STATEBRIDGE: STATE-TRANSITION SHORTCUTS FOR CONTEXT REUSE IN SSM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

State-space sequence models (SSMs) have emerged as a promising alternative to Transformers for efficient long-context language modeling, compressing context into a fixed-size recurrent state. However, unlike Transformers, which have well-established solutions for reusing precomputed KV caches of repeated text blocks across requests, SSMs lack any analogous mechanism. The reason is structural: Transformers handle context in an append-based manner that naturally supports modular reuse, whereas SSMs operate in an in-place manner where each block is fused into the recurrent state inseparably. This suggests a natural question: can we learn to approximate the effect of a fixed block on a different recurrent state, without actually prefilling it? In this paper, STATEBRIDGE answers this affirmatively, learning a compact input matrix for each reusable context block that approximates its in-place state update for different preceding contexts. At serving time, SSM scans this learned input matrix from the live recurrent state, applying a shared transition shortcut in place of the raw block to approximate the original behavior. Across four benchmarks, STATEBRIDGE retains 88.69%–101.07% of performance and achieves up to $2.71\times$ speedup in time to first token (TTFT).

1 INTRODUCTION

State-space sequence models (SSMs) are efficient alternatives to Transformers for long-context language modeling (Gu & Dao, 2024; Dao & Gu, 2024; Lahoti et al., 2026). At serving time, an SSM scans n prompt tokens with $O(n)$ recurrent updates and stores the result as a fixed-size recurrent state. This fixed-size state keeps state storage independent of context length, while prefill still requires one recurrent update for each prompt token. Repeatedly performing these updates becomes costly when many requests share the same fixed text. Shared-prefix caching avoids this repeated computation for identical prefixes, but shared text following different prefixes still needs to be processed for each request. For example, different dialogue histories or user queries may precede the same instruction, document, or task template. Avoiding this repeated prefill is the problem of *non-prefix context reuse* (Yao et al., 2025; Hu et al., 2025). We refer to the shared fixed text as a *reusable context block*. A context reuse system must precompute a representation of the reusable context block before requests arrive and reuse it after processing each request-specific prefix.

Existing context reuse solutions mainly address this serving problem for Transformers, where the reusable object is the KV cache of the context block. Because the block's KV cache depends on the preceding prefix, these systems selectively recompute or adapt its prefix-dependent KV entries when appending the precomputed block KV cache to the current request's prefix cache (Yao et al., 2025; Hu et al., 2025). These adjustments allow the precomputed block KV cache to be reused across different prefixes while leaving the prefix KV cache unchanged.

However, SSM serving requires a different reusable representation. After scanning a request-specific prefix, the model enters the reusable context block with a prefix-dependent recurrent state. Scanning the block updates this live state in place, producing a post-block state that depends on the incoming prefix state. Thus, a post-block state cached for one prefix cannot generally be reused directly for a different prefix. The reusable representation must instead capture the block-induced transition: a mapping from the incoming recurrent state to the corresponding post-block state. Figure 1 summarizes this contrast. Transformer reuse can append a precomputed block representation to the prefix cache, whereas SSM reuse needs an executable representation of this block-induced transition.

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.

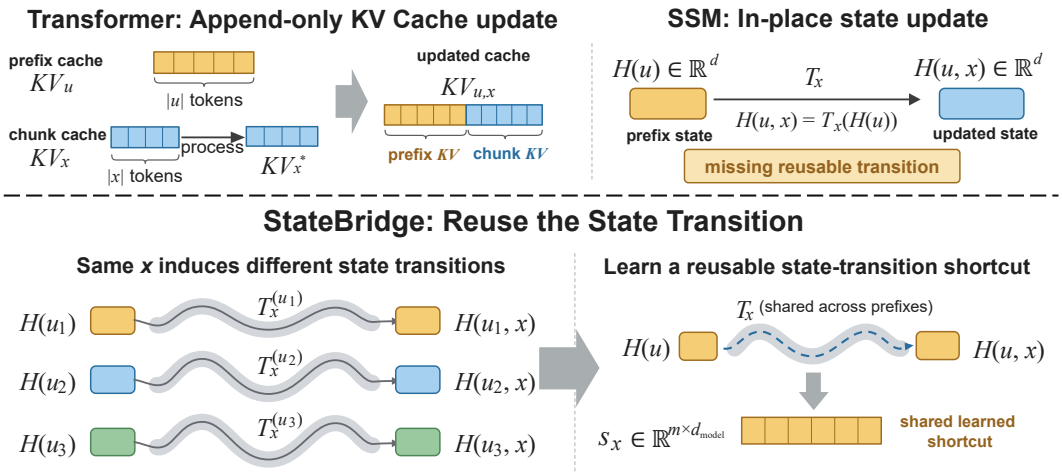

Figure 1: Context reuse differs between Transformer and SSM serving: Transformers append block KV entries to the prefix cache, whereas SSMs update the recurrent state in place. StateBridge makes the block-induced state transition reusable across prefixes by learning a compact input matrix s_x, whose short scan from the current prefix state approximates the full block's state update.

To make this block-induced transition reusable, we propose StateBridge, which learns a compact input matrix for each reusable context block. The matrix encodes a reusable state-transition shortcut, implemented as a short sequence of learned input vectors scanned by the SSM. Before serving, StateBridge optimizes the matrix across several incoming prefix states, so that executing the matrix from each state approximates the recurrent state produced by scanning the original block. During serving, the model scans the request prefix normally and then scans this shorter matrix from the live recurrent state, allowing one stored matrix to replace the raw block scan across different prefixes. Our experiments show that StateBridge preserves downstream QA quality across varying request contexts while reducing TTFT. Our contributions are as follows.

- We identify a key mismatch between non-prefix context reuse in Transformer and SSM serving, formalizing the reusable object for SSMs as the block-induced state transition that maps each live prefix state to the state obtained after scanning the block.
- We introduce StateBridge for SSM context reuse. StateBridge learns a compact surrogate for each reusable context block, allowing the SSM to approximate the block transition from each request's current recurrent state with a much shorter scan.
- We evaluate StateBridge on four QA benchmarks with Mamba backbones. StateBridge retains 88.69%–101.07% of raw-document F1 and achieves TTFT speedups of up to 2.71×.

2 Background and Related Work

State-space model inference. State-space sequence models (SSMs) process a prompt by scanning its tokens through recurrent state updates rather than storing a per-token KV cache (Gu et al., 2022; Gu & Dao, 2024; Dao & Gu, 2024). For a token position t, let x_t denote the current input vector and let h_{t-1} denote the recurrent state before that input is processed. At one SSM layer, a discrete recurrent update can be written abstractly as

$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t, \qquad y_t = C_t h_t, \tag{1}$$

where h_t is the updated recurrent state, y_t is the layer output, and $\bar{A}_t$, $\bar{B}_t$, and C_t are the input-conditioned state, input, and output matrices at position t. Figure 2 shows how Mamba realizes this

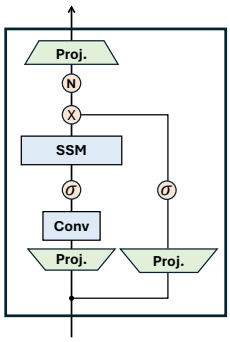

Figure 2: A Mamba Block.

update (Gu & Dao, 2024; Dao & Gu, 2024). An input projection splits into recurrent and gating branches; the recurrent branch applies local convolution and a selective SSM scan, while the gating branch modulates the scanned output before the final projection. During serving, the convolutional and SSM states are carried across tokens as the current cache. Section 3 next asks how a reusable context block should update that cache.

Transformer context reuse and compression. Transformer context reuse is the closest serving setting because it also reuses a representation that full prompt prefill would have produced. Within Transformer serving, exact-prefix systems such as SGLang cache shared initial tokens and resume requests from that cached prefix; they do not match our setting, where the repeated block appears after a request-specific prefix (Zheng et al., 2024). Non-prefix systems such as CacheBlend and EPIC are directly related because they reuse retrieved or repeated blocks by placing a precomputed block KV cache after the prefix cache and selectively recomputing or adapting prefix-dependent parts (Yao et al., 2025; Hu et al., 2025). Their reuse operation appends block KV entries while keeping the prefix KV cache unchanged, whereas STATEBRIDGE represents the block as an executable in-place update of the current recurrent state. Text compression and learned-memory methods such as LLMLingua, Gist Tokens, and AutoCompressors address the same pressure to reduce long-context work, but they optimize shortened prompts, compressed prompt tokens, or compact memories for downstream behavior (Jiang et al., 2023; Mu et al., 2023; Chevalier et al., 2023). These compression methods need not reproduce the intermediate KV cache or recurrent state produced by full prefill over the original block; STATEBRIDGE instead learns a compact input matrix whose scan approximates the cache produced by scanning the block after different prefixes.

State-based methods for SSMs. SSM and hybrid-SSM serving work is closest on architecture because it acts on recurrent state rather than appendable Transformer KV entries. Marconi studies hybrid LLM prefix caching for models whose recurrent layers update state in place, but its reusable object is still tied to exact shared prefixes rather than to a repeated block after different prefixes (Pan et al., 2025). PICASO preprocesses retrieved contexts into recurrent states and composes those states to approximate raw context concatenation, while Document Souping asks whether document states can be pooled for SSM document conditioning (Liu et al., 2025; Jafari et al., 2025). These state-composition methods show that SSM states can serve as reusable interfaces, which is relevant because our serving problem also acts on the current recurrent state. The missing operation is the update of that current recurrent state, since a state saved after one context or a pool of states does not describe how a fixed block updates an arbitrary prefix state in place. STATEBRIDGE stores a learned input matrix for each reusable context block and lets the SSM scan that matrix from the current prefix state, so the stored object approximates the block-induced transition and tests whether that in-place update has exploitable regularity across prefixes.

3 MOTIVATION

Two observations motivate STATEBRIDGE. First, SSM context reuse cannot simply reuse a cache saved after scanning some prefix and the reusable block; instead, it requires approximating the block-induced transition from the current prefix state. Second, when the prefix is held fixed, a short learned scan can approximate this update in a probing experiment.

Observation 1: Reuse must transform the current prefix state, not replace it. When the same reusable context block x follows different prefixes, a state saved after scanning one prefix and the block cannot be reused after another prefix. If $H(a)$ denotes the whole-model recurrent state after scanning a token sequence a, then $H(u_0, x)$ contains information from both the prefix u_0 and the block x. Using this state after another prefix u would discard the current state $H(u)$, rather than apply the block to it. The reusable object must therefore be the effect of the block on the current state: scanning x should map $H(u)$ to $H(u, x)$.

STATEBRIDGE stores a compact learned input matrix $s_x = (s_1, \ldots, s_m) \in \mathbb{R}^{m \times d_{\text{model}}}$ for a reusable block $x : \{x_1, \ldots, x_L\}$, with $m \ll L$. The rows s_i are optimized continuous inputs scanned by the SSM, and thus induce their own cache update. For a token sequence or learned input matrix z, write $\Phi_z(H(u)) = H(u, z)$ for scanning z from the current cache $H(u)$. The target is to make this induced

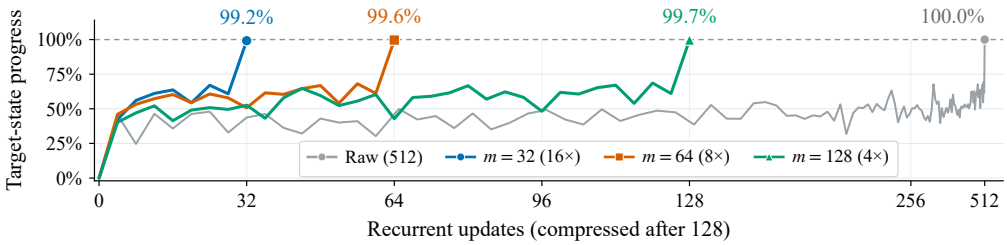

Figure 3: Projected state progress for a 512-token block with one fixed prefix.

update reach approximately the same cache as scanning the original block:

$$\Phi_{s_x}(H(u)) = H(u, s_x) \approx \Phi_x(H(u)) = H(u, x). \tag{2}$$

If this target is achievable, reuse can replace L raw recurrent updates with m learned recurrent updates while preserving the cache from which downstream tokens continue. The remaining question is whether such a short learned update exists even in the simplest setting with one fixed prefix.

Observation 2: Short learned scans can approach raw replay after one fixed prefix.

We test the easiest version of the target: one block, one fixed prefix, and one learned matrix optimized only for that case. For block lengths $L \in \{512, 1024, 2048, 4096\}$, we sample a prefix u, hold it fixed, and optimize s_x so that scanning u followed by s_x approaches the cache $H(u, x)$ obtained by scanning the original block. The loss is the normalized squared ℓ_2 cache distance d, averaged over recurrent-cache tensors. Table 1 reports $\mathrm{StateMatch} = 100(1-d)$, where higher values indicate closer agreement with raw replay.

Table 1: Mean state match (%) with one fixed prefix, averaged over Mamba-2 1.3B/2.7B.

Raw length $\lvert x\rvert$	$m = 16$	$m = 32$	$m = 64$	$m = 128$
512	96.1%	97.5%	98.5%	99.2%
1024	92.4%	95.8%	97.7%	98.8%
2048	93.1%	95.8%	97.7%	98.6%
4096	74.4%	83.7%	89.3%	92.7%

This one-prefix result supports the core possibility. With 128 learned rows, StateMatch reaches 92.7–99.2% across 512–4096-token blocks, and every block length improves as m increases. Figure 3 complements these endpoint measurements by visualizing projected state progress: for a 512-token block, 32 learned updates already move close to the cache reached by replaying all 512 raw tokens. However, this probe still does not solve serving, because it trains a different matrix for each sampled prefix and block. Serving requires a single surrogate for x that works after different incoming prefixes, so training must expose the same s_x to multiple prefix states. Section 4.3 turns this requirement into a multi-prefix transition-matching objective, and Section 5 evaluates its performance in full task settings.

4 METHOD

This section presents STATEBRIDGE, which learns shortcuts for the state transitions induced by reusable context blocks. The key idea is to approximate the combined effect of a block's recurrent updates with fewer steps, using a compact input matrix scanned by the frozen SSM. The learned matrix is shared across requests containing the same block and is scanned from each request's current prefix state. We first formalize the reuse problem, then describe how the shortcut is represented, learned offline, and applied at serving time, following the workflow illustrated in Figure 4.

4.1 PROBLEM FORMULATION

Consider requests that share a reusable context block but arrive with different prefixes and suffixes. We write one request as (u, x, p), where u is the request-specific prefix, x is the reusable context block, and p is the post-block prompt segment. A reuse system must prepare an object for x before a request arrives and then execute that object after the SSM has scanned the live prefix.

Using the state and transition notation introduced in Section 3, raw prefill for this request produces

$$H(u, x, p) = \Phi_p(\Phi_x(H(u)))\,. \tag{3}$$

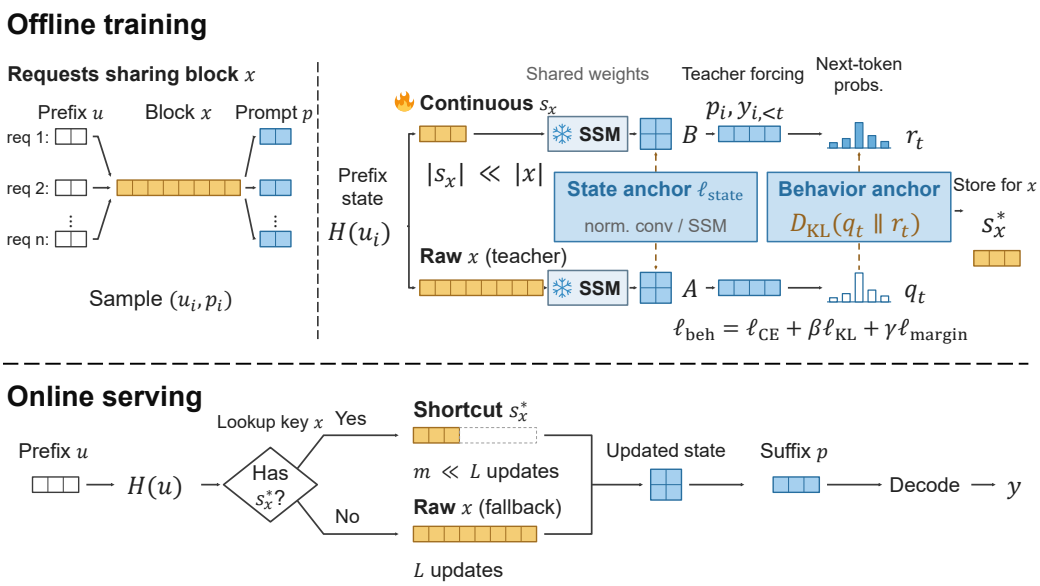

Figure 4: Overview of StateBridge. StateBridge learns a compact surrogate s_x that approximates the state transition induced by a reusable context block x. Offline training (top) uses different prefixes to support reuse across incoming states, with complementary supervision of state fidelity and downstream response behavior. Online serving (bottom) applies the learned surrogate $s_x^\star$ to each request's current prefix state $H(u)$, approximating the block's effect with fewer recurrent updates.

In Mamba-style models, H is materialized by the layerwise `conv_state` and `ssm_state` tensors carried forward during serving. These tensors are the quantities compared by the training loss below.

A state such as $H(u_0, x)$ is a cached endpoint, not the block update itself. It has already absorbed the particular prefix state $H(u_0)$, so using it after another prefix would replace, rather than update, the live state $H(u)$. StateBridge therefore learns a short executable object whose scan approximates Φ_x from the prefix states observed for the same block.

4.2 Executable Surrogate for the Block Transition

StateBridge uses a learned input matrix as the stored object for x. For a block $x = (\tau_1, \ldots, \tau_L)$, the stored surrogate is

$$s_x = (s_1, \ldots, s_m) \in \mathbb{R}^{m \times d_{\text{model}}}, \qquad m \ll L. \tag{4}$$

Each row is a continuous input supplied at the embedding interface. Extending H from token sequences to these input rows, executing the surrogate after prefix u means

$$\Phi_{s_x}(H(u)) = H(u, s_1, \ldots, s_m) \approx \Phi_x(H(u)). \tag{5}$$

The matrix bypasses vocabulary lookup, but the SSM processes its rows with the same recurrent updates, layers, state tensors, and decoder interface used for ordinary token embeddings. This constraint makes the surrogate executable by the deployed model. Replacing x with s_x changes the block scan from L recurrent updates to m recurrent updates, while the suffix and decoder continue from an ordinary frozen-model state.

4.3 Offline Training with Dual Anchors

StateBridge learns a single matrix s_x for each reusable block x under multiple prefixes, using the frozen model's raw-document path as the teacher. Training combines a document-end state anchor with a response-behavior anchor. For training pairs $\{(u_i, p_i)\}_{i=1}^n$, p_i contains the question and answer prompt, and y_i denotes the teacher answer generated from (u_i, x, p_i).

The state anchor compares the raw and shortcut states $A = \Phi_x(H(u))$ and $B = \Phi_{s_x}(H(u))$, obtained from the same prefix state $H(u)$ before processing p_i:

$$\ell_{\text{state}}(u; s_x) = \frac{1}{N} \sum_{\ell=1}^{N} \sum_{a \in \{\text{conv,ssm}\}} \frac{\|A_{\ell,a} - B_{\ell,a}\|_F^2}{\max\{\|A_{\ell,a}\|_F^2, \varepsilon\}}, \tag{6}$$

where N is the number of layers, a indexes the `conv_state` and `ssm_state` tensors, and $\varepsilon > 0$ stabilizes the normalization.

The behavior anchor complements state matching by supervising the shortcut path's predictions over the full teacher answer. For a prefix u, the raw and shortcut paths process (u, x, p_i) and (u, s_x, p_i), respectively, and are teacher-forced with the same answer history $y_{i,<t}$. Let q_t and r_t denote their next-token distributions. The behavior loss is

$$\ell_{\text{beh}}(u, p_i, y_i; s_x) = \ell_{\text{CE}} + \beta \ell_{\text{KL}} + \gamma \ell_{\text{margin}}, \tag{7}$$

where the terms are cross-entropy against y_i, KL divergence $D_{\text{KL}}(q_t \| r_t)$, and a logit-margin penalty, all averaged over answer positions. The margin term encourages the shortcut path's logit for $y_{i,t}$ to exceed the largest competing logit by a prescribed margin.

Training proceeds in two stages. We first optimize $\ell_{\text{beh}}(u_i, p_i, y_i; s_x)$ on the original requests to obtain an initial shortcut. In Stage B, we replace the original prefix u_i with an augmented prefix $\tilde{u}$, keeping the document x and suffix p_i unchanged. This exposes the same shortcut to additional incoming states $H(\tilde{u})$, encouraging it to approximate the block-induced transition from different starting states. The teacher answers y_i remain fixed, but we recompute teacher distributions under $(\tilde{u}, x, p_i)$ using the same answer history $y_{i,<t}$.

Refinement uses three supervision terms: the original-prefix behavior loss $\ell_{\text{beh}}(u_i, p_i, y_i; s_x)$, the augmented-prefix behavior loss $\ell_{\text{beh}}(\tilde{u}, p_i, y_i; s_x)$, and the augmented-prefix state loss $\ell_{\text{state}}(\tilde{u}; s_x)$. We compute these three gradients separately, rescale the two augmented-prefix gradients relative to the norm of the original-prefix behavior gradient, and combine them for each update. Both stages backpropagate through the frozen model and update only s_x. The resulting matrix $s_x^\star$ is stored for online serving. Held-out evaluation prompts are excluded from training.

4.4 Online Serving

Online serving keeps the original text block as the external request object and uses the surrogate only inside the SSM scan. Given (u, x, p), the prefill state is

$$H_{\text{serve}}(u, x, p) = \begin{cases} \Phi_p\big(\Phi_{s_x^\star}(H(u))\big), & \text{if a trained surrogate for } x \text{ is stored,} \\ \Phi_p(\Phi_x(H(u))), & \text{otherwise.} \end{cases} \tag{8}$$

If no trained surrogate is available, Eq. (8) reduces to raw prefill over (u, x, p). If $s_x^\star$ is available, serving scans the prefix normally to form $H(u)$, executes $s_x^\star$ from that state in place of x, and then scans the unchanged suffix p. The block portion of prefill changes from $|x|$ recurrent updates to m recurrent updates, while the suffix and decoder continue from an ordinary recurrent state. StateBridge therefore turns fixed-text reuse in SSMs into learned transition reuse without changing request semantics, model weights, or the decoding interface.

5 Experiments

We evaluate whether one surrogate per document preserves raw-document QA performance across varying request contexts while accelerating TTFT. We then analyze the quality–latency trade-off, applicability across backbones, and sensitivity to document length and query-type shifts.

5.1 Experimental Setup

Benchmarks and protocol. Doc2Dial (Feng et al., 2020) contains conversations grounded in public-service documents; MultiDoc2Dial (Feng et al., 2021) extends this setting to conversations spanning multiple documents. DuoRC (Saha et al., 2018) provides movie-plot QA, and CoQA (Reddy et al.,

2019) contains multi-turn QA over passages. For each document, the same learned surrogate is reused across held-out requests whose prefixes and suffixes vary. Within each request, the full-document run (Raw) and STATEBRIDGE use identical prefixes, suffixes, and input order; only the reusable document block is replaced. Held-out evaluation questions are excluded from surrogate construction, and no request-specific optimization is performed. Support documents are provided without retrieval. Document controls remove or replace the support document while keeping the prefix and query fixed.

Models and surrogate construction. Unless stated otherwise, experiments use frozen Mamba-2 2.7B (Dao & Gu, 2024). We also evaluate Mamba-2 1.3B and Mamba-1 1.4B (Gu & Dao, 2024) on Doc2Dial to test applicability across model sizes and Mamba variants. Doc2Dial's repeated-document dialogues closely match the intended reuse setting. For each document, backbone, and budget, the two-stage procedure in Section 4.3 learns a separate matrix of m input vectors. The backbone remains frozen throughout construction and evaluation. We use $m \in \{32, 64, 128\}$, with $m = 96$ also evaluated on DuoRC. Raw and surrogate runs share the same checkpoint, tokenizer, prompt template, and decoding settings.

Implementation and hardware. We measure latency on both edge and server hardware. The edge configuration uses an NVIDIA GeForce RTX 4060 Ti (16 GB) with FP16 inference at batch size 16. The server configuration uses an NVIDIA H800 at batch sizes 32 and 64. All experiments use PyTorch 2.10.0, Transformers 5.1.0, and the latest official `mamba-ssm` implementation.

Metrics and latency. Task quality is measured by grounding-span F1 on Doc2Dial and MultiDoc2Dial, and by exact match (EM) and token-level F1 on DuoRC and CoQA. For each metric M, retention is $100 \times M_{\mathrm{SB}}/M_{\mathrm{raw}}$ on the same evaluation set. Whole-prompt TTFT includes the prefix, document or surrogate, and suffix up to the first output token. Block latency isolates the document or surrogate scan starting from the incoming recurrent state. For both metrics, speedup is Raw latency divided by surrogate latency on the same device and batch size. The DuoRC H800 block-latency result at $m = 64$ and batch size 64 uses random sequences of 840 and 64 tokens as proxies for the two scan lengths. Timing excludes tokenization and request scheduling; offline training time is reported separately.

Table 2: Task performance on Mamba-2 2.7B. Parentheses show F1 retention relative to Raw.

Setting	**Doc2Dial**	**MultiDoc2Dial**	**DuoRC**		**CoQA**	
	F1	F1	EM	F1	EM	F1
Raw	24.33 (100%)	23.54 (100%)	27.94	38.15 (100%)	45.70	60.10 (100%)
$m = 32$	22.14 (91.00%)	21.34 (90.70%)	25.29	35.38 (92.74%)	38.90	53.30 (88.69%)
$m = 64$	22.78 (93.70%)	22.13 (94.00%)	27.06	36.98 (96.93%)	39.20	53.70 (89.35%)
$m = 128$	23.03 (94.70%)	23.04 (97.90%)	27.35	38.56 (101.07%)	39.70	54.60 (90.85%)
No doc.	12.42 (51.05%)	11.28 (47.92%)	0.59	3.10 (8.13%)	18.70	27.50 (45.76%)
Mismatched doc.	11.92 (48.99%)	10.92 (46.39%)	2.35	4.31 (11.30%)	18.50	26.20 (43.59%)

5.2 MAIN RESULTS

Performance preservation. Effective document reuse requires a surrogate to retain the quality of full-document processing as the surrounding request context changes. We first evaluate this requirement by comparing STATEBRIDGE with Raw on four QA benchmarks, holding each document surrogate fixed while varying the prefix and suffix across requests. Table 2 shows that STATEBRIDGE preserves high QA quality relative to Raw, with the strongest performance at larger budgets. At $m = 128$, it retains 94.70% and 97.90% of raw-document F1 on Doc2Dial and MultiDoc2Dial, respectively. At the same budget, retention reaches 101.07% on DuoRC and 90.85% on CoQA. These results show that a shared document surrogate can approach full-document quality on several tasks while maintaining high retention across all four benchmarks. Quality remains robust at smaller budgets: at $m = 32$, STATEBRIDGE still retains 88.69%–92.74% of raw-document F1. Relative to $m = 128$, this amounts to a 2.38%–8.25% decrease in F1 across benchmarks, showing that most of the quality achieved at the larger budget is preserved after a fourfold reduction in surrogate length.

These results directly test the central reuse requirement: each document surrogate remains fixed while both the prefix and suffix change across requests. The retained QA performance shows that the same shortcut can preserve the document's task-relevant effect from different incoming states, without adapting the backbone or optimizing the surrogate for each request.

Table 3: TTFT and block-prefill speedups on Mamba-2 2.7B (FP16). Each range spans RTX 4060 Ti (batch 16) and H800 (batches 32 and 64) at the given budget.

	TTFT			Block prefill		
Dataset	$m = 32$	$m = 64$	$m = 128$	$m = 32$	$m = 64$	$m = 128$
Doc2Dial	1.92–1.95×	1.83–1.86×	1.67–1.70×	5.69–9.41×	5.26–6.74×	3.58–4.21×
MultiDoc2Dial	1.89–1.93×	1.81–1.84×	1.65–1.68×	5.77–9.33×	5.34–6.68×	3.63–4.17×
DuoRC	2.67–2.71×	2.51–2.55×	2.21–2.25×	7.70–12.34×	7.12–8.84×	4.84–5.51×
CoQA	2.20–2.31×	1.97–2.05×	1.62–1.66×	3.37–5.34×	3.11–3.82×	2.11–2.39×

Table 4: Doc2Dial F1 across Mamba backbones.

Backbone	**Raw**	$\boldsymbol{m = 32}$	$\boldsymbol{m = 64}$	$\boldsymbol{m = 128}$
Mamba-2 2.7B	24.33 (100%)	22.14 (91.00%)	22.78 (93.70%)	23.03 (94.70%)
Mamba-2 1.3B	23.57 (100%)	21.22 (90.00%)	22.17 (94.00%)	22.55 (95.70%)
Mamba-1 1.4B	20.81 (100%)	19.35 (92.97%)	19.49 (93.65%)	19.79 (95.08%)

The document controls establish the importance of the correct support document. With the prefix and query unchanged, removing the document retains at most 51.05% of raw-document F1, while supplying a mismatched document retains at most 48.99%. On DuoRC, the 32-vector shortcut retains 92.74% of raw-document F1, compared with 8.13% and 11.30% for these controls. This indicates that the shortcut retains document-specific information for answering held-out requests.

Speedup. Table 3 shows TTFT speedups of $1.62\times$–$2.71\times$ and block-prefill speedups of $2.11\times$–$12.34\times$ across the four benchmarks and tested hardware configurations. Even at $m = 128$, TTFT speedups remain $1.62\times$–$2.25\times$, combining the quality benefits of larger budgets with lower first-token latency. The larger gains on DuoRC than CoQA are consistent with their average document lengths of 837.7 and 342.2 tokens: longer blocks offer more scan steps to replace. Per-configuration TTFT and the quality–latency trade-off are provided in Appendices D and C, respectively.

Generality across Mamba backbones. Quality preservation extends across the tested model sizes and Mamba variants. We repeat the Doc2Dial evaluation with Mamba-2 1.3B and Mamba-1 1.4B, learning separate surrogates for each frozen backbone under the same reuse protocol. As Table 4 shows, all three models retain at least 90% of their own raw-document F1 with 32 input vectors, and increasing the budget improves F1 for every backbone. This consistency supports the applicability of learned transition shortcuts to both Mamba-1 and Mamba-2 on this benchmark.

Offline training time. On DuoRC with frozen Mamba-2 2.7B, $m = 64$ surrogates reach an estimated training-time break-even after approximately $4{,}100$ requests per document. This estimate combines 51.2 s of training per document under eight-way parallel normalization with 12.59 ms of block-processing savings per request averaged across batches of 64 on H800, excluding model loading and teacher-target/state preparation. Production RAG studies report concentrated document access and sustained chunk reuse over months (Jin et al., 2026; Agarwal et al., 2025), suggesting that this reuse level is attainable for popular, slowly changing documents.

5.3 Analysis

Role of the state anchor. We isolate the contribution of document-end state supervision on 40 DuoRC documents, learning one $m = 64$ surrogate per document with a frozen backbone. Across these documents, we use 200 questions for construction and 30 held-out questions for evaluation. Both variants distill Raw-generated answers and differ only in whether state supervision is included. Table 5 shows that adding the state anchor raises held-out HOW F1 retention from 14.54% to 76.39%

Table 5: State-anchor ablation.

Setting	**Construction F1**	**Held-out HOW F1**
Raw	45.36	49.09
Behavior only	45.12	7.14
Behavior + state	44.83	**37.50**

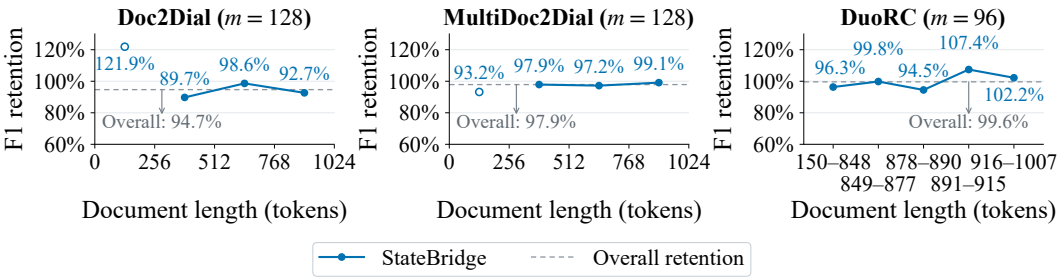

Figure 5: F1 retention by document length. Dashed lines mark overall retention; hollow markers indicate groups with fewer than 10 documents.

of Raw, while both variants remain close to Raw on construction questions. This supports the state anchor's role in transferring the shortcut beyond the construction questions.

Effect of document length. We examine whether fixed-budget surrogates preserve quality across document lengths. Doc2Dial and MultiDoc2Dial use $m = 128$ with 256-token bins; DuoRC uses $m = 96$ with five equally sized document groups ordered by length. Retention is computed from mean F1 over requests within each group. Figure 5 shows no consistent decline with length. MultiDoc2Dial retains over 97% of raw-document F1 in all three groups containing at least 10 documents; Doc2Dial and DuoRC vary more, but neither has its lowest retention in the longest group. These results support fixed-budget reuse across the evaluated length range.

Generalization to unseen query types. Beyond varying request prefixes and suffixes, we test whether a document surrogate transfers to question types absent from its construction data. DuoRC surrogates are learned from WHO, WHAT, and WHERE questions. Evaluation uses held-out questions of these types (ID) and WHICH, YES-NO, and OTHER questions (OOD) about the same documents. Table 6 shows that OOD F1 retention increases from 55.98% at $m = 32$ to 74.34% at $m = 128$. Larger budgets thus improve transfer to unseen query types, although retention remains below ID at every budget, indicating sensitivity to the downstream query distribution.

Table 6: Query-type generalization on DuoRC.

Setting	**Unseen questions (ID)**		**Unseen query types (OOD)**	
	EM	F1	EM	F1
Raw	27.94	38.15 (100%)	27.38	36.56 (100%)
$m = 32$	25.29	35.38 (92.74%)	11.90	20.47 (55.98%)
$m = 64$	27.06	36.98 (96.93%)	13.10	21.89 (59.88%)
$m = 128$	27.35	38.56 (101.07%)	17.86	27.18 (74.34%)

6 Conclusion and Discussion

We introduced StateBridge, which enables context reuse in frozen SSMs through learned state-transition shortcuts. The same compact input matrix operates from each request's current recurrent state, supporting varying prefixes and suffixes without request-specific optimization. Across four QA benchmarks, StateBridge retains 88.69%–101.07% of raw-document F1 and achieves TTFT speedups of up to $2.71\times$. The state-anchor ablation further shows that document-end state supervision improves transfer to unseen question types.

Our results show that a document's effect on recurrent state can be approximated by a reusable shortcut. This raises a broader question for SSMs: what do these context-induced state transformations represent? In particular, how does a context block interact with existing memory, and which aspects of this interaction shape subsequent model behavior? Understanding these mechanisms could explain why a short learned input can preserve the behavior induced by a much longer document, connecting efficient context reuse with a mechanistic account of how SSMs incorporate and use context.

AI USE STATEMENT

We used generative AI tools to assist with literature search, manuscript writing, and the implementation of parts of the experimental code. The authors reviewed and revised the AI-assisted material and take full responsibility for the final manuscript, including its scientific claims, results, and references.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 3(3):136:1–136:28, 2025. doi: 10.1145/3725273. URL https://doi.org/10.1145/3725273.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Song Feng, Hui Wan, Chulaka Gunasekara, Siva Patel, Sachindra Joshi, and Luis Lastras. doc2dial: A goal-oriented document-grounded dialogue dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8118–8128, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.652. URL https://aclanthology.org/2020.emnlp-main.652/.

Song Feng, Siva Sankalp Patel, Hui Wan, and Sachindra Joshi. MultiDoc2Dial: Modeling dialogues grounded in multiple documents. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6162–6176, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.498. URL https://aclanthology.org/2021.emnlp-main.498/.

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.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

Yasaman Jafari, Zixian Wang, Leon Bergen, and Taylor Berg-Kirkpatrick. Studying the soupability of documents in state space models. *arXiv preprint arXiv:2505.24033*, 2025. doi: 10.48550/arXiv.2505.24033. URL https://arxiv.org/abs/2505.24033.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1): 2:1–2:27, 2026. doi: 10.1145/3768628. URL https://doi.org/10.1145/3768628.

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 *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10010352.

Tian Yu Liu, Alessandro Achille, Matthew Trager, Aditya Golatkar, Luca Zancato, and Stefano Soatto. Picaso: Permutation-invariant context composition with state space models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=88TC1AWV27.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross Position-Aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 83–99. USENIX Association, 2026. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract-Conference.html.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/7c180af017258d239bac6248d1eb26ac-Abstract-Conference.html.

Siva Reddy, Danqi Chen, and Christopher D. Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016/.

Amrita Saha, Rahul Aralikatte, Mitesh M. Khapra, and Karthik Sankaranarayanan. Duorc: Towards complex language understanding with paraphrased reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1683–1693, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1156.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Non-Prefix Reuse in Practice

Non-prefix reuse naturally arises when repeated content follows request-specific context. In multi-document RAG, the same document can be retrieved alongside different preceding documents across requests, as studied by CacheBlend and EPIC (Yao et al., 2025; Hu et al., 2025). Agent workloads can also place updated context before reusable history; CacheSlide discusses why moving these static segments to the front can disrupt context organization or dependencies (Liu et al., 2026). These examples motivate reuse that respects the original input order.

A natural alternative is to move the reusable document to the beginning of the request and cache its resulting recurrent state. We evaluate this document-first arrangement, (x, u, p), against the original order (u, x, p), where u, x, and p denote the request prefix, document, and suffix, respectively. This comparison additionally includes documents with insufficient examples for surrogate construction, which are excluded from the main experiments. Since the comparison requires no surrogate training, these documents remain eligible for evaluating the effect of input order. Both orders are evaluated on the same requests.

Table 7: Effect of moving the document before the request prefix. Relative F1 decreases are measured against the original input order.

Dataset	Original F1	Document-first F1	Relative drop
Doc2Dial	24.33	21.78	10.48%
DuoRC	40.88	28.04	31.41%

Table 7 shows that document-first reordering reduces F1 by 10.48%–31.41% across the two benchmarks. Although moving the document first enables prefix caching, it does not preserve the original QA quality in these evaluations. The results support the motivation for StateBridge: reusing a document from the live prefix state while preserving the original input order.

B Construction and Evaluation Details

B.1 Surrogate Construction

Unless stated otherwise, we use `state-spaces/mamba2-2.7b` with the `EleutherAI/gpt-neox-20b` tokenizer. Each surrogate is initialized from the embeddings of the document's final m tokens and optimized in FP32, while the frozen backbone runs in FP16. The behavior objective uses Raw-generated answers as targets, with KL temperature 1 and logit margin 0.5.

Following the two-stage procedure in Section 4.3, Stage A uses AdamW with learning rate 0.02 and gradient-norm clipping at 1 for $9\lceil n_i/4 \rceil$ updates (9–54 in our experiments), where n_i is the number of construction questions for document i. Stage B continues from the resulting surrogate for 25 updates using a fresh AdamW optimizer at learning rate 0.001. On DuoRC, the combined Stage B gradient is used without clipping.

For Stage B, augmented prefixes are constructed using different few-shot demonstrations selected from a fixed public SQuAD pool. The pool is fixed, while the selected demonstrations vary across augmented prefixes; the document x and suffix p_i remain unchanged.

The behavior objective is

$$\ell_{\text{beh}} = \ell_{\text{CE}} + 0.5\, D_{\text{KL}}(p_{\text{teacher}} \| p_{\text{student}}) + 0.25\, \ell_{\text{margin}}, \tag{9}$$

with all terms averaged over answer tokens. For state matching, the relative squared errors of the convolution and SSM states are summed within each layer, then averaged over layers and samples.

For the refinement procedure in Section 4.3, g_a denotes the behavior gradient on original requests, while g_u and g_s denote the behavior and state gradients under the augmented prefixes. Each gradient is averaged over its own samples before combination, with norms computed in FP32:

$$g = g_a + \frac{0.5\|g_a\|_2}{\max(\|g_u\|_2, 10^{-12})}\, g_u + \frac{0.1\|g_a\|_2}{\max(\|g_s\|_2, 10^{-12})}\, g_s. \tag{10}$$

Table 8: Construction and evaluation protocols for the reported results.

Protocol	Construction / selection	Evaluation set
Doc2Dial	12 train questions per document; validation questions held out.	150 documents; 914 questions
MultiDoc2Dial	12 train questions per document; validation questions held out.	150 documents; 1,213 questions
DuoRC (main / ID)	20 development documents excluded; four held-out evaluation questions per document.	85 documents; 340 questions
DuoRC (OOD)	Evaluation questions belong to query types excluded from surrogate construction.	84 questions
DuoRC (state-anchor ablation)	200 construction questions across 40 documents; HOW questions held out.	30 HOW questions
CoQA	Shared evaluation questions for Raw, all surrogate budgets, and document controls.	100 documents; 1,583 questions

Table 9: Request components and maximum numbers of generated tokens. Gold history denotes the reference dialogue history supplied in the request.

Dataset	Prefix u	Block x	Suffix p	Max. tokens
Doc2Dial / MultiDoc2Dial	Instruction, two demonstrations, and gold dialogue history.	Support document.	Current question followed by `Agent:`.	64
DuoRC	Instruction and two SQuAD demonstrations.	Movie plot.	Current question and answer cue.	32
CoQA	Q/A history.	Story.	Gold Q/A history and current question.	8

The coefficients 0.5 and 0.1 specify gradient-norm ratios.

B.2 Data Selection and Evaluation Splits

Construction and held-out evaluation questions are strictly disjoint across all experiments. Documents with insufficient construction examples are excluded from surrogate evaluation. Table 8 summarizes the evaluation sets and their selection protocols. The document-first comparison in Section A additionally admits documents that do not meet the construction requirements.

For DuoRC query-type generalization, WHO, WHAT, and WHERE questions form the construction pool. ID evaluation uses held-out questions of these types, while WHICH, YES-NO, and OTHER questions are reserved for OOD evaluation. The main-quality results in Table 2 and the ID results in Table 6 use the same 340-question evaluation set. Construction-question order uses seed $42 + \mathrm{epoch}$, and the ID partition uses a fixed hash seed.

The state-anchor ablation in Table 5 is a separate diagnostic experiment on 40 DuoRC documents. Its 200 construction questions and 30 held-out HOW questions are totals across these documents; construction-question F1 is reported only as a fitting diagnostic.

B.3 Request Layout and Scoring

Table 9 summarizes the components of (u, x, p) and the generation limits. Where few-shot demonstrations are used, they are randomly resampled during both construction and evaluation. Each trained surrogate is held fixed across evaluation requests and replaces only x, without request-specific optimization.

Doc2Dial and MultiDoc2Dial use greedy decoding for both Raw and StateBridge. EM measures exact match after answer normalization, and F1 measures token overlap. DuoRC takes the highest score over reference answers; CoQA uses the official evaluator's multi-reference aggregation.

C Quality–Latency Trade-off

Figure 6 combines the F1 retention in Table 2 with block-prefill speedups on H800 at batch size 64 to visualize the effect of surrogate budget. At $m = 32$, StateBridge achieves $4.17\times$–$9.71\times$ block speedup while retaining 88.69%–92.74% of raw-document F1 across the four benchmarks. Reducing m from 128 to 32 nearly doubles block speedup on every benchmark, with a relative F1 decrease of 2.38%–8.25%. Thus, the smallest tested budget delivers the greatest acceleration, while larger budgets recover additional quality at the cost of a longer scan.

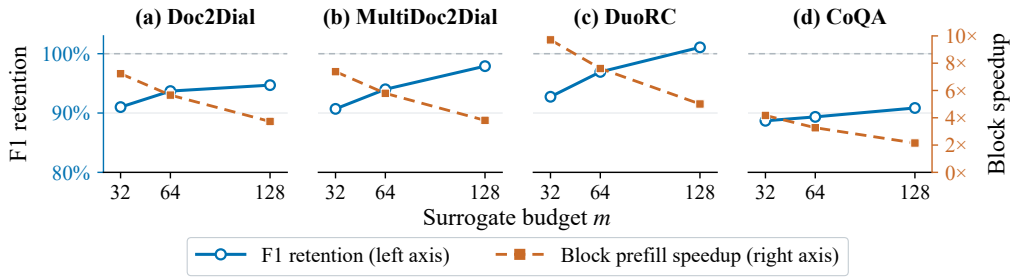

Figure 6: F1 retention and block speedup on H800 at batch size 64.

D Time to First Token

Table 10 reports TTFT for Mamba-2 2.7B in FP16, including the prefix, document or surrogate, and suffix. The TTFT columns in Table 3 summarize the three hardware configurations at each surrogate budget.

Table 10: TTFT in milliseconds. Parentheses show speedup relative to Raw under the same hardware and batch configuration.

Dataset	Raw	$m = 32$	$m = 64$	$m = 128$
RTX 4060 Ti, batch 16				
Doc2Dial	3333.27	1710.51 (1.95×)	1795.56 (1.86×)	1966.49 (1.70×)
MultiDoc2Dial	3352.99	1735.82 (1.93×)	1821.11 (1.84×)	1991.35 (1.68×)
DuoRC	3423.03	1261.44 (2.71×)	1342.76 (2.55×)	1521.02 (2.25×)
CoQA	1468.87	635.02 (2.31×)	718.17 (2.05×)	886.52 (1.66×)
H800, batch 32				
Doc2Dial	658.92	341.65 (1.93×)	358.41 (1.84×)	392.11 (1.68×)
MultiDoc2Dial	660.59	347.12 (1.90×)	363.92 (1.82×)	397.48 (1.66×)
DuoRC	678.19	253.16 (2.68×)	268.99 (2.52×)	304.84 (2.22×)
CoQA	294.03	133.90 (2.20×)	149.63 (1.97×)	181.87 (1.62×)
H800, batch 64				
Doc2Dial	1326.51	692.34 (1.92×)	726.71 (1.83×)	795.16 (1.67×)
MultiDoc2Dial	1329.87	702.36 (1.89×)	736.57 (1.81×)	804.71 (1.65×)
DuoRC	1364.97	510.76 (2.67×)	543.66 (2.51×)	618.04 (2.21×)
CoQA	595.15	263.90 (2.26×)	296.18 (2.01×)	362.21 (1.64×)