DOES MAMBA NEED AN INDEPENDENT GATE?

Anonymous authors
Paper under double-blind review

ABSTRACT

We study whether Mamba's independent gate projection is necessary, or whether its routing signal can instead be derived from a representation the block has already computed. We propose **Content-Derived Gating (CDG)**: a low-rank map applied to Mamba's content projection in place of the full gate. CDG substantially reduces gate parameters while preserving performance below flagship (130M-parameter) scale, where it trains $\sim$19% faster and uses $\sim$23% less peak memory than the full gate but incurs a persistent $\sim$2.6% relative bpb gap.

We then show this behavior is not explained by expressivity: CDG and a matched content-blind low-rank gate parameterize *exactly the same set* of reachable gate functions whenever the content projection has full rank, as it does throughout this study (Proposition 1). Yet the two behave differently in practice, and the direction reverses with scale: comparable or suggestively better for content-derivation at small scale, but at 130M the content-blind gate improves bpb by 1.4% over CDG despite fewer parameters, closing more than half its gap to the full gate. Any derived-routing scheme's gains should therefore be checked against a content-blind low-rank baseline of matched size before being credited to the derivation, since that credit can reverse with scale.

1 INTRODUCTION

Selective state space models are a strong sub-quadratic alternative to Transformer attention, but spend most of their parameter budget outside the recurrence itself. In a Mamba block, the gate projection W_g is a full $(d_{\text{model}} \times d_{\text{inner}})$ matrix whose only job is deciding, after the fact, which recurrence output channels to amplify or suppress (Section 3) – unlike the content and output projections, it never writes into or reads out of the recurrence. This raises a question already asked of Transformer attention: can this routing signal be derived from a representation the block has already computed, rather than requiring an independent projection, the way Keyless Attention derives attention's routing from the value representation (Gao, 2026)? We propose **Content-Derived Gating (CDG)** (Section 4): a low-rank operator $R(u_t) = UV^\top u_t$ applied to the content projection u_t, replacing W_g entirely.

Tested at four scales up to a $\sim$130M-parameter LLM, CDG matches the full gate and beats a hard-tie baseline below flagship scale, then underperforms it by a small, persistent margin at 130M parameters, confirmed on real fused kernels ($\sim$19% faster, $\sim$23% less peak memory). Two natural fixes fail to close that gap. Most basically: does CDG's success require the gate to be *content-derived*, or would any low-rank gate of the same size do as well? Testing this missing baseline reveals a reversal within the reversal: content-derivation is inert at loose compression, suggestively better on sMNIST's steepest cut, and loses to a content-blind gate by a large margin at the 130M flagship.

Three hypotheses are worth separating. **H1** (the full gate is unnecessary) is well supported. **H2** (its signal can be low-rank) is *strongly* supported and does most of H1's work. **H3** (the signal should specifically be *derived from content*) is only partially supported and *contradicted* at flagship scale. The strongest conclusion is H2, not H3: content-derivation and a free low-rank parameterization reach identical functions (Proposition 1), so any content-derivation-specific advantage is scale-dependent, not guaranteed by capacity.

This paper's contribution is therefore not CDG in isolation, but the missing control it makes possible: CDG itself, validated across four scales and real fused kernels; the content-blind control missing from prior instances of this pattern; a formal reachability-equivalence result tested with a controlled subspace-alignment experiment (Section 4.4); and the recommendation that any derived-routing proposal report this control before crediting its gains to the derivation.

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.

2 Related Work

Value- and content-derived routing. Gao (2026) derive a Transformer's attention routing signal directly from the value representation rather than an independent key projection, cutting a full projection (and its KV-cache footprint) at little accuracy cost – one instance of a broader pattern of deriving a routing signal from an already-computed representation rather than paying for an independent projection. CDG is the low-rank, content-derived instance we build for Mamba's gate, tested against a matched content-blind control (Section 4.3). Qiu et al. (2025) give the same bet direct support inside attention, finding that gating derived from the query removes attention sinks and improves stability at scale, and noting that production architectures (Qwen3-Next) already do this operationally. Neither work reports a content-blind control of matched size, so neither isolates whether its gains are due to derivation or simply to size; our ablation is, to our knowledge, the first to run that control for this pattern, and finds its answer is not scale-invariant.

Projection sharing and low-rank adaptation. Zhan et al. (2025) scale Mamba's capacity via sparsely-activated mixture-of-experts projection sharing – a mechanistically different route to the same bet that a gate's nominal capacity is not fully needed per token. Ali et al. (2026) find that sharing a Mamba adapter's up/down projections across layers regularizes with little cost, a smaller-scale analogue of this paper's Shared Basis Extension, which finds the opposite result at the scales tested here (Section 6). Press & Wolf (2017) tie an LM's input embedding to its output projection with no perplexity loss, and Hu et al. (2021) show a low-rank update suffices to adapt a pretrained model – the same low-rank-is-enough intuition our low-rank-gate ablation tests directly against CDG's content-derivation rather than assumes.

Tying a Mamba block's own projections. In our own broader exploration of this block, fully tying the gate to the content projection ($W_{\text{gate}} := W_{\text{ssm}}$, no new parameters) costs nothing at a single block but is reliably among the most costly schemes once stacked to depth 2–3, on sequential MNIST, and at 130M parameters. This paper's central question is whether CDG's small amount of new, learned low-rank capacity avoids that failure mode – Section 6 shows it does, below 130M parameters.

Selective state space models. Gu & Dao (2023) introduce Mamba; Dao & Gu (2024) reformulate it as structured state-space duality (Mamba-2), the architecture this paper's implementation follows, whose `mamba_ssm` package provides the fused kernels used for the real-kernel confirmation (Section 6.5). Huang et al. (2025) give a mechanistic account of why Mamba's input-dependent selectivity matters for approximation power, memorization, and recall; that result is complementary, not competing – it addresses selectivity inside the recurrence, while this paper asks a narrower question about the independent gate projection outside it (Section 3).

3 Background

Ignoring SSM internals, a standard selective SSM block computes

$$u_t = W_u x_t \tag{1}$$
$$g_t = W_g x_t \tag{2}$$
$$h_t = \mathrm{SSM}(u_t) \tag{3}$$
$$y_t = W_o \left(\sigma(g_t) \odot h_t\right), \tag{4}$$

where $x_t \in \mathbb{R}^d$ is the block's input, $d = d_{\text{model}}$ is the model dimension, and $e = d_{\text{inner}}$ is the expanded channel dimension the state-space recurrence operates in. The dominant parameter cost is the three $(d \times e)$-shaped matrices W_u, W_g, W_o:

$$P_{\text{baseline}} = de + de + ed = 3de,$$

with the gate contributing de parameters despite never participating in the recurrence, while A, B, C, Δ are a small fraction of block parameters at practical widths (B/C stay untied throughout, orthogonal to this study). This paper targets the gate: it only determines which content dimensions get amplified or suppressed, suggesting its routing signal may be derivable from an already-computed representation rather than a full projection of its own.

4 METHOD

4.1 CONTENT-DERIVED GATING

CDG replaces the independent gate projection with a low-rank function of the block's own content projection:

$$u_t = W_u x_t, \qquad g_t = R(u_t), \qquad h_t = \mathrm{SSM}(u_t), \qquad y_t = W_o\left(\sigma(g_t) \odot h_t\right).$$

The routing operator R is a pure two-matrix low-rank map, with no nonlinearity between the two factors:

$$U \in \mathbb{R}^{e\times r}, \quad V \in \mathbb{R}^{e\times r}, \quad r \ll e, \qquad g_t = UV^\top u_t.$$

Parameter count becomes $P_R = 2er$, versus the original gate projection's de; the reduction factor is

$$\frac{P_R}{de} = \frac{2er}{de} = \frac{2r}{d}.$$

This factor is denominated in $d = d_{\text{model}}$, not $e = d_{\text{inner}}$ — a distinction that matters in practice (Section 5): at $d_{\text{model}} = 4096$, $r = 128$ gives $2r/d = 6.25\%$ of the original gate cost (a 93.75% reduction); at the small synthetic-task scale used in Section 6 ($d_{\text{model}} = 16$), the same formula means $r = 8$ is already break-even with the original gate, and $r = 16$ is strictly more expensive.

Remark: CDG is a constrained low-rank linear gate, not a distinct function class. Because R has no nonlinearity and u_t is itself linear in x_t, composing gives

$$g_t = UV^\top u_t = UV^\top W_u x_t = U\underbrace{\left(V^\top W_u\right)}_{\in\,\mathbb{R}^{r\times d}} x_t,$$

a rank-$\leq r$ linear map of x_t – exactly the functional form of the Plain Low-Rank Gate ablation's $g_t = AB^\top x_t$ (Section 4.3). CDG is therefore not a different *function class* from the plain low-rank gate; both are rank-$\leq r$ linear maps from x_t to g_t. What differs is the *parameterization*: the plain low-rank gate learns its $(r \times d)$ down-projection $B^\top$ freely and independently, while CDG's implicit down-projection $V^\top W_u$ is constrained to factor through the block's own content matrix W_u – so CDG's routing subspace is tied to whatever subspace the content branch has already learned to represent u_t in, rather than a freely-learned subspace of its own. This reframes the paper's central ablation (Section 4.3, Section 6.6): it does not test "linear" against "content-derived," since both variants are linear in x_t; it tests whether *constraining* the gate's low-rank down-projection to share structure with W_u, instead of learning it freely, helps or hurts – and Section 6.6 shows the answer reverses with scale. That constraint has a precise form, given next.

Write CDG's effective gate matrix as $W_g^{\text{CDG}} = U(V^\top W_u) \in \mathbb{R}^{e\times d}$ (so $g_t = W_g^{\text{CDG}} x_t$) and the Plain Low-Rank Gate's as $W_g^{\text{LR}} = AB^\top \in \mathbb{R}^{e\times d}$, and let $\mathrm{row}(\cdot)$ denote a matrix's row space, i.e. the subspace of $\mathbb{R}^d$ spanned by its rows – the directions in x_t-space the gate actually reads. A first guess is that $\mathrm{row}(W_g^{\text{CDG}}) \subseteq \mathrm{row}(W_u)$ – CDG's rows are, after all, linear combinations of W_u's rows – and that this containment is itself the source of the scale-dependent reversal. This containment is true but, at every width used in this study, vacuous: a trained W_u is comfortably full rank at every scale we checked, including the 130M flagship (Appendix A.4 gives the singular-value evidence); $\mathrm{row}(W_u) = \mathbb{R}^d$ then, so the containment holds for *every* rank-$\leq r$ matrix, not just CDG's, and restricts nothing.

Proposition 1 (Reachability equivalence under full-rank W_u)**.** *If* $\mathrm{rank}(W_u) = d$*, then for every matrix* $M \in \mathbb{R}^{e\times d}$ *with* $\mathrm{rank}(M) \leq r$*, there exist* $U, V \in \mathbb{R}^{e\times r}$ *with* $M = UV^\top W_u$*. Consequently CDG and the Plain Low-Rank Gate parameterize* exactly the same set *of reachable gate functions at matched rank* r*. (Proof in Appendix A.3.)*

This sharpens the puzzle rather than resolving it. Because the two parameterizations can reach the identical set of gate functions, the reversal reported in Section 6 cannot be an expressivity difference between CDG and the plain low-rank gate – there isn't one, at full-rank W_u. It must instead be a difference in what gradient descent actually finds within that shared reachable set: an inductive-bias or optimization-dynamics effect, not a hard capacity restriction. The mechanism we conjecture is parameter *coupling*, not subspace confinement: CDG ties W_u into two roles at once – constructing u_t for the recurrence, and, via U, V, supplying the gate's realized directions – so every training step's

gradient on W_u carries signal from both the content/reconstruction path and the gating path, whereas the plain low-rank gate's B receives gradient only from the gating path. Whether this coupling helps (by biasing the gate, especially early in training, toward directions W_u already emphasizes for content) or hurts (by forcing W_u to compromise between two objectives once they diverge) is a question about training trajectories that Proposition 1 cannot settle by itself – it only rules out the simpler, capacity-based explanation. Section 4.4 tests the coupling hypothesis directly with a controlled experiment, and Section 7 returns to its results.

4.2 Two Proposed Fixes for the Flagship-Scale Gap

Section 6 finds that CDG matches the full gate below 130M parameters but trails it at that flagship scale. Two natural architectural changes were proposed as fixes.

Residual Routing Variant. One concern is that forcing routing entirely through content features may lose expressive capacity that an independent projection would have retained. The residual routing variant adds back a light-weight, independently-parameterized correction term on top of the content router:

$$g_t = \underbrace{UV^\top u_t}_{\text{content routing}} + \underbrace{AB^\top x_t}_{\text{independent residual}}, \qquad A \in \mathbb{R}^{e\times r_s}, \quad B \in \mathbb{R}^{d\times r_s}, \quad r_s \ll d, e.$$

The second term is itself a low-rank factorization of the *original* independent gate, applied directly to the block's (normed) residual-stream input x_t rather than to u_t – not a reuse of any other matrix in the block. At matched rank $r_s = r$, its own parameter cost is $r_s(d + e)$, which eats into CDG's savings but keeps the variant strictly cheaper than the full independent gate at every rank tested (Section 6).

Shared Basis Extension. A more aggressive parameter-reduction strategy shares a single latent basis $U_b \in \mathbb{R}^{e\times k}$, $k \ll d$, among the content, gate-routing, and output computations:

$$W_u = U_b M_u, \qquad W_o = M_o U_b^\top, \qquad g_t = R(u_t),$$

with $M_u \in \mathbb{R}^{k\times d}$, $M_o \in \mathbb{R}^{d\times k}$ small private factors either side of the one physically-shared matrix U_b. This creates one common feature space for content construction, routing, and output generation simultaneously, going further than CDG alone by also factoring W_u and W_o rather than only the gate.

4.3 Ablation: Does Tying the Gate's Subspace to the Content Matrix Matter?

Given the Remark above, CDG and an ordinary low-rank gate are the same rank-$\leq r$ linear function class; the open question is whether *constraining* that rank-r down-projection to factor through the block's own content matrix W_u, rather than learning it freely, actually matters. Neither the original CDG proposal nor either fix above was ever checked against this most basic alternative: a freely-parameterized low-rank gate of the same rank, with no tie to W_u at all. We test this directly with a **Plain Low-Rank Gate**: the residual routing variant's $AB^\top x_t$ correction term, used *alone* as the entire gate, with its own independent A, B factors (no constraint linking them to W_u):

$$g_t = AB^\top x_t, \qquad A \in \mathbb{R}^{e\times r}, \quad B \in \mathbb{R}^{d\times r}.$$

At matched rank r, this has parameter count $r(d + e)$ against CDG's router's $2er$: equal only when $d = e$ (true only of this study's sMNIST config; slightly smaller otherwise, including single-block/stacked-depth and the 130M flagship, since $d < e$ in every other case tested). This makes the comparison equal-*rank*, not always exactly equal-*parameter* (Section 6.6 gives each stage's true count). If the plain low-rank gate matches or beats CDG despite never having more parameters, tying the gate's down-projection to W_u is not earning its keep – a freely-parameterized low-rank gate of the same rank does just as well or better; if CDG is measurably better, that tie is doing real work by routing through a subspace the content branch has already learned, rather than one the gate must learn independently. Results are in Section 6.6.

4.4 Subspace-Alignment Gate

Proposition 1 rules out expressivity as the reason CDG and the plain low-rank gate diverge, but it does not test whether *subspace alignment* – the informal mechanism CDG's parameter tie suggests – actually matters; it only shows that tying to W_u does not prevent the free gate from reaching the same solutions. To test alignment directly, independent of whether it is tied or learned, we introduce a **Subspace-Alignment Gate**: a rank-r gate whose down-projection is a *fixed, non-trainable* orthonormal basis $Q \in \mathbb{R}^{d\times r}$, rather than a learned matrix,

$$g_t = A(Q^\top x_t), \qquad A \in \mathbb{R}^{e\times r}, \quad Q^\top Q = I_r,$$

with Q built from a reference model's own already-trained content matrix W_u^{ref} (a separately-trained `free` run at the same scale) in one of three ways: **aligned** (Q = its top-r right singular vectors, the directions W_u^{ref} reads most), **random** (Q = r uniformly random orthonormal directions, ignoring W_u^{ref} entirely), or **orthogonal** (Q = its bottom-r right singular vectors, the directions it reads least – W_u^{ref}'s full rank rules out a literal orthogonal complement, Proposition 1, so this is the closest well-defined stand-in). Only A is trained; Q is frozen throughout. This directly manipulates the one thing CDG's parameter tie leaves uncontrolled: whether the gate's (now genuinely fixed) reading subspace overlaps a reference content projection's dominant directions, holding rank, function class, and every other architectural choice fixed across the three conditions.

4.5 Overlap Analysis

As a complementary, non-interventional check, we also measure, for independently-trained `cdg` and `low_rank_gate` models, how much of each one's own *learned* down-projection overlaps that same model's own (independently trained) W_u: writing $B \in \mathbb{R}^{r\times d}$ for a gate's effective down-projection ($B = V^\top W_u$ for `cdg`, B = the free down-projection for `low_rank_gate`; Section 4.1) and $P = QQ^\top$ for the projector onto that model's own top-r right singular directions of W_u,

$$\text{overlap}(B) = \frac{\|BP\|_F^2}{\|B\|_F^2} \in [0, 1],$$

we correlate $\text{overlap}(B)$ across seeds with final task performance. Unlike the Subspace-Alignment Gate, this does not manipulate alignment – it asks whether gradient descent's own choice of down-projection, when left free (`low_rank_gate`) or tied (`cdg`), ends up more or less aligned with the content projection's dominant directions, and whether that alignment predicts performance.

4.6 Computational Complexity

Baseline gating costs $O(de)$ multiply-adds per token. Content-derived routing costs $O(de) + O(er)$ for the content projection plus the router; since $r \ll d$, the additional routing computation is negligible relative to the content projection, and the independent gate projection is removed entirely.

4.7 Implementation

R is a bias-free `down`/`up` linear pair with no nonlinearity between the factors, applied to u_t immediately after the content projection and before the causal convolution – matching $u_t = W_u x_t$'s definition literally. The Plain Low-Rank Gate ablation reuses this module unchanged, applied to x_t instead. PyTorch code is in Appendix A.

5 Experimental Setup

We evaluate using our own pure-PyTorch selective-SSM block (a parallel Hillis–Steele scan, no `mamba_ssm` CUDA dependency), which separates the packed $W_{\text{ssm}}/W_{\text{gate}}$ projection into two independently-registered matrices and provides free/tied baselines at every scale below. We extend it with `cdg`, `cdg_residual`, `shared_basis`, and `low_rank_gate` modes (Section 4).

Four stages, cheap to expensive, each proceeding only if the previous stage showed no degradation:

1. **Single-block memory capacity** ($d_{\text{model}} = 16$, $d_{\text{inner}} = 32$, $d_{\text{state}} = 8$, $dt_{\text{rank}} = 4$, 24 lags, $T = 200$), 5 seeds.

2. **Stacked depth** ($L = 2, 3$) on the same task, 15 fresh seeds — the regime where the hard gate/content identity tie first showed a large standardized cost despite no single-block effect.
3. **Sequential MNIST** ($d_{\text{model}} = 64$, $d_{\text{inner}} = 64$, $d_{\text{state}} = 16$, $dt_{\text{rank}} = 8$, 8 epochs, batch 32), 3 seeds.
4. **A ∼130M-parameter byte-level causal LM on SlimPajama** ($d_{\text{model}} = 1024$, $d_{\text{inner}} = 2048$, $d_{\text{state}} = 16$, $dt_{\text{rank}} = 64$, $n_{\text{layers}} = 20$, $T = 512$, batch 4), matching our own prior flagship scale-up. Run at 1000 and 10000 steps (Section 6), and, for `free`, `cdg`, and `cdg_residual`, at 3 seeds each on real `mamba_ssm` CUDA kernels rather than the custom scan (Section 6.5).

At each stage, rank r is chosen so the reduction factor $2r/d_{\text{model}}$ (Section 4.1) lands at meaningful savings points rather than at fractions of d_{inner}: $r \in \{1, 2, 4\}$ for stages 1–2 ($d_{\text{model}} = 16$, giving 87.5%/75%/50% gate reduction), $r \in \{8, 16, 32\}$ for stage 3 ($d_{\text{model}} = 64$, giving 75%/50%/0%-break-even reduction), and $r = 128$ for stage 4 (75% reduction, the worked example of Section 4.1). The residual routing variant uses matched rank $r_s = r$ throughout; the shared basis extension fixes its gate-router rank at $r = 4$ (stages 1–2) or $r = 8$ (stage 3) and sweeps its basis rank k instead; the low-rank-gate ablation uses the same rank grid as `cdg` at every stage, for a directly matched-parameter comparison. Free and tied baselines are reused directly from our own existing (unpublished) results rather than rerun.

For the real-kernel confirmation (Section 6.5), we reimplement the block's content/gate/output projections and their tying structure against `mamba_ssm`'s actual fused `causal_conv1d_fn` and `selective_scan_fn` kernels, keeping the block otherwise norm/residual-free to mirror `mamba_ssm`'s own mixer scope exactly, and verified to parity against a stock `mamba_ssm` block before use (Appendix A.6 gives the exact numerical tolerance).

Statistical convention. Every "gap/SE" figure in Section 6 is a *standardized difference* (a two-sample z-style statistic), not a hypothesis test: given two modes' seed means m_1, m_2 and sample standard deviations s_1, s_2 over n_1, n_2 seeds, gap/SE $= (m_1 - m_2)/\sqrt{s_1^2/n_1 + s_2^2/n_2}$, with no seed pairing and no correction for the many rank/basis/depth cells swept in each table. We accordingly avoid the words "significant"/"significance" throughout, using $|\text{gap/SE}| \gtrsim 2$ only as a magnitude cutoff for prose (a *large standardized difference*), not a p-value-backed claim – most importantly where $n \leq 3$ (every 130M-scale comparison, and sMNIST). Full definition, rationale, and seed-level data are in Appendix A.7.

6 Results

6.1 Single-block memory capacity

Among `free`, `tie_ssm_gate`, CDG at any rank, and the Residual Routing Variant at any rank, none shows a statistically meaningful gap from `free` (all $|\text{gap/SE}| < 1$, well below the ∼2.0 bar used throughout this project). This mirrors the sibling study's own finding that its hard gate tie was also invisible at a single block. Shared basis is the exception, and already the first sign of a pattern Section 7 returns to.

At $k = 8$ (50% total io-parameter savings – already a deeper cut than any CDG rank tested, at fewer total parameters than `cdg`, $r = 2$'s 1152), shared basis matches both `free` and `tie_ssm_gate` within noise. But this does not scale down monotonically with parameter count: $k = 4$ (a further cut to 512 params, matching `tie_all`'s parameter budget) collapses to a real, low-variance capacity loss (-2.53 gap/SE vs. `free`). Stranger still, $k = 16$ – despite having *more* parameters than $k = 8$, and the same parameter count as `cdg`, $r = 4$ – does not recover: its mean capacity is lower than $k = 8$'s, with a 12× jump in seed-to-seed variance (std $0.140 \rightarrow 1.680$). Whatever drives that non-monotonicity is not explained by parameter count alone, and we do not discuss shared basis stage-by-stage further (Section 7 gives the one-paragraph verdict).

The residual routing variant behaves like plain CDG here, not shared basis: all three ranks match `free` within noise, and the added AB^T correction term produces no visible improvement over plain `cdg` at matched r either (e.g. $r = 4$: 6.272 for cdg_residual vs. 6.719 for plain cdg, both within noise of `free`'s 6.211) – at this single-block scale the correction term is, at best, inert.

Mode	io params	savings	Capacity (mean ± std, $n = 5$)	vs. free	vs. tie_ssm_gate
free	1536	0.0%	6.211 ± 1.962	–	-0.90
tie_ssm_gate	1024	33.3%	6.726 ± 0.395	$+0.90$	–
cdg, $r = 1$	1088	29.2%	5.763 ± 3.003	$+0.28$	$+0.71$
cdg, $r = 2$	1152	25.0%	6.906 ± 0.334	-0.78	-0.78
cdg, $r = 4$	1280	16.7%	6.719 ± 1.142	-0.50	$+0.01$
shared_basis, $k = 4$	512	66.7%	3.993 ± 0.007	-2.53	-15.45
shared_basis, $k = 8$	768	50.0%	6.817 ± 0.140	$+0.69$	$+0.49$
shared_basis, $k = 16$	1280	16.7%	5.860 ± 1.680	-0.30	-1.12
cdg_residual, $r = 1$	1136	26.0%	6.834 ± 0.879	-0.65	-0.25
cdg_residual, $r = 2$	1248	18.8%	6.685 ± 1.139	-0.47	$+0.08$
cdg_residual, $r = 4$	1472	4.2%	6.272 ± 1.307	-0.06	$+0.74$
low_rank_gate, $r = 1$	1072	30.2%	5.716 ± 3.104	$+0.30$	–
low_rank_gate, $r = 2$	1120	27.1%	4.583 ± 3.693	$+0.87$	–
low_rank_gate, $r = 4$	1216	20.8%	7.099 ± 0.314	-1.00	–

Table 1: Single-block memory capacity, all modes. "Savings" is io-parameter reduction vs. `free`'s 1536: gate-only for `cdg` (87.5%/75%/50% at $r = 1, 2, 4$); shared_basis also factors $W_{\text{ssm}}/W_{\text{out}}$. low_rank_gate's gate uses $r(d+e)$ params vs. `cdg`'s $2er$ (Sec. 4.3), never exceeding its matched-r `cdg` budget. shared_basis fixes router rank $r = 4$; k is basis rank. cdg_residual adds a matched-rank AB^T correction, lowering its savings. gap/SE uses pooled SE.

Mode	savings	$L = 2$ mean ± std	gap/SE vs. free	$L = 3$ mean ± std	gap/SE vs. free
free	0.0%	10.242 ± 0.557	–	12.345 ± 0.492	–
tie_ssm_gate	33.3%	9.401 ± 0.435	$+4.61$	11.718 ± 0.536	$+3.34$
cdg, $r = 1$	29.2%	10.018 ± 0.439	$+1.23$	12.324 ± 0.727	$+0.09$
cdg, $r = 2$	25.0%	10.153 ± 0.970	$+0.31$	12.457 ± 0.483	-0.63
cdg, $r = 4$	16.7%	9.350 ± 2.007	$+1.66$	12.035 ± 0.961	$+1.11$
shared_basis, $k = 4$	66.7%	7.736 ± 0.408	$+14.06$	10.348 ± 0.594	$+10.02$
shared_basis, $k = 8$	50.0%	9.456 ± 0.525	$+3.97$	11.673 ± 0.891	$+2.56$
shared_basis, $k = 16$	16.7%	8.871 ± 2.269	$+2.27$	12.123 ± 0.472	$+1.26$
cdg_residual, $r = 1$	26.0%	9.969 ± 0.422	$+1.51$	12.359 ± 0.519	-0.08
cdg_residual, $r = 2$	18.8%	9.110 ± 1.958	$+2.15$	12.575 ± 0.603	-1.14
cdg_residual, $r = 4$	4.2%	9.528 ± 1.562	$+1.67$	12.551 ± 0.726	-0.91
low_rank_gate, $r = 1$	30.2%	10.026 ± 0.360	$+1.26$	12.511 ± 0.590	-0.84
low_rank_gate, $r = 2$	27.1%	10.292 ± 0.551	-0.25	12.511 ± 0.422	-0.99
low_rank_gate, $r = 4$	20.8%	9.897 ± 0.562	$+1.69$	12.484 ± 0.350	-0.89

Table 2: Stacked-depth memory capacity, $n = 15$ seeds/cell. "savings" is the same io-parameter reduction defined in the single-block table (Section 6.1), depth-independent since parameter count scales linearly with n_{layers}. Every mode beats `tie_ssm_gate` by a large margin in at least half its cells (gap/SE not shown; full breakdown in Appendix A.1), including low_rank_gate, at a margin at least as strong as CDG's own. A single seed collapsed to near-zero capacity in one cell each for `cdg` ($r = 4$, $L = 2$: 2/15) and cdg_residual ($r = 2$, $L = 2$: 1/15), inflating their variance – resembling instability characterized for `free` elsewhere, not mode-specific. low_rank_gate's gap/SE vs. `cdg` (Appendix A.1) stays short of the bar at every cell, the same pattern as the single-block table – being low-rank is enough to avoid `tie_ssm_gate`'s depth failure here.

The low-rank-gate ablation's own comparison to `cdg` at matched rank (at or below `cdg`'s own parameter count, never above) is essentially a wash: gap/SE is $+0.02$ at $r = 1$, $+1.40$ at $r = 2$, -0.72 at $r = 4$ – none crossing the significance bar, and no consistent direction. At this stage, being low-rank alone already matches both `free` and `cdg`; whether content-derivation earns its keep independent of rank is not yet distinguishable here (Section 6.6).

6.2 Stacked depth

This is the discriminating test: in the sibling study, `tie_ssm_gate` showed no single-block cost but a large one once stacked (gap/SE $+4.61$ at $L = 2$, $+3.34$ at $L = 3$, reproduced here). CDG does not reproduce that failure at any rank – no large gap from `free` at either depth, and at $r = 1, 2$ it *beats* `tie_ssm_gate` by a large margin. Shared basis does not share this good behavior (large loss in 5 of 6 cells; full detail in the table below). The residual routing variant again tracks plain CDG: no large gap from `free` anywhere (largest $|\text{gap/SE}| = 2.15$, a single-seed collapse), and it beats `tie_ssm_gate` in half its cells. The plain low-rank-gate ablation matches both `free` and plain `cdg` at every one of its six cells while beating `tie_ssm_gate` by a margin at least as large as CDG's own – confirmed under the one stress test in this study that has previously separated a working low-rank scheme from a failing one.

Table 3: sMNIST final-epoch accuracy, $n = 3$ seeds; last two columns are gap/SE. "savings" is total io-parameter reduction vs. `free`'s 12288 (gate-only for `cdg`; shared_basis also factors W_{ssm}/W_{out}, not comparable at matched k/r). cdg_residual r=32 pushes io params to 16384, *above* `free`'s (a net cost). low_rank_gate's io params match `cdg` exactly here (d=e); its gap/SE vs. `cdg` is $-2.18/-0.46/-0.07$ at $r = 8, 16, 32$ – $r = 8$ crosses the significance bar (Section 6.6).

Mode	savings	Final acc. (mean ± std)	vs. free	vs. tie_ssm_gate
free	0.0%	90.78% ± 1.40	–	+2.10
tie_ssm_gate	33.3%	87.92% ± 1.35	−2.10	–
cdg, $r = 8$	25.0%	90.93% ± 0.35	+0.18	+3.74
cdg, $r = 16$	16.7%	91.58% ± 1.03	+0.79	+3.73
cdg, $r = 32$	0.0%	90.34% ± 0.82	−0.47	+2.66
shared_basis, $k = 8$	79.2%	87.28% ± 2.67	+2.02	+0.37
shared_basis, $k = 16$	66.7%	89.58% ± 0.50	+1.40	−1.99
shared_basis, $k = 32$	41.7%	88.89% ± 0.05	+2.33	−1.25
cdg_residual, $r = 8$	16.7%	90.87% ± 1.30	−0.08	−2.73
cdg_residual, $r = 16$	0.0%	89.21% ± 1.22	+1.47	−1.23
cdg_residual, $r = 32$	−33.3%	90.82% ± 0.14	−0.05	−3.70
low_rank_gate, $r = 8$	25.0%	90.27% ± 0.39	−0.61	+2.90
low_rank_gate, $r = 16$	16.7%	91.13% ± 1.33	+0.31	+2.94
low_rank_gate, $r = 32$	0.0%	90.27% ± 1.61	−0.42	+1.94

Mode	savings	Val bpb (mean ± std)	gap/SE vs. free
free	0.0%	1.8939 ± 0.0061 ($n = 3$)	–
CDG, $r = 128$	25.0%	1.9431 ± 0.0055 ($n = 3$)	−10.31
cdg_residual, $r = 128$ (3 seeds)	18.8%	2.1071 ± 0.2624 ($n = 3$)	−1.41
cdg_residual, $r = 128$ (2 clean seeds)	18.8%	1.9556 ± 0.0006 ($n = 2$)	−17.30
low_rank_gate, $r = 128$	27.1%	1.9158 ± 0.0028 ($n = 3$)	−5.63

Table 4: 130M flagship, real `mamba_ssm` kernels, 10K steps. "savings" is total io-parameter reduction vs. `free`'s 125,829,120. The two cdg_residual rows are the same three seeds read two ways (collapse discussion below). low_rank_gate has fewer io params here (91,750,400 vs. CDG's 94,371,840, $d \neq e$ at this scale) – so the result below is harder on CDG, not easier: it loses despite the larger budget.

6.3 Sequential MNIST

The real-task check confirms the synthetic-task pattern with the largest standardized differences yet: no CDG rank shows a meaningful gap from `free`, while every rank beats `tie_ssm_gate` by a large margin. Shared basis is weaker here too (table below, not discussed stage-by-stage further; Section 7). The residual variant again tracks plain CDG: no rank shows a meaningful gap from `free` ($r = 16$'s $+1.47$ stays just under the bar), and all three beat `tie_ssm_gate`. Unlike the smaller stages, the correction is no longer free at this task's smaller $d_{model} = 64$: its own cost grows faster than the router's savings, so $r = 32$ is 33.3% *more* expensive than `free` – the first cell in this study where cdg_residual costs more than it compresses.

The plain low-rank-gate ablation completes its final stage here, and the pattern does not hold cleanly throughout. At $r = 8$ – the steepest cut – low_rank_gate underperforms plain `cdg` at matched parameter count by a large margin (90.27% vs. 90.93%, -2.18 gap/SE), the first cell in either ablation stage to cross the bar. At $r = 16, 32$ the gap from `cdg` shows no large difference (-0.46, -0.07), matching the earlier pattern. low_rank_gate still clearly beats `tie_ssm_gate` at $r = 8, 16$ ($+2.90/+2.94$), though at $r = 32$ that margin falls just short ($+1.94$). None of the three ranks shows a large gap from `free` in either direction.

6.4 130M-parameter SlimPajama byte-LM

At the planned 1000-step budget, single seed, CDG ($r = 128$) trails `free` by a small but real margin ($\sim$2.0% relative val bpb) – the first stage where CDG does not simply match `free` – and the gap persists (if anything slightly larger, $\sim$3.0% relative) when re-run step-matched at 10000 steps, ruling out a short-budget artifact. Shared basis's own flagship run is far worse still ($\sim$32.7% relative, the worst result at any stage in this study). Full curves and figures for this single-seed preliminary check are in Appendix A.8; both findings are reconfirmed, at 3 seeds on real fused kernels, next.

6.5 Real-kernel, multi-seed confirmation

Every result above this point was obtained on this study's own pure-PyTorch scan (Sec 5), at a single seed for the 130M flagship. Both are addressed together here: `free`, CDG ($r = 128$), the residual variant ($r_s = 128$), and low_rank_gate ($r = 128$) were rerun at the identical architecture/step budget on `mamba_ssm`'s actual CUDA kernels rather than the custom scan, at 3 seeds each.

The free-vs-CDG gap is confirmed, essentially unchanged from the single-seed measurement: 2.60% relative here vs. 3.05% there, now backed by $n = 3$ seeds with a tight spread (std ≤ 0.0061)

– gap/SE -10.31 is among the largest standardized differences in this study. The gap is not a scan or seed-variance artifact.

The plain low-rank-gate ablation's flagship result is the most surprising finding in this paper, reversing the sMNIST reading (Section 6.3). low_rank_gate *underperforms* `free` by a large margin (-5.63 gap/SE, 1.16% relative) – expected, since it removes the same gate CDG does – but *outperforms* CDG itself by a large margin ($+7.62$ gap/SE, 1.40% relative, tighter spread than either), despite fewer gate parameters. A content-blind gate closes more than half of CDG's own gap to `free` (1.16% vs. 2.60%). Content-derivation is not merely unhelpful at flagship scale; it actively costs accuracy against a cheaper alternative.

This run is also the first point with an actual efficiency measurement, not just a parameter count: CDG trains $\sim$19% faster than `free` (0.304s vs. 0.366s/step) with $\sim$23% less peak memory (2.07GB vs. 2.70GB); low_rank_gate is fractionally cheaper still (0.292s/step, 2.02GB), consistent with its smaller gate. A FLOP-accounting sanity check confirms this wall-clock measurement is genuine gate-compute savings rather than a harness artifact (Appendix A.5).

The residual correction did not improve on plain CDG here and showed a sharp, unrecovered late-training instability in one of three seeds – the first outright instability at 130M scale in this study, enough alone to weigh against recommending it (Appendix A.2). A single-block cross-check on real kernels reproduced the custom-scan numbers for every mode, including shared basis's $k = 4$ collapse to three decimals: the story does not depend on scan implementation.

6.6 Plain Low-Rank Gate ablation

The single-block, stacked-depth, and sMNIST results are reported inline above (Sections 6.1–6.3), since low_rank_gate vs. `cdg` is a same-table comparison at each stage. The first two stages show no large difference (largest gap/SE -1.70, $L = 3, r = 4$) – consistent with CDG's success being attributable to being low-rank rather than content-derived. sMNIST breaks that pattern: at $r = 8$, the steepest compression, low_rank_gate underperforms `cdg` by a large margin (-2.18 gap/SE, the first cell to cross the bar), while $r = 16, 32$ show no large difference. This $r = 8$ result rests on $n = 3$ seeds and was not independently reconfirmed the way the flagship result was, so we read it as suggestive rather than conclusive – consistent with this study's pattern that failure modes appear on the real-task check or at scale, never on synthetic tasks alone.

The ablation's fourth and final stage reverses that pattern. Rerun at the 130M flagship on real kernels across 3 seeds, the plain low-rank gate *outperforms* CDG by a large margin ($+7.62$ gap/SE) despite a smaller router, closing more than half of CDG's gap to `free`. At the one scale large enough to inform deployment, content-derivation is a measurable liability, not the isolated asset it was at sMNIST or the non-factor it was elsewhere. The regime that matters most for practical use is the one where content-derivation hurts.

7 Discussion and Conclusion

The result that matters most is not about CDG specifically: a derived-routing scheme that looks like a clean win at small scale can have that win reattributed, at the scale that matters for deployment, to mere compression rather than derivation – invisible without the content-blind control this paper introduces. CDG matches the full gate at no measurable cost below 130M parameters and, even at flagship scale, stays within a small persistent gap while training $\sim$19% faster and using $\sim$23% less peak memory (gap/SE -10.31); neither follow-up fix closes that gap, and the plain-low-rank-gate ablation shows content-derivation is inert at loose compression but loses to a content-blind gate by a large margin ($+7.62$ gap/SE) at 130M – evidence that CDG's routing-from-u_t design itself, not just its capacity, is part of the flagship-scale cost. Proposition 1 rules out expressivity as the explanation and points to optimization dynamics instead, tested directly by the Subspace-Alignment Gate and overlap analysis (Sections 4.4–4.5); the sMNIST $r = 8$ result rests on only $n = 3$ seeds and the shared-basis flagship result is single-seed, so both should be read with appropriate caution. The strongest conclusion is therefore that the gate's signal can be made low-rank (H2), not that it must specifically be content-derived (H3): CDG is one particular low-rank mechanism, not the source of the low-rank gate's success, giving a concrete recipe (content-derived below flagship scale, plain low-rank at flagship) and a broader lesson – architectural changes validated at small scale should not be trusted to transfer without testing at the scale, kernel, and depth they ultimately target.

8 AI USE STATEMENT

An AI coding agent (Claude, Anthropic) was used extensively throughout this project's experimental and writing process, under the direction and review of the human author(s) at every stage. The agent implemented the experiment code for every mode and scale reported here, including the pure-PyTorch and real-`mamba_ssm`-kernel Mamba-block variants; launched, monitored, and resumed the training runs whose logged results populate every table and figure; and computed the statistical comparisons (means, standard deviations, and gap/SE figures) directly from the resulting result files. It also identified and corrected an error in an earlier draft's parameter-count accounting for the low-rank-gate ablation (Section 4.3), which required revising two result tables, and drafted and iteratively revised this manuscript's text based on the underlying results and the author's direction. The human author(s) formulated the research questions, directed each experimental stage, reviewed the code and results for correctness, made the final judgment calls on experimental design and interpretation, and take full responsibility for the accuracy and integrity of everything reported here. No result in this paper was fabricated, extrapolated, or altered beyond what the underlying logged training data supports.

9 REPRODUCIBILITY STATEMENT

Every architecture, hyperparameter, rank grid, seed count, and training budget used to produce the results in Section 6 is stated explicitly in Section 5, including the exact statistical convention (gap/SE formula and significance threshold) used for every comparison in the paper. The two datasets used are both public and standard: MNIST (Section 6, sequential-pixel task) and SlimPajama (byte-level, Section 6.4). The Content-Derived Gating router and its integration into a Mamba block are given as complete, runnable PyTorch code in Appendix A, and the exact per-cell statistics omitted from the main tables for space are given in Appendix A.1 and Appendix A.2. Free and hard-tied baselines are reused directly from our own existing (unpublished) results rather than rerun, as noted in Section 5; every other number in this paper comes from a training run whose seed is reported alongside it. The real-kernel results (Section 6.5) additionally require the `mamba_ssm` and `causal-conv1d` packages and a CUDA-capable GPU, noted where first used. Code and raw result files will be released upon publication; they are withheld from this submission to preserve double-blind review.

REFERENCES

Ali, S. H., Cappellazzo, U., and Ravanelli, M. MambAdapter: Lightweight Mamba-Based Adapters for Parameter-Efficient Transfer Learning in Speech and Audio. *arXiv:2606.15638*, 2026.

Dao, T. and Gu, A. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. *arXiv:2405.21060*, 2024.

Gao, X. Keyless Attention: Value-Space Routing and Value-Only Caching for Efficient Transformers. *arXiv:2606.21848*, 2026.

Gu, A. and Dao, T. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv:2312.00752*, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685*, 2021.

Huang, N., Sarabia, M., Moudgil, A., Rodriguez, P., Zappella, L., and Danieli, F. Understanding Input Selectivity in Mamba: Impact on Approximation Power, Memorization, and Associative Recall Capacity. *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Press, O. and Wolf, L. Using the Output Embedding to Improve Language Models. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2017.

Qiu, Z., Wang, Z., Zheng, B., Huang, Z., Wen, K., Yang, S., Men, R., Yu, L., Huang, F., Huang, S., Liu, D., Zhou, J., and Lin, J. Gated Attention for Large Language Models: Non-linearity, Sparsity, and Attention-Sink-Free. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Zhan, Z., Ren, L., Wang, S., Liu, L., Liu, Y., Gong, Y., Wang, Y., and Shen, Y. Routing Mamba: Scaling State Space Models with Mixture-of-Experts Projection. *arXiv:2506.18145*, 2025.

A PyTorch Implementation

The Content-Derived Gating router (Section 4.1), as actually implemented and used to produce every result in Section 6:

```
class ContentDerivedGate(nn.Module):
    def __init__(self, expand_dim, rank=128):
        super().__init__()
        self.down = nn.Linear(expand_dim, rank, bias=False)
        self.up = nn.Linear(rank, expand_dim, bias=False)

    def forward(self, u):
        return self.up(self.down(u))
```

Integrated into a Mamba block, replacing the independent gate projection:

```
# Original                        # Content-Derived Gating
u = self.ssm_proj(x)              u = self.ssm_proj(x)
g = self.gate_proj(x)             g = self.content_gate(u)
h = self.ssm(u)                   h = self.ssm(u)
y = self.out_proj(F.silu(g)*h)    y = self.out_proj(F.silu(g)*h)
```

The Plain Low-Rank Gate ablation (Section 4.3) reuses `ContentDerivedGate` unchanged, constructed with `expand_dim` set to the block's inner dimension exactly as above, but called on x_t (the block's residual-stream input) in place of u_t – a one-line change at the call site, with no modification to the module itself, keeping the two gates' implementations identical apart from which tensor they see.

A.1 Stacked-Depth Per-Cell Standardized-Difference Detail

Full per-(rank, depth)-cell gap/SE vs. `tie_ssm_gate`, referenced from the stacked-depth table's caption (Section 6.2): for `cdg` $r = 1, 2, 4$, gap/SE is $-3.87/-2.74/+0.10$ at $L = 2$ and $-2.60/-3.97/-1.12$ at $L = 3$ (large in 4 of 6 cells, always in CDG's favor). For shared_basis $k = 4, 8, 16$, gap/SE is $+10.81/-0.31/+0.89$ at $L = 2$ and $+6.63/+0.17/-2.20$ at $L = 3$ (large in 2 of 6, both at $k = 4$'s expense; the lone win is $k = 16$ at $L = 3$). For cdg_residual $r = 1, 2, 4$, gap/SE is $-3.63/+0.56/-0.30$ at $L = 2$ and $-3.33/-4.11/-3.58$ at $L = 3$ (large in 4 of 6, always in its favor, tracking plain CDG). For low_rank_gate $r = 1, 2, 4$, gap/SE vs. `tie_ssm_gate` is $-4.29/-4.92/-2.70$ at $L = 2$ and $-3.85/-4.50/-4.63$ at $L = 3$ (every cell large, always in its favor); its gap/SE vs. plain `cdg` at matched rank is $-0.05/-0.48/-1.02$ at $L = 2$ and $-0.77/-0.33/-1.70$ at $L = 3$ (none large).

A.2 cdg_residual's 130M Late-Training Collapse

Full detail referenced from Section 6.5: of cdg_residual's three 130M real-kernel seeds ($r = 128$, $r_s = 128$), two (val bpb 1.9560, 1.9552) land right next to CDG's own range. Seed 0 trained cleanly and comparably (val bpb tracking $\sim$1.94–1.98 through step 9700) before a sharp late-training divergence – val bpb $1.9413 \to 2.2299 \to 2.5789 \to 2.4100$ across the final three eval points (every 100 steps from step 9700 to step 10000) – that never recovered by step 10000. Across the final four eval points of all eight other seeds run on real kernels in Section 6.5 (all three `free`, all three CDG, and cdg_residual's own two clean seeds), the largest eval-to-eval move observed is 0.0153 bpb, against 0.29 then 0.35 bpb here – roughly 19–23$\times$ that ceiling, and the first outright training instability observed at the 130M scale by any mode in this study (`free` and CDG stayed smooth across all three seeds, and no small-scale cdg_residual run showed anything like it either).

A.3 Proof of Proposition 1

Write $M = U_0 N_0$ for some rank factorization ($U_0 \in \mathbb{R}^{e\times r}$, $N_0 \in \mathbb{R}^{r\times d}$). Since $\mathrm{rank}(W_u) = d$, $W_u^\top W_u \in \mathbb{R}^{d\times d}$ is invertible; set $V^\top = N_0(W_u^\top W_u)^{-1}W_u^\top$ and $U = U_0$. Then $UV^\top W_u = U_0 N_0 (W_u^\top W_u)^{-1} W_u^\top W_u = U_0 N_0 = M$.

A.4 Full-Rank Check for W_u at Flagship Scale

Referenced from Section 4.1: $e \geq d$ at every width used in this study (Section 5), and a trained W_u generically has full column rank d. We do not merely assume this: for the one pretrained checkpoint available at flagship scale (`free`, 130M, Section 6.4), every block's W_u has smallest singular value ≥ 0.25 against a largest of ≤ 4.08 (ratio ≥ 0.06, zero singular values below $10^{-4}\times$ the largest) – comfortably full rank, not merely nominally so.

A.5 Real-Kernel Efficiency Detail

Full detail referenced from Section 6.5.

Mode	Total params	tok/s	step time	peak mem	io-FLOPs/tok	val bpb
free	133.9M	5,597	0.366s	2.70GB	251.7M	1.890
CDG, $r = 128$	102.4M	6,737	0.304s	2.07GB	188.7M	1.949
low_rank_gate, $r = 128$	99.8M	7,011	0.292s	2.02GB	183.5M	1.919

Table 5: 130M flagship, real `mamba_ssm` kernels, seed 0 of the 3 seeds in the main-text table (tok/s and step time are wall-clock measurements, not repeated per-seed; val bpb here is seed 0's own final value, for consistency with the other columns measured on that run). "Total params" is the whole model (embedding, all three blocks' matrices, SSM internals, head), *not* the gate-only or io-only figures used for "savings" elsewhere in this paper (Section 6.1); tok/s = batch$\times T$ / step time $= 4 \times 512$ / step time. "io-FLOPs/tok" is the estimated forward-pass multiply-add FLOPs of the content, gate, and output projections only (the three matrices this paper's savings claims are about), $2\times$ their combined parameter count per token – it excludes the selective-scan kernel, convolution, and $\Delta/B/C$ projections, which are identical across all three modes and so cannot explain any difference between rows.

The io-FLOPs/tok column is a sanity check on the wall-clock claim in Section 6.5, not just a restatement of it: CDG's io-projections need exactly 25.0% fewer FLOPs/token than `free`'s (188.7M vs. 251.7M – matching this study's own gate-only parameter-savings figure for $r = 128$ exactly, Section 6.1, since every projection involved is linear), yet its wall-clock speedup is only $\sim$19%, smaller than that FLOP reduction. That gap is expected, not a discrepancy: the selective-scan kernel, causal convolution, and $\Delta/B/C$ projections are identical across all three modes and contribute a fixed per-step cost the gate change cannot touch, diluting the io-projection FLOP savings' effect on total wall-clock time. The direction and rough magnitude of the measured speedup are therefore consistent with genuinely removing gate compute, not an artifact of the real-kernel harness – and low_rank_gate gets the same efficiency profile as CDG without CDG's accuracy cost.

A.6 Real-Kernel Parity Check

Referenced from Section 5. The real-kernel block keeps its content/gate/output projections and tying structure identical to the custom-scan implementation used everywhere else in this paper, but otherwise mirrors `mamba_ssm`'s own mixer scope exactly (norm/residual-free). A parity check against a stock `mamba_ssm.modules.mamba_simple.Mamba` block, given identical weights, matches to within 2.98×10^{-8} absolute (float32, CUDA) – confirming the split-matrix real-kernel plumbing is wired correctly before trusting any comparison built on it.

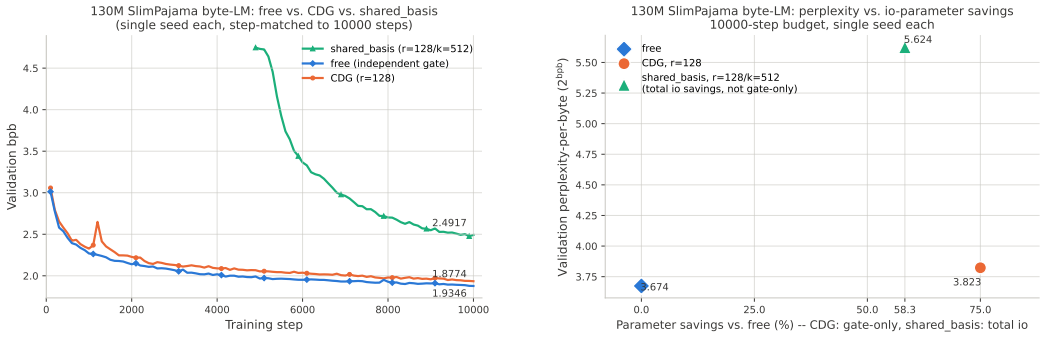
A.7 Statistical Convention: Full Definition and Rationale

Referenced from Section 5. Every "gap/SE" figure in Section 6 is a *standardized difference* (a two-sample z-style statistic), not a hypothesis test: given two modes' seed means m_1, m_2 and sample standard deviations s_1, s_2 over n_1, n_2 seeds, gap/SE $= (m_1 - m_2)/\sqrt{s_1^2/n_1 + s_2^2/n_2}$, with no

seed pairing (each mode's seeds are trained independently, not matched by initialization) and no correction for the many rank/basis/depth cells swept in each table. Because this paper reports dozens of such comparisons per table without a multiple-comparison correction, we deliberately avoid the words "significant"/"significance" throughout and do not treat $|\text{gap/SE}| \gtrsim 2$ as a p-value-backed claim. Instead we use $|\text{gap/SE}| \gtrsim 2$ only as a magnitude cutoff for prose, and describe cells above it as showing a *large standardized difference* (or, for the losing side, a *large standardized deficit*) and cells below it as showing *no large standardized difference* – i.e., a difference too small, relative to seed variance, to describe as a directional effect at a glance. This threshold is a heuristic consistent with this project's prior work, not a formally corrected level, and it is reported here purely as an effect-size indicator rather than a test statistic, most explicitly where $n \leq 3$ (every 130M-scale comparison, and the sMNIST stage). Every mean is reported with its standard deviation in every table, and the underlying seed values are given directly in Appendix A.1 and Appendix A.2 wherever they are not already in the main tables, so a reader wanting a corrected or interval-based analysis (e.g. bootstrap or t-based confidence intervals) has what is needed to compute one; we leave that formal treatment to future work rather than attempt it here.

A.8 130M Single-Seed Preliminary Check

Referenced from Section 6.4. At the planned 1000-step budget, single seed, CDG ($r = 128$) trails `free` by a small but real margin (2.3269 vs. 2.2818 val bpb, ∼2.0% relative) – the first stage where CDG does not simply match `free`. Re-run step-matched at 10000 steps (single seed each), `free` remains ahead by ∼3.0% relative – if anything a slightly larger gap, so this is not a short-budget artifact. Shared basis's own flagship run (10000 steps, r=128/k=512) reaches val bpb 2.4917, a ∼32.7% relative gap behind `free` – an order of magnitude worse than CDG's, and the worst result at any stage in this study, consistent with the collapses already seen at smaller scale.



(a) Validation bpb over training.

(b) Perplexity vs. parameter savings.

Figure 1: 130M single-seed results at the confirmed 10000-step budget. (a) The `free`/CDG gap is present from early training and holds roughly steady rather than closing; shared_basis's curve begins around step 4900 because a log-corrupting disk/IO fault lost its earlier steps (the run itself trained the full budget continuously). (b) The shared_basis triangle is plotted at its own 58.3% *total* io-parameter savings, not CDG's gate-only x-axis convention (Section 6.1), so only its y-position (perplexity, far above both other points) is on the same footing as them.