DART: DECODED ATTENTION OVER RECURRENT STATES FOR EFFICIENT LONG-CONTEXT SEQUENCE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern language models are built primarily from Transformers, recurrent models, and their hybrid architectures. Transformers rely on token-level attention memories, while recurrent models such as state space models (SSMs) and linear attention maintain compact recurrent states. These architectures are typically instantiated separately or interleaved at the layer level, leaving open whether a shared memory representation can support both recurrent compression and attention-style retrieval. We study this question through the state space duality (SSD) view of Mamba-2, where the SSM state can be interpreted as a compressed associative key–value (KV) cache. We observe that Mamba-2 decodes token-conditioned values from this state but does not decode token-conditioned keys. Based on this observation, we propose DART (Decoded Attention over Recurrent sTates), which retains the chunk state contributions produced by Mamba-2's chunkwise SSD computation as chunk state memories, decodes token-conditioned keys and values from these memories, and performs state-memory attention (SMA) over the resulting KV pairs. The retrieved output is then combined with the native Mamba-2 output through a gated residual connection. DART supports practical training by reusing the Mamba-2 chunkwise SSD computation and implementing SMA as a FlashAttention-style computation. Our analysis and experiments show that DART substantially reduces the inference cache compared with a matched attention baseline (e.g., 75% savings when the chunk size is $S = 256$ and the state size is $N = 128$). Compared with Mamba-2, DART substantially improves associative recall and retrieval while preserving general language-modeling quality.

1 INTRODUCTION

Modern large language models (LLMs) are built mainly from Transformers, recurrent models, and their hybrid architectures. Transformers are the dominant backbone of LLMs and can directly attend to all previous tokens (Vaswani et al., 2017; Brown et al., 2020; Touvron et al., 2023). However, full attention has quadratic training compute in sequence length, and autoregressive inference requires a token-level KV cache that grows linearly with context length. Recurrent models provide a representative alternative for addressing this efficiency bottleneck, ranging from classical RNNs and LSTMs to modern linear-time language models such as RWKV, RetNet, Mamba, and DeltaNet (Elman, 1990; Hochreiter & Schmidhuber, 1997; Peng et al., 2023; Sun et al., 2023; Gu & Dao, 2023; Dao & Gu, 2024; Yang et al., 2024). These models process sequences through recurrent state updates and maintain a compact recurrent state during inference. While these advantages make recurrent models attractive for long sequences, compressing long contexts into compact states creates a memory bottleneck. Prior studies on associative recall, copying, and position-structured retrieval show that recurrent models can lose long-range recall capacity relative to attention (Arora et al., 2024a; Jelassi et al., 2024; Waleffe et al., 2024).

These complementary tradeoffs have motivated hybrid architectures that combine attention and recurrent models. By mixing efficient recurrent layers with attention layers, hybrid models can improve the trade-off between compute efficiency and recall. Recent systems have achieved strong language-modeling and retrieval performance (Lieber et al., 2024; Glorioso et al., 2024; Ren et al., 2025; Lahoti et al., 2026). Most hybrid designs, however, keep token-level attention memories and

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.

compact recurrent states as separate memory forms and combine them by interleaving or stacking layers. This leaves open a more basic question: can the same memory representation support both recurrent compression and attention-style retrieval?

Mamba-2 provides a natural starting point through state space duality (SSD) (Dao & Gu, 2024). SSD shows that a selective state space model (SSM) layer can be written as structured causal attention, connecting the SSM coefficients (A, B, C, X) with the attention roles of decay, keys, queries, and values. In this view, the Mamba-2 state is a compressed KV cache. The recurrent update writes token-dependent information into this cache, while the readout $C_t H_t$ extracts the value needed by the current token. SSD also leads to the hardware-efficient chunkwise SSD algorithm.

The key observation of this paper is that Mamba-2 reads values from the compressed KV cache but does not extract keys from it. If keys can also be decoded from this compressed state, attention can retrieve from a much smaller set of compressed recurrent memories rather than operating over all token-level keys and values. This suggests a state-memory attention (SMA) mechanism between full token-level attention and purely recurrent models: recurrent models learn to compress input tokens into compact states, and attention retrieves from the compressed memories according to the current query.

Based on this observation, we propose DART (Decoded Attention over Recurrent sTates), an architecture that augments Mamba-2 with an SMA branch for attention-style retrieval over compressed recurrent states. DART preserves Mamba-2's chunkwise SSD computation, treats each chunk state contribution as a chunk state memory, extracts token-conditioned keys and values from these memories, and adds the retrieved state readout as a gated residual correction to the Mamba-2 output. Our contributions can be summarized as follows:

- We introduce DART, an architecture that connects recurrent compression with attention-style retrieval through SMA. Mamba-2's chunkwise SSD computation produces chunk state contributions that serve as chunk state memories, and SMA retrieves from them with token-conditioned keys and values.
- We show that DART is practical to train by reusing the Mamba-2 chunkwise SSD algorithm and implementing SMA with a FlashAttention-style computation (Dao et al., 2022; Dao, 2024).
- We evaluate DART on MQAR, NIAH, real-world retrieval tasks, Pile pretraining, and zero-shot downstream evaluations. Compared with Mamba-2, DART substantially improves associative recall and long-context retrieval while preserving language-modeling quality. Compared with matched attention, DART substantially reduces the length-dependent inference cache without relying on sparse or sliding-window mechanisms.

2 BACKGROUND

State Space Models and State Space Duality. State space models (SSMs) model sequences through dynamics described by continuous-time ordinary differential equations (Gu et al., 2021; 2022; Smith et al., 2022; Qian et al., 2026). Modern selective SSMs make these dynamics input-dependent, allowing the model to control what information is written, retained, and read (Gu & Dao, 2023; Dao & Gu, 2024; Lahoti et al., 2026). For $X, Y \in \mathbb{R}^{L\times P}$ and scalar transition $A_t = a_t I_N$, Mamba-2 has the recurrence

$$H_t = A_t H_{t-1} + B_t^\top X_t, \qquad Y_t = C_t H_t, \tag{1}$$

where $H_t \in \mathbb{R}^{N\times P}$ and $B_t, C_t \in \mathbb{R}^{1\times N}$. The update $B_t^\top X_t$ writes the current input into the state, and $C_t H_t$ reads from it. We write

$$A_{u:v} = A_v A_{v-1} \cdots A_u = \prod_{k=u}^{v} A_k, \qquad u \le v, \tag{2}$$

and set $A_{u:v} = I_N$ when $u > v$. State space duality (SSD) expresses this recurrence as structured causal attention. Letting $B, C \in \mathbb{R}^{L\times N}$ stack the token-wise vectors, SSD gives

$$Y = \left(L_{\mathrm{SSD}} \circ CB^\top\right) X. \tag{3}$$

Here $(L_{\text{SSD}})_{ij} = \prod_{k=j+1}^{i} a_k$ for $i \geq j$, and $(L_{\text{SSD}})_{ij} = 0$ otherwise. Without the decay mask, $CB^\top X$ has the linear-attention form $QK^\top V$ with $Q = C$, $K = B$, and $V = X$. Thus, H_t can be viewed as a compressed KV cache from which $C_t H_t$ decodes the current value.

Recurrent Models and Transformers. Recurrent models compress history into compact states, enabling linear-time sequence processing and memory-efficient inference. However, encoding long histories into fixed-size states can limit precise long-range recall because future queries are unknown when information is written, as observed in associative recall, copying, and position-structured retrieval (Arora et al., 2024a; Jelassi et al., 2024; Waleffe et al., 2024). Delta-rule models improve memory editing through targeted updates and adaptive gating (Yang et al., 2024; 2025; Hatamizadeh et al., 2026), while log-linear attention maintains a logarithmically growing hierarchy of recurrent states (Guo et al., 2026). Transformers preserve token-level keys and values, providing precise content-dependent retrieval, but incur quadratic training compute and a KV cache that grows with context length. FlashAttention removes the quadratic activation-memory cost of standard attention (Dao et al., 2022; Dao, 2024), while sparse attention, cache selection, latent KV representations, and quantization further reduce inference costs (Beltagy et al., 2020; Zaheer et al., 2020; Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024; Liu et al., 2024a;b).

Hybrid Architectures. The complementary strengths and limitations of recurrent models and Transformers have motivated hybrid architectures such as Jamba, Zamba, and Samba, which interleave recurrent and attention layers (Lieber et al., 2024; Glorioso et al., 2024; Ren et al., 2025). These models improve the tradeoff between recurrent efficiency and attention-based retrieval, but the placement and frequency of attention layers remain architecture-specific choices that are largely guided by empirical tuning. Other works move the interaction between recurrence and attention inside a layer. Attamba retains chunk-boundary outputs from separate key and value SSMs, while RAT applies recurrence within chunks and softmax attention across chunk summaries (Akhauri et al., 2024; Wei et al., 2025). A comparison between interleaved hybrids and DART is shown in Figure 1.

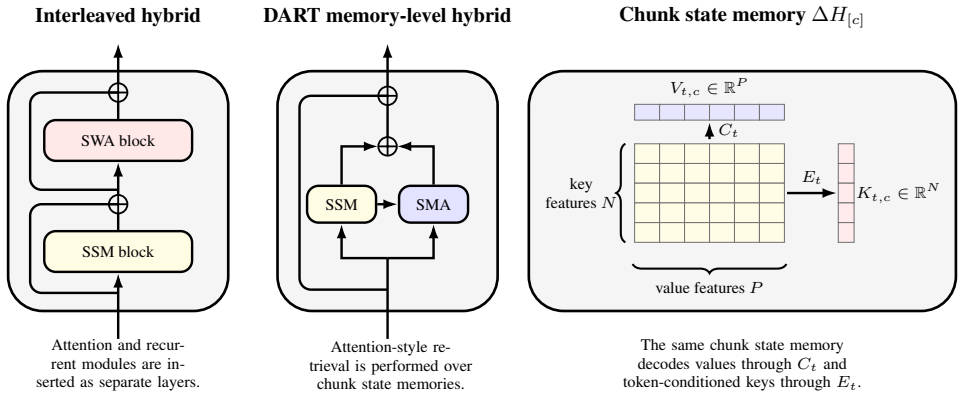

Figure 1: Comparison between interleaved hybrid architectures and DART. Standard interleaved hybrids place attention and recurrent modules in separate layers, whereas DART integrates attention-style retrieval within each recurrent block through SMA over chunk state memories. The matrix view shows that a chunk state contribution $\Delta H_{[c]} \in \mathbb{R}^{N \times P}$ supports both value readout $C_t \Delta H_{[c]}$ and token-conditioned key decoding $\Delta H_{[c]} E_t$.

3 METHODOLOGY

DART augments Mamba-2 with state-memory attention (SMA) over its chunk state contributions, which DART retains as chunk state memories. In the SSD view, the recurrent state H_t is a compressed KV cache: Mamba-2 reads values through $C_t H_t$, while DART also decodes token-conditioned keys from chunk state memories. We refer to the original Mamba-2 as the SSM branch,

and to the retrieval module over chunk state memories as the SMA branch. More details about Mamba-2 chunkwise SSD computation are provided in Appendix A.

Chunk State Memories. For a sequence of length L and chunk size S, DART divides the sequence into chunks. We write $[c]$ for the set of token indices in the c-th chunk, let $c(t) = \lceil t/S \rceil$ denote the chunk containing token t, and let $\Delta H_{[c]}$ denote its chunk state contribution:

$$\Delta H_{[c]} = \sum_{s \in [c]} A_{s+1:e(c)} B_s^\top X_s \in \mathbb{R}^{N \times P}. \tag{4}$$

Here $e(c)$ is the last index in $[c]$, and $A_{s+1:e(c)} = \left(\prod_{k=s+1}^{e(c)} a_k \right) I_N$, with the empty product equal to the identity. In the recurrent computation, we refer to $\Delta H_{[c]}$ as a *chunk state contribution* because it is the contribution of chunk c to the chunk-boundary state. When retained for SMA, the same tensor serves as a *chunk state memory*. The matrix $\Delta H_{[c]}$ has a key axis N and a value axis P. Reading it from the left with $C_t \in \mathbb{R}^{1 \times N}$ produces a value, while contracting its value axis with $E_t \in \mathbb{R}^{P \times 1}$ produces a key:

$$C_t \Delta H_{[c]} \in \mathbb{R}^{1 \times P} \quad \text{(value-side readout)}, \qquad \Delta H_{[c]} E_t \in \mathbb{R}^{N \times 1} \quad \text{(key-side readout)}. \tag{5}$$

State-Memory Attention. Let $U_t \in \mathbb{R}^{1 \times d_{\text{model}}}$ denote the input representation to the DART block. For a single SSM head and token t, DART forms three token-conditioned vectors from the block input U_t:

$$Q_t = U_t W_Q \in \mathbb{R}^{1 \times N}, \qquad C_t = U_t W_C \in \mathbb{R}^{1 \times N}, \qquad E_t = (U_t W_E)^\top \in \mathbb{R}^{P \times 1}, \tag{6}$$

where $W_Q, W_C \in \mathbb{R}^{d_{\text{model}} \times N}$ and $W_E \in \mathbb{R}^{d_{\text{model}} \times P}$. Here Q_t is a query vector in the SSD key-side space and is used for chunk routing. The vector C_t is the native Mamba-2 read vector shared with the SMA branch, and $C_t \Delta H_{[c]}$ extracts the value needed by token t from the chunk state memory. Empirically, this sharing is important for optimization, as an independent SMA read vector did not learn useful recall behavior in our experiments. The vector E_t is a value-side verifier, and $\Delta H_{[c]} E_t$ turns each chunk state memory into a token-conditioned key.

For each historical chunk c, DART computes:

$$V_{t,c} = C_t \Delta H_{[c]} \in \mathbb{R}^{1 \times P}, \qquad K_{t,c} = \Delta H_{[c]} E_t \in \mathbb{R}^{N \times 1}. \tag{7}$$

The corresponding chunk logit is

$$\ell_{t,c} = \frac{Q_t K_{t,c}}{\sqrt{N}} \in \mathbb{R}. \tag{8}$$

Gated Residual Integration. The SMA logits are normalized over historical chunks:

$$g_{t,c} = \text{softmax}_{c < c(t)}(\ell_{t,c}), \qquad R_t = \sum_{c < c(t)} g_{t,c} V_{t,c} \in \mathbb{R}^{1 \times P}. \tag{9}$$

For tokens in the first chunk, the historical memory set is empty; in this case we define the SMA readout as $R_t = 0$. In practice, we use RMS-normalized Q_t, E_t, and chunk states for stable routing, with details provided in Appendix C. DART then adds the SMA readout as a scalar-gated residual correction to the SSM branch readout:

$$G_t = \text{SiLU}(U_t W_G) \in \mathbb{R}, \qquad Y_t = C_t H_t + G_t R_t, \tag{10}$$

where $W_G \in \mathbb{R}^{d_{\text{model}} \times 1}$. We initialize W_G to zero, so the SMA residual starts from $G_t R_t = 0$ and is learned as a residual correction during training.

The architecture of DART is shown in Figure 2. The SSM branch follows the Mamba-2 block and passes chunk state memories to the SMA branch. The SMA branch uses token-conditioned Q_t and E_t for retrieval, and its readout is added to the SSM output through a gated residual connection.

4 Efficient Training and Complexity Analysis

DART can be trained efficiently because it reuses the Mamba-2 chunkwise SSD algorithm to produce chunk state memories, and implements SMA with FlashAttention-style computation. The complexity analysis shows that DART can substantially reduce inference cache size. With a common setting of chunk size $S = 256$ and state size $N = 128$, DART saves 75% cache memory compared with token-level attention.

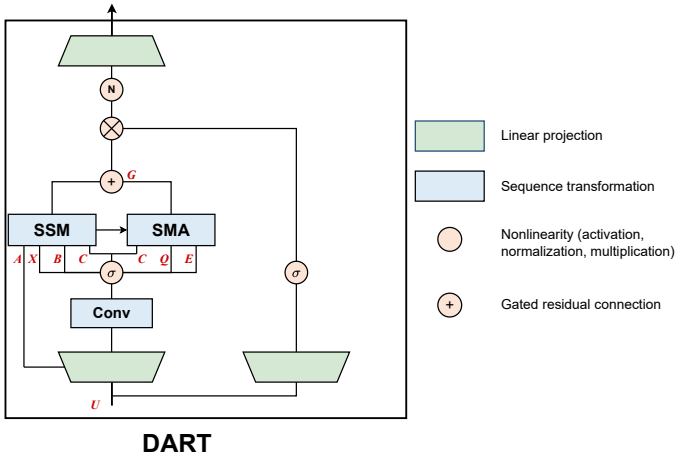

Figure 2: Architecture of DART. The SSM branch provides chunk state memories to the SMA branch, whose retrieved readout is combined with the SSM output through a gated residual connection. In practice, the B, C, Q, and E projections use a single projection group shared across all heads, analogous to multi-value attention (MVA) in Mamba-2.

Efficient Training. Figure 3 illustrates the Mamba-2 chunkwise SSD computation and SMA readout used by DART. DART reuses the efficient Mamba-2 chunkwise SSD algorithm (Dao & Gu, 2024), which produces both the SSM branch output $C_t H_t$ and the chunk state contributions $\Delta H_{[c]}$. The SMA branch is implemented with a custom FlashAttention-style kernel (Dao et al., 2022; Dao, 2024). Following FlashAttention, the kernel fixes a query tile and streams over historical chunks. For each historical chunk c, the kernel constructs $K_{t,c}$ and $V_{t,c}$ from $\Delta H_{[c]}$ on chip, updates the online softmax over chunks, and accumulates the SMA readout R_t. During the forward computation, the kernel writes only the SMA readout and logsumexp statistics to HBM. Chunk keys, values, logits, and attention weights remain on-chip temporaries and are recomputed during backpropagation. Appendix B provides the background on online softmax and recomputation, and the complete forward procedure is given in Algorithm 1 in Appendix C.2.

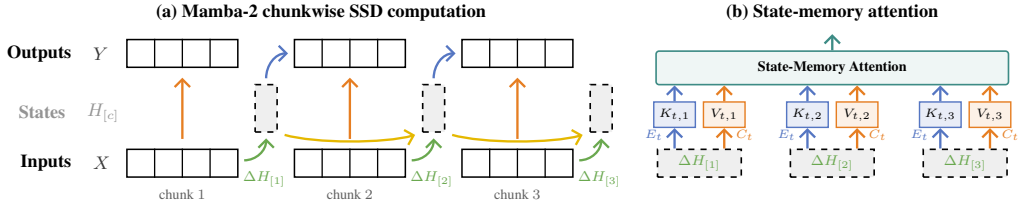

Figure 3: Construction and retrieval of chunk state memories in DART. **Left**: the Mamba-2 chunkwise SSD computation produces local outputs and chunk state contributions while passing compact boundary states across chunks. **Right**: for each token t, E_t and C_t decode a key $K_{t,c}$ and value $V_{t,c}$ from each historical chunk state memory. SMA attends over the decoded key–value pairs and produces the readout R_t.

Training Complexity. Let the sequence length be L, chunk size be S, number of chunks be $M = \lceil L/S \rceil$, state size be N, and head dimension be P. In training, the SMA branch lets each token read from at most $M - 1$ historical chunk state memories. For each token–chunk pair, the kernel forms a token-conditioned key $K_{t,c} \in \mathbb{R}^N$ and value $V_{t,c} \in \mathbb{R}^P$ through bilinear contractions with $\Delta H_{[c]} \in \mathbb{R}^{N \times P}$. The resulting compute cost is

$$O(LMNP) = O(L^2 NP/S). \tag{11}$$

Compared with Mamba-2, DART adds this SMA branch on top of the linear-time SSD computation, trading additional compute for query-dependent access to historical compressed states. Compared

with token-level attention implemented by FlashAttention, the SMA branch reads M chunk state memories rather than L token-level KV entries. Its retrieval compute has a leading-order ratio

$$\frac{O(L^2NP/S)}{O(L^2P)} = \frac{N}{S} \tag{12}$$

relative to token-level attention per head, and is smaller when $N < S$.

Inference Cache. DART is compared against a matched token-level attention baseline during autoregressive inference. This baseline stores keys and values for each token, giving a length-dependent KV cache of size $O(2LP)$ per head. DART instead stores one chunk state memory per chunk, with cache size $O(MNP) = O(LNP/S)$. The cache ratio to this matched attention baseline is therefore

$$\frac{LNP/S}{2LP} = \frac{N}{2S}. \tag{13}$$

For example, with chunk size $S = 256$ and state size $N = 128$, the chunk state-memory cache is 25% of the token-level KV cache, saving 75% of the length-dependent cache memory.

5 Experiments

We evaluate DART on synthetic MQAR to test associative recall, pretrain it on the Pile for 100B tokens, and evaluate on general language-modeling and retrieval tasks. We further analyze runtime and memory behavior of the SMA, with more details provided in Appendix D.

5.1 Synthetic Tasks: Associative Recall

Following Mamba-2 (Dao & Gu, 2024), we use synthetic associative recall to test whether DART improves the ability to look up information from context. We adopt multi-query associative recall (MQAR), which requires an autoregressive model to memorize multiple in-context key–value associations and predict the associated value when queried by a previously seen key (Arora et al., 2024a). We compare DART against Mamba-2 under the same sequence length, model width, and SSD state size, and include the Transformer++ attention baseline (Dao & Gu, 2024) under the same sequence length and model width. The chunk size is set to $S = 16$. For DART, this relatively small chunk size tests whether SMA can retrieve the stored associations from chunk state memories.

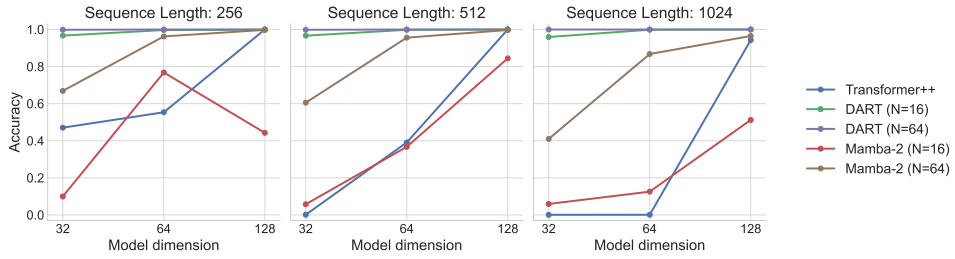

Figure 4: MQAR recall accuracy. Each panel fixes the sequence length L and sweeps the model width d_{model}. DART and Mamba-2 are evaluated with SSD state sizes $N \in \{16, 64\}$. All markers report test accuracy.

Results are shown in Figure 4. We sweep sequence length $L \in \{256, 512, 1024\}$ and model width $d_{\text{model}} \in \{32, 64, 128\}$. For Mamba-2 and DART, we evaluate SSD state sizes $N \in \{16, 64\}$. More details are provided in Appendix D.1. DART consistently improves MQAR recall over Mamba-2 under matched model width and state size. With $N = 64$, DART reaches near-perfect accuracy across all tested sequence lengths and model widths. With the smaller state size $N = 16$, DART also remains highly accurate, and even exceeds Mamba-2 with $N = 64$ in the matched-width configurations. In contrast, Mamba-2 degrades substantially when either the state size or model width is small, especially at longer sequence lengths. The Transformer++ baseline is strong when the width is large enough, but is unreliable in low-width and long-sequence settings. These results indicate that SMA makes compressed SSD memories much more accessible than the native Mamba-2 readout, especially when recurrent state capacity is limited.

5.2 LANGUAGE MODELING

General Sequence Modeling. We start from the baseline Mamba-2 model sizes of 130M, 370M, and 780M parameters, and construct DART by adding the SMA branch to the same backbone shapes. We additionally compare with Attamba, a related chunk-level hybrid architecture (Akhauri et al., 2024). We instantiate its key and value SSM branches with independent Mamba-2 encoders. We evaluate Attamba with chunk size 4 at comparable parameter scales. Its length-dependent cache ratio approaches 25% of the Transformer KV cache, matching DART with $S = 256$ and $N = 128$. All models are pretrained on the Pile for 100B tokens, and the resulting models are evaluated with Pile validation metrics and zero-shot downstream evaluations. Details of the Pile pretraining setup are provided in Appendix D.2.

Table 1 reports validation perplexity and accuracy on the Pile for the trained models, while Table 2 reports zero-shot downstream results. For a public full-attention reference trained under a comparable token budget, we also include Pythia checkpoints (Biderman et al., 2023) released at approximately 100B pretraining tokens in the downstream evaluation. DART adds fewer than 3% parameters through the SMA branch at these model sizes. Across the evaluated model sizes, DART preserves Pile perplexity, improves Pile validation accuracy, and obtains higher average zero-shot downstream accuracy than Mamba-2. Compared with Attamba, DART obtains lower Pile perplexity and higher average zero-shot accuracy with its main $S = 256$ setting at all three parameter scales. Most standard downstream examples in Table 2 are short-context inputs, with lengths below 256 tokens. Thus, the $S = 256$ setting mainly evaluates whether training with SMA preserves the general capability of the SSM branch. With smaller evaluation chunk sizes $S = 64$ and $S = 32$, SMA is active on more examples, and DART maintains downstream performance when the SMA branch contributes at evaluation time.

Table 1: Model parameters and Pile validation metrics after 100B-token pretraining. Lower perplexity and higher accuracy are better.

	Mamba-2	DART	Attamba	Mamba-2	DART	Attamba	Mamba-2	DART	Attamba
Model scale	130M	133M	135M	370M	380M	371M	780M	795M	781M
Parameters	128.99M	132.70M	134.57M	368.35M	378.22M	371.21M	780.16M	794.95M	781.39M
Pile PPL ↓	12.23	12.13	12.19	9.99	9.90	10.23	8.85	8.86	8.95
Pile Acc. ↑	50.58	50.75	51.02	53.27	53.45	53.25	54.86	54.88	54.90

Table 2: Zero-shot downstream evaluation of models pretrained on the Pile. LAMBADA PPL is reported as perplexity; LAMBADA, PIQA, ARC-Easy, and WinoGrande report accuracy; HellaSwag, ARC-Challenge, and OpenBookQA report length-normalized accuracy. All accuracy entries are percentages, and Avg. is the mean over the seven accuracy metrics. Best and second-best entries within each model-size group are shown in bold and underline, respectively.

Model	Chunk Size	LAMBADA PPL	LAMBADA	HellaSwag	PIQA	ARC-E	ARC-C	WinoGrande	OpenBookQA	Avg.
Metric	–	ppl ↓	acc ↑	acc_n ↑	acc ↑	acc ↑	acc_n ↑	acc ↑	acc_n ↑	acc ↑
Mamba-2 130M	–	**22.44**	38.09	32.45	62.89	44.61	23.21	50.04	28.40	39.96
DART 133M	256	22.77	38.35	**32.69**	62.95	**45.37**	**24.23**	**51.14**	29.80	**40.65**
DART 133M	64	25.28	36.66	32.66	**63.00**	**45.37**	**24.23**	**51.14**	29.80	40.41
DART 133M	32	24.13	37.07	32.68	62.73	45.29	23.81	50.59	**30.00**	40.31
Attamba 135M	4	25.64	**39.74**	32.43	62.02	44.28	23.38	**51.62**	29.40	40.41
Pythia 160M	–	35.67	33.30	31.10	62.73	44.36	23.38	50.20	26.40	38.78
Mamba-2 370M	–	13.53	46.85	38.93	66.81	**51.35**	24.15	51.07	29.40	44.08
DART 380M	256	**12.68**	**47.45**	**39.17**	67.52	50.21	24.66	50.99	**30.80**	44.40
DART 380M	64	13.18	46.40	39.07	67.52	50.34	24.57	**52.17**	**30.80**	44.41
DART 380M	32	12.92	47.22	39.08	**67.68**	50.46	24.40	52.09	**30.80**	**44.53**
Attamba 371M	4	15.40	46.09	37.51	65.89	48.78	**25.51**	51.30	29.00	43.44
Pythia 410M	–	16.20	43.76	37.09	65.13	50.55	24.15	51.07	30.40	43.16
Mamba-2 780M	–	9.50	52.32	**44.42**	**69.04**	52.53	25.94	52.88	31.40	46.93
DART 795M	256	9.44	52.22	44.03	68.34	**55.72**	**27.13**	**54.46**	30.80	**47.53**
DART 795M	64	9.87	51.27	44.16	68.66	55.68	27.05	54.22	31.20	47.46
DART 795M	32	9.76	51.50	44.18	68.34	55.51	27.05	**54.46**	31.00	47.43
Attamba 781M	4	**9.33**	**53.27**	43.49	68.66	54.38	26.02	52.96	**33.00**	47.40
Pythia 1B	–	10.77	50.81	42.47	68.01	53.28	24.91	51.78	30.80	46.01

Retrieval Capacity. We further evaluate DART on retrieval tasks, with details provided in Appendix D.3. As shown in Table 3, DART substantially improves SWDE and FDA over Mamba-2 across all three model scales. The gains on QA-style retrieval tasks are smaller but generally positive. DART also improves many NIAH settings that require retrieving information from the provided context. The hardest 4096-token NIAH-Single-2/3 settings remain challenging for both

models. Table 3 reports DART at its pretraining chunk size $S = 256$ as the main configuration, and includes evaluation-time chunk-size ablations at $S = 64$ and $S = 32$ using the same trained checkpoints. At $S = 256$, DART already substantially improves over Mamba-2 across extraction tasks and many NIAH settings. The chunk-size ablations further measure sensitivity to inference granularity: smaller chunks improve several extraction-heavy metrics such as SWDE and FDA, whereas $S = 64$ is strongest on some NIAH-Single-2/3 settings. Attamba is particularly strong on several NIAH-Single-2/3 settings at 1024 and 2048 tokens, whereas DART performs better on extraction-heavy tasks and NIAH-Single-1. Pythia remains a strong full-attention reference on several QA and NIAH columns, indicating that DART narrows the retrieval gap to attention while operating under the compact-memory constraints of recurrent states.

Table 3: Retrieval evaluation of models pretrained on the Pile. The real-world suite includes extraction-heavy tasks (SWDE, SQuAD Completion, and FDA) and QA-style retrieval tasks (TriviaQA, NQ, and DROP). NIAH reports synthetic retrieval accuracy with 500 examples per context length. All real-world tasks are evaluated with contains accuracy. All entries are percentages, and higher is better. Best and second-best entries within each model-size group are shown in bold and underline, respectively.

		Real-world retrieval						NIAH-Single-1			NIAH-Single-2			NIAH-Single-3		
Model	Chunk Size	SWDE	SQD	FDA	TQA	NQ	DROP	1024	2048	4096	1024	2048	4096	1024	2048	4096
Mamba-2 130M	–	17.1	25.6	15.4	38.4	5.7	16.6	98.6	97.0	**74.2**	2.4	1.0	**1.6**	1.0	1.0	1.2
DART 133M	256	34.9	23.3	19.7	37.8	**9.1**	17.3	**100.0**	**99.4**	57.2	59.0	47.2	**1.6**	36.0	46.6	**5.0**
DART 133M	64	44.8	26.6	26.2	38.8	7.1	18.1	**100.0**	**99.4**	65.2	65.6	42.2	1.4	71.4	47.2	4.4
DART 133M	32	46.1	26.7	29.0	38.4	6.4	**18.2**	**100.0**	98.4	66.8	50.6	27.6	1.4	37.0	24.8	3.0
Attamba 135M	4	38.9	**28.8**	19.3	**40.7**	6.7	**18.2**	87.0	73.4	0.0	93.0	77.4	0.0	**74.0**	**81.8**	0.0
Pythia 160M	–	**50.9**	26.5	**37.0**	35.4	9.0	**18.2**	98.2	99.2	0.0	**98.0**	**83.2**	0.0	52.8	25.4	0.0
Mamba-2 370M	–	33.2	29.9	16.6	45.9	9.5	20.9	**100.0**	**100.0**	95.4	1.4	1.4	1.8	2.6	3.4	2.8
DART 380M	256	46.9	29.8	31.3	46.3	12.6	18.9	**100.0**	**100.0**	**95.8**	13.0	14.0	**3.2**	2.8	4.4	**5.8**
DART 380M	64	56.7	31.2	45.8	45.3	11.6	18.9	99.8	99.4	86.6	24.0	32.6	1.8	22.8	18.4	4.6
DART 380M	32	56.2	31.5	50.4	44.4	10.4	19.0	99.6	97.0	65.8	36.0	35.4	1.4	35.4	34.8	3.4
Attamba 371M	4	49.2	29.6	23.0	46.4	10.0	**21.4**	28.4	81.4	0.0	67.8	70.6	0.0	97.2	**94.4**	0.0
Pythia 410M	–	**68.8**	**32.9**	**62.7**	**49.1**	**14.3**	20.8	99.2	99.2	0.0	**99.6**	**85.4**	0.0	**98.2**	91.8	0.0
Mamba-2 780M	–	39.9	32.6	17.2	50.5	11.8	21.3	85.8	91.2	99.8	40.6	24.6	**1.8**	70.6	19.4	**3.6**
DART 795M	256	56.5	32.4	28.9	52.0	16.1	23.5	**100.0**	**100.0**	99.8	60.6	70.0	1.4	67.6	43.4	2.6
DART 795M	64	63.7	**35.5**	41.4	51.1	15.7	**23.6**	**100.0**	**100.0**	**100.0**	83.0	90.6	1.4	**99.6**	80.6	2.4
DART 795M	32	65.9	34.8	47.9	50.4	14.5	23.4	**100.0**	**100.0**	**100.0**	80.6	89.4	1.4	99.2	88.8	2.4
Attamba 781M	4	60.5	34.8	24.4	**54.7**	13.3	23.2	98.2	80.6	0.0	**99.8**	96.4	0.0	94.4	82.8	0.0
Pythia 1B	–	**68.7**	34.8	**66.6**	52.5	**17.6**	23.2	99.0	97.4	0.0	99.2	**99.8**	0.0	91.0	**97.0**	0.0

5.3 Ablations and Performance Analysis

SMA Branch. We assess the contribution of the SMA branch through an evaluation-time ablation. For each trained DART checkpoint, we remove the SMA residual readout during evaluation and keep only the native SSM branch. The resulting performance change measures how much associative-recall and retrieval capacity is provided by the SMA branch beyond the recurrent state readout. Table 4 reports the retrieval-task ablation, and Table 5 reports the MQAR ablation. Removing SMA substantially weakens MQAR and real-world extraction performance, indicating that the retrieval gains mainly come from making chunk state memories explicitly addressable.

Table 4: Evaluation-time ablation of the SMA branch on NIAH and real-world extraction tasks for Pile-pretrained checkpoints. The ablated rows use the same trained checkpoints as the full model, but remove the SMA residual readout during evaluation. All evaluations use $S = 256$, and all entries are accuracies in percent.

	NIAH-Single-1			NIAH-Single-2			NIAH-Single-3			Real	
Model	1024	2048	4096	1024	2048	4096	1024	2048	4096	SWDE	FDA
DART 133M w/o SMA	12.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.2	0.7
DART 133M	100.0	99.4	57.2	59.0	47.2	1.6	36.0	46.6	5.0	34.9	19.7
DART 380M w/o SMA	100.0	95.8	66.2	4.6	4.2	3.0	2.4	4.6	3.2	13.8	7.1
DART 380M	100.0	100.0	95.8	13.0	14.0	3.2	2.8	4.4	5.8	46.9	31.3
DART 795M w/o SMA	99.8	98.0	90.8	11.6	13.2	3.0	29.6	18.6	3.2	17.3	3.0
DART 795M	100.0	100.0	99.8	60.6	70.0	1.4	67.6	43.4	2.6	56.5	28.9

Sharing the SSM Read Vector. DART uses the native Mamba-2 read vector C_t as the value-side read vector in the SMA branch. This design aligns the SMA readout with the SSM branch when decoding historical chunk state memories. We train otherwise matched variants that replace C_t with an independently parameterized SMA read vector $C_t^{\mathrm{SMA}} = U_t W_C^{\mathrm{SMA}}$. Table 6 reports the MQAR

comparison in the memory-constrained setting $N = 16$. The shared design consistently succeeds across widths and sequence lengths, whereas the independently parameterized read vector fails to learn useful recall behavior in this setting and stays at zero accuracy. This shows that aligning the SMA value readout with the native SSM readout is important for decoding historical chunk state memories.

Table 5: Evaluation-time ablation of the SMA branch on MQAR. Each row uses a DART model with SSD state size $N = 16$ and chunk size $S = 16$. All entries are test accuracies in percent.

Model	d_{model}	$L = 256$	$L = 512$	$L = 1024$
DART w/o SMA	32	12.40	0.30	0.11
DART	32	96.70	96.67	95.94
DART w/o SMA	64	0.55	1.07	0.06
DART	64	99.70	99.83	99.88
DART w/o SMA	128	0.23	0.08	0.08
DART	128	99.94	99.99	99.98

Table 6: Ablation of C_t sharing on MQAR with SSD state size $N = 16$ and chunk size $S = 16$. All entries are test accuracies in percent.

Model	d_{model}	$L = 256$	$L = 512$	$L = 1024$
DART, shared C_t	32	96.70	96.67	95.94
DART, independent C_t^{SMA}	32	0.00	0.00	0.00
DART, shared C_t	64	99.70	99.83	99.88
DART, independent C_t^{SMA}	64	0.00	0.00	0.00
DART, shared C_t	128	99.94	99.99	99.98
DART, independent C_t^{SMA}	128	0.00	0.00	0.00

Speed Benchmarks and Inference Cache. We evaluate the system efficiency of the SMA branch using speed benchmarks and inference cache size. For speed, we measure the forward time of the core sequence-mixing kernels under matched batch size, head dimension, state size, chunk size, and bfloat16 inputs on a single A800 80GB GPU. The comparison measures FlashAttention-2 with precomputed Q, K, V, SSD with precomputed X, Δ, A, B, C, and SMA with precomputed Q, E, C, chunk state memories $\{\Delta H_{[c]}\}$. For inference, we compare the persistent decode cache associated with these sequence modules: attention KV cache for FlashAttention-2, conv/recurrent state cache for SSD, and chunk state-memory cache for SMA. As discussed in Section 4, the ratio of the chunk state-memory cache to the matched attention KV cache is $N/(2S)$ per head, giving a 25% ratio when $N = 128$ and $S = 256$.

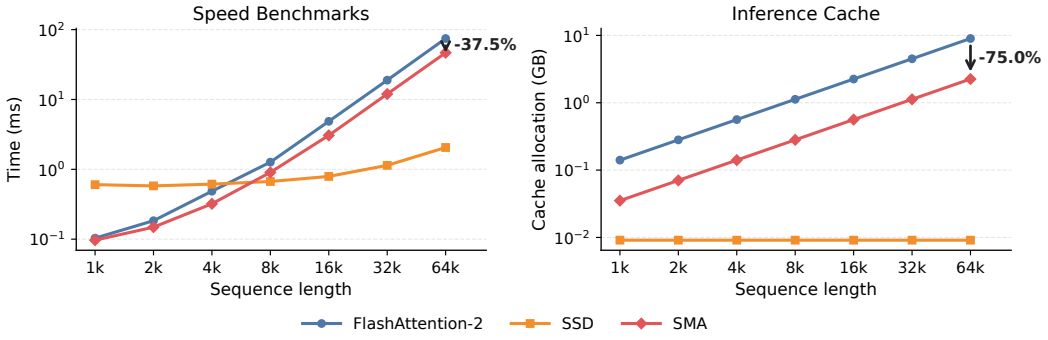

Figure 5: Speed benchmarks and inference cache comparison. **Left**: forward time of the kernels on an A800 80GB GPU with batch size 1, head dimension 64, SSD state size 128, chunk size 256, and bfloat16 inputs. **Right**: measured CUDA allocation for persistent inference cache tensors. The results show that SMA consistently has lower runtime than FlashAttention-2 in this setting, while also requiring substantially smaller persistent inference cache than token-level attention.

6 Conclusion

We presented DART, an architecture that augments Mamba-2 with state-memory attention over compact recurrent states. Mamba-2's chunkwise SSD computation produces chunk state contributions that DART retains as chunk state memories. SMA decodes token-conditioned keys and values from these memories and adds its readout as a gated correction to the SSM branch. Experiments on MQAR, Pile pretraining, downstream evaluation, retrieval benchmarks, and efficiency analysis show that DART improves associative recall and retrieval while preserving general language-modeling quality. These results demonstrate that recurrent compression and attention-style access can be supported by the same recurrent-state representation.

AI USE STATEMENT

Generative AI tools were used for language editing, literature discovery, experimental design, and interpretation of results. All AI-assisted suggestions and outputs were reviewed and verified by the authors. The authors take responsibility for the final content of the paper, including its text, claims, experiments, and artifacts.

REPRODUCIBILITY STATEMENT

The architecture, mathematical formulation, and computational analysis are described in Sections 3 and 4. The appendices provide a derivation of Mamba-2's chunkwise SSD computation, SMA implementation details, training configurations, and evaluation protocols.

REFERENCES

Yash Akhauri, Safeen Huda, and Mohamed S Abdelfattah. Attamba: Attending to multi-token states. *arXiv preprint arXiv:2411.17685*, 2024.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Y Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International conference on learning representations*, volume 2024, pp. 15664–15730, 2024a.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024b.

Simran Arora, Aman Timalsina, Aaryan Singhal, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*, 2024c.

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

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *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, 2019.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7B SSM hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

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

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in neural information processing systems*, 35:35971–35983, 2022.

Han Guo, Songlin Yang, Tarushii Goel, Eric P. Xing, Tri Dao, and Yoon Kim. Log-linear attention. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.04761.

Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 775–784, 2011.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Jia Fei, Zhang Yang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. *arXiv preprint arXiv:2603.15569*, 2026.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2-bit quantization for KV cache. *arXiv preprint arXiv:2402.02750*, 2024b.

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 *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

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 *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, et al. RWKV: Reinventing RNNs for the transformer era. In *Findings of the association for computational linguistics: EMNLP 2023*, pp. 14048–14077, 2023.

Yixiao Qian, Song Chen, Jiaxu Liu, Shengze Cai, and Chao Xu. DSSMs: State space models with explicit memory via delay differential equations. *arXiv preprint arXiv:2607.10244*, 2026.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

Liliang Ren, Yang Liu, Yadong Lu, Chen Liang, Weizhu Chen, et al. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *International Conference on Learning Representations*, volume 2025, pp. 53551–53575, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, et al. An empirical study of Mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

Xiuying Wei, Anunay Yadav, Razvan Pascanu, and Caglar Gulcehre. RAT: Bridging RNN efficiency and attention accuracy via chunk-based sequence modeling. *Advances in Neural Information Processing Systems*, 38:34735–34762, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A Mamba-2 Chunkwise SSD Computation

This appendix reviews the chunkwise SSD algorithm used by Mamba-2. Let $I_c = \{a_c, \dots, b_c\}$ denote the c-th chunk, and let $H_{[c]} = H_{b_c}$ be its chunk-boundary state. The token-level recurrence can be written as

$$H_t = A_t H_{t-1} + B_t^\top X_t, \qquad Y_t = C_t H_t. \tag{14}$$

For compactness, define

$$A_{u:v} = A_v A_{v-1} \cdots A_u = \prod_{k=u}^{v} A_k, \qquad u \le v, \tag{15}$$

and set $A_{u:v} = I$ when $u > v$. Thus a value written at position s reaches position t through $A_{s+1:t}$.

Chunkwise Computation. Using arrow notation to indicate that the decay has been absorbed toward the chunk boundary,

$$\overrightarrow{H}_{[c-1]} = A_{a_c:b_c} H_{[c-1]}, \qquad \overleftarrow{C}_t = C_t A_{a_c:t}, \tag{16}$$

the Mamba-2 chunkwise SSD computation produces the chunk-boundary state and decomposes each token output into intra- and inter-chunk terms:

$$H_{[c]} = \overrightarrow{H}_{[c-1]} + \Delta H_{[c]}, \tag{17}$$

$$Y_t = Y_t^{\text{intra}} + Y_t^{\text{inter}}, \qquad t \in I_c. \tag{18}$$

Here $\Delta H_{[c]}$ is the chunk state contribution to the chunk-boundary state:

$$\Delta H_{[c]} = \sum_{s \in I_c} A_{s+1:b_c} B_s^\top X_s. \tag{19}$$

The zero-initial-state output is

$$Y_t^{\text{intra}} = \sum_{s \in I_c,\, s \le t} C_t A_{s+1:t} B_s^\top X_s, \qquad t \in I_c. \tag{20}$$

The inter-chunk readout is

$$Y_t^{\text{inter}} = \overleftarrow{C}_t H_{[c-1]}. \tag{21}$$

Thus the sequence computation is reduced to chunk-local structured attention blocks, followed by inter-chunk state passing over the compact boundary states $\{H_{[c]}\}$. The matrices $\Delta H_{[c]}$ are the chunk state contributions retained by DART as chunk state memories.

Matrix Form. Stacking the tokens in a chunk gives the same decomposition in block form:

$$Y_{[c]} = \overleftarrow{C}_{[c]} H_{[c-1]} + \mathcal{L}_{[c]} X_{[c]}, \tag{22}$$

where the lower-triangular SSD block is

$$\mathcal{L}_{[c]}[t,s] = \begin{cases} C_t A_{s+1:t} B_s^\top, & s \le t, \quad s,t \in I_c, \\ 0, & s > t, \quad s,t \in I_c. \end{cases} \tag{23}$$

B FlashAttention Background

FlashAttention reduces the memory cost of exact attention by matching the attention computation to the GPU memory hierarchy (Dao et al., 2022; Dao, 2024). Instead of materializing the full score matrix in HBM, it streams tiles of keys and values through on-chip SRAM. Here a tile denotes a small contiguous block of query, key, or value vectors processed as a matrix block. For a fixed query tile Q_i, the kernel iterates over key-value tiles (K_j, V_j), computes the local scores $S_{ij} = Q_i K_j^\top$, updates the softmax statistics online, and accumulates the output tile.

Online Softmax. The key idea is to compute the stable softmax while reading a vector in pieces. For a score vector x, let $m = \max_k x_k$. The stable softmax is

$$\mathrm{softmax}(x)_j = \frac{e^{x_j - m}}{\sum_k e^{x_k - m}}. \tag{24}$$

Suppose the entries of x are processed one by one. After seeing $x_1, \ldots, x_{j-1}$, online softmax maintains

$$m^{(j-1)} = \max_{k<j} x_k, \qquad \ell^{(j-1)} = \sum_{k<j} e^{x_k - m^{(j-1)}}, \tag{25}$$

with $m^{(0)} = -\infty$ and $\ell^{(0)} = 0$. Here $m^{(j-1)}$ records the largest score seen so far, and $\ell^{(j-1)}$ records the softmax denominator under the current maximum. After seeing a new entry x_j, the statistics are updated as

$$m^{(j)} = \max\left(m^{(j-1)}, x_j\right), \qquad \ell^{(j)} = e^{m^{(j-1)} - m^{(j)}} \ell^{(j-1)} + e^{x_j - m^{(j)}}. \tag{26}$$

When softmax is used to form a weighted value sum, the numerator accumulator is rescaled by the same factor and then receives the new value contribution:

$$O^{(j)} = e^{m^{(j-1)} - m^{(j)}} O^{(j-1)} + e^{x_j - m^{(j)}} v_j. \tag{27}$$

After all entries have been processed, the output is O/ℓ. FlashAttention applies this same procedure to every query row while streaming key-value tiles.

Recomputation. FlashAttention uses recomputation to avoid storing attention score and probability tiles. During the forward pass, the kernel computes each local score tile $S_{ij} = Q_i K_j^\top$ on chip, updates the online softmax statistics, accumulates the output, and stores only the output together with one logsumexp value per query row. During the backward pass, the same local score tile $S_{ij} = Q_i K_j^\top$ is recomputed from Q_i and K_j, and the stored logsumexp is used to recover the corresponding softmax probabilities.

Concretely, for a score vector x, let $m = \max_j x_j$. Its logsumexp can be computed in the stable form

$$\mathrm{LSE}(x) = m + \log \sum_j e^{x_j - m}. \tag{28}$$

The softmax probability can then be written as

$$\mathrm{softmax}(x)_j = \frac{e^{x_j - m}}{\sum_k e^{x_k - m}} = e^{x_j - \mathrm{LSE}(x)}. \tag{29}$$

Thus recovering the softmax probabilities only requires the recomputed score vector and its logsumexp. The online softmax statistics provide this quantity because $\ell = \sum_j e^{x_j - m}$ is the stable softmax denominator, and

$$L_{\mathrm{lse}} = m + \log \ell. \tag{30}$$

For a recomputed score tile S_{ij}, the probability tile is recovered as

$$P_{ij} = \exp(S_{ij} - L_{\mathrm{lse},i}). \tag{31}$$

This trades additional arithmetic for lower activation memory, because the attention probabilities and intermediate score tiles do not need to be stored in HBM.

C Method Details

C.1 Normalization in DART

In the implemented architecture, the state-memory attention uses a normalized score branch:

$$\bar{Q}_t = \mathrm{RMSNorm}(Q_t), \qquad \bar{E}_t = \mathrm{RMSNorm}(E_t). \tag{32}$$

For each chunk state memory, define

$$r_{c,n} = \left(\frac{1}{P} \sum_{p=1}^{P} (\Delta H_{[c]})^2_{n,p} + \epsilon \right)^{-1/2}, \qquad K_{t,c} = r_c \odot (\Delta H_{[c]} \bar{E}_t), \tag{33}$$

where $r_c \in \mathbb{R}^{N \times 1}$ is the inverse RMS of $\Delta H_{[c]}$ along the P axis. This stabilizes the score branch without materializing a normalized copy of $\Delta H_{[c]}$. The normalized chunk logit and SMA readout are

$$\bar{\ell}_{t,c} = \frac{\bar{Q}_t K_{t,c}}{\sqrt{N}}, \qquad g_{t,c} = \mathrm{softmax}_{c<c(t)}(\bar{\ell}_{t,c}), \qquad \bar{R}_t = \sum_{c<c(t)} g_{t,c} V_{t,c}. \tag{34}$$

C.2 FlashAttention-Style SMA Kernel

Algorithm 1 gives the tiled forward procedure. For clarity, it omits the RMS normalization above. The kernel writes only the SMA readout and logsumexp statistics to HBM; keys, values, logits, and attention weights are formed on chip and recomputed during the backward pass.

C.3 Multi-Value Details

Let $\mathcal{B}$ denote the batch size, L the sequence length, H the number of heads, P the value dimension, N the SSD state size, $M = \lceil L/S \rceil$ the number of chunks, and G the number of grouped projections. In the current experiments, DART uses grouped B, C, Q, and E with $G = 1$.

Table 7: Compact tensor shapes used by DART with MVA-style grouped projections.

Tensor	Shape
X	$\mathcal{B} \times L \times H \times P$
ΔH	$\mathcal{B} \times M \times H \times N \times P$
SSM B, C, Q	$\mathcal{B} \times L \times G \times N$
E	$\mathcal{B} \times L \times G \times P$

D Experiment Configuration Details

D.1 Synthetic Tasks: Associative Recall

Following the harder MQAR variant used in Mamba-2 (Dao & Gu, 2024), non-special positions are filled with random vocabulary tokens rather than zeros. This prevents the model from relying on a sparse blank background and makes the task closer to a dense language-modeling sequence.

Algorithm 1 FlashAttention-Style State-Memory Attention

Input: Matrices $Q, C \in \mathbb{R}^{L\times N}$, $E \in \mathbb{R}^{P\times L}$, chunk state memories $\Delta H_{[1]}, \ldots, \Delta H_{[M]} \in \mathbb{R}^{N\times P}$ in HBM, chunk size S, token tile size T.
Output: SMA readout $R \in \mathbb{R}^{L\times P}$ and logsumexp $L_{\text{lse}} \in \mathbb{R}^{L}$.

Divide the sequence into $M = \lceil L/S \rceil$ chunks; divide each chunk into token tiles of size T.
for target chunk $i = 1, \ldots, M$ **do**
  for token tile τ inside chunk i **do**
    Load $Q_\tau, C_\tau \in \mathbb{R}^{T\times N}$ and $E_\tau \in \mathbb{R}^{P\times T}$ from HBM to on-chip SRAM.
    On chip, initialize $O_\tau^{(0)} = 0 \in \mathbb{R}^{T\times P}$, $d_\tau^{(0)} = 0 \in \mathbb{R}^{T}$, and $m_\tau^{(0)} = -\infty \in \mathbb{R}^{T}$.
    for historical chunk $j < i$ **do**
      Load $\Delta H_{[j]}$ from HBM to on-chip SRAM.
      On chip, compute $K_\tau^{(j)} = (\Delta H_{[j]} E_\tau)^\top \in \mathbb{R}^{T\times N}$.
      On chip, compute $\ell_\tau^{(j)} = \mathrm{rowsum}(Q_\tau \odot K_\tau^{(j)})/\sqrt{N} \in \mathbb{R}^{T}$.
      On chip, compute $V_\tau^{(j)} = C_\tau \Delta H_{[j]} \in \mathbb{R}^{T\times P}$.
      On chip, compute $m_\tau^{(j)} = \max(m_\tau^{(j-1)}, \ell_\tau^{(j)})$ and $\widetilde{g}_\tau^{(j)} = \exp(\ell_\tau^{(j)} - m_\tau^{(j)})$.
      On chip, update $d_\tau^{(j)} = e^{m_\tau^{(j-1)} - m_\tau^{(j)}} d_\tau^{(j-1)} + \widetilde{g}_\tau^{(j)}$.
      On chip, update $O_\tau^{(j)} = \mathrm{diag}(e^{m_\tau^{(j-1)} - m_\tau^{(j)}}) O_\tau^{(j-1)} + \mathrm{diag}(\widetilde{g}_\tau^{(j)}) V_\tau^{(j)}$.
    end for
    On chip, compute $R_\tau = \mathrm{diag}(d_\tau^{(i-1)})^{-1} O_\tau^{(i-1)}$, keeping $R_\tau = 0$ when no historical chunk exists.
    On chip, compute $(L_{\text{lse}})_\tau = m_\tau^{(i-1)} + \log d_\tau^{(i-1)}$, keeping $(L_{\text{lse}})_\tau = -\infty$ when no historical chunk exists.
    Write R_τ and $(L_{\text{lse}})_\tau$ to HBM.
  end for
end for
Return R and L_{lse}.

Task construction. For a sequence of length L, let R be the number of key–value pairs. Each example samples R distinct keys $k_1, \ldots, k_R$ from the key vocabulary and R distinct values $v_1, \ldots, v_R$ from a disjoint value vocabulary. The first $2R$ positions contain the source associations:

$$x_{2i-1} = k_i, \qquad x_{2i} = v_i, \qquad i = 1, \ldots, R. \tag{35}$$

The remaining sequence provides alternating candidate query slots at positions $\{2R+1, 2R+3, \ldots, L-1\}$. We choose R distinct slots and insert a randomly permuted copy of the source keys into them. At a query position containing key k_i, the target is its associated value v_i. Training and evaluation use causal model forward passes with supervised labels only at query positions: the model predicts the associated value from the prefix ending at the query key, while all other labels are ignored.

Lengths and number of pairs. We evaluate sequence lengths

$$L \in \{256, 512, 1024\}, \tag{36}$$

and use $R = L/4$ key–value pairs at evaluation time. Since the number of query slots is also $L/4$, all query slots are occupied in this setting.

Optimization. The training set contains 2^{18} examples per run. Validation and test examples use the full evaluation setting $R = L/4$, while training uses a four-stage pair-count curriculum $R \in \{L/16, L/8, 3L/16, L/4\}$ with 8 epochs per stage. Thus each run uses 32 epochs in total. The effective batch size is chosen so that each optimization step contains $2^{18} \approx 0.25M$ tokens: 1024 sequences for $L = 256$, 512 sequences for $L = 512$, and 256 sequences for $L = 1024$. We train with AdamW, gradient clipping at norm 1.0, and a linear learning-rate decay over the full training run. For each model configuration, we run peak learning rates 10^{-3} and 3×10^{-4} and report the run with the better validation accuracy.

Model settings. All MQAR models use two layers and untied input and output embeddings. DART and Mamba-2 use chunk size 16. We sweep sequence length, SSD state size, and model width:

$$L \in \{256, 512, 1024\}, \qquad N \in \{16, 64\}, \qquad d_{\text{model}} \in \{32, 64, 128\}. \tag{37}$$

The Transformer++ attention baseline uses the same (L, d_{model}) grid.

D.2 Pile Pretraining

We pretrain all language models on the Pile for approximately $100B$ tokens with context length 2048. The training batch is fixed to 1024 sequences per optimizer step. We train with AdamW, $\beta = (0.9, 0.95)$, weight decay 0.1, gradient clipping at norm 1.0, bfloat16 mixed precision, and a cosine learning-rate schedule with 1% warmup and minimum learning-rate ratio 0.1.

Table 8: Pile pretraining settings. DART uses the same Mamba-2 backbone shape as the corresponding Mamba-2 baseline and adds the SMA branch. Attamba uses chunk-level attention with independent SSM encoders for the key and value SSM branches.

Model	Parameters	Layers	Hidden size	State size	Chunk size	Peak LR
Mamba-2 130M	128.99M	24	768	128	256	6.0×10^{-4}
DART 133M	132.70M	24	768	128	256	6.0×10^{-4}
Attamba 135M	134.57M	24	512	32	4	6.0×10^{-4}
Mamba-2 370M	368.35M	48	1024	128	256	3.0×10^{-4}
DART 380M	378.22M	48	1024	128	256	3.0×10^{-4}
Attamba 371M	371.21M	48	640	32	4	3.0×10^{-4}
Mamba-2 780M	780.16M	48	1536	128	256	2.5×10^{-4}
DART 795M	794.95M	48	1536	128	256	2.5×10^{-4}
Attamba 781M	781.39M	48	960	32	4	2.5×10^{-4}

All Mamba-2 and DART models use head dimension 64, one shared B, C group, convolution width 4, and RMSNorm. DART uses chunk size $S = 256$ during pretraining; different inference chunk sizes are evaluated from the same trained model in the downstream and retrieval experiments. In all language-modeling experiments, DART uses grouped Q, B, C, and E projections and a scalar SMA residual gate per token for parameter efficiency. Attamba uses 64-dimensional attention heads with rotary embeddings, chunk size 4, a 32-token local window of uncompressed recent tokens, and cyclically shifted chunk boundaries. Its key and value representations for earlier context are produced by independent Mamba-2 SSM branches with state size 32, SSM head dimension 32, and one SSM group.

We include public Pythia checkpoints (Biderman et al., 2023) as full-attention references. To match our 100B-token training budget, we evaluate the publicly released Pythia checkpoints corresponding to approximately 100B pretraining tokens. These models follow the GPT-NeoX Transformer architecture with rotary position embeddings, rotary fraction 0.25, and native context length 2048. The official architecture settings are summarized in Table 9.

Table 9: Pythia reference model settings from the official configurations. All models use full attention with RoPE and native context length 2048.

Model	Layers	Hidden size	Attention heads	Head dim.	Context length
Pythia 160M	12	768	12	64	2048
Pythia 410M	24	1024	16	64	2048
Pythia 1B	16	2048	8	256	2048

D.3 Downstream Evaluation Tasks

Evaluation protocol. All downstream and retrieval evaluations are zero-shot. We use the standard multiple-choice or completion formulation for the general language-modeling tasks. For the retrieval suite, SWDE, SQuAD Completion, and FDA use the completion-format evaluation from the Based benchmark, while TriviaQA, Natural Questions, and DROP are evaluated in the cloze-style format. These retrieval tasks are scored by contains accuracy: an example is counted as correct if the generated response contains any accepted answer string. For NIAH, we use the RULER single-needle

generators with 500 examples per context length and report accuracy separately at 1024, 2048, and 4096 tokens. All generation-based retrieval tasks are evaluated with greedy decoding. The real-world retrieval tasks use maximum context length 2048 and generate up to 48 tokens, while NIAH uses maximum context length 4096 and generates up to 128 tokens. Attamba is evaluated at 4096 tokens with its pretrained rotary embedding configuration and no additional RoPE scaling. For the 4096-token NIAH setting, manual inspection of Attamba and Pythia generation logs shows that failed outputs often copy parts of the prompt template, nearby instructions, or local context fragments, rather than generating the target number or UUID. These outputs may restate that the special value appears in the provided context or repeat filler sentences from the haystack. This behavior accounts for the zero contains-accuracy entries of Attamba and Pythia at 4096 tokens in Table 3.

General Language Modeling Tasks.

- **Pile.** The Pile (Gao et al., 2020) is a large-scale English corpus that combines web text, books, code, academic papers, and other written sources. We report language-modeling perplexity to measure general next-token prediction quality on this broad text distribution.
- **LAMBADA.** LAMBADA (Paperno et al., 2016) evaluates long-context word prediction by asking the model to infer the final word of a passage, with results reported as both perplexity and exact-match accuracy.
- **HellaSwag.** HellaSwag (Zellers et al., 2019) is a commonsense sentence-completion benchmark where the model selects the most plausible continuation among adversarially constructed choices.
- **PIQA.** PIQA (Bisk et al., 2020) measures physical commonsense reasoning by asking the model to choose the more plausible solution for everyday situations.
- **ARC-Easy.** ARC-Easy (Clark et al., 2018) evaluates multiple-choice elementary science questions that are usually solvable with relatively direct factual knowledge.
- **ARC-Challenge.** ARC-Challenge (Clark et al., 2018) uses the harder subset of multiple-choice science questions that more often require multi-step reasoning or stronger commonsense knowledge.
- **WinoGrande.** WinoGrande (Sakaguchi et al., 2021) evaluates pronoun and coreference resolution through commonsense sentence pairs designed to reduce simple lexical shortcuts.
- **OpenBookQA.** OpenBookQA (Mihaylov et al., 2018) tests open-book-style elementary science reasoning by combining a short scientific fact with commonsense inference over multiple-choice answers.

Retrieval Tasks. Following the retrieval suite adopted by Mamba-3 (Lahoti et al., 2026), we include real-world retrieval tasks from the evaluation suite of Based and Just Read Twice (Arora et al., 2024b;c), together with synthetic needle-in-a-haystack tasks from RULER (Hsieh et al., 2024).

- **SWDE.** SWDE (Hao et al., 2011) evaluates semi-structured web information extraction by asking the model to recover an attribute value from an HTML page or page-derived text.
- **SQuAD Completion.** SQuAD Completion (Rajpurkar et al., 2018; Arora et al., 2024b) converts extractive reading comprehension into a cloze-style completion task where the model generates the answer span from the provided passage and query context.
- **FDA.** FDA (Arora et al., 2024b) evaluates information extraction from regulatory documents by asking the model to generate the value associated with a requested key in a document chunk.
- **TriviaQA.** TriviaQA (Joshi et al., 2017) measures open-domain factual retrieval by asking the model to generate a short answer to a trivia question.
- **Natural Questions.** Natural Questions (Kwiatkowski et al., 2019) evaluates open-domain question answering over naturally occurring search queries with short answer targets.
- **DROP.** DROP (Dua et al., 2019) tests paragraph-level retrieval and discrete reasoning by asking the model to answer questions whose evidence is located in a passage.

- **NIAH-Single-1.** NIAH-Single-1 is a synthetic single-needle retrieval task that places a word-to-number fact in a repeated-text haystack and asks the model to recover the number.
- **NIAH-Single-2.** NIAH-Single-2 is a synthetic single-needle retrieval task that places a word-to-number fact in an essay-style haystack and asks the model to recover the number.
- **NIAH-Single-3.** NIAH-Single-3 is a synthetic single-needle retrieval task that places a word-to-UUID fact in an essay-style haystack and asks the model to recover the UUID.