VOLTIC: DISTINGUISHING VOLATILITY FROM STOCHASTICITY IN RECURRENT MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent sequence models must decide how strongly to overwrite their memory at each token. Read as Bayesian filtering, this write is the gain of a Kalman update, set by uncertainty from two sources that pull it in opposite directions: volatility, how quickly the underlying associations change, and stochasticity, how noisy each observation of them is. First, we show that the update of gated delta-rule memories is the form this filter takes under isotropic uncertainty. Next, we introduce Voltic, a recurrent memory that keeps the covariance anisotropic and makes both noise variances input-dependent, so the write is vector-valued and carries uncertainty accumulated over the sequence. A dense covariance would have to be propagated token by token, ruling out the parallel training these models depend on. We therefore give two assumed-density approximations, diagonal and quasi-diagonal, both of which leave the memory update in delta-rule form and reuse its chunked kernels. On controlled recall tasks in which associations change and observations are corrupted, Voltic leads all baselines. On the task combining volatility and stochasticity, its margin over the strongest baseline is larger at both extrapolation sizes than at the training sizes. In 45M-parameter language models it leads an eight-task reasoning average and achieves higher retrieval accuracy beyond the training context length than gated baselines, at throughput close to those baselines. Deriving the write from an uncertainty recursion therefore makes memory more responsive to change.

1 INTRODUCTION

Recurrent sequence models compress a stream of tokens into a fixed-size memory, and at every token they must decide how strongly the new token should change what is stored. This decision has a natural statistical reading. If the memory is taken to be the posterior mean of a latent state that the token stream observes, then updating it is Bayesian filtering, and the strength of each write is the Kalman gain (Kalman, 1960): how much weight a new observation receives relative to the evidence already accumulated. The gain is set by uncertainty, so the question of how a memory should write becomes a question of what that uncertainty is made of.

Under this view, the uncertainty that sets the gain has two sources: *Volatility* is how quickly the latent state changes, the magnitude of the process noise; *stochasticity* is how noisy each observation is, the magnitude of the observation noise. The two act on the gain in opposite directions. Higher volatility makes past evidence less relevant and calls for a stronger write; higher stochasticity makes the new observation less informative and calls for a weaker one. Both raise uncertainty, so a filter that represents only their combined effect cannot respond appropriately to either. Estimating the two online is a long-standing problem in adaptive filtering (Särkkä & Nummenmaa, 2009; Piray & Daw, 2021), and normative accounts of learning treat their dissociation as central, showing that failing to make it produces characteristic failures of adaptation (Fang & Piray, 2026; Smith et al., 2026). These accounts infer both noise parameters jointly with the latent state, using a Rao-Blackwellized particle filter (Doucet et al., 2000; Piray & Daw, 2021) or a deterministic quantile grid in place of the particles (Chen & Piray, 2026). Such methods are designed for behavioral tasks, a few static parameters inferred from hundreds of observations. Their representation of the noise parameters grows exponentially with the number of parameters, so they do not extend to the input-dependent, per-coordinate rates a high-dimensional memory trained on billions of tokens would require.

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.

Existing recurrent memories can be read as filters in which this structure is heavily restricted. The delta-rule update used in these models (Schlag et al., 2021) is the classical least-mean-squares rule of adaptive filtering, and the Kalman filter is its uncertainty-weighted counterpart, setting the learning rate from the filter's uncertainty rather than fixing it (Sutton, 1992; Dayan et al., 2000). Gated delta-rule models such as GDN (Yang et al., 2025) and KDA (Kimi Team et al., 2025) update a key-value memory by erasing its component along the current key and writing the new value, scaled by a learned write strength. We show that this update is a filtering step under isotropic uncertainty: the write strength is the gain of a filter whose uncertainty is the same in every direction of key space, so it is a single scalar per head, and it is produced by a projection of the current token rather than by accumulated uncertainty. Filtering has also been used directly as a sequence mixer, most closely in Kalman Linear Attention (Shaj et al., 2026), which filters each channel with a diagonal precision and learned, time-invariant process noise. What remains missing is a memory in which both sources of uncertainty depend on the input, uncertainty varies across key space, and the update keeps the delta-rule form that makes these models efficient to train.

We introduce **Voltic**, a recurrent memory that treats the sequence as *vol*atile and stochas*tic*, and derives its write strength accordingly. Both noise variances are learned functions of the input, entering the filter at the prediction and observation steps, so the write responds to the two sources in opposite directions without inference over either. The resulting gain is vector-valued along the key dimension and carries uncertainty accumulated over the sequence. Exact inference would propagate a dense covariance, which cannot be evaluated in parallel across tokens, so we develop two assumed-density approximations (Opper, 1999; Minka, 2001): Voltic-D keeps a diagonal covariance with a learned per-coordinate decay, and Voltic-Q retains covariance within short blocks. Both leave the memory update in delta-rule form and reuse its parallel algorithms. Both properties are ordinary in real sequences: what a name or a value refers to can change over a document, and a mention of it can be wrong. Existing recall benchmarks assume neither, observing each association once and faithfully (Arora et al., 2024a), so a model is never asked to respond differently to a change and to a corrupted observation. We therefore vary the two sources separately and together.

We make four contributions. (i) We introduce Voltic, a recurrent memory whose write strength is derived from a filtering recursion rather than learned as a free gate, with two tractable approximations that keep the update in delta-rule form. (ii) We give efficient forward and backward kernels for both variants, bringing throughput close to gated recurrent-attention baselines at the sequence lengths and batch sizes used for language-model training. (iii) We introduce three controlled recall tasks in which associations change during the sequence, observations of them are corrupted, or both. (iv) We evaluate Voltic against recurrent-attention, state-space, and full-attention baselines on these tasks, language modeling, in-context retrieval, and throughput.

2 Related Work

Recurrent memories and online learning. Linear attention expresses sequence mixing through a fixed-size recurrent state (Katharopoulos et al., 2020). Delta-rule models improve how this state stores key–value associations through error-correcting writes; Section 3.2 shows that the write strength of their gated extensions plays the role of a filtering gain. State-space models provide another route to recurrent sequence mixing, with input-dependent state updates (Gu & Dao, 2024; Lahoti et al., 2026). A parallel line reads these updates as solutions to a per-token regression problem, unifying delta-rule and gated variants as online learners (Liu et al., 2025; Wang et al., 2026a).

Filtering-based sequence models. Deep Kalman Filters (Krishnan et al., 2015), Recurrent Kalman Networks (Becker et al., 2019), and learned latent dynamics for planning (Hafner et al., 2019) combine neural representations with probabilistic state estimation, but their recurrent updates limit use at long context. Closest to this work is Kalman Linear Attention (KLA; Shaj et al., 2026), which derives sequence mixing from Bayesian filtering and parallelizes its information-form update through a scan. KLA keeps a diagonal precision per channel and treats process noise as a learned time-invariant parameter; Voltic instead carries uncertainty over key coordinates, retaining short-range covariance in Voltic-Q, and makes both noise variances input-dependent. We compare KLA on MQAR, language modeling, in-context retrieval, and throughput; Appendix D.1 specifies the different KLA backends used for these studies. MesaNet (von Oswald et al., 2026) and Gated

KalmaNet (Peng et al., 2026) solve a related regularized least-squares objective at each step, but assume a static latent state and require iterative solvers.

Assumed-density filtering. Projecting the posterior back onto a fixed family at each step, minimizing KL divergence to the exact posterior, is a standard route to keeping the state bounded (Opper, 1999; Minka, 2001); Section 3.3 applies it here. Temporal parallelization of Bayesian filters has also been studied directly (Särkkä & García-Fernández, 2021), using lifted representations to obtain associativity; we instead keep the memory update in delta-rule form so that it reuses the chunked algorithms already developed for that family.

3 VOLTIC: MEMORY UPDATES FROM UNCERTAINTY

3.1 A FILTERING MODEL OF RECURRENT MEMORY

In language modeling, a model predicts each token from the tokens before it. Linear recurrent models, such as linear attention and Mamba (Katharopoulos et al., 2020; Gu & Dao, 2024), summarize that history in a fixed-size state, updated as each token arrives. We treat this summary as the result of inference: the token stream is modeled as the output of a dynamical system whose latent state evolves over time, and each observation $x_t \in \mathbb{R}^P$, a learned projection of the current token, is generated noisily from it.

The latent state is a matrix $Z_t \in \mathbb{R}^{N\times P}$, and each observation reads it along a direction $k_t \in \mathbb{R}^N$, also computed from the token and normalized to unit length. The model makes two structural assumptions: the latent state evolves as a Markov process, and each observation depends only on the current state. For each observation dimension j, with latent column $z_{t,j} \in \mathbb{R}^N$,

$$\begin{aligned} z_{t,j} \mid z_{t-1,j} &\sim \mathcal{N}\big(A_t z_{t-1,j},\, \mathrm{diag}(v_t)\big), \\ x_{t,j} \mid z_{t,j} &\sim \mathcal{N}\big(k_t^\top z_{t,j},\, s_t\big), \end{aligned} \tag{1}$$

with diagonal transition $A_t = \mathrm{diag}(\alpha_t)$, $\alpha_t \in (0,1]^N$. The variance of the transition, $v_t \in \mathbb{R}^N_{>0}$, is the volatility of the latent state and may differ across its N coordinates; the variance of the observation, $s_t \in \mathbb{R}_{>0}$, is its stochasticity. Volatility is a vector because the latent state is, while stochasticity is a scalar because each observation value is a single number per step. The Gaussian form is chosen for tractability, since it makes the posterior available in closed form; it is not a claim about the distribution of tokens. In the terminology of recurrent memories, k_t is the key, x_t the value, and the posterior mean of Z_t is the memory that maps keys to values.

Given the parameters at each step, the posterior over each latent column is Gaussian and is computed by the Kalman filter (Kalman, 1960; Bishop, 2006). The columns share their covariance U_t, since they are observed along the same direction with the same noise. The prediction step propagates the covariance through the transition and adds volatility, and the observation step weights the new observation against its prediction:

$$P_t = A_t U_{t-1} A_t^\top + \mathrm{diag}(v_t), \qquad g_t = \gamma_t\, P_t k_t, \qquad \gamma_t = \frac{1}{s_t + k_t^\top P_t k_t}, \qquad U_t = P_t - g_t k_t^\top P_t. \tag{2}$$

The posterior mean of column j is $m_{t,j} = A_t m_{t-1,j} + g_t\big(x_{t,j} - k_t^\top A_t m_{t-1,j}\big)$. Because the gain is shared across columns, stacking them into $M_t \in \mathbb{R}^{N\times P}$, the memory, gives

$$M_t = A_t M_{t-1} + g_t\big(x_t^\top - k_t^\top A_t M_{t-1}\big) = \big(I - \gamma_t P_t k_t k_t^\top\big) A_t M_{t-1} + \gamma_t P_t k_t\, x_t^\top, \tag{3}$$

where the second form makes the erase-and-write structure explicit: the memory's component along k_t is removed and the new observation written in its place, both scaled by the gain. The readout $o_t = M_t^\top q_t$ is the posterior predictive mean of the observation along a query direction q_t, computed from the token. The gain weights the difference between the observation and its prediction: larger predicted uncertainty writes more strongly, and larger observation noise writes less. Because each observation dimension is scalar, the denominator $1/\gamma_t$ is scalar and no matrix inverse is required. In a neural sequence model, all parameters of Equation 1 are supplied by the network: learned projections of the token representation h_t produce the observation, key, and query, and separate projections produce the transition and both noise variances (Appendix B.2). That the variances depend on the

input is what inference over them would produce: a learner that treats v and s as unknown maintains a posterior, and the rates entering its state update at time t are posterior-weighted and therefore functions of $x_{1:t}$, as particle-filter and grid-based treatments compute explicitly (Piray & Daw, 2021; Chen & Piray, 2026). We instead amortize that dependence, producing the rates directly as learned functions of the input and training them end-to-end through the sequence-prediction loss, which treats them as parameters rather than as generative random variables.

One dependence is deliberately absent. A learner that infers v and s updates its beliefs about them from the predictive likelihood of each observation, so the inferred rates depend on the current state estimate. Here the rates depend on the input alone, which leaves the gain computable without the memory and the two recursions separable.

The obstacle to using this filter directly as a recurrent memory is the covariance. The update of U_t subtracts a rank-one term from a matrix that the prediction step has already made dense, so the exact posterior covariance is a full $N \times N$ matrix costing $O(N^2)$ per token. Recurrent memories are trainable at scale only because their state update can be rewritten as matrix products over blocks of tokens, which lets a GPU process many tokens at once instead of stepping through them one at a time (Yang et al., 2024). The dense covariance admits no such rewriting, and at $O(N^2)$ per token it is as expensive as the memory update it is meant to inform, so it would dominate training cost.

3.2 The isotropic case: gated delta rules

Consider the simplest case of Equation 1: a latent state with no preferred directions, in which volatility is shared across coordinates and the covariance is isotropic, $P_t = \rho_t I$. For a unit key, $P_t k_t = \rho_t k_t$, so both factors in Equation 3 collapse onto the key,

$$g_t = \gamma_t \rho_t \, k_t \;\equiv\; \beta_t k_t, \qquad \beta_t = \frac{\rho_t}{s_t + \rho_t} \in (0, 1), \tag{4}$$

and the memory update becomes $M_t = (I - \beta_t k_t k_t^\top) A_t M_{t-1} + \beta_t k_t x_t^\top$. This is exactly the form of two recent recurrent memories, adopted as the sequence mixer in frontier open-weight models such as Qwen3-Next (Qwen Team, 2025) and Kimi Linear, and both derived from the delta rule and its gating rather than from a filter:

$$\text{GDN:} \quad M_t = \alpha_t \big(I - \beta_t k_t k_t^\top\big) M_{t-1} + \beta_t k_t x_t^\top, \tag{5}$$

$$\text{KDA:} \quad M_t = \big(I - \beta_t k_t k_t^\top\big) \operatorname{diag}(\alpha_t) M_{t-1} + \beta_t k_t x_t^\top, \tag{6}$$

differing in whether the decay α_t is a scalar or a vector along the state dimension (Yang et al., 2025; Kimi Team et al., 2025). The correspondence is one of form, and it says what the two components of these models do: the rank-one erase occupies the position of the filter's covariance correction, and the learned write strength β_t that of its gain. GDN-2 (Hatamizadeh et al., 2026) uses separate erase and write vectors and so falls outside this correspondence, but its write is likewise scaled by a quantity that does not vary along the key axis.

In each case the write is fixed to the direction of the key, and its strength is one number per head: a covariance with a single scale produces a gain with a single magnitude. Voltic keeps the covariance anisotropic, so that the gain varies along the key axis according to where uncertainty has accumulated. We develop two approximations that preserve this structure while keeping the per-token uncertainty work well below that of the memory update, so it remains a small addition to a computation that still runs in block form: a diagonal covariance (Section 3.3) and a quasi-diagonal one that retains correlations within short blocks of tokens (Section 3.4).

3.3 Voltic-D: Diagonal uncertainty

Both variants of Voltic are assumed-density filters (Opper, 1999; Minka, 2001): after each observation, the posterior is projected onto a tractable Gaussian family by minimizing the KL divergence to the exact posterior. For Gaussians this projection matches moments, so projecting onto diagonal covariances keeps the exact marginal variances and discards only the correlations between coordinates. The variants differ in how often they project.

Volitc-D projects after every token, so the covariance is a diagonal u_t throughout. With coordinate-wise decay α_t, the update is

$$p_t = \alpha_t^{\odot 2} \odot u_{t-1} + v_t, \qquad d_t = s_t + \sum_{i=1}^{N} p_{t,i} k_{t,i}^2, \tag{7}$$

$$g_t = (p_t \odot k_t)/d_t, \qquad u_t = p_t - (p_t \odot k_t)^{\odot 2}/d_t, \tag{8}$$

where $\odot$ denotes element-wise multiplication and $d_t = 1/\gamma_t$. The gain is the key reweighted by predicted uncertainty: coordinates in which the state is more uncertain receive a larger share of the write. Because the covariance is shared across the P columns of the memory, this costs $O(N)$ work and N stored scalars per head, against the NP scalars of the memory itself.

3.4 VOLTIC-Q: QUASI-DIAGONAL UNCERTAINTY

Discarding correlations at every token is the coarsest projection that keeps the uncertainty cheap enough to run alongside the memory update. Voltic-Q keeps more of the covariance: within a block of Q tokens the exact recursion runs with no projection, so correlations between coordinates accumulate and are discarded only at the block boundary. Inside a block,

$$P_t = U_{t-1} + \mathrm{diag}(v_t), \qquad g_t = \gamma_t P_t k_t, \qquad U_t = P_t - \gamma_t (P_t k_t)(P_t k_t)^\top, \tag{9}$$

and at the boundary the off-diagonal entries of U_t are set to zero, the same moment-matching projection applied every Q tokens rather than every one, with the diagonal passed to the next block along with the memory. The two variants are endpoints of one family: $Q = 1$ recovers Section 3.3, and Q equal to the sequence length would be the exact filter. Because the projection applies to the uncertainty alone, Q controls the covariance approximation without restricting the context length.

Voltic-Q uses fixed decay, $A_t = I$. A coordinate-wise decay inside a block would require rescaling the accumulated terms by cumulative products of the decays, which shrink toward zero over a block and destabilize the gradients; with $A_t = I$ no such rescaling arises.

Within a block the covariance never has to be formed. Each observation adds a diagonal term through the prediction step and subtracts a rank-one term through the correction, so U_t is always a known diagonal minus at most Q rank-one terms, and the gain follows from those terms alone. This costs $O(QN)$ work per token, a factor Q/P of the memory update's $O(NP)$, so the uncertainty remains the cheaper of the two computations as long as Q stays well below P (Appendix B.3). The main experiments use $Q = 4$; Appendix C examines the effect of the block length.

3.5 EFFICIENT IMPLEMENTATION

The efficiency of the gated linear-attention family rests on one algorithm: updates of the form $M_t = T_t M_{t-1} + w_t x_t^\top$, with T_t diagonal plus rank one, can be evaluated with matrix products over chunks of tokens rather than stepping through them (Yang et al., 2024; Yang & Zhang, 2024). Voltic inherits it: because the gain does not depend on the memory or the observation, Equation 3 has exactly this form, with transition $(I - g_t k_t^\top) A_t$, and we evaluate it with a specialized version of the corresponding FLA operator. The per-coordinate gain costs nothing here: g_t enters where a gated delta rule places a scalar multiple of the key, and the kernel does not care which.

What the filter adds is the uncertainty recursion, which is sequential. Its cost per token is $O(N)$ for Voltic-D and $O(QN)$ for Voltic-Q, against $O(NP)$ for the memory update, so it is smaller by a factor of P and P/Q. Note that Q sets the covariance approximation, while the chunk length only organizes the same recurrence. Fused projections, a fused kernel for the noise rates, and reuse of the gain's matrix products between its write and erase roles reduce memory traffic and kernel launches; uncertainty updates use FP32. The surrounding block follows standard practice for recurrent memories, with short causal convolutions, a gated output, and normalization before the projection back to d_{model}. Appendix B details the block, kernels, and gradients; Appendix D lists software, hardware, and training time. Section 4.5 reports throughput against gated recurrent-attention baselines.

4 EXPERIMENTS AND RESULTS

Voltic separates changes in stored associations from noise in new observations. We test this design first with controlled memory tasks, where both sources can be specified, then with language modeling and in-context retrieval, where useful information must be recovered from natural or synthetic text. Training-throughput and cached-decoding measurements test computational cost as sequence length and batch size vary, and when generating tokens one at a time.

4.1 MODELS COMPARED

We compare Voltic-D and Voltic-Q with KLA, which provides another filtering-based memory, and with KDA and GDN-2, which adapt memory through learned gates. Mamba-3 supplies state-space baselines in single-input single-output (SISO) and multiple-input multiple-output (MIMO) forms; the latter uses several input and output directions per update. For MQAR only, xLSTM (Beck et al., 2024) provides another recurrent-memory comparison, with either two matrix-memory (mLSTM) layers or a matrix/scalar-memory (mLSTM/sLSTM) hybrid. A causal softmax Transformer (Vaswani et al., 2017) provides a full-attention comparison for quality and throughput. Each benchmark specifies the included models and matched dimensions.

4.2 MULTI-QUERY ASSOCIATIVE RECALL

Multi-query associative recall (MQAR) presents key–value pairs and asks the model to predict the value of each queried key (Arora et al., 2024a). Since associations are sampled separately for each sequence, the task isolates the memory update: nothing can be answered from the weights, and performance depends on what the model writes and retains. We use four matched conditions with three observations per key: associations may change during the sequence or stay fixed, and observations may be corrupted or faithful. In our Standard MQAR condition, associations stay fixed and all observations are faithful. This gives a 2×2 design that varies volatility and stochasticity while holding the observation and query layout fixed. Keys and values are categorical tokens with no metric, so none of the variants matches the Gaussian model of Section 3.1; we use them to test the structural distinction that model encodes rather than its distributional form.

Tasks. Each sequence has K distinct keys, three observation rounds, and one query per key, with length $T = 8K$. Keys and values come from disjoint parts of an 8,192-token vocabulary. An observation is two consecutive tokens; a query supplies a key and asks for its value. *Standard MQAR* repeats each unchanged association in all three rounds. In *Volatile MQAR*, each key's hidden value changes independently with probability 0.30 before each of the last two rounds. The target is the most recent value, so the model must replace those associations while retaining untouched ones. *Stochastic MQAR* holds every hidden value fixed, but replaces each displayed value with an incorrect one with probability 0.25. A conflicting observation is therefore always noise, and the target is the hidden value, which the model must recover from the observations. *Vol & Sto MQAR* combines the two: the hidden value changes with probability 0.30 before each of the last two rounds, and each displayed value is independently corrupted with probability 0.25. A conflicting observation can now reflect either a change or noise, so the latest observation is insufficient on its own, and neither source is observable to the model.

Training and evaluation. Models share two layers, hidden width 128, training data, and 10,000 AdamW updates (Loshchilov & Hutter, 2019) at $K = 16/64$. We average accuracy at the two memory sizes equally within each run, then report the mean and its standard error (SEM) over five runs. Appendix E gives the full protocol, individual-seed results, and extrapolation tests.

Results. Standard MQAR is saturated: seven models are above 99.8%, so it does not separate them (Table 1). Varying either source does, and the baselines separate unevenly. KDA, tied for the best Standard accuracy, is among the weakest once values change, and both it and GDN-2 succeed on some seeds and fail on others. xLSTM holds up best among the baselines but trails on all three variants. The Voltic variants are the only models strong across all four tasks, with Voltic-Q leading Volatile MQAR and Vol & Sto and Voltic-D leading Stochastic MQAR; Voltic-Q also exceeds every baseline on Vol & Sto at both larger memory sizes (Table 7). Figure 4 shows individual-run accuracy. The Transformer saturates Standard MQAR at a lower learning rate, and warm-up training on a one-

Table 1: MQAR accuracy (%), averaging $K = 16$ and $K = 64$ equally within each run: mean $\pm$ SEM over five runs. Bold/underline mark the top two displayed means, including ties.

Model	Standard	Volatile	Stochastic	Vol & Sto
Voltic-D	<u>99.98</u> $\pm$ 0.02	<u>99.29</u> $\pm$ 0.31	**88.33** $\pm$ 0.20	69.86 $\pm$ 1.63
Voltic-Q	**100.00** $\pm$ 0.00	**99.48** $\pm$ 0.23	<u>87.61</u> $\pm$ 0.33	**72.89** $\pm$ 0.26
KLA	84.94 $\pm$ 2.14	75.26 $\pm$ 3.53	69.44 $\pm$ 1.55	47.62 $\pm$ 8.10
KDA	**100.00** $\pm$ 0.00	48.76 $\pm$ 19.41	84.08 $\pm$ 3.55	<u>70.88</u> $\pm$ 0.31
GDN-2	99.85 $\pm$ 0.14	70.25 $\pm$ 16.43	71.40 $\pm$ 16.53	65.12 $\pm$ 6.83
Mamba-3 SISO	11.67 $\pm$ 2.08	5.14 $\pm$ 2.00	4.84 $\pm$ 0.94	1.55 $\pm$ 0.63
Mamba-3 MIMO	12.96 $\pm$ 1.51	2.65 $\pm$ 0.64	26.72 $\pm$ 10.25	0.29 $\pm$ 0.08
xLSTM (mLSTM)	99.95 $\pm$ 0.02	86.39 $\pm$ 2.73	84.52 $\pm$ 1.52	66.78 $\pm$ 1.70
xLSTM (hybrid)	99.93 $\pm$ 0.03	87.11 $\pm$ 1.04	85.64 $\pm$ 0.08	64.97 $\pm$ 2.64
Transformer	<u>99.98</u> $\pm$ 0.00	2.96 $\pm$ 0.01	3.39 $\pm$ 0.02	3.16 $\pm$ 0.01

observation task recovers much of its accuracy on the other three at matched total budget, so its failure under the shared recipe reflects optimization rather than capacity (Appendix E.6).

Variance ablations. Fixing v_t, s_t, or both to 0.5 during evaluation tests how trained models rely on their noise branches (Table 9). Fixing v_t has two effects at once: a constant volatility makes the predicted covariance isotropic, so the gain becomes a scalar multiple of the key, and accuracy collapses on every task, including Standard MQAR, where no value changes. This is a direct test of the anisotropy of Section 3.2, and removing it costs most of the model's accuracy. Fixing s_t instead leaves the covariance anisotropic and removes only the stochasticity response, with the largest effect on Stochastic MQAR.

4.3 Language modeling

Language modeling tests whether Voltic's memory supports prediction in natural text.

Data, models, training, and evaluation. Models share FineWeb-Edu data (Penedo et al., 2024), a tokenizer, and ordered 2,048-token sequences, respecting document and sequence boundaries. The approximately 45.5M-parameter models share depth, width, and 4,800 AdamW updates over about 2.52 billion input tokens. Voltic, KLA, KDA, GDN-2, and both Mamba-3 variants receive the same base-learning-rate search. Selection and final training use separate seeds. Results use final checkpoints from three fresh seeds, reported as mean $\pm$ sample standard deviation. Table 10 gives model dimensions and selected rates; Appendix F gives the full recipe. We report zero-shot accuracy with shared prompts through Language Model Evaluation Harness (Biderman et al., 2026), and record FineWeb-Edu NLL and WikiText-103 perplexity (Merity et al., 2017) as a convergence check (Table 12). The eight-task average reflects reasoning and comprehension tasks; the nine-task average adds LAMBADA's final-word prediction (Paperno et al., 2016).

Results. Voltic-Q achieves the highest eight-task average and Voltic-D the second, with the seven recurrent models spanning about one point (Figure 1). Voltic-Q and KDA have nine-task averages within 0.01 percentage points. Voltic-Q has the highest mean on ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), and OpenBookQA (Mihaylov et al., 2018). Table 13 reports individual task scores.

4.4 In-context retrieval

In-context retrieval tests whether a trained language model can generate answers from details supplied in its prompt. For a recurrent model this is a direct test of the memory: the answer must have been written to a fixed-size state and survived everything written after it. We compare the recurrent language models from Section 4.3, without retrieval fine-tuning or prompt tuning. The full-attention RULER comparison appears in Figure 5.

Tasks and evaluation. Four RULER tasks (Hsieh et al., 2024) test key–value retrieval at 1K–8K tokens, exceeding the training length at 4K/8K without adaptation. Six BASED tasks (Arora et al., 2024b) test natural-text field extraction and question answering within 2K tokens.

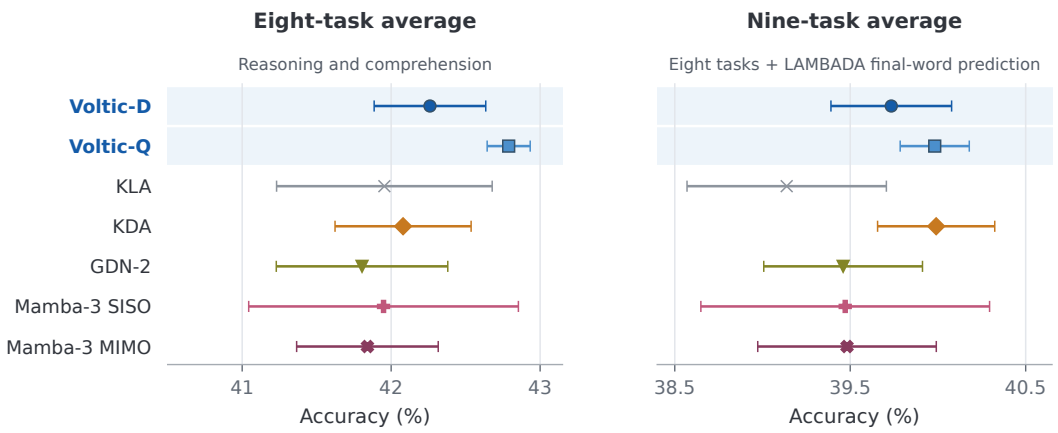


Figure 1: 45M downstream accuracy after 2.52 billion training tokens: mean ± sample SD over three seeds. The nine-task average adds LAMBADA to the eight-task average (Appendix F.4).

Selected LongBench tasks (Bai et al., 2024) test question answering, summarization, and classification; BABILong tasks (Kuratov et al., 2024) require one to three supporting facts. These short-context subsets complement RULER, without covering either full benchmark.

Models share examples, prompts, and greedy decoding. Groups average task/context scores equally within each seed, then report mean and SD across seeds. Appendix G gives the full protocol.

Table 2: 45M in-context scores (0–100) for recurrent models: mean ± sample SD over three seeds. LongBench combines F1, ROUGE-L (Lin, 2004), and classification scores; other groups use answer-matching accuracy. Bold/underline mark the top two means.

Model	RULER All 16	RULER 4K/8K	BASED	LongBench	BABILong
Voltic-D	3.12 ± 3.73	0.49 ± 0.29	10.23 ± 0.49	**7.04** ± 0.78	**7.26** ± 1.91
Voltic-Q	**15.15** ± 8.55	**9.53** ± 8.42	9.29 ± 0.37	4.99 ± 0.38	3.89 ± 1.28
KLA	0.07 ± 0.08	0.02 ± 0.01	7.37 ± 0.17	4.52 ± 0.91	3.41 ± 2.06
KDA	2.03 ± 0.26	0.33 ± 0.20	**11.58** ± 0.40	<u>6.46</u> ± 0.84	6.70 ± 1.73
GDN-2	<u>5.05</u> ± 2.85	<u>0.82</u> ± 0.61	<u>10.86</u> ± 0.21	6.44 ± 0.35	6.30 ± 1.83
Mamba-3 SISO	0.56 ± 0.15	0.07 ± 0.09	9.47 ± 0.32	4.92 ± 1.04	6.19 ± 1.53
Mamba-3 MIMO	1.37 ± 1.13	0.46 ± 0.45	9.83 ± 0.49	5.99 ± 0.81	<u>7.15</u> ± 0.61

Results. Voltic variants have the highest mean on four of the five groups: both RULER groups, LongBench, and BABILong (Table 2). Voltic-Q leads the average over all sixteen RULER conditions by 10.10 points over GDN-2, and leads the 4K/8K group in all three training seeds; its largest advantage is on S1, which asks for a numeric value in repeated filler text, while long identifiers and multiple key–value pairs remain more difficult. Voltic-D has the highest mean on the LongBench tasks and a small lead over Mamba-3 MIMO on BABILong, smaller than the variation across seeds. The two variants differ in both covariance structure and decay. Voltic-D's learned decay removes content that is never queried again, which plausibly accounts for its advantage where many passages compete. Voltic-Q's block covariance and fixed decay instead favor preserving a single stored value, since nothing decays and the block covariance suppresses repeated writes along directions already used. We do not separate the two factors here.

4.5 Throughput

Voltic computes an uncertainty state on top of the memory all recurrent models carry. This computation is much smaller than the memory update, but we evaluate it sequentially rather than as a parallel scan (Shaj et al., 2026), so the question is what it costs in practice. We measure training throughput

in two sweeps, varying sequence length at a fixed token budget and batch size at a fixed length, then measure single-token decoding latency, the regime in which recurrent models are deployed.

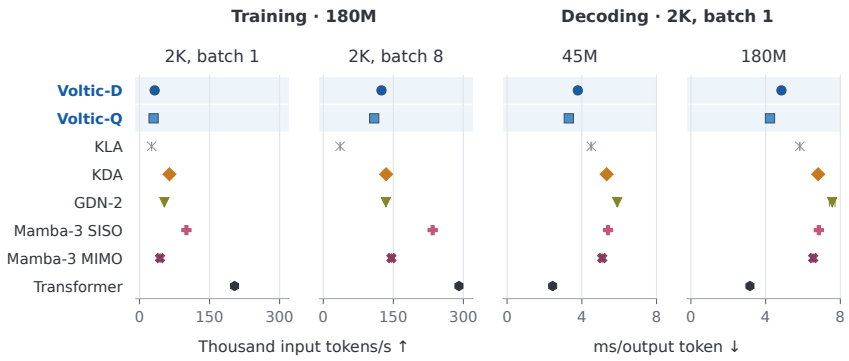


Figure 2: Training throughput (180M) and cached decoding latency (45M/180M) at 2K context on one H200 NVL: mean ± SD over three timing processes. Decoding excludes prompt processing.

Models and measurement. We benchmark the seven recurrent language-model architectures and the Transformer on one physical H200 NVL, using shared inputs and matched 45M/180M configurations distinct from the trained language models. Training timing covers forward, loss, and backward, excluding optimizer updates, data loading, compilation, and warmup. Appendix H specifies dimensions, precision, execution settings, and repetitions.

Results. We hold 8,192 input tokens per operation, increasing length from 1K to 8K as batch size falls from eight to one (Figure 6). The Transformer is fastest, followed by Mamba-3 SISO. Voltic-D and Voltic-Q exceed the updated KLA training backend at all these settings except 45M, 8K, batch one, where KLA is faster. This design also reduces the number of independent sequences as length grows. At 180M, the change from 1K to 8K reduces Voltic-D throughput by 65.4%, compared with 3.7% for KDA. Holding sequence length fixed separates the batch-size effect.

Next, we test batches of 1, 2, 4, and 8; Figure 2 shows the 180M endpoints at 2K tokens. Larger batches substantially improve Voltic's throughput. At 180M, 2K, and batch eight, Voltic-D reaches 92.6% of KDA and 93.0% of GDN-2 throughput. It is the fastest Voltic variant at this setting and reaches 3.5 times the throughput of KLA's updated training backend. At 180M and 8K tokens, increasing the batch from two to four raises Voltic-D throughput by 67.6%, compared with 5.2% for KDA. The uncertainty recursion processes tokens in order within a head, while independent sequences provide parallel work.

Cached autoregressive decoding. Training throughput does not determine decoding latency, which is the regime in which recurrent models are deployed. At batch one, timing covers output tokens including greedy selection and excludes prompt processing. Among the recurrent models, Voltic-Q and Voltic-D have the lowest latency at both scales and all prompt lengths, with one exception at 8K where KLA is slightly faster than Voltic-D. The Transformer is faster still at these lengths, though its per-token cost grows with context while the recurrent models' does not. Figure 2 shows 2K prompts; Appendix H.5 gives the timing protocol and cache checks.

5 Conclusion

Voltic derives recurrent writes from separate volatility and stochasticity while retaining efficient delta-rule computation. Its two variants lead the three MQAR tasks that vary these sources, four of five in-context retrieval groups, and the eight-task language-model average. Three limitations remain. Quality evaluation is limited to 45M models. Throughput depends on batch parallelism, and Voltic falls furthest behind gated baselines at long sequences with small batches. The two variants differ in both covariance structure and decay, so their complementary strengths are not attributed to one mechanism. Larger-scale validation remains future work. Deriving the write from an uncertainty recursion nonetheless makes recurrent memory more responsive to change, at a cost that stays close to gated baselines where batch parallelism is available.

AI USE STATEMENT

The authors used OpenAI's Codex assistant with GPT-6 Astra (OpenAI, 2026a;b) for drafting assistance and language editing, including revisions to improve the manuscript's grammar, clarity, and organization. The assistant also helped find and verify references, implement code and correctness tests, execute experiments across multiple GPUs, aggregate results, and prepare figures. The authors are responsible for the final text, claims, and artifacts.

ETHICS STATEMENT

This work studies recurrent sequence models using synthetic memory tasks and existing text datasets; it does not involve new human-subject data collection. We use public corpora and benchmarks under their applicable terms. Filtering educational web text does not eliminate biases, personal information, or harmful content, and the reported benchmarks do not establish suitability for deployment in sensitive settings. The work concerns memory and computational efficiency rather than deployment of a user-facing language model.

REPRODUCIBILITY STATEMENT

Appendix A derives the updates, and Appendix B specifies their optimized evaluation and gradients. Appendix D records software versions, hardware, and approximate training time. The benchmark appendices document data generation and preparation, model dimensions, optimizer searches, seeds, and scoring (Appendix E, Appendix F, Appendix G, and Appendix H). MQAR error bars report standard errors of the mean across training runs; the other benchmarks report sample standard deviations across training runs or timing processes, as specified in their captions. We will release the implementation and experiment configurations upon publication.

REFERENCES

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proc. VLDB Endow.*, 17(2):92–105, October 2023. ISSN 2150-8097. doi: 10.14778/3626292.3626294. URL https://doi.org/10.14778/3626292.3626294.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Y Zou, Atri Rudra, and Christopher Re. Zoology: Measuring and improving recall in efficient language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 15664–15730, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/448fc91f669c15d10364ee01d512cc10-Paper-Conference.pdf.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. Simple linear attention language models balance the recall-throughput tradeoff. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1763–1840. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/arora24a.html.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM:

Extended long short-term memory. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 107547–107603. Curran Associates, Inc., 2024. doi: 10.52202/079017-3417. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c2ce2f2701c10a2b2f2ea0bfa43cfaa3-Paper-Conference.pdf.

Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, C. James Taylor, and Gerhard Neumann. Recurrent Kalman Networks: Factorized inference in high-dimensional deep feature spaces. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 544–552. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/becker19a.html.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2026. URL https://arxiv.org/abs/2405.14782.

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006. ISBN 978-0-387-31073-2.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, April 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

Junxi Chen and Payam Piray. Categorical Bayes filtering for computational phenotyping in adaptive learning. *bioRxiv*, 2026. doi: 10.64898/2026.05.14.725268. URL https://www.biorxiv.org/content/early/2026/05/18/2026.05.14.725268.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Tri Dao. causal-conv1d: Causal depthwise convolution in CUDA. Software, 2023. URL https://github.com/Dao-AILab/causal-conv1d.

Tri Dao and Albert Gu. Mamba: State-space model software. Software, 2023. URL https://github.com/state-spaces/mamba.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Peter Dayan, Sham Kakade, and P. Read Montague. Learning and selective attention. *Nature Neuroscience*, 3(11):1218–1223, 2000. doi: 10.1038/81504. URL https://doi.org/10.1038/81504.

Arnaud Doucet, Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pp. 176–183, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607099.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, November 2018. ISSN 0893-6080. doi: 10.1016/j.neunet.2017.12.012. URL https://doi.org/10.1016/j.neunet.2017.12.012.

Xiaotong Fang and Payam Piray. Inferring the causes of noise from binary outcomes: A normative theory of learning under uncertainty. *Psychological Review*, 2026. doi: 10.1037/rev0000638. URL https://doi.org/10.1037/rev0000638.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In Lu Wang, Jackie Chi Kit Cheung, Giuseppe Carenini, and Fei Liu (eds.), *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5409. URL https://aclanthology.org/D19-5409/.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention, 2026. URL https://arxiv.org/abs/2605.22791.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi Linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep Kalman Filters, 2015. URL https://arxiv.org/abs/1511.05121.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 106519–106554. Curran Associates, Inc., 2024. doi: 10.52202/079017-3381. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c0d62e70dbc659cc9bd44cbcf1cb652f-Paper-Datasets_and_Benchmarks_Track.pdf.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Aakash Sunil Lahoti, Kevin Li, Berlin Chen, Caitlin Wang, Aviv Bick, Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 86173–86202, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8abd2043b71a074278d5f687947bff9c-Paper-Conference.pdf.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In Heike Adel and Shuming Shi (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. URL https://aclanthology.org/2021.emnlp-demo.21/.

Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL https://aclanthology.org/C02-1150/.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013/.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 95419–95434, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ee0e45ff4de76cbfdf07015a7839f339-Paper-Conference.pdf.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. OpenCeres: When open information extraction meets the semi-structured web. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3047–3056, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1309. URL https://aclanthology.org/N19-1309/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Thomas Peter Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001. URL https://hdl.handle.net/1721.1/86583.

Anthony Moi and Nicolas Patry. HuggingFace's Tokenizers. Software, 2023. URL https://github.com/huggingface/tokenizers.

NVIDIA. *CUDA C++ Programming Guide*, 2025. URL https://docs.nvidia.com/cuda/archive/13.0.0/cuda-c-programming-guide/index.html. Release 13.0.

OpenAI. Codex, 2026a. URL https://developers.openai.com/learn/codex. Online documentation. Accessed September 26, 2026.

OpenAI. GPT-6 Astra, 2026b. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Online documentation. Accessed September 26, 2026.

Manfred Opper. A Bayesian approach to on-line learning. In David Saad (ed.), *On-Line Learning in Neural Networks*, Publications of the Newton Institute, pp. 363–378. Cambridge University Press, Cambridge, 1999. doi: 10.1017/CBO9780511569920.017. URL https://doi.org/10.1017/CBO9780511569920.017.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.),

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Liangzu Peng, Aditya Chattopadhyay, Luca Zancato, Elvis Nunez, Wei Xia, and Stefano Soatto. Gated KalmaNet: A fading memory layer through test-time ridge regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20212–20222, June 2026.

Payam Piray and Nathaniel D. Daw. A model for learning based on the joint estimation of stochasticity and volatility. *Nature Communications*, 12(1):6587, Nov 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-26731-9. URL https://doi.org/10.1038/s41467-021-26731-9.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. Qwen Blog, September 2025. URL https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, April 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

Simo Särkkä and Ángel F. García-Fernández. Temporal parallelization of Bayesian smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, 2021. doi: 10.1109/TAC.2020.2976316. URL https://ieeexplore.ieee.org/document/9013038/.

Simo Särkkä and Aapo Nummenmaa. Recursive noise adaptive Kalman filtering by variational Bayesian approximations. *IEEE Transactions on Automatic Control*, 54(3):596–600, 2009.

doi: 10.1109/TAC.2008.2008348. URL https://ieeexplore.ieee.org/document/4796261/.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Vaisakh Shaj, Cameron Barker, Aidan Scannell, Andras Szecsenyi, Elliot J. Crowley, and Amos Storkey. Kalman linear attention: Parallel bayesian filtering for efficient language modeling and state tracking. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=9h7sSJe4jN.

Karen E. Smith, Payam Piray, Alyssa Teal, Selin Gok, and Seth D. Pollak. Early life stress affects human decision making by increasing expectations of volatility. *Developmental Science*, 29(4): e70238, 2026. doi: 10.1111/desc.70238. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/desc.70238.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Richard S. Sutton. Gain adaptation beats least squares? In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 161–166, New Haven, CT, 1992. Yale University. URL http://incompleteideas.net/papers/sutton-92b-remastered.pdf.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, PLDI '19, pp. 10–19. ACM, June 2019. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Guillaume Lajoie, Rif A. Saurous, Charlotte Frenkel, Razvan Pascanu, Blaise Aguera y Arcas, and Joao Sacramento. MesaNet: Sequence modeling by locally optimal test-time training. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 26397–26447, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/2c3d5a4eff20d24390a445762d61bfbb-Paper-Conference.pdf.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *Journal of Machine Learning Research*, 27 (169):1–41, 2026a. URL https://www.jmlr.org/papers/v27/25-0903.html.

Lei Wang, Yu Cheng, Yining Shi, Zhiwen Mo, Zhengju Tang, Wenhao Xie, Tong Wu, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, and Zhi Yang. TileLang: Bridge programmability and performance in modern neural kernels. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 73099–73134, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/76fb92288bf90360c527efb0d1c2aba6-Paper-Conference.pdf.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick

von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Songlin Yang and Yu Zhang. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522. Curran Associates, Inc., 2024. doi: 10.52202/079017-3668. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d13a3eae72366e61dfdc7eea82eeb685-Paper-Conference.pdf.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with delta rule. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4904fad153f6434a7bcf04465d4be2cc-Paper-Conference.pdf.

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

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/1e8a19426224ca89e83cef47f1e7f53b-Paper.pdf.

A Filtering Derivation and Approximation

A.1 Conditional feature-space model

For one head and value channel j, let $m_{t-1,j}$ be column j of the previous memory. The model in Equation 1 treats projected keys, transitions, and noise variances as given at each filtering step. Assume independent, zero-mean Gaussian process and observation noises, and a Gaussian previous state with mean $m_{t-1,j}$ and covariance U_{t-1}. Value channels share this covariance and have independent noises under the conditional model. Prediction gives

$$m^-_{t,j} = A_t m_{t-1,j}, \qquad P_t = A_t U_{t-1} A_t^\top + \mathrm{diag}(v_t). \tag{10}$$

The joint conditional distribution of the latent state and its scalar observation is

$$\begin{bmatrix} z_{t,j} \\ x_{t,j} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m^-_{t,j} \\ k_t^\top m^-_{t,j} \end{bmatrix}, \begin{bmatrix} P_t & P_t k_t \\ k_t^\top P_t & s_t + k_t^\top P_t k_t \end{bmatrix} \right). \tag{11}$$

Conditioning on $x_{t,j}$ therefore gives

$$d_t = s_t + k_t^\top P_t k_t = \gamma_t^{-1}, \qquad g_t = \gamma_t P_t k_t, \tag{12}$$

$$m_{t,j} = m^-_{t,j} + g_t(x_{t,j} - k_t^\top m^-_{t,j}), \qquad U_t = P_t - \gamma_t (P_t k_t)(P_t k_t)^\top. \tag{13}$$

Stacking the channel means yields Equation 3. The covariance update does not depend on the observed values, which makes it possible to compute gains separately from mean-memory updates.

The conditional construction motivates a neural memory update; it is not a Gaussian observation model for discrete language tokens. Projections and noise parameters are trained through the task loss. In MQAR, the model receives no labels identifying hidden changes or observation noise. Its feature-space variance parameters therefore need not equal the generator's change and replacement probabilities.

A.2 Connection to scalar delta writes

Suppose the predicted covariance at a step is $P_t = \rho_t I$ with $\rho_t > 0$. For a unit-normalized key,

$$g_t = \beta_t k_t, \qquad \beta_t = \frac{\rho_t}{s_t + \rho_t}, \qquad M_t = (I - \beta_t k_t k_t^\top) A_t M_{t-1} + \beta_t k_t x_t^\top. \tag{14}$$

With $A_t = I$, this has a delta-rule update; scalar decay $A_t = \alpha_t I$ gives the Gated DeltaNet form, and diagonal decay gives its coordinate-wise extension. This statement concerns the algebra of the mean update under an isotropic prediction. A rank-one observation correction generally produces non-isotropic posterior covariance, so isotropy is not automatically preserved by subsequent filtering steps. Nor does the connection identify separately learned erase and write vectors with a single Kalman gain.

In the isotropic case, the opposite effects of the two uncertainty terms follow directly from

$$\frac{\partial \beta_t}{\partial \rho_t} = \frac{s_t}{(s_t + \rho_t)^2} > 0, \qquad \frac{\partial \beta_t}{\partial s_t} = -\frac{\rho_t}{(s_t + \rho_t)^2} < 0. \tag{15}$$

Process noise increases predicted covariance in Equation 10. Observation noise instead increases the gain denominator. With non-isotropic covariance, the gain can also change direction; the scalar derivatives above do not assert that every signed gain coordinate changes monotonically with every variance parameter.

A.3 Diagonal projection and short-block covariance

The assumed-density projection in Section 3.3 minimizes the forward KL divergence from the Gaussian posterior to a diagonal Gaussian. To make its direction explicit, let the posterior at the current step be $p = \mathcal{N}(\mu, \Sigma)$ and the approximation be $q = \mathcal{N}(m, D)$, where D is positive and diagonal. Then

$$\mathrm{KL}(p\|q) = \frac{1}{2}\left[\mathrm{tr}(D^{-1}\Sigma) + (m-\mu)^\top D^{-1}(m-\mu) - N + \log \frac{\det D}{\det \Sigma} \right]. \tag{16}$$

Minimizing first over m gives $m = \mu$. For a diagonal entry D_{ii}, the remaining derivative is $(D_{ii} - \Sigma_{ii})/(2D_{ii}^2)$, so the minimum is $D_{ii} = \Sigma_{ii}$. The projection therefore retains the mean and marginal variances and discards the covariances between coordinates. Here p is the posterior obtained from the incoming approximate state.

Voltic-D applies this projection after every token. With diagonal A_t, its predicted covariance is diagonal, and taking the diagonal of Equation 13 gives Equation 8.

Voltic-Q applies the same projection only at boundaries of Q-token blocks. Within each block, it uses the full covariance update implied by the diagonal covariance received at the block's start. For this fixed-decay family, $Q = 1$ gives the diagonal update with $A_t = I$, while a single block covering the whole sequence gives the full fixed-decay filter. The evaluated Voltic-D additionally learns A_t, as specified in Section 3.3. The diagonal-plus-low-rank representation in Appendix B.3 evaluates the within-block updates without forming a dense matrix.

For positive s_t and positive semidefinite P_t, the posterior covariance remains positive semidefinite. To see this, set $w = P_t^{1/2} k_t$. Then

$$U_t = P_t^{1/2} \left(I - \frac{w w^\top}{s_t + w^\top w} \right) P_t^{1/2}. \tag{17}$$

The middle matrix has eigenvalues one orthogonal to w and $s_t/(s_t + \|w\|^2) > 0$ along w. Keeping only a covariance diagonal also preserves nonnegative marginal variances. These are properties of the mathematical updates; the precision and evaluation choices of the GPU implementation are specified in Appendix B.6.

A.4 Computational scope

A direct dense covariance update uses $O(N^2)$ work and storage per head at each step, in addition to the $N \times P$ mean memory. Voltic-D reduces uncertainty work and recurrent storage to $O(N)$. For fixed-decay Voltic-Q, at most Q rank-one corrections give $O(QN)$ work per token and active uncertainty storage. Both variants preserve the same affine mean-memory form for the chunked kernel. These costs describe uncertainty computation; full training also stores or recomputes activations needed for gradients.

B Implementation Details

B.1 Sequence-mixing block

The MQAR and 45M language-model configurations use the following block. Let $h_t \in \mathbb{R}^{d_{\text{model}}}$ be its normalized input and H its number of heads. Each head has a key of dimension N, a value of dimension P, and a mean memory of size $N \times P$. Separate linear projections produce queries, keys, and values. To include nearby tokens in these features, each projected channel passes through a causal convolution of width four, followed by SiLU (Elfwing et al., 2018). Writing the resulting vectors as $\widetilde{q}_{t,h}$, $\widetilde{k}_{t,h}$, and $x_{t,h}$, we use

$$q_{t,h} = N^{-1/2} \widetilde{q}_{t,h}, \qquad k_{t,h} = \frac{\widetilde{k}_{t,h}}{\|\widetilde{k}_{t,h}\|_2}, \qquad o_{t,h} = M_{t,h}^\top q_{t,h}. \tag{18}$$

Thus, q_t in the memory equations already includes the factor $N^{-1/2}$; queries receive no unit-length normalization. The implementation protects key normalization against division by zero.

The output combines the memory readout $o_{t,h}$ with the current value $x_{t,h}$. A trainable scalar D_h, initialized to one, controls this direct value contribution. After concatenating the heads, an input-dependent gate reweights the value channels before RMSNorm (Zhang & Sennrich, 2019) rescales the resulting vector:

$$\begin{aligned} y_t &= \operatorname{concat}_{h=1}^{H}(o_{t,h} + D_h x_{t,h}), \\ \operatorname{Mixer}(h_t) &= W_{\text{out}}\, \operatorname{RMSNorm}\big(y_t \odot \operatorname{SiLU}(W_{\text{gate}} h_t)\big). \end{aligned} \tag{19}$$

RMSNorm operates over all HP concatenated channels, with $\epsilon = 10^{-5}$ and trainable weights initialized to one. The output projection returns to d_{model} for the residual addition. The benchmark appendices specify the number of layers, head dimensions, and separate feed-forward branches.

B.2 Noise and decay parameterization

The model must produce positive noise variances from unconstrained projection outputs. Let r_t^v and r_t^s be the volatility and observation-noise projections of h_t, respectively. We first compute their log rates,

$$\ell_t^v = -\exp(\theta_v) \odot \operatorname{softplus}(r_t^v + b_v), \qquad \ell_t^s = -\exp(\theta_s)\operatorname{softplus}(r_t^s + b_s), \tag{20}$$

and then set $v_t = \exp(\ell_t^v)$ and $s_t = \exp(\ell_t^s)$. The volatility bias and log scale have one entry per head and state coordinate; the observation-noise bias and log scale have one entry per head. All are trainable. For finite real-valued parameters, each log rate is negative, so $0 < v_{t,i} < 1$ and $0 < s_t < 1$. The filtering equations allow any positive variances; the evaluated networks use this bounded parameterization in their learned feature space. The selected configurations use no additional variance floor or convolution in these branches.

For learned-decay Voltic-D, the decay should remain between $\alpha_{\min} = 0.2$ and one. Given the decay projection r_t^α, define

$$a_t^{\mathrm{raw}} = \exp(\theta_A)\operatorname{softplus}(r_t^\alpha + b_\alpha), \qquad F = -\log \alpha_{\min} > 0. \tag{21}$$

We obtain a smoothly bounded decay through

$$\log \alpha_t = F\left[\exp\left(-a_t^{\mathrm{raw}}/F\right) - 1\right]. \tag{22}$$

The operations are coordinate-wise, with a learned scale per head. Since a_t^{raw} is nonnegative, this maps the projection to $\alpha_{\min} \le \alpha_t \le 1$. Voltic-Q uses $\alpha_t = 1$ and has no learned decay projection.

B.2.1 Initialization

Each independent sequence starts with $M_0 = 0$ and a diagonal uncertainty u_0 stored as a fixed buffer. The two Voltic variants share the following initialization of u_0 and the noise parameters, which spreads initial retention across heads and state coordinates. The rates below are nominal values at zero projection output; actual token-dependent rates also depend on the initialized projection weights.

For head $h = 1, \dots, H$, the target retention logit is

$$\eta_h = \operatorname{logit}(0.98) + \left[\operatorname{logit}(0.20) - \operatorname{logit}(0.98)\right]\left(\frac{h-1}{H-1}\right)^{1.1}, \qquad \lambda_h = \operatorname{sigmoid}(\eta_h). \tag{23}$$

For one head, the fraction in this expression is defined as zero, giving $\lambda_1 = 0.98$. Set the nominal noise ranges to $s_{\min} = 0.1$, $s_{\max} = 0.98$, $v_{\min} = 0.01$, and $v_{\max} = 0.45$. We choose

$$\begin{aligned} \log s_h^{(0)} &= \tfrac{1}{2}\left[\max(\log s_{\min}, \log v_{\min} + \eta_h) + \min(\log s_{\max}, \log v_{\max} + \eta_h)\right], \\ \xi_h^* &= 1 - \frac{\lambda_h v_{\min}}{(1-\lambda_h)s_h^{(0)}}, \qquad \xi_h = \min(\max(\xi_h^*, 0), 1/2). \end{aligned} \tag{24}$$

The fraction ξ_h assigns part of the initial variance budget to u_0 and the rest to process noise. To spread that budget across coordinates, define

$$\eta_h^{\mathrm{hi}} = \log \frac{(1-\xi_h)s_h^{(0)}}{v_{\min}}, \qquad \eta_h^{\mathrm{lo}} = \log \frac{(1-\xi_h)s_h^{(0)}}{v_{\max}}. \tag{25}$$

For $\xi_h^* > 1/2$, coordinate $i = 1, \dots, N$ has retention logit

$$\eta_{h,i} = \eta_h^{\mathrm{hi}} - (\eta_h^{\mathrm{hi}} - \eta_h^{\mathrm{lo}})\left(\frac{i-1}{N-1}\right)^{a_h}, \qquad a_h = \frac{\log\left[(\eta_h^{\mathrm{hi}} - \eta_h)/(\eta_h^{\mathrm{hi}} - \eta_h^{\mathrm{lo}})\right]}{\log(1/2)}. \tag{26}$$

Otherwise, the first half of coordinates use η_h^{hi} and the second half use

$$\eta_{h,i} = \eta_h^{\mathrm{hi}} - (\eta_h^{\mathrm{hi}} - \eta_h^{\mathrm{lo}})\left(\frac{i - N/2}{N/2}\right)^2, \qquad i = N/2+1, \dots, N. \tag{27}$$

The evaluated state widths are even. These logits determine

$$B_{h,i} = s_h^{(0)}\exp(-\eta_{h,i}), \qquad u_{0,h,i} = \xi_h B_{h,i}, \qquad v_{h,i}^{(0)} = \operatorname{clip}\big((1-\xi_h)B_{h,i}, v_{\min}, v_{\max}\big). \tag{28}$$

The budget satisfies $s_h^{(0)}/(s_h^{(0)} + B_{h,i}) = \mathrm{sigmoid}(\eta_{h,i})$, specifying the nominal retention along an individual coordinate at unit decay. With $\phi_0 = 0.01$, the noise biases and log scales are initialized as

$$(b_s)_h = (b_v)_{h,i} = \mathrm{softplus}^{-1}(\phi_0),$$
$$\theta_{s,h} = \log \frac{-\log s_h^{(0)}}{\phi_0}, \qquad \theta_{v,h,i} = \log \frac{-\log v_{h,i}^{(0)}}{\phi_0}. \tag{29}$$

Substituting zero projection outputs in Equation 20 recovers the nominal rates above. Voltic-D initializes $\theta_A = 0$ and $b_\alpha = \mathrm{softplus}^{-1}(10^{-4})$, giving $\alpha \approx 0.9999$ at zero decay-projection output. Voltic-Q uses exact unit decay throughout.

B.2.2 Fused rate computation and backward

The noise projections have no recurrent dependencies, so one token-parallel kernel evaluates Equation 20 and its exponentials together. It writes the positive rates directly, without storing the log-rate arrays. For Voltic-D, the same kernel also computes α_t from $\log \alpha_t$, moving this exponential outside the uncertainty loop.

The backward pass reuses the saved rates to avoid retaining each intermediate activation. For one scalar noise coordinate, write $c = \exp(\theta)$, $\phi = \mathrm{softplus}(r + b)$, and $\nu = \exp(-c\phi)$. Here ν denotes either a volatility coordinate or the observation-noise variance. When $\nu > 0$, the kernel reconstructs

$$\phi = -\frac{\log \nu}{c}, \qquad \frac{\partial \nu}{\partial r} = -c\nu(1 - \exp(-\phi)), \qquad \frac{\partial \nu}{\partial \theta} = \nu \log \nu. \tag{30}$$

The bias has the same local derivative as r. The kernel multiplies these derivatives by the incoming loss gradient and sums parameter contributions over tokens. Rates that underflow to zero receive zero local gradients, avoiding logarithms of zero. Under deterministic training, parameter-gradient sums use separate partial reductions instead of unordered atomic additions.

B.3 Evaluating Voltic-Q with diagonal and low-rank terms

Each observation subtracts a rank-one term from the Voltic-Q covariance. We retain the vector and scalar defining each subtraction, allowing later updates to use these terms without constructing the full matrix. Consider a block starting after token r and ending at $e = \min(r + Q, T)$, with uncertainty diagonal u_r passed from the previous block. Define the accumulated diagonal c_t, update direction w_t, and inverse denominator γ_t as

$$c_t = u_r + \sum_{j=r+1}^{t} v_j, \qquad w_t = P_t k_t, \qquad \gamma_t = (s_t + k_t^\top w_t)^{-1}. \tag{31}$$

With γ_t as defined in Equation 2, the covariance before the observation update at token t is

$$P_t = \mathrm{diag}(c_t) - \sum_{j=r+1}^{t-1} \gamma_j w_j w_j^\top. \tag{32}$$

Multiplying this expression by k_t gives an update using only the previous direction vectors:

$$w_t = c_t \odot k_t - \sum_{j=r+1}^{t-1} \gamma_j w_j (w_j^\top k_t), \qquad g_t = \gamma_t w_t. \tag{33}$$

At the block's end, the uncertainty diagonal passed to the next block is

$$u_e = c_e - \sum_{j=r+1}^{e} \gamma_j w_j^{\odot 2}. \tag{34}$$

These equations evaluate Equation 9 without forming its dense matrices. The direction computation takes $O(QN)$ work per token and the active block contains $O(QN)$ direction entries. Training saves per-token directions, accumulated diagonals, and inverse denominators for the backward pass.

The main MQAR and language-model quality experiments use $N = 128$ with $Q = 4$. The uncertainty kernel processes tokens in order, distributing the vector coordinates across GPU threads. Threads synchronize before the next token uses a newly stored direction and inverse denominator. The throughput studies use $N = 64$ and the register-based kernels described in Appendix H.2.

B.3.1 BACKWARD THROUGH THE UNCERTAINTY BLOCK

Let a bar denote a loss gradient, and let $\overline{g}_t$ and $\overline{u}_e$ be the incoming gradients from the memory updates and the next covariance block. The following reverse computation differentiates Equation 33 and Equation 34 using vectors and scalars. Initialize all accumulated gradients to zero, then set

$$\begin{aligned} \overline{w}_t &= \gamma_t \overline{g}_t - 2\gamma_t \overline{u}_e \odot w_t, \\ \overline{\gamma}_t &= \overline{g}_t^\top w_t - \overline{u}_e^\top w_t^{\odot 2}, \qquad \overline{c}_e = \overline{u}_e. \end{aligned} \tag{35}$$

Visit tokens in reverse order, from e to $r+1$. Differentiating the denominator gives $\delta_t = -\gamma_t^2 \overline{\gamma}_t$. At each token, first apply

$$\overline{s}_t = \delta_t, \qquad \overline{k}_t = \delta_t w_t, \qquad \overline{w}_t \mathrel{+}= \delta_t k_t, \tag{36}$$

where $+=$ denotes adding a contribution to an accumulated gradient. Using the updated $\overline{w}_t$, the diagonal product contributes

$$\overline{c}_t \mathrel{+}= \overline{w}_t \odot k_t, \qquad \overline{k}_t \mathrel{+}= \overline{w}_t \odot c_t. \tag{37}$$

For every earlier token j in the same block, define $\tau_{jt} = w_j^\top k_t$ and $\chi_{jt} = \overline{w}_t^\top w_j$. The rank-one terms contribute

$$\begin{aligned} \overline{\gamma}_j &\mathrel{-}= \tau_{jt}\chi_{jt}, \\ \overline{w}_j &\mathrel{-}= \gamma_j(\tau_{jt}\overline{w}_t + \chi_{jt}k_t), \\ \overline{k}_t &\mathrel{-}= \gamma_j \chi_{jt} w_j. \end{aligned} \tag{38}$$

After reversing the block, the cumulative diagonal in c_t gives

$$\overline{v}_t = \sum_{i=t}^{e} \overline{c}_i, \qquad \overline{u}_r = \sum_{i=r+1}^{e} \overline{c}_i. \tag{39}$$

Passing $\overline{u}_r$ to the preceding block accounts for all later uncertainty updates. Key gradients from the memory operator are added to $\overline{k}_t$ before differentiating key normalization and the input projection. The noise gradients pass through Equation 30. This reverse computation needs $O(QN)$ active direction gradients, in addition to the forward arrays saved for training.

B.4 CHUNKED MEMORY COMPUTATION AND OVERLAP

To use the generalized delta-rule operator in Flash Linear Attention (Yang & Zhang, 2024), expand Equation 3 as

$$M_t = (A_t + b_t a_t^\top) M_{t-1} + \kappa_t x_t^\top, \qquad a_t = \alpha_t \odot k_t, \quad b_t = -g_t, \quad \kappa_t = g_t. \tag{40}$$

Because A_t is diagonal, $a_t^\top = k_t^\top A_t$, and the right-hand side equals $(I - g_t k_t^\top) A_t M_{t-1} + g_t x_t^\top$. The kernel receives the diagonal decay as $\log \alpha_t$ and organizes the affine memory updates into chunks of $C = 16$ tokens.

The gains depend on keys, noise variances, decay, and previous uncertainty. They do not depend on values or the mean memory, so the uncertainty kernel can run ahead of the memory kernel. We divide the sequence into groups of S tokens, called panels, and assign the two kernels to separate CUDA streams (NVIDIA, 2025). Once the gains for a panel are ready, the memory stream processes that panel while the uncertainty stream prepares the next one. CUDA events enforce the required dependencies. This overlap is available when a sequence contains multiple panels.

The covariance block length Q determines when Voltic-Q discards off-diagonal covariance. The memory chunk length C and panel length S organize the evaluation of the resulting recurrence. Choosing S as a multiple of both C and Q keeps panel boundaries aligned with memory chunks and covariance blocks. For Voltic-D, only alignment with C is required. The benchmark appendices give the selected panel lengths.

B.5 STATES AND GRADIENTS ACROSS PANELS

At each panel boundary, the forward pass carries the uncertainty and mean memory to the next panel. The backward pass carries their loss gradients in the opposite direction, allowing a later prediction

to affect an earlier memory update. A panel boundary therefore does not truncate either gradient path.

In Equation 40, the same gain enters the memory operator's write term through $\kappa_t = g_t$ and its erase term through $b_t = -g_t$. With the gradient notation above, the chain rule gives

$$\overline{g}_t = \overline{\kappa}_t - \overline{b}_t, \qquad \kappa_t = g_t, \quad b_t = -g_t. \tag{41}$$

The first term accounts for the gain's effect on writing the new value; the second accounts for its effect on erasing the old prediction. Both must reach the uncertainty backward pass because both depend on the same gain.

Learned decay also affects more than one computation: it changes predicted uncertainty, the factor $a_t = \alpha_t \odot k_t$, and the memory kernel's $\log \alpha_t$ input. The contribution through a_t is

$$\overline{\log \alpha_t}^{(a)} = \overline{a}_t \odot \alpha_t \odot k_t. \tag{42}$$

We add this term to the direct decay-input gradient and the contribution from the uncertainty update before differentiating the decay parameterization in Appendix B.2.

B.6 Kernel specialization and execution settings

We implement model layers and automatic differentiation in PyTorch (Paszke et al., 2019), with custom GPU kernels written in Triton (Tillet et al., 2019). Table 4 lists the software versions. The following details describe the MQAR and language-model quality configurations; Appendix H.2 gives the choices for throughput measurements.

B.6.1 Fused projections and short convolutions

Query, key, value, output-gate, and noise projections share one matrix multiplication; Voltic-D includes the decay projection in the same operation. Their weights remain separately trainable parameters. The selected short-convolution backend is `causal-conv1d` (Dao, 2023), which applies the convolution bias and SiLU activation in the convolution kernel. The three query, key, and value convolutions are invoked separately.

B.6.2 Panel inputs and saved intermediates

The Voltic-D uncertainty kernel writes the gain, $a_t = \alpha_t \odot k_t$, and the memory decay input into contiguous arrays for the current panel. The memory kernel consumes these arrays directly. This avoids concatenating all panels into a full-sequence gain array before starting memory computation. We retain the matrix products needed by the analytic backward pass. In the sign-specialized kernels, a product involving $-g_t$ is reconstructed by negating the corresponding product involving g_t after loading it into registers. This removes separate arrays and GPU launches for these negations in both forward and backward computation.

B.6.3 Compact memory computation for Voltic-Q

For Voltic-Q, fixed decay implies $\log \alpha_t = 0$ and $a_t = k_t$. When the processed sequence fits in one panel, the specialized path omits cumulative decay calculations, multiplication by unit decay factors, and their backward derivatives. A single preparation kernel forms the within-chunk products, replacing separate input-packing and decay-prefix operations. It also omits the stored zero-decay array and reuses the normalized key directly. This path applies to processed document segments as well as complete sequences. Longer inputs use the panel pipeline while carrying uncertainty and memory between panels.

B.6.4 Precision and execution choices

Parameters and optimizer states are FP32. Under BF16 autocasting, the projections and memory-kernel inputs use BF16, while uncertainty updates and their reductions use FP32. For Voltic-D, the kernel evaluates the posterior diagonal as $u_{t,i} = p_{t,i}(d_t - p_{t,i}k_{t,i}^2)/d_t$, an algebraic rearrangement of Equation 8. The surrounding PyTorch layers execute eagerly. MQAR fixes $S = 256$. The 45M language models select S by timing supported values before training, restoring model parameters and random-number state afterward. Appendix F.3 gives the timing candidates.

C MQAR COVARIANCE-BLOCK ABLATION

The main experiments use Voltic-Q with $Q = 4$. Here we compare it with $Q = 16$ on MQAR. The longer block retains covariance between state coordinates over more tokens before discarding the off-diagonal entries. Both configurations use fixed decay and the same model dimensions, parameter count, and training protocol in Appendix E.2. Each selects its learning rate independently by the procedure in Appendix E.3. For $Q = 16$, the selected rates are 0.003 on Standard and Volatile MQAR and 0.01 on Stochastic MQAR and Vol & Sto.

Table 3 compares the block lengths at all four memory sizes. Averaging the two training sizes within each run, their mean accuracies differ by less than 0.3 percentage points on every task. Both are nearly perfect on Standard MQAR; $Q = 16$ has a slightly higher mean on Volatile MQAR, while $Q = 4$ has slightly higher means on Stochastic MQAR and Vol & Sto.

Table 3: MQAR covariance-block comparison: accuracy (%), mean ± SEM over five runs. Bold marks the higher displayed mean for each task and memory size; ties are both bold.

Model	Standard	Volatile	Stochastic	Vol & Sto
(a) $K = 16$, $T = 128$				
Voltic-Q ($Q = 4$)	**100.00** ± 0.00	99.82 ± 0.06	**88.66** ± 0.13	**73.52** ± 0.23
Voltic-Q ($Q = 16$)	**100.00** ± 0.00	**99.88** ± 0.05	88.45 ± 0.24	73.14 ± 0.29
(b) $K = 64$, $T = 512$				
Voltic-Q ($Q = 4$)	**100.00** ± 0.00	99.13 ± 0.42	**86.56** ± 0.53	72.25 ± 0.34
Voltic-Q ($Q = 16$)	**100.00** ± 0.00	**99.53** ± 0.11	86.39 ± 0.59	**72.42** ± 0.26
(c) $K = 128$, $T = 1,024$				
Voltic-Q ($Q = 4$)	97.65 ± 0.62	69.36 ± 5.21	68.96 ± 1.44	61.23 ± 1.16
Voltic-Q ($Q = 16$)	**98.01** ± 0.96	**70.46** ± 2.91	**73.27** ± 2.89	**62.21** ± 0.93
(d) $K = 256$, $T = 2,048$				
Voltic-Q ($Q = 4$)	**43.51** ± 6.97	15.33 ± 3.98	27.74 ± 5.57	23.70 ± 2.81
Voltic-Q ($Q = 16$)	41.28 ± 6.24	**20.67** ± 4.32	**32.34** ± 5.83	**25.52** ± 2.31

At the extrapolation sizes, $Q = 16$ has higher mean accuracy on Volatile MQAR, Stochastic MQAR, and Vol & Sto at both $K = 128$ and $K = 256$. On Standard MQAR, $Q = 16$ is slightly higher at $K = 128$, while $Q = 4$ is higher at $K = 256$. Thus, retaining covariance over a longer block can help with extrapolation, even though its effect at the training sizes is small in these experiments.

D SOFTWARE, HARDWARE, AND TRAINING TIME

D.1 SOFTWARE ENVIRONMENT

Table 4 lists the main packages in the MQAR and 45M language-model environment. PyTorch and Triton provide the framework and custom kernels described in Appendix B.6; FLA provides the generalized delta-rule operations. The Mamba package (Dao & Gu, 2023) and TileLang (Wang et al., 2026b) support the Mamba-3 baselines. For language modeling, Transformers (Wolf et al., 2020) and Accelerate (Gugger et al., 2022) provide model and training utilities, while Datasets (Lhoest et al., 2021) and Tokenizers (Moi & Patry, 2023) support data preparation and tokenization. NumPy (Harris et al., 2020) supports data generation and array processing, and Einops supplies tensor reshaping operations. Language Model Evaluation Harness is used as described in Section 4.3.

The xLSTM baselines use the original mLSTM and sLSTM blocks from the official xLSTM repository, version 2.0.6 at revision `ab22eadbd245`. The source is unchanged; the MQAR interface and removal of the hybrid's external feed-forward branch are described in Appendix E.2.

Retrieval evaluations use the PyTorch and CUDA versions in Table 4. Training-throughput and cached-decoding measurements use PyTorch 2.13.0+cu130 and Triton 3.7.1; their Python, CUDA, FLA, `causal-conv1d`, `mamba-ssm`, and TileLang versions match the corresponding table entries.

KLA language modeling and retrieval use the `cuda_v3_fast` backend based on the official implementation at revision `ad5e64de903e`, with deterministic gradient reductions. Training-throughput measurements use that release's unmodified CUDA backend; cached decoding uses its state-producing Triton prefill and single-token step implementation (Appendix H). LongBench's official metrics use the `rouge` package, version 1.0.1; benchmark code and dataset revisions are specified in Appendix G.

Table 4: Main software versions for MQAR and 45M language modeling.

Package or runtime	Version
Python	3.11.14
PyTorch	2.12.0+cu130
CUDA	13.0
Triton	3.7.0
Flash Linear Attention / FLA Core	0.5.1 / 0.5.1
`causal-conv1d`	1.6.2.post1
`mamba-ssm`	2.3.2.post1
TileLang	0.1.8
Transformers	5.9.0
Accelerate	1.13.0
Datasets	4.8.5
Tokenizers	0.22.2
NumPy	2.4.6
Einops	0.8.2
Language Model Evaluation Harness	0.4.12

D.2 Hardware

Each training run uses one GPU. In the MQAR campaign, Voltic, KLA, and Mamba-3 MIMO use NVIDIA H200 NVL GPUs; the other models use RTX A6000 GPUs. Evaluation-only variance interventions use H200 NVL or RTX A6000, with each intervention and its full control evaluated on the same device. At 45M scale, Voltic, KLA, and both Mamba-3 variants use H200-class GPUs, while KDA and GDN-2 use RTX A6000 GPUs; the Transformer runs use both GPU types. All runs within each study retain the model dimensions, data, and optimization settings specified in its experimental setup.

The original RULER and BASED retrieval evaluations use NVIDIA Ampere and Ada GPUs: RTX 3090, RTX A4000/A6000, L4/L40, RTX 4060 Ti, RTX 4070 Ti SUPER, RTX 4080 SUPER, RTX 4090, and RTX 4500/6000 Ada. LongBench and BABILong use RTX 3090 and RTX A6000 GPUs. KLA's evaluations across all four groups use H200 and H200 NVL GPUs. The same prompts and scoring rules are used across devices.

All reported training-throughput and cached-decoding measurements run serially on the same physical H200 NVL at a 600 W power limit. The fixed-token training view is a subset of this device's batch-size grid; it replaces the earlier standalone sequence-length measurements on a different H200 NVL. Baseline updates and recovered large-batch measurements retain the same device and runtime.

D.3 Approximate training time

Table 5 gives approximate training time over the recorded optimizer updates. For language modeling, we reconstruct this time from the token counts and processing rates in the training logs, counting each completed update once across checkpoint restarts. MQAR records its training-loop duration directly. These times include data loading and checkpoint writes within the training intervals. They exclude data preparation, model initialization, final downstream evaluation, and time waiting to restart a job.

The MQAR row covers completed learning-rate searches and fresh runs for the four tasks, including the covariance-block study in Appendix C and the 20 additional Transformer runs at peak rate 0.001 in Appendix E.6. It excludes twelve numerically failed search candidates, evaluation-only interventions, and other Transformer diagnostics, including the curriculum experiment. The 45M row covers

Table 5: Approximate training time: completed MQAR search, fresh-seed, and learning-rate follow-up runs, and 45M final runs.

Study	Runs	Hours per run	Summed hours
MQAR, 10,000 updates	316	0.02–0.24	31.6
45M, 4,800 updates	24	2.1–34.5	285

three final seeds per reported model. These times cover the recorded intervals of completed runs, rather than total allocated GPU time.

E MQAR Experimental Details

E.1 Sequence generation

The four tasks in Section 4.2 use sequences of length $T = 8K$ with K distinct keys. Keys are sampled without replacement from token IDs 1–4,095. Initial values are sampled without replacement from IDs 4,096–8,191. Unassigned positions contain uniform random tokens from the full vocabulary, including IDs that may also appear as keys or values. Each key has one designated query. Only predictions at these query-key positions contribute to the loss and accuracy; the output at a query is compared directly with its target value, without shifting the labels to another position.

Observation rounds and queries. All four tasks share three observation rounds and the same token layout. Each round visits all K keys in an independently shuffled order, occupying $2K$ tokens. Queries occupy every other position in the last $2K$ tokens, in an independently shuffled key order. The target is the final hidden value.

Let h be the probability that a key's hidden value changes before rounds two and three, and let e be the probability that a displayed value is incorrect. A hidden change samples uniformly from the 4,095 values different from the preceding hidden value. Independently, an incorrect observation samples uniformly from the 4,095 alternatives to the current hidden value. Observation noise applies in all three rounds, including the first. Values may coincide across keys or rounds after a change or replacement. The four settings are

$$(h, e) \in \{(0, 0), (0.30, 0), (0, 0.25), (0.30, 0.25)\}, \tag{43}$$

corresponding to Standard MQAR, Volatile MQAR, Stochastic MQAR, and Vol & Sto MQAR. Models receive no change or noise indicators. Random draws for keys, initial values, observation order, query order, and filler positions are shared across the four conditions; the thresholds h and e determine which candidate changes and replacements take effect. Thus, these comparisons hold the number and positions of observations fixed.

Data sharing and splits. Training, validation, and test generation use separate random-number seeds so that the splits can be reproduced independently. Training data depend on the task, run seed, batch index, and sequence configuration. They do not depend on the model or learning-rate candidate. All rate candidates share initialization and training data within a model–task pair. For each run seed, all models also receive identical initial embedding weights, sampled from a zero-mean normal distribution with standard deviation 0.02. These embeddings are then trained independently. Validation and test examples are fixed across run seeds and generated independently of training data. Evaluation generates examples in batches of 16, then processes each batch through the model in groups of four. Changing this forward-batch size therefore does not change the examples.

E.2 Models and optimization

Table 6 gives parameter counts and selected learning rates. All models use two layers, hidden width $d_{\text{model}} = 128$, and 1,048,576 parameters in a shared embedding/output matrix. We remove the separate feed-forward branch and its normalization while retaining each mixer's internal gates and projections. Total counts span 1.18–1.39 million parameters.

Voltic. Both variants use one head with key/state width $N = 128$ and value width $P = 128$, giving a 128×128 mean memory per layer. They use the block in Appendix B.1, including width-four

Table 6: MQAR parameter counts and selected peak learning rates. Other parameters exclude the shared embedding/output matrix.

Model	Total	Other	Std.	V	S	Vol & Sto
Voltic-D	1,283,464	234,888	0.003	0.003	0.003	0.003
Voltic-Q	1,250,438	201,862	0.003	0.003	0.003	0.010
KLA	1,263,234	214,658	0.003	0.003	0.003	0.003
KDA	1,315,202	266,626	0.003	0.030	0.003	0.003
GDN-2	1,380,482	331,906	0.003	0.003	0.003	0.003
Mamba-3 SISO	1,286,800	238,224	0.003	0.010	0.003	0.003
Mamba-3 MIMO	1,394,320	345,744	0.003	0.003	0.010	0.003
xLSTM (mLSTM)	1,267,600	219,024	0.003	0.003	0.003	0.003
xLSTM (hybrid)	1,192,328	143,752	0.003	0.003	0.003	0.003
Transformer	1,180,032	131,456	0.001	0.001	0.003	0.003

query, key, and value convolutions and gating before full-width RMSNorm. The memory-kernel chunk length is $C = 16$ and the panel length is $S = 256$, with fused projections, precomputed noise rates, and saved intermediates in the specialized memory kernel. Voltic-D's learned decay is initialized near one and bounded below by 0.2 (Appendix B.2.1). Voltic-Q uses fixed decay $\alpha_t = 1$ and covariance-block length $Q = 4$.

KDA and GDN-2. Both use head dimension 128 and causal short convolutions of width four. GDN-2 uses value expansion one.

KLA. We use state dimension 64, expansion one, rank-64 time-step projection, Ornstein–Uhlenbeck dynamics, and learned process noise initialized at 0.01. The value stream comes from the width-four convolution. The observation-variance projection uses the time-step rank, and its minimum observation variance is 10^{-4}. Queries and keys are normalized, the output is gated, and the composed Triton backend uses scan checkpointing.

Mamba-3. Both variants use expansion two, head dimension 64, state dimension 64, one group, and rotary fraction 0.5. The chunk lengths are 64 for SISO and 16 for MIMO, with MIMO rank four.

xLSTM. The mLSTM configuration uses two matrix-memory blocks. Each expands the hidden width to 256 and uses four heads with 64×64 matrix memories, giving 16,384 matrix-state elements per layer, equal to Voltic's mean-memory size. The hybrid uses an mLSTM block followed by a scalar-memory sLSTM block of width 128 with four heads. Its scalar layer has fewer recurrent-state elements, and its total parameter count is smaller. Both use width-four causal convolutions, tied embeddings, zero dropout, and the upstream output normalization. The hybrid's separate feed-forward branch is removed to follow the common MQAR configuration; it is an adaptation of the original architecture. We use the official parallel mLSTM implementation and the official CUDA sLSTM implementation with FP32 recurrent arithmetic. The source revision is recorded in Appendix D.1.

Transformer. We use four causal attention heads and rotary position embeddings (Su et al., 2024) over the full head dimension, with base 10,000 and maximum position 2,048. Attention dropout is zero. Like the other baselines, this is a small configuration of the model defined in Section 4.1, using the common MQAR network described above. Tables and figures place this full-attention reference after the recurrent models.

Optimizer and numerical settings. AdamW uses moment coefficients $(0.9, 0.95)$, numerical-stability constant $\epsilon = 10^{-8}$, and global gradient-norm clipping at one. The learning rate warms up for 50 updates, then follows cosine decay to 0.1 times its peak at update 10,000. Every batch produces one optimizer update, without gradient accumulation. All parameter groups use learning-rate multiplier one, including KLA dynamics and Voltic rate parameters. Weight decay is 0.1 for the matrix parameter groups, including the shared embedding and Voltic rate projections; biases, normalization parameters, and static dynamics marked for exclusion by the model implementation use zero decay.

Every run uses 10,000 updates with batch size 16, alternating $(T, K) = (128, 16)$ and $(512, 64)$. Loss is computed only at designated query positions. Parameters are stored in FP32. The forward

pass uses bfloat16 (BF16) autocasting while retaining each implementation's recurrent arithmetic. Query logits are converted to FP32 for the loss. Both matrix-multiplication and cuDNN TensorFloat-32 (TF32) modes are disabled. There is no outer `torch.compile` wrapper. Appendix D lists the software versions and hardware.

E.3 Selection and metrics

The initial peak learning-rate grid is $\{0.003, 0.01, 0.03\}$. For each model and task, we train each candidate with seed 600 and select the finite final checkpoint with the highest validation accuracy. Validation uses 2,048 sequences at each of $K = 16$ and $K = 64$, with equal weight for the two sizes. Ties are resolved by lower query negative log-likelihood (NLL), then by the smaller rate.

After investigating the Transformer's low Standard MQAR accuracy, we add 0.001 to its grid and apply the same validation rule (Appendix E.6). The recurrent models retain the original grid. At each selected rate, we report seeds 600–604: the tuning run and four fresh runs. Test data contain 4,096 shared sequences at each size. Test accuracy is not used for learning-rate selection.

For each query, the model predicts the vocabulary token with the highest score. Accuracy is the percentage of predictions equal to the target value. Each matched task has one condition per K, without averaging over update fractions. The main table averages $K = 16$ and $K = 64$ equally within each run before summarizing across runs; Appendix E.4 reports each size separately. All MQAR summaries use mean $\pm$ standard error of the mean (SEM), computed as $s/\sqrt{n}$ from the sample standard deviation s across n training runs. We include all five evaluated runs, including the selected tuning seed. SEM describes uncertainty in the mean; individual-seed scores are shown in Appendix E.5.

To test generalization to longer sequences containing more associations, we evaluate $(T, K) = (1024, 128)$ and $(2048, 256)$. We evaluate the same five checkpoints at every test size. We also record query NLL, the fraction of sequences with every query correct, and accuracy grouped by whether the last observation was noisy or the hidden value changed. Hidden indicators are used only for this analysis after prediction.

Twelve xLSTM search candidates stop with nonfinite gradients. Each reported model–task pair has five selected and fresh runs with finite evaluations. Numerical failures are recorded separately and are never assigned zero test accuracy.

E.4 Results at every memory size

Table 7 reports the four tasks at each memory size, using mean $\pm$ SEM over all five runs. Bold and underline mark the highest and second-highest means at the displayed precision. Appendix C gives the separate covariance-block comparison.

Table 7: MQAR accuracy (%) at each memory size: mean ± SEM over five runs. $K = 128/256$ are extrapolation settings. Bold/underline mark the top two displayed means within each task and size, including ties.

Model	Standard	Volatile	Stochastic	Vol & Sto
(a) $K = 16$, $T = 128$				
Voltic-D	**100.00** ± 0.00	**99.86** ± 0.04	**88.92** ± 0.05	72.69 ± 0.40
Voltic-Q	**100.00** ± 0.00	<u>99.82</u> ± 0.06	88.66 ± 0.13	**73.52** ± 0.23
KLA	99.69 ± 0.16	95.64 ± 2.86	84.86 ± 0.54	60.55 ± 7.97
KDA	**100.00** ± 0.00	59.03 ± 22.66	86.95 ± 2.00	<u>73.09</u> ± 0.10
GDN-2	99.98 ± 0.02	80.10 ± 17.71	73.04 ± 15.88	69.41 ± 3.69
Mamba-3 SISO	21.81 ± 3.92	7.62 ± 2.96	8.79 ± 1.74	2.71 ± 1.11
Mamba-3 MIMO	24.11 ± 2.81	4.72 ± 1.15	36.62 ± 14.00	0.47 ± 0.14
xLSTM (mLSTM)	**100.00** ± 0.00	89.43 ± 4.22	88.64 ± 0.15	70.04 ± 0.91
xLSTM (hybrid)	**100.00** ± 0.00	85.63 ± 0.55	<u>88.70</u> ± 0.03	67.86 ± 1.04
Transformer	<u>99.99</u> ± 0.00	5.19 ± 0.02	5.93 ± 0.03	5.51 ± 0.00
(b) $K = 64$, $T = 512$				
Voltic-D	99.96 ± 0.04	<u>98.72</u> ± 0.60	**87.75** ± 0.36	67.03 ± 2.87
Voltic-Q	**100.00** ± 0.00	**99.13** ± 0.42	<u>86.56</u> ± 0.53	**72.25** ± 0.34
KLA	70.19 ± 4.14	54.89 ± 4.47	54.03 ± 2.56	34.70 ± 8.57
KDA	<u>99.99</u> ± 0.00	38.50 ± 16.63	81.21 ± 5.12	<u>68.67</u> ± 0.63
GDN-2	99.73 ± 0.25	60.40 ± 15.35	69.75 ± 17.17	60.84 ± 9.97
Mamba-3 SISO	1.52 ± 0.26	2.66 ± 1.05	0.89 ± 0.15	0.39 ± 0.14
Mamba-3 MIMO	1.82 ± 0.32	0.59 ± 0.12	16.82 ± 6.62	0.10 ± 0.01
xLSTM (mLSTM)	99.89 ± 0.05	83.35 ± 4.65	80.40 ± 2.90	63.52 ± 3.40
xLSTM (hybrid)	99.87 ± 0.06	88.58 ± 1.54	82.57 ± 0.14	62.08 ± 4.24
Transformer	99.97 ± 0.00	0.73 ± 0.00	0.85 ± 0.01	0.81 ± 0.01
(c) $K = 128$, $T = 1,024$				
Voltic-D	94.62 ± 3.92	<u>67.04</u> ± 6.54	**72.02** ± 2.91	45.72 ± 8.18
Voltic-Q	<u>97.65</u> ± 0.62	**69.36** ± 5.21	<u>68.96</u> ± 1.44	**61.23** ± 1.16
KLA	24.42 ± 2.94	14.78 ± 1.80	23.60 ± 3.05	10.98 ± 2.91
KDA	96.03 ± 0.37	16.25 ± 8.76	65.55 ± 5.70	<u>49.63</u> ± 1.85
GDN-2	91.34 ± 5.15	17.03 ± 6.44	59.98 ± 14.90	47.17 ± 10.07
Mamba-3 SISO	0.44 ± 0.03	0.72 ± 0.28	0.31 ± 0.04	0.17 ± 0.05
Mamba-3 MIMO	0.50 ± 0.06	0.25 ± 0.03	5.60 ± 2.21	0.07 ± 0.00
xLSTM (mLSTM)	92.55 ± 1.93	42.61 ± 4.66	61.39 ± 5.72	46.83 ± 5.65
xLSTM (hybrid)	92.36 ± 1.98	45.47 ± 1.91	64.91 ± 0.34	43.37 ± 6.12
Transformer	**99.08** ± 0.27	0.12 ± 0.02	0.32 ± 0.00	0.27 ± 0.00
(d) $K = 256$, $T = 2,048$				
Voltic-D	44.97 ± 6.77	<u>14.47</u> ± 2.16	27.83 ± 4.73	16.65 ± 4.90
Voltic-Q	43.51 ± 6.97	**15.33** ± 3.98	27.74 ± 5.57	**23.70** ± 2.81
KLA	1.37 ± 0.57	1.15 ± 0.35	3.36 ± 1.08	0.79 ± 0.33
KDA	46.10 ± 0.94	3.18 ± 1.97	30.06 ± 3.35	16.15 ± 1.14
GDN-2	40.57 ± 7.62	2.62 ± 1.32	31.83 ± 7.98	15.43 ± 4.55
Mamba-3 SISO	0.16 ± 0.01	0.19 ± 0.06	0.13 ± 0.01	0.08 ± 0.02
Mamba-3 MIMO	0.17 ± 0.00	0.10 ± 0.01	0.69 ± 0.27	0.04 ± 0.00
xLSTM (mLSTM)	50.81 ± 3.73	13.35 ± 1.93	<u>33.11</u> ± 5.53	<u>18.31</u> ± 3.25
xLSTM (hybrid)	<u>50.89</u> ± 3.70	14.08 ± 0.97	**35.54** ± 0.43	15.12 ± 2.73
Transformer	**61.74** ± 2.85	0.02 ± 0.00	0.11 ± 0.01	0.09 ± 0.01

Extrapolation. Figure 3 shows five-run results beyond the training sizes. Voltic-Q retains a higher Vol & Sto mean than the other main models at both sizes. Voltic-D and Voltic-Q both remain strong on Volatile MQAR at $K = 128$, while accuracy drops substantially at $K = 256$. The Transformer has the highest Standard MQAR mean at both extrapolation sizes. The xLSTM hybrid leads Stochastic MQAR at $K = 256$.

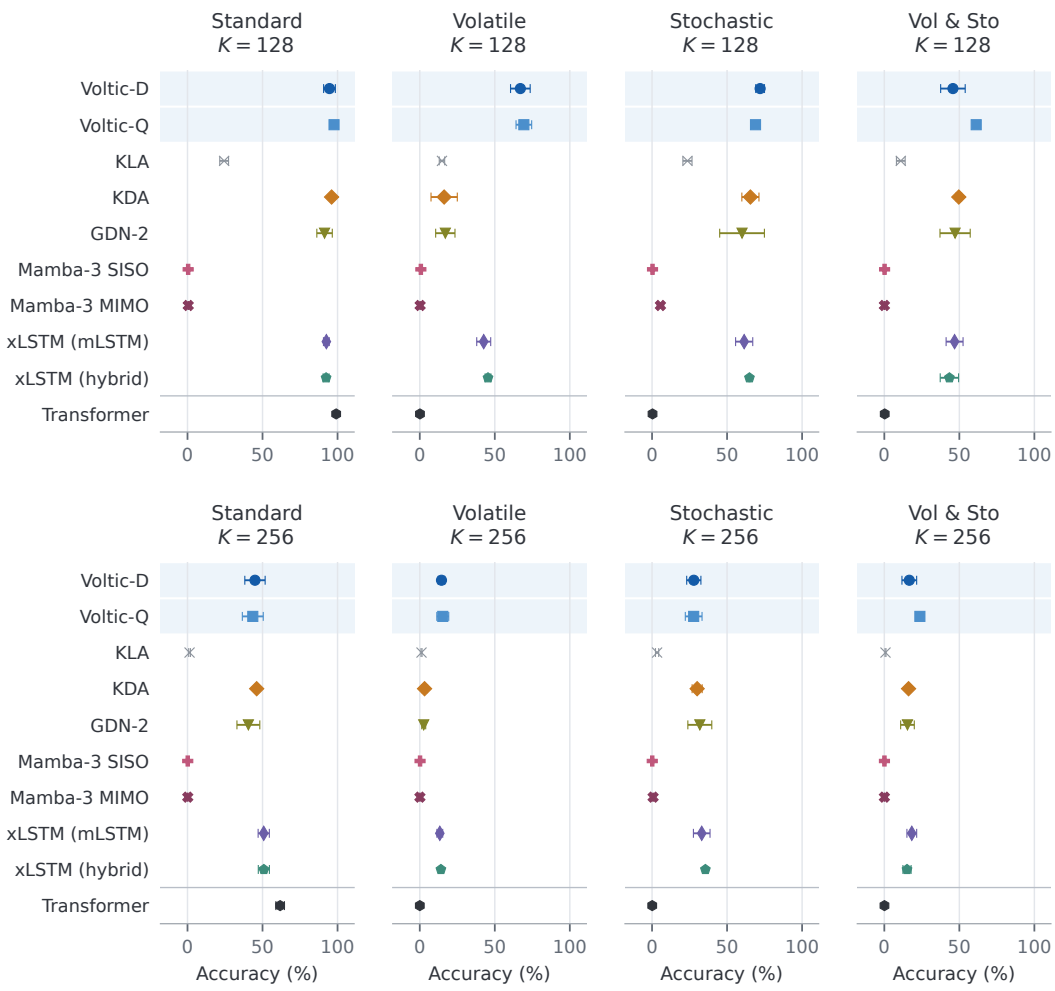


Figure 3: MQAR extrapolation at $K = 128$ ($T = 1,024$) and $K = 256$ ($T = 2,048$). Points and horizontal bars show mean accuracy and SEM over five runs.

E.5 Variation across seeds

Figure 4 shows every run at each training size. On Stochastic MQAR, Mamba-3 MIMO shows substantial variation across initializations at both training sizes. These evaluations remain numerically finite. The individual points show the variation across runs that SEM alone does not describe.

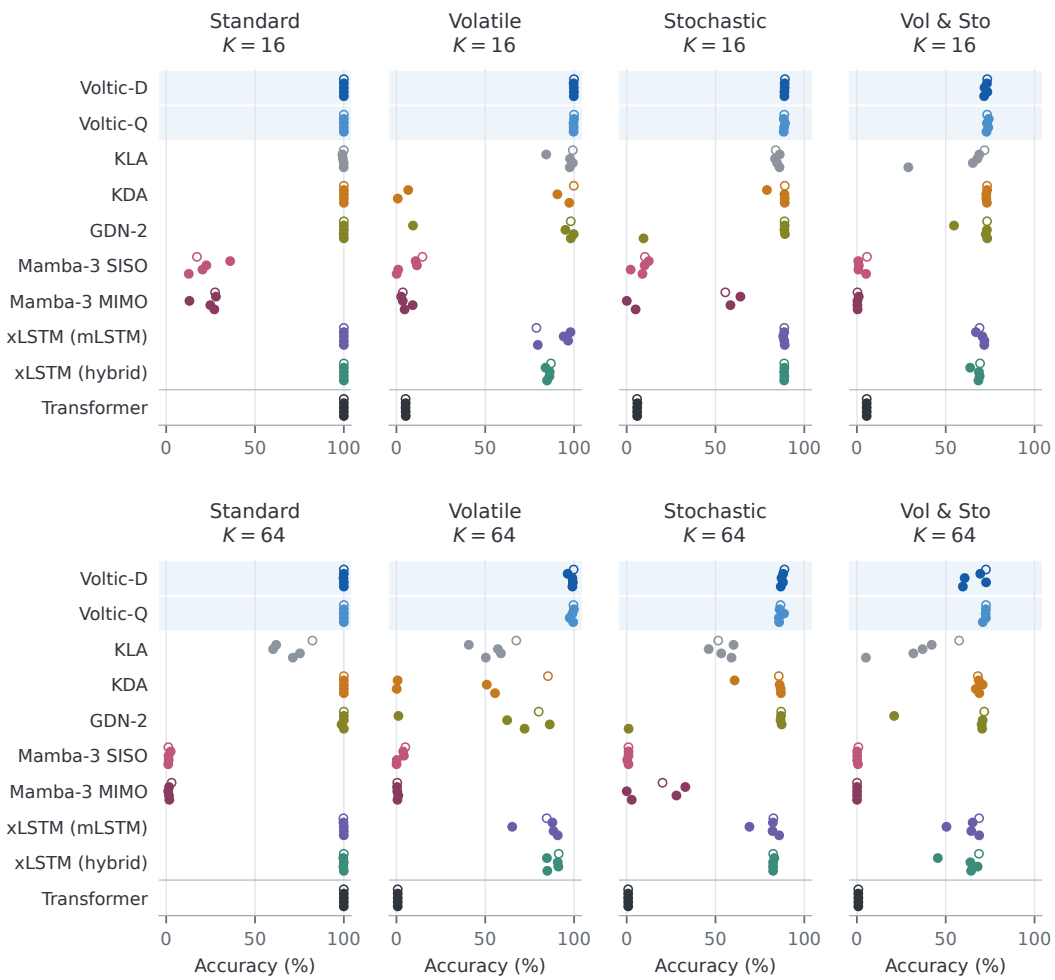


Figure 4: MQAR test accuracy for individual runs at $K = 16$ and $K = 64$. Open points mark the tuning seed; filled points mark the four fresh seeds.

E.6 Transformer learning rate and curriculum controls

Learning-rate follow-up. The Transformer's low Standard MQAR accuracy in the initial search was unexpected, since this task provides three faithful observations of each fixed association. This prompted a check of the optimization settings. On all four tasks, the initial search selected 0.003, its lowest candidate. We therefore evaluate an additional peak rate of 0.001 on all four tasks and seeds 600–604, keeping the architecture, initial weights, ordered training batches, and 10,000-update budget unchanged. The checkpoints use the same validation and test sequences as the main comparison.

The smaller rate restores nearly perfect Standard MQAR accuracy, while accuracy remains low on Volatile MQAR, Stochastic MQAR, and Vol & Sto. Applying the validation rule in Appendix E.3 to the expanded grid selects 0.001 for Standard and Volatile MQAR and retains 0.003 for Stochastic MQAR and Vol & Sto. These are the Transformer rates used in the reported results.

One-observation warm-up task. We use Zoology's `multiquery_ar` generator with one query per key, random non-query tokens, and `power_a=0.01`. The first $2K$ tokens alternate between keys and their values. The remaining region has candidate query positions at every other token. We sample K of these positions without replacement, using weights proportional to $j^{-0.99}$, where j numbers the candidate positions starting from one. We assign one key to each sampled position, and its target is that key's initial value. This task is used only for the curriculum diagnostic below.

Matched training-budget comparison. To test whether learning the one-observation task first helps with Volatile MQAR, Stochastic MQAR, and Vol & Sto, we compare two training paths for each target task and seed 600–604. One path trains on the one-observation task for 10,000 updates and then on the target task for 2,000 updates. The other trains on the target task for all 12,000 updates. Both paths use the same original initialization and unchanged Transformer architecture. Each receives 61.44 million input tokens and 7.68 million supervised query positions.

Both paths use peak learning rate 0.003 in the first phase. At update 10,000, both reset AdamW and the scheduler, then use peak rate 0.001 for the first 2,000 steps of the same 10,000-step warmup/cosine schedule. The second phase uses identical ordered target batches that do not repeat sequences from either first phase or the test set. All training uses RTX A6000 GPUs; the warm-up runs are retrained on this device to match the target runs. Other optimizer and numerical settings follow Appendix E.2.

We evaluate the final checkpoints on the same 4,096 held-out sequences per target task and memory size. Table 8 averages the two sizes within each run, then reports mean ± SEM over all five seeds. This is a separate diagnostic with a larger training budget than the main benchmark. It uses no further learning-rate search or checkpoint selection.

Table 8: Transformer curriculum control after 12,000 updates. Warm-up uses the one-observation task described in Appendix E.6. Accuracy (%) averages $K = 16/64$: mean ± SEM over five paired seeds.

Target task	Target → target	Warm-up → target
Volatile	21.05 ± 18.22	99.43 ± 0.12
Stochastic	3.36 ± 0.01	88.49 ± 0.11
Vol & Sto	3.11 ± 0.01	72.29 ± 0.13

Interpretation. Training first on the one-observation task improves accuracy in every paired seed/task comparison. Extra target-only training produces uneven outcomes on Volatile MQAR: one seed reaches high accuracy, while the others remain low. On Stochastic MQAR and Vol & Sto, all target-only runs remain low. Thus, the unchanged architecture can achieve much higher accuracy, and extra updates or the optimizer restart alone do not explain the curriculum's advantage under this recipe.

These results support difficulty learning a useful retrieval computation from scratch under the tested training settings. They do not establish a representational limitation of full attention. The warm-up changes both the first-phase task and the placement of observations and queries, so the experiment does not identify which learned computation produces the benefit. Different optimization settings or longer target-only training may also improve performance.

E.7 Volatility and stochasticity ablations

Evaluation-only interventions. We fix the effective observation variance $s_t = 0.5$, the volatility vector $v_t = 0.5\mathbf{1}$, or both, at every token and layer during evaluation. The fixed value lies within the implemented variance range (Appendix B.2). Each branch is fixed by zeroing its projection, bias, and log scale; under the parameterization in Equation 20, this gives $\exp[-\operatorname{softplus}(0)] = 0.5$. The memory update, keys, values, initial uncertainty, and decay configuration are retained.

We evaluate each full-model checkpoint without changing its remaining weights and compare each intervention with an unmodified replay on the same device and examples. All evaluated runs contribute to the reported mean and SEM.

The reported interventions use 40 full-model checkpoints, four variance settings (including the unmodified control), and four memory sizes. Table 9 summarizes accuracy averaged equally over $K = 16/64$.

Reliance in trained models. Replacing learned volatility at evaluation sharply reduces accuracy on every task, including Standard MQAR. Replacing stochasticity has a more variable effect: it strongly affects Voltic-D on Stochastic MQAR, while Voltic-Q largely retains Vol & Sto accuracy. These interventions measure sensitivity to changing trained internal values. The strong effect on

Standard MQAR also shows why this sensitivity alone is not evidence that a branch specifically detects changes or observation noise.

These evaluation-only interventions do not establish that input-dependent variance branches are necessary, or that their outputs are calibrated estimates of the generator's probabilities.

Table 9: Evaluation-only variance interventions: accuracy (%) averaged equally over $K = 16/64$, mean ± SEM over five runs. Full controls are unmodified matched replays.

Model	Full	$s_t = 0.5$	$v_t = 0.5$	$s_t = v_t = 0.5$
Standard				
Voltic-D	99.98 ± 0.02	93.49 ± 6.27	22.33 ± 12.10	21.10 ± 11.51
Voltic-Q	100.00 ± 0.00	80.08 ± 19.91	13.72 ± 6.09	13.27 ± 5.38
Volatile				
Voltic-D	99.29 ± 0.31	89.09 ± 7.23	23.45 ± 15.86	22.99 ± 15.88
Voltic-Q	99.48 ± 0.23	96.60 ± 1.78	1.02 ± 0.38	1.19 ± 0.52
Stochastic				
Voltic-D	88.33 ± 0.20	53.88 ± 13.26	2.31 ± 1.71	2.17 ± 1.40
Voltic-Q	87.61 ± 0.33	86.42 ± 0.68	12.47 ± 5.63	12.66 ± 5.76
Vol & Sto				
Voltic-D	69.86 ± 1.63	66.88 ± 2.94	7.90 ± 3.35	7.07 ± 3.17
Voltic-Q	72.89 ± 0.26	72.98 ± 0.24	24.26 ± 8.06	22.40 ± 7.96

F Language-Modeling Experimental Details

F.1 Data preparation and document boundaries

The 45M data use FineWeb-Edu's `sample-10BT` release at revision `87f09149ef47`. Source documents are processed in shard and row order. A fixed hash of the document ID assigns each document to one of four disjoint partitions: 1% for tokenizer training, 0.25% for hyperparameter selection, 0.75% for final reporting, and 98% for training. The tokenizer uses byte-level byte-pair encoding with 32,000 entries, including padding, beginning-of-sequence, end-of-sequence, and unknown-token symbols. It is trained on a deterministic sample of at least 400 MiB from its own partition and then shared by all models.

To reduce overlap with evaluation data, we normalize source and evaluation text with Unicode NFKC normalization, lowercasing, ASCII-punctuation removal, and whitespace splitting. We reject a source document if it contains any contiguous 13-word sequence from an evaluation prompt. For prompts of 4–12 words, we instead match the complete normalized prompt. This filter applies to all four partitions and includes the language-task prompts and the six BASED retrieval datasets. LongBench and BABILong were added after the training corpus was fixed and are not included in this overlap filter.

The selection set has 8,388,608 tokens in 4,096 sequences, and the reporting set has 25,165,824 tokens in 12,288 sequences.

Each accepted document ends with an end-of-sequence token. The resulting stream is split into sequences of 2,048 tokens, with document boundaries retained. A recurrent model resets its memory at every sequence and document start, and a Transformer uses causal attention separately within each document segment. Targets at these starts are excluded from the loss, preventing prediction from the preceding document or sequence. Training reads the same ordered sequences for all models and seeds.

F.2 Architecture and parameter matching

We fix the token-representation dimension d_{model}, layer count L, vocabulary, and input/output embedding sharing. We then adjust the intermediate dimension d_{ff} of each block's gated feed-forward branch to match the parameter target. Table 10 gives the resulting sizes and the rates selected by

the procedure in Appendix F.3. The target is 45,508,352 parameters, corresponding to the chosen Mamba-3 SISO configuration; every model is within 0.5% of this target.

Table 10: 45M model dimensions and selected learning rates, with $L = 8$ and $d_{\text{model}} = 512$. Projection and dynamics columns give learning-rate multipliers.

Model	Parameters	d_{ff}	Peak rate	Projection	Dynamics
Voltic-D	45,322,880	1,152	0.0032	1.0	0.3
Voltic-Q	45,580,896	1,344	0.0032	1.0	0.1
KLA	45,576,712	1,712	0.0032	—	0.1
KDA	45,434,400	1,504	0.0032	—	—
GDN-2	45,483,552	1,168	0.0032	—	—
Mamba-3 SISO	45,508,352	1,280	0.0032	—	—
Mamba-3 MIMO	45,655,808	1,152	0.0032	—	—
Transformer	45,425,152	1,680	0.0016	—	—

Voltic, KDA, and GDN-2. Each head uses a 128-dimensional key and a 128-dimensional value, with four heads. This gives the same main memory size as the Mamba-3 configurations below. All use causal short convolutions of width four. Voltic uses the block and noise parameterization in Appendix B.1 and Appendix B.2. Voltic-D's learned decay is initialized near one and bounded below by 0.2 (Appendix B.2.1). GDN-2 uses value expansion one. The internal memory chunk lengths are 16 for Voltic and 64 for KDA and GDN-2.

KLA. KLA uses state dimension 64 and expansion one, giving 512 channels. Its observation-noise and time-step projections use rank 64, while values come from the convolution output. The model uses a causal convolution of width four with SiLU, normalized queries and keys, learned process noise, Ornstein–Uhlenbeck discretization, output gating, and a learned scalar skip connection. Readout uses the updated state. We use the optimized CUDA implementation specified in Appendix D.1, with checkpointing inside the state-space scan.

Mamba-3. Both variants use state dimension 64, head dimension 64, expansion two, one group, and rotary fraction 0.5. The expanded dimensions give 16 heads. SISO uses chunk length 64; MIMO uses chunk length 16 and rank four. These configurations include the separate gated feed-forward branch in every block.

Transformer. The full-attention baseline uses four heads, with head dimension 128. Rotary position embeddings cover the full head dimension, with base 10,000 and maximum position 2,048. Attention dropout is zero, and attention respects the document boundaries used by the recurrent models.

State budget. Voltic, KDA, GDN-2, and Mamba-3 each store 65,536 main-memory scalars per layer. KLA stores two 512×64 arrays, its information vector and diagonal precision, giving 65,536 recurrent-state scalars in total. Convolution histories and Voltic's uncertainty state are additional storage. Transformer attention retains the keys and values of previous tokens, so its stored history grows with context length.

F.3 Optimization and hyperparameter selection

Every update processes 256 sequences of 2,048 tokens, or 524,288 input tokens. The 4,800-update budget is 2,516,582,400 input tokens. All models accumulate gradients over 32 batches of eight sequences. Hardware and software are specified in Appendix D. Parameters and optimizer states use FP32; the forward pass uses BF16 autocasting. Dropout and general gradient checkpointing are disabled.

All models use AdamW with moment coefficients $(0.9, 0.95)$ and global gradient-norm clipping at one. The learning rate warms up over 48 updates (1% of training), then follows cosine decay to 10% of its peak at update 4,800. Weight decay 0.1 applies to matrix, convolution, and embedding weights, including Voltic's rate-projection weights. Biases, normalization parameters, and the architectures' designated static dynamics parameters use zero decay.

Learning-rate selection. All eight reported configurations, including the Transformer, begin with the same peak-rate candidates: 10^{-4}, 2×10^{-4}, 4×10^{-4}, 8×10^{-4}, 1.6×10^{-3}, and 3.2×10^{-3}. We score candidates by mean selection-set NLL over seeds 100 and 101. An initial screen uses 477 updates; the two best candidates are evaluated after 954 updates. If their mean NLLs differ by at most 0.01, both are evaluated after 1,717 updates. Each stage restarts with the same seed and uses the corresponding prefix of the 4,800-update learning-rate schedule. When the best screened rate is a grid endpoint, one additional rate is tested outward from that endpoint; this adds 6.4×10^{-3} for seven models, while the Transformer requires no extension.

For Voltic, multiplier candidates are $\{0.3, 1\}$ for noise/decay projection weights and $\{0.1, 0.3, 1\}$ for learned log-rate scales and biases. We select these settings after the base rate, using the same screening and confirmation procedure. Exact ties prefer the lower base rate, then the lower projection multiplier, then the lower dynamics multiplier. The selected settings are fixed before training three fresh final seeds, 0–2; reported scores use their final checkpoints.

KLA retains a fixed learning-rate multiplier of 0.1 and zero weight decay for its static dynamics parameters. Its learned projection matrices use the base learning rate and weight decay 0.1. KLA's base rate is selected by the same search above; its confirmation stage resolves the two candidates without a near-tie extension.

Voltic uses fused projections, precomputed noise rates, and saved memory-kernel intermediates. Panel length is selected for runtime from $\{256, 512, 1024, 2048\}$ before training, with model parameters and random-number state restored afterward.

F.4 Evaluation tasks and scoring

The downstream tasks cover final-word prediction with LAMBADA; continuation and reference resolution with HellaSwag (Zellers et al., 2019) and WinoGrande (Sakaguchi et al., 2020); passage comprehension with BoolQ; physical and social commonsense with PIQA (Bisk et al., 2020) and Social IQa (Sap et al., 2019); and science questions with ARC-Easy, ARC-Challenge, and OpenBookQA. Table 11 lists the splits and their full sizes.

We use the frozen task definitions from Language Model Evaluation Harness, with document-level WikiText-103 and the OpenAI-preprocessed LAMBADA task. Evaluation uses batches of 64, no beginning-of-sequence token, and a maximum model input of 2,048 tokens. Candidate log probabilities are computed with FP32 log-softmax. Inputs are right-padded, and padding positions are excluded from scores. For a candidate that exceeds the context limit, the evaluator retains the rightmost 2,049 tokens before forming the shifted input and targets. WikiText uses rolling windows to score the full documents.

Table 11: Language-model evaluation splits and scoring metrics.

Task	Split	Examples	Metric
WikiText-103	Test	62 documents	Word perplexity
LAMBADA	Test	5,153	Accuracy; perplexity
HellaSwag	Validation	10,042	Normalized accuracy
PIQA	Validation	1,838	Accuracy
ARC-Easy	Test	2,376	Accuracy
ARC-Challenge	Test	1,172	Normalized accuracy
OpenBookQA	Test	500	Normalized accuracy
WinoGrande	Validation	1,267	Accuracy
Social IQa	Validation	1,954	Accuracy
BoolQ	Validation	3,270	Accuracy

For each accuracy task, the harness applies its fixed prompt and answer-scoring rule. In HellaSwag, ARC-Challenge, and OpenBookQA, it divides each candidate's summed log probability by its character count before choosing the highest score. The remaining multiple-choice tasks use unnormalized conditional log probabilities. LAMBADA accuracy requires the final word's tokens to be predicted correctly. Its perplexity uses the log probability of that target word. WikiText word per-

plexity divides the summed negative log probability by the number of words before exponentiation; byte perplexity and bits per byte are also recorded.

The eight-task reasoning and comprehension average includes HellaSwag, PIQA, ARC-Easy, ARC-Challenge, OpenBookQA, WinoGrande, Social IQa, and BoolQ. The nine-task average adds LAMBADA. Both give equal weight to their tasks and were recorded by the frozen evaluator for every final checkpoint. We compute each average within each run, then summarize the run scores with their mean and sample standard deviation. Table 12 reports WikiText perplexity and FineWeb-Edu NLL alongside the downstream averages.

F.5 Complete 45M results

Table 12 and Table 13 report all eight reported 45M configurations, including the Transformer, using their final checkpoints.

Table 12: 45M results after 4,800 updates: mean ± sample SD over three seeds. NLL uses FineWeb-Edu; WikiText-103 reports word perplexity. The nine-task average adds LAMBADA. Bold/underline mark the top two means.

Model	NLL ↓	WikiText PPL ↓	8-task (%) ↑	9-task (%) ↑
Voltic-D	3.284 ± 0.004	59.09 ± 0.40	<u>42.26</u> ± 0.37	39.73 ± 0.34
Voltic-Q	3.306 ± 0.002	60.80 ± 0.21	**42.79** ± 0.14	<u>39.98</u> ± 0.20
KLA	3.329 ± 0.003	66.52 ± 0.13	41.95 ± 0.72	39.14 ± 0.57
KDA	**3.229** ± 0.002	**54.72** ± 0.07	42.08 ± 0.46	**39.99** ± 0.33
GDN-2	<u>3.262</u> ± 0.003	<u>56.79</u> ± 0.43	41.80 ± 0.58	39.46 ± 0.45
Mamba-3 SISO	3.284 ± 0.008	60.14 ± 0.66	41.95 ± 0.91	39.47 ± 0.82
Mamba-3 MIMO	3.269 ± 0.001	58.58 ± 0.52	41.84 ± 0.48	39.48 ± 0.51
Transformer	3.347 ± 0.003	62.08 ± 0.68	42.13 ± 0.42	39.59 ± 0.39

Table 13: 45M zero-shot accuracy (%) after 4,800 updates: mean ± sample SD over three training seeds. Bold/underline mark the top two means.

(a) LAMBADA, HellaSwag, PIQA, and ARC

Model	LAMBADA	HellaSwag	PIQA	ARC-E	ARC-C
Voltic-D	19.52 ± 0.46	28.98 ± 0.22	60.97 ± 0.60	48.64 ± 0.23	23.75 ± 0.88
Voltic-Q	17.52 ± 0.76	28.84 ± 0.35	60.61 ± 0.25	47.91 ± 1.42	**24.37** ± 0.44
KLA	16.61 ± 0.69	29.12 ± 0.17	**62.15** ± 0.32	48.25 ± 0.60	23.52 ± 0.36
KDA	**23.27** ± 0.90	**29.80** ± 0.27	60.66 ± 0.38	**49.21** ± 0.02	23.81 ± 0.74
GDN-2	<u>20.70</u> ± 0.57	<u>29.52</u> ± 0.05	60.74 ± 0.35	48.20 ± 0.95	23.24 ± 0.34
Mamba-3 SISO	19.65 ± 0.36	28.99 ± 0.53	60.70 ± 0.72	48.02 ± 0.78	23.35 ± 0.73
Mamba-3 MIMO	20.61 ± 1.01	29.40 ± 0.24	<u>61.30</u> ± 0.23	<u>48.71</u> ± 0.50	23.75 ± 0.57
Transformer	19.30 ± 0.73	28.07 ± 0.06	59.88 ± 0.44	47.42 ± 0.72	<u>23.86</u> ± 0.58

(b) OpenBookQA, WinoGrande, Social IQa, and BoolQ

Model	OpenBookQA	WinoGrande	Social IQa	BoolQ
Voltic-D	29.20 ± 1.74	51.09 ± 0.59	36.63 ± 0.75	58.83 ± 2.91
Voltic-Q	**31.07** ± 1.10	50.99 ± 0.48	36.78 ± 0.44	**61.74** ± 0.43
KLA	28.47 ± 0.42	<u>51.93</u> ± 1.11	36.22 ± 0.18	55.98 ± 6.25
KDA	29.67 ± 1.10	51.85 ± 0.44	36.52 ± 0.78	55.11 ± 2.55
GDN-2	28.87 ± 1.22	50.57 ± 1.28	<u>36.90</u> ± 0.09	56.41 ± 5.04
Mamba-3 SISO	29.67 ± 1.70	51.25 ± 0.43	36.44 ± 0.37	57.18 ± 4.63
Mamba-3 MIMO	<u>29.93</u> ± 0.46	51.25 ± 1.03	**37.34** ± 0.34	53.05 ± 1.47
Transformer	29.40 ± 0.40	**52.22** ± 0.56	36.63 ± 0.34	<u>59.52</u> ± 3.88

G In-Context Retrieval Experimental Details

G.1 Checkpoints and generation

We evaluate eight 45M configurations, including the Transformer, using the final checkpoints reported in Section 4.3, with the same model dimensions, tokenizer, and weights. All models and seeds use identical evaluation examples. There is no additional training, instruction tuning, or model-specific prompt selection.

Generation uses BF16 autocasting and selects the highest-scoring token from FP32 logits. Each example starts with a fresh state. The evaluator scores tokens using causal full-prefix computation; right padding used for kernel alignment cannot provide information to earlier positions. Where a recurrent decoder proposes several tokens together, the full-prefix model verifies them before acceptance, preserving greedy output. These are quality evaluations; their execution times are not used in the throughput comparisons.

Appendix D lists the retrieval software and hardware.

For the Transformer, we extend the allowed position range to the evaluation budget. Rotary position embeddings retain their training frequency base of 10,000, without rescaling or long-context adaptation. The recurrent models likewise receive no adaptation for the longer inputs.

KLA uses the same CUDA implementation as its language-model evaluation, with full-prefix greedy decoding. Its input-length limit is raised to 8,192 for RULER without changing model weights; this leaves its 2K logits unchanged. The benchmark scores do not depend on a cached-state decoder.

G.2 RULER tasks and context budgets

We use revision `c3f5e3b4f87f` of the official RULER generator. S1, S2, and S3 correspond to `niah_single_1`, `niah_single_2`, and `niah_single_3`; MK1 is `niah_multikey_1`. S1 uses repeated filler text; the other three use the generator's essay background. Keys are word strings. S1, S2, and MK1 use numeric values, while S3 uses 36-character random identifier strings. MK1 contains four key–value facts with distinct keys and one query; the other tasks contain one fact and one query.

Each task/context combination contains 500 frozen prompts, shared by every model and training seed. Context budgets are 1,024, 2,048, 4,096, and 8,192 tokens, including a reserved generation allowance of 128 tokens. These are upper bounds: generated prompts need not fill the entire budget. Table 14 gives the observed mean prompt lengths. Prompts retain the generator's question and answer prefix. Generation stops at the end-of-sequence token or the 128-token limit, without newline stopping.

The RULER scorer strips surrounding whitespace, replaces control characters with newlines, and checks whether the reference string occurs in the answer, ignoring case. Each selected task has one reference answer, so the score is the percentage of examples recovered correctly. We evaluate $4 \times 4 \times 500 = 8,000$ examples per checkpoint. Our overall RULER score averages all sixteen task/context combinations equally, including the S3 and MK1 8K conditions. It is a score for this retrieval subset, not the full thirteen-task RULER benchmark.

Table 14: Mean RULER prompt length at each total context budget, reserving 128 tokens for generation.

Task	1K	2K	4K	8K
S1	880.9	1,905.8	3,955.8	8,039.4
S2	703.7	1,535.7	3,750.1	7,699.2
S3	519.5	1,767.9	3,639.1	7,693.8
MK1	656.0	1,817.9	3,766.9	7,967.1

G.3 BASED DATA AND SCORING

We use the six BASED-format validation splits in Table 15, with the shared language-model tokenizer. Prompt formatting follows the released `prefix-linear-attention` evaluation code at revision `7490f22bda3e`. SWDE (Lockard et al., 2019) and FDA (Arora et al., 2023) append a requested field to the source text. SQuAD (Rajpurkar et al., 2016) appends an unfinished statement whose missing ending is the answer, following the released prompt format. TriviaQA (Joshi et al., 2017), Natural Questions (NQ) (Kwiatkowski et al., 2019), and DROP (Dua et al., 2019) retain the released context and question formatting.

We tokenize each formatted prompt and retain its last 2,000 tokens. This leaves 48 tokens for generation within the 2,048-token budget. Generation stops at the first newline, the end-of-sequence token, or the output limit. This rule shortens some FDA and NQ inputs; the other four datasets fit without truncation. All 12,129 examples are retained, including cases where truncation removes answer text. We use no answer-centered cropping or filtering based on answer location.

The score is the fraction of generated answers containing a reference string, using case-insensitive matching. It does not require equality with the entire generated string. For the question-answering datasets, this differs from their usual exact-match or token-F1 evaluation. The six-task mean averages task percentages equally within each training seed. Evaluation produces 12,129 predictions per checkpoint.

Table 15: BASED validation splits and frozen revisions under `hazyresearch/`.

Task	Dataset	Revision	Examples
SWDE	`based-swde-v2`	`27f0223028a5`	1,111
SQuAD	`based-squad`	`d152c14c2d7d`	2,984
FDA	`based-fda`	`42569d301e12`	1,102
TriviaQA	`based_triviaqa`	`5c39988c6a2b`	1,688
NQ	`based_nq_2048`	`d3338c4e6d8e`	3,157
DROP	`based_drop`	`ad796806f1b9`	2,087

G.4 SELECTED LONGBENCH TASKS

We use the official LongBench prompts and scorers at code revision `2e00731f8d0b` and dataset revision `5e628be450b7`. Qasper (Dasigi et al., 2021) asks questions about research papers, while MultiFieldQA-en covers passages from several domains. SAMSum (Gliwa et al., 2019) summarizes a dialogue, and TREC (Li & Roth, 2002) predicts a question category from demonstrations in the prompt. We retain the released prompt examples and instructions.

For each task, we rank example IDs by the SHA256 hash of the selection seed 1729, task name, and original ID, then select the first 100. The same examples are used at both 1,024- and 2,048-token input budgets. When a prompt is too long, we remove tokens from its middle, keeping the beginning and a suffix long enough to contain the complete final question or dialogue. The suffix contains at least half the input budget and at least the tokenized final input plus 32 tokens. This retains the query but can remove supporting text; examples are not filtered by answer visibility. The budget limits the input alone. Generation allows up to 128 tokens for Qasper and SAMSum and 64 for MultiFieldQA-en and TREC.

Generation stops at the end-of-sequence token or output limit. SAMSum also stops at a newline after generating at least one token. Following the official scorer, SAMSum and TREC discard leading newlines and score the first remaining line. Question answering uses token F1 and summarization uses ROUGE-L, taking the best score over the supplied references. TREC gives full credit for the correct category alone; if several category names occur, it divides credit among them. We scale scores to 0–100 and average the eight task/budget scores within each seed. This gives 800 predictions per checkpoint.

G.5 Selected BABILong tasks

We use the official BABILong prompts and answer scorer at code revision `7a6efee29f5c` and dataset revision `ee0d588794c7`. QA1 asks for a fact stated in the context; QA2 and QA3 require combining two and three supporting facts, respectively. We use all 100 examples in each released short (`0k`), `1k`, and `2k` setting for these tasks. The released instructions and demonstrations remain in every prompt.

Prompts are not truncated. The release names describe nominal lengths: with our tokenizer and the full prompt, the 1K examples contain 896–1,057 tokens and the 2K examples contain 1,775–2,139 tokens. Short controls contain the task statements without the added long distractor text, but their lengths also vary; QA3 controls can reach 1,252 tokens. The releases are not fully paired across lengths, so their differences can also reflect which questions are included. We report these conditions as a short-context reasoning evaluation; RULER supplies the controlled 4K/8K context extrapolation.

Generation allows 32 new tokens and stops at the end-of-sequence token or this limit. The official scorer lowercases the answer, retains text before the first period or a generated context/example delimiter, and checks whether the reference answer occurs in that text. We average the nine task/length accuracies within each seed. This gives 900 predictions per checkpoint.

G.6 RULER across context lengths

Figure 5 compares RULER accuracy at the four context budgets, including the Transformer as a full-attention reference. We average the four task accuracies within each training seed before computing the mean and sample standard deviation across seeds.

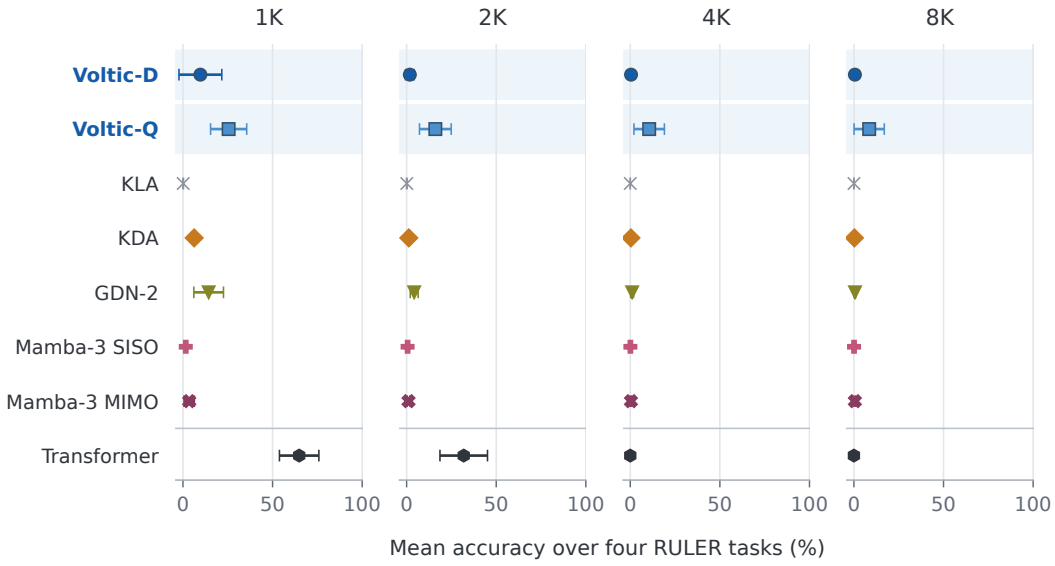


Figure 5: 45M RULER accuracy averaged over four tasks at each context budget: mean ± sample SD over three seeds. Training uses 2K sequences.

H Throughput Experimental Details

H.1 Model dimensions and timed operation

The throughput models use $L = 7$, $d_{\text{model}} = 512$ at 45M and $L = 9$, $d_{\text{model}} = 1024$ at 180M. Both use a vocabulary of 32,000 tokens, tied input and output embeddings, and a gated feed-forward branch. We adjust the feed-forward dimension to match parameter targets of 44,620,384 and 180,134,592. Every model is within 0.3% of its target. Table 16 gives the exact counts.

Voltic, KDA, and GDN-2 use key and value dimensions of 64, with 16 heads at 45M and 32 at 180M. Mamba-3 uses state and head dimensions of 64, expansion two, and rank four for MIMO. KLA uses state dimension 64 and expansion one. Transformer head dimension is 64. The main recurrent budget is 65,536 scalars per layer at 45M and 131,072 at 180M: this counts the mean memory for matrix recurrences and the two sufficient-statistic vectors for KLA. Auxiliary uncertainty, convolution state, and intermediate activations are additional storage.

The training and decoding measurements reuse these configurations. Models start from seed 1701; weights are fixed throughout timing. All architectures receive the same synthetic token IDs at a given input shape. In training measurements, there is no document packing or recurrent state carried between operations. Each operation runs `zero_grad(set_to_none=True)`, causal forward computation, FP32 full-vocabulary cross-entropy, and backward. Throughput counts BT input tokens per operation; next-token loss uses $B(T-1)$ targets. Cached decoding is measured separately below.

Table 16: Dedicated throughput configurations: seven layers at 45M and nine at 180M, with matched parameter budgets.

	45M		180M	
Model	Parameters	d_{ff}	Parameters	d_{ff}
Voltic-D	44,684,224	224	180,152,448	528
Voltic-Q	44,619,600	560	180,281,184	1,216
KLA	44,680,199	1,968	180,142,089	4,112
KDA	44,648,496	1,120	180,060,000	2,448
GDN-2	44,705,840	448	180,354,912	1,104
Mamba-3 SISO	44,620,384	1,536	180,134,592	3,200
Mamba-3 MIMO	44,749,408	1,408	180,466,368	3,072
Transformer	44,547,584	1,936	180,243,456	3,968

H.2 Execution settings

Appendix D gives the shared software versions and hardware. BF16 autocasting is enabled, TF32 is disabled, and FP32 matrix multiplication uses the highest precision setting. Native kernels retain their supported internal precision; KLA's recurrence is FP32.

Execution settings are selected before the final timing repetitions. Every baseline is offered eager execution and compilation of supported model components. Voltic also tests panel lengths, short-convolution backends, separate or combined query/key/value input preparation, and the available Voltic-Q launch settings. Selection uses synchronized trial times after warmup; its measurements are excluded from the reported results. The batch-size study retains the selected settings at each model size and sequence length, without retuning them for each batch.

Voltic uses fused noise rates and separate query, key, and value convolution calls in the selected settings. Voltic-D uses $S = 512$ throughout. Voltic-Q panel lengths and convolution choices are listed in Table 17. At state dimension 64, Voltic-Q holds a block's update vectors in GPU registers while computing its token-ordered forward updates.

KDA and Mamba-3 SISO use eager execution at both scales; GDN-2 uses eager execution at 45M and compiled model components at 180M. Eager here describes the outer model: KDA, GDN-2, and Mamba-3 still call their native accelerated recurrence kernels. Mamba-3 MIMO and the Transformer use compiled model components. Voltic compiles the projections and output operations around custom kernels, except for the 45M Voltic-D 1K setting, which compiles the full model.

KLA training uses the unmodified official `cuda_v3_fast` backend at revision `ad5e64de903e`, replacing the earlier `triton_composed` measurements. The original weights are loaded strictly into the updated model; dimensions, parameter counts, noise parameterization, and inputs are retained. Its execution settings are selected again over eager execution, compiled components, and compiled components with scan checkpointing, using five warmup and five pilot operations. Compiled components without additional scan checkpointing are selected at all sizes and lengths; the

CUDA kernel's internal recomputation remains enabled. Each selection is fixed across batches. Thus the comparison uses optimized baseline kernels, not Python reference recurrences, without claiming that every implementation is equally optimized.

Table 17: Selected Voltic-Q S/convolution backend: C = causal-conv1d, F = FLA Triton, P = PyTorch. Voltic-D uses 512/C throughout.

Scale	Variant	1K	2K	4K	8K
45M	Voltic-Q	1024/C	256/C	512/C	512/F
180M	Voltic-Q	1024/F	2048/F	256/C	512/C

H.3 Training timing and checks

The measured training grid covers 256 combinations: two scales, eight models, four sequence lengths, and batches in $\{1, 2, 4, 8\}$. The fixed-token view selects its 64 entries with $BT = 8,192$; these are not additional independent measurements. All reported training and decoding timings use the same physical H200 NVL. The updated KLA measurements replace its previous backend, and newly completed large-batch cases fill the previous omissions; other batch-grid measurements are retained.

Each combination is measured in three fresh processes. Model and shape order are randomized within each repetition using seeds 9013–9015. Ten warmup operations precede fifty synchronized wall-clock measurements. For each process, we divide the input-token count by its mean operation time, then report the mean and sample standard deviation of the three process throughputs. The 150 operations are not treated as 150 independent runs. Compilation, optimizer updates, data loading, packing, and gradient accumulation are excluded.

Before timing, each selected execution setting is compared with its accepted eager implementation at the full input shape, checking outputs, loss, and parameter gradients. KLA's updated backend is compared with its previous Triton implementation and additionally checks recurrence outputs, variance, and input gradients against the upstream sequential reference. Voltic checks complete-sequence memory outputs and output-loss gradients against an independent FP64 recurrence at sampled first and final layers. These checks apply to the measured training operation, which does not reuse returned states.

Reference-memory limits. Where the whole-batch reference runs out of memory, we compute each independent row separately, concatenate all logits, and average the equal-length row losses and parameter gradients. This reference procedure is first checked against whole-batch execution at length 1K and batch two, then used at the complete target shape with unchanged numerical bounds. The candidate is qualified and timed with the actual full batch, never split into smaller timed batches. Together with the updated KLA backend, this completes all 256 measured combinations, including the previously omitted 180M Voltic cases at 8K and batch eight. Their first- and final-layer sequence-output/gradient oracle checks pass, but the separately recorded final-state API comparisons exceed their oracle bounds for both Voltic variants. The training timing checks therefore do not certify that separate API or arbitrary cached execution.

H.4 Training throughput

The fixed-token view in Figure 6 uses the entries with $BT = 8,192$, so length and batch size change together. Figure 7 holds sequence length at 2K and compares batches one and eight at both model scales.

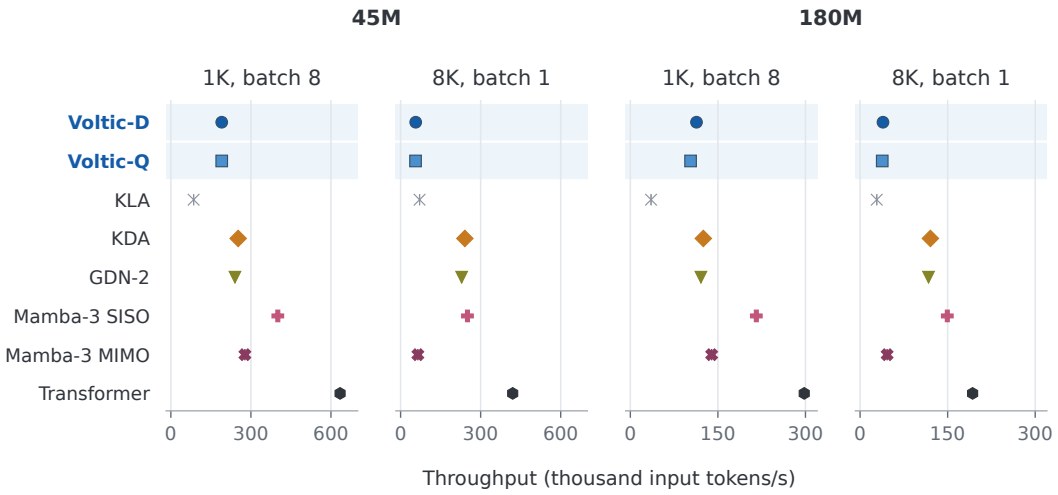


Figure 6: Training throughput at 8,192 input tokens per operation: mean ± sample SD over three timing processes.

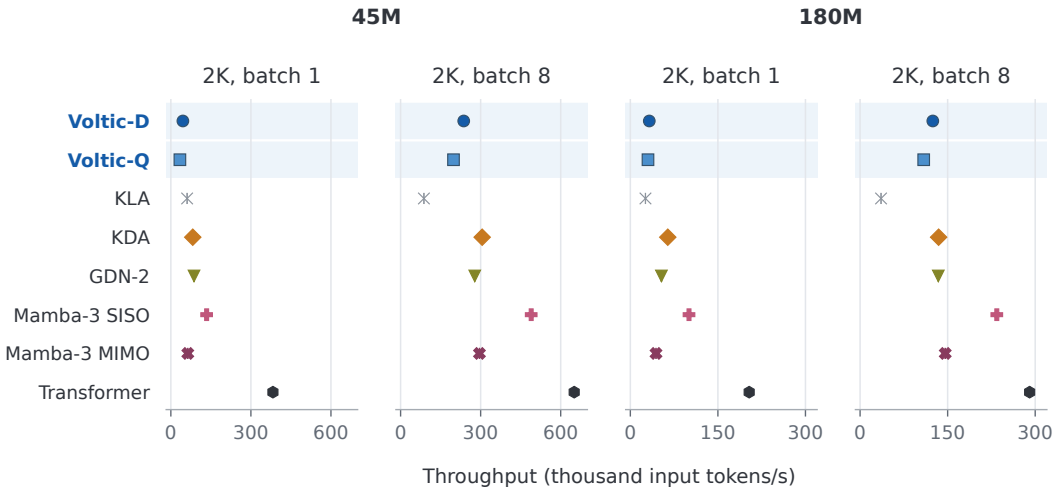


Figure 7: Training throughput at 2K sequence length, batches one and eight: mean ± sample SD over three timing processes.

H.5 Cached autoregressive decoding

We measure all eight configurations at both scales, with batch one, prompt lengths 1K, 2K, 4K, and 8K, and 128 cached decode steps per generation. The prompt cache is reconstructed before each timed block. Time per output token (TPOT) is the synchronized wall-clock time for the complete decode block divided by 128. It includes host dispatch, every model layer, the full vocabulary projection, and GPU greedy selection. It excludes prompt processing, tokenization, networking, and backward computation. Each of three fresh timing processes uses two warmup generations and five measured generations. We average each process's five block times before computing the three-process mean and sample standard deviation. These output-token latencies are not training input-token throughput.

Cached implementations. Voltic uses its Triton recurrent inference path; the Voltic-Q cache retains within-block information and chunk phase. KDA and GDN-2 use their chunked prefill and fused recurrent decode kernels. Mamba-3 uses its official cached step kernels with a singleton-sequence shape adapter. The Transformer uses its frozen projections, rotary positions, normalization, and feed-forward layers with a key/value-cache adapter and PyTorch scaled dot-product attention. No

full-prefix recomputation is substituted for cached decoding. KLA uses its updated official release, but its CUDA training backend cannot return a prefill state: cached generation instead uses the official state-producing Triton prefill and `kla_step`. We do not label these KLA decoding results as CUDA-v3-fast timings.

Checks and limits. Before timing, cached teacher-forced logits are compared with complete-sequence computation at all 64 continuation positions following each full-length prompt, including Voltic-Q block boundaries. Admission requires finite logits and global and per-token relative ℓ_2 error at most 0.05. Pointwise errors and argmax agreement are recorded as diagnostics, not additional pass criteria. All 64 tested configurations pass this finite continuation check; it does not establish exact greedy-token agreement for arbitrary prompts or replace a final-state oracle certificate.

Results. Figure 2 shows the decoding latencies for 2K prompts. Across the measured 1K–8K range, most latencies change little. KLA's 8K points were measured separately after increasing only the wrapper's length limit to cover the prompt and continuation. Its roughly 18% lower 8K latency than the earlier 1K–4K points has no established cause; we do not interpret this as improved scaling with context length. Weights, kernels, precision, timing rules, and physical GPU were unchanged.