely, the dVM framing, and the event-native interface.

4 MOSAIC as the Control Kernel

4.1 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.

4.2 What MOSAIC is not

MOSAIC is a kernel controller, not a full agent:

- It is **not** a planning algorithm and does not inherently produce explicit plans.

- It is **not** a safety system; CCP policy/evaluator gates provide constraints.

- It is **not** a benchmark claim; the goal is controllable interfaces and falsifiable diagnostics.

4.3 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.

4.4 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 {close, neutral, 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.

4.5 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:

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

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

4.6 Multiscale continuous state

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

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

4.7 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.

4.8 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.

Why addressing is the core unsolved problem. Hard addressing is a discrete, credit-assignment-heavy problem: the model must emit the correct key *before* it can receive gradients through the retrieved value. This makes addressing the primary axis where MOSAIC can fail even when the memory capacity is sufficient. CCP therefore treats addressing performance as a first-class diagnostic via telemetry (routing entropy, gate utilization) and targeted stress tests (collision/interference probes).

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., $K{=}64, G{=}2 \Rightarrow 4{,}096$ buckets). Straight-through estimators allow end-to-end training while preserving discrete routing.

In the current implementation (`MosaicMemory`), VQ routing is configured via:

- `mem_vq_groups`: number of groups G (default 2).

- `mem_vq_codebook_size`: codes per group K (e.g., 64 yields $64^2 = 4096$ buckets).

- `mem_vq_group_dim`: embedding dimension per group (default 16).

- `mem_vq_beam`: number of top codes to consider for neighbor reads (default 2).

Separate read/write codebooks (`mem_vq_codebook_r`, `mem_vq_codebook_w`) allow the model to learn different routing strategies for retrieval versus storage.

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.

4.9 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.

4.10 Fusion

We combine streams with learned gates:

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

4.11 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:

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

4.12 Streaming inference

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

4.12.1 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 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 $x_{t+1} \sim \text{softmax}(\ell_t)$

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 $r.\texttt{commitment\_delta} \leftarrow \delta$ (Mode B)
8: Update commitment ledger and publish r to the EventBus

4.13 Implementation (Caramba)

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

- **Core layers:** `MosaicBlockLayer` (`layer/mosaic/block/layer.py`) implements the full block including local mixer, state bank, and memory fusion. `MosaicMemory` (`layer/mosaic/memory/memory.py`) implements the hard-addressed associative cache with VQ or bits routing, set-associative buckets, and sparse writes.

- **State management:** `MosaicState` (`layer/mosaic/state.py`) maintains conv buffer, state bank vectors, registers, and memory tables across streaming decode steps.

- **Control surfaces:** Opcode head, register gates, and commitment head are integrated into `MosaicBlockLayer` with configurable enable flags (`opcodes_enabled`, `reg_slots`, `commitment_head_enabled`).

- `MosaicNGramCacheLogitsLayer` (`layer/mosaic/ngram\_cache.py`) (optional inference-time logit bias).

- **Event primitives:** `EventEnvelope` (`core/event.py`), `EventEncoder`/`EventDecoder` (`core/event\_codec/`), and `EventBus` (`core/event\_bus.py`).

- **Curricula:** Memory curriculum (`dataset.mosaic_memory_curriculum`), ICL rule induction (`dataset.icl_rule_induction`), and event traces (`dataset.mosaic_event_traces`).

- **Phase 2 runtime:** Commitment ledger (`core/commitments.py`) and event runtime (`infer/event\_runtime.py`).

The training loop in `StandardTrainer` collects MOSAIC-specific telemetry including per-block write gate utilization, read gate utilization, and routing entropy (normalized bucket distribution entropy). These metrics are logged to WandB and console for debugging memory usage patterns.

5 Experiments

5.1 Setup

We define several presets for systematic evaluation:

- **Token-stream MOSAIC** in `config/presets/mosaic.yml` (base architecture).

- **MOSAIC ICL** in `config/presets/mosaic\_icl.yml` (few-shot rule induction with VQ routing).

- **SSM baseline** in `config/presets/mamba\_ssm.yml` (pure SSM without explicit memory).

- **Event-native Phase 1** in `config/presets/mosaic\_event\_native.yml` (opcodes + memory gating).

- **Event-native Phase 2** in `config/presets/mosaic\_commitment.yml` (commitment lifecycle).

All experiments use 6-layer models with $d = 256$ on the synthetic memory curriculum or ICL rule induction datasets, trained for 2000 steps on Apple M-series hardware (MPS backend).

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, 400 steps).

Phase	Capability	Final loss	tok/s
Phase 1	Event traces + opcodes + memory gating	0.36	∼970
Phase 2	Commitment lifecycle ($\Delta \in \{-1, 0, +1\}$)	0.32	∼930

5.2 ICL Rule Induction (Table 2 Row D)

We trained MOSAIC on a synthetic in-context learning task: given 4 demonstrations of a simple input→output rule (with distractors), predict the correct output for a query. The dataset binds gap distances (number of tokens between demonstration and query) into bins for fine-grained analysis.

After 2000 steps, MOSAIC achieved a training loss of ∼2.5 (PPL ∼12.7). The per-bin accuracy remained at 0% across all gap bins, indicating that MOSAIC has not yet learned to perform reliable few-shot rule induction—a known weakness of memory-augmented networks without careful curriculum design. However, memory telemetry shows:

- Routing entropy decreasing from ∼0.25 to ∼0.1 (routing is becoming more concentrated).

- Memory read gates at ∼60% utilization (memory is being queried).

- Teacher annealing reaching 0% by step 2000 (student is in control).

This suggests the memory is being used but not yet in a way that solves the ICL task. Further curriculum design (e.g., utility prediction, contrastive recall losses) is needed.

5.3 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. (Current results: 0% accuracy; see above.)

- **Non-verbatim manipulation:** tasks like "repeat this list sorted" to distinguish mere copying from compositional reuse.

5.4 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: MOSAIC vs SSM baseline on memory curriculum (2000 steps, synthetic retrieval task).

Model	Final loss	PPL	Params	tok/s
SSM baseline (6L, d=256)	1.53	4.62	6.8M	∼96k
MOSAIC ICL (6L, d=256)	2.54	12.7	13.2M	∼200

The SSM baseline achieves lower perplexity on the memory curriculum task, but this is expected: the curriculum is designed to stress-test discrete retrieval, which is precisely where SSMs (smooth continuous state) excel when patterns are predictable. MOSAIC's higher perplexity reflects the harder optimization surface of learning discrete addressing. Crucially, MOSAIC's memory telemetry shows healthy routing entropy ($\sim$0.1–0.25) and active memory gates ($\sim$50% read/write utilization), indicating the memory subsystem is being used rather than ignored.

Table 3: Planned ablation results (pending full runs).

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

6 Discussion

6.1 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.

6.2 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.

6.3 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.

6.4 Idle-time compute, latent impulses, and native tool-building

While MOSAIC is a language model architecture, the Caramba runtime layers on system behaviors that are useful for long-lived agents:

- **Idle-time compute spend:** when no external events arrive, the runtime can spend bounded compute on maintenance (e.g., replay-based consolidation updates to small adapters) rather than sitting idle (see preset `config/presets/mosaic\_idle.yml`).

- **Latent impulses:** the runtime can emit internal "wake" or "drive" events (via a homeostatic loop) that keep the system active, producing self-initiated event traces instead of only prompt→response behavior.

- **Native tool-building:** tools can be defined, tested, and registered as first-class events (a minimal toolchain + evaluator). This supports workflows where the system proposes a tool, runs unit tests, and only then adopts it (e.g., an "unknown format decoder" demo runner built on deterministic trace/replay and lab datasets).

These are presented here as *integration points* rather than claims about agent performance; they define how MOSAIC can be embedded into a controlled, test-driven runtime.

7 Limitations and Non-Goals

This work is infrastructure-first. We explicitly do *not* claim:

- **General intelligence:** CCP is not an AGI claim; it is a control substrate.

- **Open-world safety:** sandboxing and evaluator gates reduce risk but do not make an arbitrary tool-safe agent.

- **SOTA language modeling:** experiments here are diagnostic (telemetry + falsifiability), not benchmark competitions.

- **Solved in-context learning:** current ICL probes remain unsolved; addressing and curriculum design are active work.

7.1 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.

8 Conclusion

We are not proposing a better chatbot. We present the **Cognitive Control Plane (CCP)**, a missing systems layer for persistent, event-native intelligent systems: deterministic trace/replay, explicit control surfaces, and a test-driven tool lifecycle. MOSAIC serves as a concrete, implemented **control kernel** for this substrate: an attention-free controller over fixed-size explicit state (local mixer, multiscale state bank, associative cache).

Our experiments are intentionally diagnostic. They show that supervised control surfaces (opcodes, memory gating, commitments) can be trained as smoke-test curricula, and that explicit-memory telemetry provides falsifiable signals about whether the model is using its memory pathways. The central technical bottleneck remains addressing; CCP is designed to make addressing failures measurable and tool-driven remediation testable.

If long-lived intelligent systems are built, they will require a cognitive control plane. This work provides a concrete, implemented starting point.

Statements and Declarations

A Manifest Presets

Reference presets for reproducing experiments:

- `config/presets/mosaic.yml` — Token-stream baseline.

- `config/presets/mosaic\_icl.yml` — ICL rule induction (Table 2 Row D).

- `config/presets/mamba\_ssm.yml` — SSM baseline for comparison.

- `config/presets/mosaic\_event\_native.yml` — Event-native Phase 1.

- `config/presets/mosaic\_commitment.yml` — Event-native Phase 2 (commitments).

- `config/presets/mosaic\_memory\_curriculum.yml` — Memory curriculum training.

- `config/presets/mosaic\_idle.yml` — Idle-time compute / maintenance loop.

B   Implementation Notes

Core implementation:

- **Block layer:** `layer/mosaic/block/layer.py` — local mixer, state bank, fusion, control surfaces.

- **Memory subsystem:** `layer/mosaic/memory/memory.py` — VQ/bits routing, set-associative read/write, last-write-wins conflict resolution.

- **State:** `layer/mosaic/state.py` — streaming state container.

- **Opcodes:** `layer/mosaic/isa.py` — instruction set enum (NOP, READ_MEM, WRITE_MEM, etc.).

- **Event primitives:** `core/event.py`, `core/event\_codec/`, `core/event\_bus.py`.

- **Commitments:** `core/commitments.py` — runtime ledger for Phase 2.

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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 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, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

[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, 2024.

[6] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[8] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

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

[10] Coleman Hooper, Sehoon Kim, Hiva Mohber, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *arXiv preprint arXiv:2401.18079*, 2024.

[11] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[12] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

[13] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[14] Shiquan Li et al. CommVQ: Reducing communication overhead for kv cache compression via importance-aware vector quantization, 2025. arXiv preprint.

[15] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infinigram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*, 2024.

[16] 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.

[17] Jack W Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Timothy P Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

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

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

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

[21] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. BigBird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, 2020.