Elastic Spectral State Space Models for Train-Once Budgeted Inference

Dachuan Song[1], **Junyu Yin**[2], **Zechen Hu**[1], **Xuan Wang**[1*]
[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
George Mason University, Fairfax, VA, USA

Abstract

Modern sequence models are typically trained at a fixed computational capacity, while real-world applications require deployment across platforms with different resource constraints. Existing approaches either train or distill separate compact models for selected budgets, or use elastic architectures that shrink dimensions such as width, depth, Feed-Forward Network (FFN) size, or SSM state dimension. In this paper, we propose Elastic Spectral State Space Models (ES-SSM), a train-once, export-many sequence modeling framework that gains elasticity through spectral approximation of the SSM sequence operator. ES-SSM builds on Hankel spectral filtering for state space models, where long-range token mixing is represented through fixed Hankel spectral channels that define an operator-level approximation resolution. To make this spectral elasticity reliably deployable, ES-SSM combines input-adaptive channel-wise gates with budget dropout, training the same spectral prefixes that are used by compact models at inference time. This encourages low-index channels to become predictive on their own, while higher-index channels provide refinements for larger budgets. We evaluate ES-SSM across byte-level language modeling, Long Range Arena, Speech Commands V2, and offline reinforcement learning benchmarks. Across these settings, a single trained ES-SSM can be truncated to competitive compact models compared with modern Transformer and SSM baselines at similar parameter scales. Furthermore, by testing under various runtime budgets, we observe smooth and stable quality-cost curves over a wide range of truncation levels.

1 Introduction

Modern sequence models are typically trained at a fixed computational capacity. However, real-world applications require AI deployment across heterogeneous computational platforms, from high-throughput cloud clusters to compute-limited endpoints and power-constrained edge devices [1]. This calls for models that can adapt to different trade-offs between inference cost and predictive performance [2, 3]. Common approaches address this challenge by training a family of model variants at different sizes [4, 5], or by distilling a large teacher into one or more smaller students [6–9]. Although these methods can produce compact models for selected budgets, each supported capacity typically requires additional training, calibration, or fine-tuning.

Elastic neural networks provide a more flexible alternative by training a single model that supports multiple deployment budgets. Slimmable Networks and Once-for-All vary architectural choices such as width and depth [4, 5], while Matryoshka representation learning, MatFormer, and MatMamba train nested representation dimensions, FFN widths, or SSM-dimension slices to be predictive [10–12]. However, their elasticity is primarily architectural: smaller models are obtained by narrowing or

*Corresponding author: xwang64@gmu.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

removing parts of the network architecture. The choice of which architectural components to shrink is indirectly related to the final model quality, and therefore often requires careful design, training, or calibration.

In this paper, we study elasticity from a new operator-approximation perspective. For long-sequence modeling, especially in state space models (SSMs), the key computation is long-range token mixing, which can be represented by a convolutional sequence operator. This suggests an alternative route to elasticity: *controlling the approximation resolution of the sequence operator*, rather than only shrinking architectural components. Hankel spectral SSMs make this possible by expressing the operator using fixed spectral channels derived from a Hankel matrix [13, 14]. Under this representation, elasticity is associated with a direct operator-level interpretation: using fewer spectral channels corresponds to a lower-resolution approximation of the sequence operator. However, this spectral-resolution axis is not automatically deployable. A spectral SSM trained only at full capacity may distribute task-relevant information across many channels. As a result, naive channel truncation can discard important learned components and lead to sharp performance degradation.

Statement of Contributions. To address the above issues, we propose Elastic Spectral State Space Models (ES-SSM), which train a full-capacity spectral SSM once and export compact models by direct prefix truncation along the Hankel spectral-channel axis. The main contributions are:

• **An elastic spectral SSM architecture.** We introduce a spectral SSM sublayer built on fixed Hankel spectral channels, learned channel projections, and input-adaptive channel-wise gates. The fixed spectral channels provide an ordered approximation axis for sequence mixing, while the learned projections and gates determine how strongly each channel contributes to the current representation. This design preserves a prefix-structured channel axis, so compact models can be obtained by truncating later spectral channels rather than redesigning or reordering the model.

• **A budget-dropout training mechanism for reliable prefix truncation.** We train ES-SSM under randomized spectral budgets that execute different retained prefixes of the spectral-channel axis during training. As a result, the same prefix computations used by exported compact models are repeatedly optimized before deployment. This training mechanism encourages early channels to carry useful predictive information independently, while later channels learn to improve the representation when larger budgets are available.

• **Train-once, export-many deployment and evaluation.** For each runtime budget, ES-SSM exports a standalone compact model by keeping only the corresponding spectral-channel prefix and its associated parameters, without budget-specific retraining, channel search, or reordering. We evaluate ES-SSM on byte-level SlimPajama against elastic and non-elastic Transformer and SSM baselines, and extend the same exported-compact-model protocol to Long Range Arena, Speech Commands V2, D4RL MuJoCo offline reinforcement learning, and a byte-level generality replication. Across these settings, ES-SSM demonstrates superior or comparable performance relative to state-of-the-art elastic methods in quality–cost trade-off curves, sweet spots and collapse boundaries, and latency and memory measurements, while incurring substantially lower multi-budget training cost than independently trained non-elastic baselines.

2 Related work

Budget-adaptive and elastic sequence models. A growing body of work studies how to adapt neural networks to different inference budgets. One class of methods produces compact models for selected deployment budgets. Knowledge distillation [6, 7, 15] transfers behavior from a larger teacher into one or more separately trained smaller students. Conditional computation [16, 17] changes the amount of active computation per input or token, for example by routing through a subset of modules or tokens, but typically does not export a standalone compact model with fewer retained parameters, so deployment may still require storing the full model. Structured pruning [18, 19] removes heads, channels, layers, or other parameter groups from a trained model. They often require additional selection criteria, calibration, or fine-tuning to recover quality.

Another line of work trains a single model that contains multiple nested subnetworks. Slimmable Networks [4] train a single network that supports multiple width multipliers, with shared weights and switchable normalization so that each width can be evaluated as a valid subnetwork. Once-for-All [5] extends this idea to a larger architectural search space, where subnetworks at different depths,

widths, kernel sizes, or resolutions are inherited from one trained supernet. Structured dropout over Transformer layers enables layer-pruned shallower subnetworks to be used at inference time [20]. Matryoshka Representation Learning introduces nested feature-vector prefixes, where each prefix is supervised to be predictive. MatFormer [11] adapts this nesting principle to Transformer language models through nested FFN blocks, while MatMamba [12] introduces nested capacity along Mamba-2-style SSM dimensions. These elastic methods primarily vary architectural axes such as width, depth, FFN size, representation dimension, or SSM hidden dimension. In contrast, the proposed ES-SSM achieves elasticity through the Hankel spectral channels of the SSM sequence operator. This operator-level elasticity is complementary to architectural nesting.

Hankel spectral filtering and spectral SSMs. In addition to attention-based modeling, state space models (SSMs) have re-emerged as an efficient alternative for long-sequence modeling. SSMs view sequence processing as a dynamical system that repeatedly compresses new inputs into latent states, which can serve as a stable long-range memory mechanism [21, 22]. Subsequent work has improved the trainability, numerical robustness, and computational efficiency of SSMs through structured parameterizations, fast convolution, and scan-based implementations [23–26]. These developments show that carefully parameterized linear recurrent layers can achieve strong long-context performance with favorable scaling in sequence length compared with attention-based models [27]. However, in most existing formulations, the training and deployment compute budgets are fixed by architectural choices such as the number of channels or state dimensions. Reducing inference-time cost therefore typically requires training a smaller model variant or applying a separate compression or pruning procedure.

Building on SSMs, a distinct line of research constructs sequence operators from Hankel structure and spectral filtering. It reformulates SSM channels in the spectral domain while retaining provable learning guarantees in dynamical settings [13, 28]. Recent spectral SSMs build on this perspective by fixing a Hankel spectral eigenbasis and learning a small set of mixing parameters over precomputed filters, enabling efficient long-range sequence computation [14]. Because the basis is fixed and ordered, this formulation induces a natural notion of spectral resolution: the sequence operator can be represented using a prefix subset of spectral components.

However, prior work typically trains and evaluates models at a single fixed spectral resolution, treating truncation mainly as a post hoc approximation. In contrast, our deployment setting requires truncation to remain reliable when all runtime budgets are derived from one trained set of parameters.

3 Preliminaries

Setup and notation. Let $u_{1:L} = (u(1), \dots, u(L))$ denote an input sequence of length L, with $u(t) \in \mathbb{R}^{d_{\text{in}}}$, and let $y_{1:L} = (y(1), \dots, y(L))$ denote the layer output, with $y(t) \in \mathbb{R}^{d_{\text{out}}}$. Here d_{in} and d_{out} are the input and output feature dimensions, respectively. In residual model blocks where the sublayer preserves the representation width, we set $d_{\text{in}} = d_{\text{out}} = d$ and refer to d as the model dimension. Whenever an expression refers to indices $t \le 0$, we assume zero-padding. All inner products $\langle \cdot, \cdot \rangle$ are the standard Euclidean inner product, i.e. $\langle v, w \rangle = v^\top w$.

Linear dynamical systems and Hankel spectral basis. A (discrete-time) state space system (SSM) evolves as $x(t) = A\,x(t-1) + B\,u(t)$, and $y(t) = C\,x(t) + D\,u(t)$, with latent state $x(t) \in \mathbb{R}^N$, input $u(t) \in \mathbb{R}^{d_{\text{in}}}$, output $y(t) \in \mathbb{R}^{d_{\text{out}}}$, and matrices $A \in \mathbb{R}^{N \times N}$, $B \in \mathbb{R}^{N \times d_{\text{in}}}$, $C \in \mathbb{R}^{d_{\text{out}} \times N}$, $D \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Assuming zero initial state, the input–output map of SSM admits a causal convolution:

$$y(t) = D\,u(t) + \sum_{\tau=0}^{t-1} G(\tau)\,u(t-\tau) = D\,u(t) + (\mathbf{G} * u)(t), \qquad t = 1, \dots, L, \tag{1}$$

where $G(\tau) = CA^\tau B \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is the *causal kernel* and $\mathbf{G} = \{G(\tau)\}_{\tau \ge 0}$.

Spectral parameterization of SSM. We first introduce the base spectral parameterization used in this work. Consider the length-L causal convolutional kernel $\mathbf{G} = \{G(\tau)\}_{\tau=0}^{L-1}$ induced by a stable linear state-space model. We follow the standard stable symmetric setting used in spectral filtering analyses [13, 14], where A is real symmetric with eigenvalues in $[0, 1)$. Thus A admits the orthogonal eigendecomposition $A = U \Lambda U^\top$, with $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_N)$ and $\lambda_\ell \in [0, 1)$. The causal kernel

can therefore be written in diagonal coordinates as

$$G(\tau) = CA^\tau B = CU\Lambda^\tau U^\top B = \sum_{\ell=1}^{N} \lambda_\ell^\tau \, (CU)_{:\ell} \, (U^\top B)_{\ell:}, \tag{2}$$

where $(CU)_{:\ell} \in \mathbb{R}^{d_{\text{out}}}$ denotes the ℓ-th column of CU, and $(U^\top B)_{\ell:} \in \mathbb{R}^{1\times d_{\text{in}}}$ denotes the ℓ-th row of $U^\top B$. Thus, the matrix-valued kernel decomposes into a sum of scalar temporal traces and input-output coefficient matrices. The only time-dependent factor in each term is the geometric sequence $[1, \lambda_\ell, \lambda_\ell^2, \ldots, \lambda_\ell^{L-1}]$. Therefore, approximating the SSM kernel over a finite horizon reduces to approximating geometric vectors $\nu(\lambda) = [1, \lambda, \lambda^2, \ldots, \lambda^{L-1}]^\top \in \mathbb{R}^L$, with $\lambda \in [0, 1)$.

Spectral filtering provides a universal way to approximate these geometric vectors. Specifically, one constructs a set of Hankel spectral channels $[\phi_1, \phi_2, \ldots, \phi_L \in \mathbb{R}^L]$, which depend only on the sequence length L, not on the hidden dimension N or on a particular realization of A, B, C. We can project the geometric vector $\nu(\lambda)$ onto the span of the first K Hankel spectral channels $\{\phi_k\}_{k=1}^K$ to obtain the low-rank approximation $\nu(\lambda) \approx \sum_{k=1}^K \langle \nu(\lambda), \phi_k\rangle \phi_k$. For each λ^τ in $\nu(\lambda)$, it follows $\lambda^\tau \approx \sum_{k=1}^K \langle \nu(\lambda), \phi_k\rangle \phi_k(\tau)$, where $\phi_k(\tau)$ denotes the $(\tau+1)$-th entry of the channel vector ϕ_k. Substituting this approximation into the causal kernel yields

$$G(\tau) = \sum_{\ell=1}^{N} \lambda_\ell^\tau \, (CU)_{:\ell} (U^\top B)_{\ell:} \approx \sum_{\ell=1}^{N} \sum_{k=1}^{K} \langle \nu(\lambda_\ell), \phi_k\rangle \, \phi_k(\tau) \, (CU)_{:\ell} (U^\top B)_{\ell:} = \sum_{k=1}^{K} M_k \phi_k(\tau)$$

for $\tau = 0, \ldots, L-1$, where each $M_k = \sum_{\ell=1}^N \langle \nu(\lambda_\ell), \phi_k\rangle \, (CU)_{:\ell} (U^\top B)_{\ell:} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ is unknown and collects the input-output coefficient matrices associated with the k-th spectral channel. The detailed derivation of M_k and the Hankel channel construction are provided in Appendix B.1.

Let $\Phi_k = \{\phi_k(\tau)\}_{\tau=0}^{L-1}$ be a fixed causal convolution filter, the corresponding spectral SSM becomes:

$$\hat{y}(t) = \sum_{k=1}^{K} M_k (\Phi_k * u)(t), \tag{3}$$

where $*$ denotes causal convolution. The direct feedthrough term $D\,u(t)$ in equation (1) is a zero-lag causal contribution. Since our convolution convention includes the $\tau = 0$ term, we absorb it into the learned kernel by setting $G(0) \leftarrow G(0) + D$ and keep the notation $\mathbf{G}$ for the absorbed kernel.

Limitations of the base spectral form. The approximation in equation (3) suggests a natural route to elastic SSM deployment through direct prefix truncation. In principle, one could train a model once at a full spectral budget $\overline{K}$ and, for a smaller runtime budget $K \le \overline{K}$, export a compact model that keeps spectral channels $1, \ldots, K$ and removes channels $k > K$. However, direct prefix truncation exposes a key limitation of the base spectral form: Hankel eigenvalue order gives an approximation order for the basis, but full-budget training of equation (3) does not make this order, in terms of information importance, for the learned channel projections $\{M_k\}$. Since the training loss is applied only to the aggregate output $\sum_{k=1}^{\overline{K}} M_k(\Phi_k * u)(t)$, predictive information can be distributed across all channels. In particular, important information may be placed in high-index channels, or encoded through cancellations and compensations between low-index and high-index channels. Consequently, simply removing channels $k > K$ after training may discard essential learned components rather than channels that only serve as refinements. This can cause sharp degradation at small budgets.

4 Methods

We aim to train a single full-capacity ES-SSM with maximum spectral budget $\overline{K}$ and export compact models for smaller runtime budgets $K \le \overline{K}$. As discussed in section 3, direct prefix truncation is useful only if the learned use of Hankel spectral channels respects the prefix order: low-index channels should already form a useful predictor, while later channels should provide higher-capacity refinements. ES-SSM enforces this through two coupled mechanisms: First, it adds input-adaptive, channel-wise sigmoid gates so that each retained Hankel spectral channel can be modulated according to the current representation. Second, it uses budget dropout, where each update trains the same prefix computation that will later be used by an exported compact model. Together, these mechanisms learn one ordered set of channel projections $\{M_k\}_{k=1}^{\overline{K}}$, rather than separate projection sets for each budget.

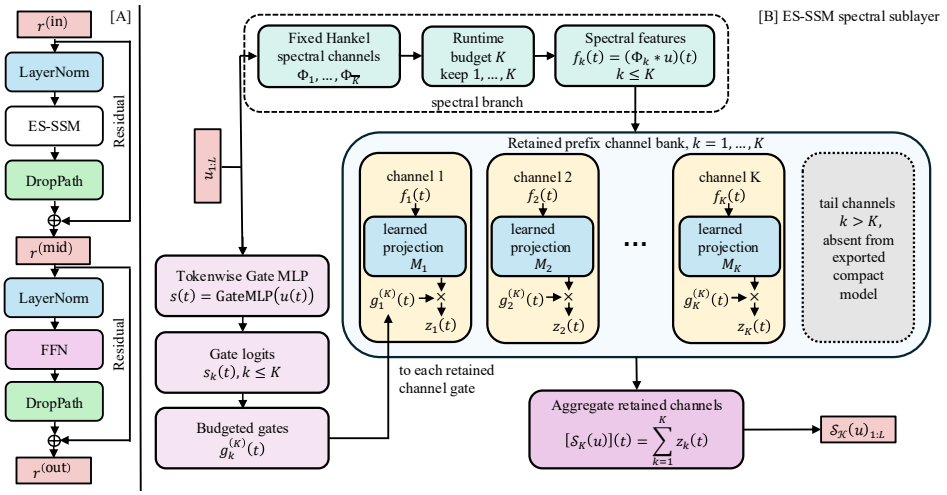

Figure 1: ES-SSM architecture overview. [A] ES-SSM is embedded in a pre-norm residual block with an ES-SSM spectral sublayer followed by an FFN; residual paths carry the block input around each sublayer. [B] The ES-SSM spectral sublayer $\mathcal{S}_K$ combines fixed Hankel spectral channels with a tokenwise Gate MLP that produces input-adaptive, channel-wise budgeted gates. For a runtime budget $K \le \overline{K}$, direct prefix truncation keeps channels $1, \ldots, K$ in the exported compact model and removes tail channels $k > K$. Each retained channel is projected by M_k, modulated by the scalar gate $g_k^{(K)}(t)$, and added to the retained-channel sum. Thus, one full-capacity training can be truncated into compact models at multiple runtime budgets.

At deployment time, for a runtime budget K, we apply direct prefix truncation to keep spectral channels $1, \ldots, K$ and their channel-indexed parameters, and remove channels $k > K$. The exported compact model is therefore physically smaller along the spectral-channel axis and requires no budget-specific retraining, channel search, or reordering. Figure 1 summarizes the pre-norm residual block and the ES-SSM spectral sublayer used inside it. Building on the base spectral form in equation (3), ES-SSM assigns each retained Hankel spectral channel a learned projection M_k and an input-adaptive scalar gate. The projection provides budget-shared channel mixing, while the gate gives the sublayer a tokenwise mechanism to suppress currently irrelevant channels and activate useful spectral refinements. Budget dropout then trains these gated prefixes to remain predictive across runtime budgets. We denote the logistic sigmoid by $\text{sigmoid}(a) = 1/(1 + \exp(-a))$.

4.1 Spectral sublayer

For a sublayer input sequence $u_{1:L}$, the gate logits are produced tokenwise by a lightweight two-layer MLP with GELU activation, $\mathbf{s}(t) = \mathbf{W}_{g,2}\text{GELU}(\mathbf{W}_{g,1}u(t) + \mathbf{b}_{g,1}) + \mathbf{b}_{g,2}$, where $\mathbf{s}(t) = [s_1(t), \ldots, s_{\overline{K}}(t)]^\top \in \mathbb{R}^{\overline{K}}$. The parameters are $\mathbf{W}_{g,1} \in \mathbb{R}^{d_g \times d_{\text{in}}}$, $\mathbf{b}_{g,1} \in \mathbb{R}^{d_g}$, $\mathbf{W}_{g,2} \in \mathbb{R}^{\overline{K} \times d_g}$, and $\mathbf{b}_{g,2} \in \mathbb{R}^{\overline{K}}$, with hidden width $d_g = d_{\text{in}}/2$. We initialize $\mathbf{b}_{g,2} \equiv -2$, so each channel starts with a small gate value, $\text{sigmoid}(-2) \approx 0.12$. This keeps the spectral residual contribution small at initialization and avoids large random channel sums before budget dropout has trained useful prefixes.

For a runtime budget $K \le \overline{K}$, the exported compact model keeps the first K logits. The applied gate coefficient for channel $k \le K$ is $g_k^{(K)}(t) = \frac{1}{\sqrt{K}}\text{sigmoid}(s_k(t))$, where the factor $1/\sqrt{K}$ is intended to keep the accumulated spectral contribution on a comparable scale across budgets. Let $\mathcal{S}_K$ denote the ES-SSM spectral sublayer at runtime budget K. Its output sequence $\mathcal{S}_K(u)_{1:L}$ is defined entrywise by

$$[\mathcal{S}_K(u)](t) = \sum_{k=1}^{K} g_k^{(K)}(t)\; M_k\,(\Phi_k * u)(t), \tag{4}$$

In figure 1, we abbreviate each retained-channel contribution as $z_k(t) \triangleq g_k^{(K)}(t) M_k(\Phi_k * u)(t)$. Here Φ_k is the fixed Hankel spectral channel viewed as a causal kernel sequence, and $M_k \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is

the corresponding learned channel projection. Compared with equation (3), the sublayer adds input-dependent channel modulation and the budget scale $1/\sqrt{K}$, while retaining the same prefix-ordered channel structure.

The spectral sublayer is embedded in a pre-norm residual block. Let $r_{1:L}^{(\mathrm{in})}$, $r_{1:L}^{(\mathrm{mid})}$, and $r_{1:L}^{(\mathrm{out})}$ denote the block input, intermediate sequence, and block output. LayerNorms $\mathrm{LN}_1, \mathrm{LN}_2$ are applied tokenwise, $\mathrm{DropPath}$ denotes stochastic depth on the residual branch, and FFN is a tokenwise two-layer MLP with widening factor 4 and GELU activation. The block updates as

$$r^{(\mathrm{mid})} = r^{(\mathrm{in})} + \mathrm{DropPath}\big(\mathcal{S}_K(\mathrm{LN}_1(r^{(\mathrm{in})}))\big), \;\; r^{(\mathrm{out})} = r^{(\mathrm{mid})} + \mathrm{DropPath}\big(\mathrm{FFN}(\mathrm{LN}_2(r^{(\mathrm{mid})}))\big)$$

The K spectral convolutions $\{(\Phi_k * u)(t)\}_{k=1}^{K}$ can be computed jointly by a single batched FFT. Direct prefix truncation slices every channel-indexed tensor along the spectral-channel axis, including the first K spectral channels, the first K rows of $\mathbf{W}_{g,2}$, the first K entries of $\mathbf{b}_{g,2}$, and the projection stack $\{M_k\}_{k=1}^{K}$. Implementation details on precision, batching, and gate-before-projection ordering are given in Appendix C.

Budgeted inference with exported compact models. At deployment time, ES-SSM exports a compact model for runtime budget K by direct prefix truncation. The exported compact model keeps spectral channels $1, \ldots, K$ and removes channels $k > K$. Equivalently, it keeps $\{\Phi_k, M_k\}_{k=1}^{K}$, the first K rows of $\mathbf{W}_{g,2}$, and the first K entries of $\mathbf{b}_{g,2}$, while retaining shared parameters such as the embedding, $\mathbf{W}_{g,1}$, $\mathbf{b}_{g,1}$, LayerNorms, FFN, and output head. Because retained gate logits $s_k(t)$ for $k \leq K$ do not depend on later channels, the export is mechanically prefix-consistent up to the deterministic scale $1/\sqrt{K}$. Thus, deployment requires no channel search or budget-specific reordering. Next, we introduce budget dropout, which facilitates the above model structure by training the exported prefixes to be effective predictors. The exported-inference procedure is given as Algorithm 1 in Appendix C.1.

4.2 Budget dropout training

Budget dropout trains the same prefixes that will later be exported for budgeted inference. Let $F_\theta(u; K)$ denote the full ES-SSM network evaluated with every spectral sublayer using runtime budget K. By default, at each optimization step, we choose a training budget $K_{\mathrm{train}} \leq \overline{K}$ and minimize $\mathcal{L}_{\mathrm{train}}(\theta) = \mathcal{L}_{\mathrm{task}}(F_\theta(u; K_{\mathrm{train}}), y)$. Thus, the forward pass during training matches the direct prefix computation used by the exported compact model.

Budget dropout changes the gradient allocation across spectral channels. At budget K_{train}, only channels $1, \ldots, K_{\mathrm{train}}$ participate in the spectral computation. For channels $k > K_{\mathrm{train}}$, the channel-specific trainable parameters, including M_k, row k of $\mathbf{W}_{g,2}$, and entry k of $\mathbf{b}_{g,2}$, receive no gradient on that step. In contrast, shared parameters such as embeddings, $\mathbf{W}_{g,1}$, $\mathbf{b}_{g,1}$, LayerNorms, FFN, and the output head are updated whenever they are used by the full network. Since a low-index channel appears in more sampled prefixes than a high-index channel, budget dropout aggregates information towards low-index channels, making them effective for many exported compact models (i.e., different K values). On the other hand, higher-index channels only act as refinements.

We sample K_{train} from a runtime-budget set $\mathcal{K}$ and set $K_{\min} = \min \mathcal{K}$. The training involves three steps. First, during the first E_{warmup} epochs, we set $K_{\mathrm{train}} = \overline{K}$ so that all channel-specific parameters are initialized under the full computation path. Second, after warmup, every n_{full} optimization steps we include a full-budget anchor update with $K_{\mathrm{train}} = \overline{K}$; this ensures that tail channels (at least) periodically receive gradients to maintain their effectiveness. Third, for all remaining steps, we draw a log-scale $\xi \sim \mathrm{Unif}[0, 1]$ (cf. figure 2) between $K_{\min}$ and $\overline{K}$, and choose the nearest element in $\mathcal{K}$:

$$K_{\mathrm{train}} = \arg\min_{k \in \mathcal{K}} \left| k - K_{\min} \, (\overline{K}/K_{\min})^{\xi} \right|. \tag{5}$$

Log-scale sampling treats budgets as multiplicative capacity levels rather than absolute channel counts. It increases training pressure on small-K prefixes where each channel has larger marginal effect, while still covering large budgets through sampling and full-budget anchor updates.

We use $n_{\mathrm{full}} = 8$, giving one full-budget anchor update every eight post-warmup steps; Appendix D and Appendix H.4 explain this setting and analyze the anchor cadence. The budget-dropout training

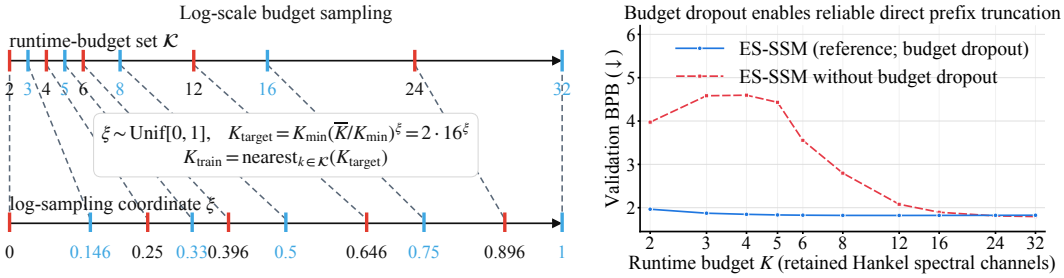

Figure 2: Budget dropout sampler and prefix-truncation behavior. Left: ES-SSM samples a log-scale target K_{target} and snaps it to the nearest element of the runtime-budget set $\mathcal{K}$, placing more training pressure on small-K prefixes while preserving coverage of larger budgets. Right: budget dropout trains the exported prefixes directly, yielding stable validation bits-per-byte (BPB) across runtime budgets; without budget dropout, a model trained only at $\overline{K}$ degrades sharply under direct prefix truncation.

procedure is given as Algorithm 2 in Appendix C.1. In our sensitivity experiments, validation BPB changes little when sampling all integer budgets or removing the full-budget warmup, as detailed in Appendices D.1 and D.2.

5 Experiments

The empirical study has two goals. The first is to verify that, at modern foundation-model scale, training a single ES-SSM at full capacity $\overline{K} = 32$ produces a family of exported compact models whose per-budget quality can approach or exceed independently trained sequence models at the same parameter counts. The second is to verify that the budgeted-inference claim transfers across modalities, not just byte-level language modeling, but also long-range classification, audio, and offline decision-making.

ES-SSM is trained once at the maximum spectral budget $\overline{K} = 32$ and exported to smaller runtime budgets $K \in \{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$ by direct prefix truncation (section 4). For ES-SSM, K has its architectural meaning as the runtime spectral-prefix budget. For baselines, K is only a matched-budget label: each point uses a model with parameter count comparable to ES-SSM at the corresponding budget. Within each benchmark suite, we match the optimizer, schedule, batch size, precision policy, and evaluation protocol across model families so that the comparison isolates architecture. Main tables report both task quality (bits-per-byte (BPB)[2] or task-appropriate accuracy / return) and inference cost (latency, throughput, peak memory) on the same hardware; full hyperparameters and parameter-matching details are provided in Appendix F.

Training-cost accounting. All ES-SSM budget points come from one training run, whereas non-elastic Transformer and SSM baselines require one training run per budget. Thus, ES-SSM pays training cost once and amortizes it across deployment budgets. In a controlled comparison with always-full training, budget dropout reaches matched full-budget validation quality with 17.3% more optimization steps but 30.2% less analytic compute and 39.4% less wall-clock time, as detailed in Appendix D.3.

5.1 Byte-level language modeling at scale

Setup. We train all models on the SlimPajama-627B byte stream [29] at sequence length $L = 2048$ with vocabulary size 258 (256 bytes plus PAD and BOS). The main configuration trains each run for 30 B byte-tokens and uses the measured per-budget parameter counts reported in Appendix G.1, Table 11; the ES-SSM full-budget checkpoint has around 1.3 B parameters. We compare ES-SSM against two prior elastic models, MatFormer [11] and MatMamba [12], and three non-elastic backbones, Transformer++ [30], Pythia-style [31], and GLA [32]. The elastic families train once and export all ten runtime budgets, while the non-elastic baselines are trained independently at four

[2]For a byte sequence $x_{1:T}$, $\mathrm{BPB} = -(T \log 2)^{-1} \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})$.

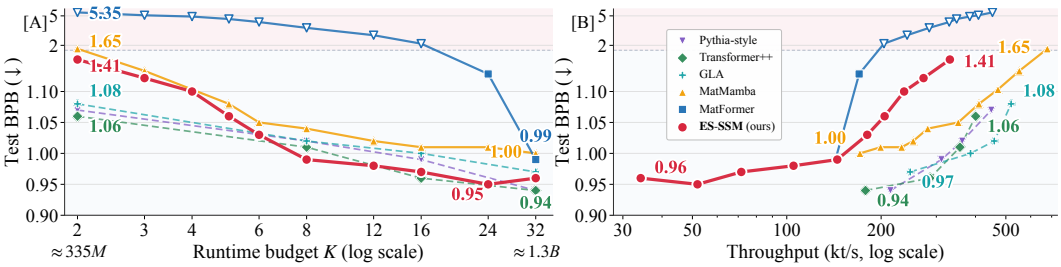

Figure 3: SlimPajama main-table visualization for the 1.3B-family, 30B-token scaling experiment. [A] Test BPB (lower is better) vs. runtime budget K (log scale). [B] Test BPB vs. inference throughput (kt/s, log scale; sequence length L=2048, batch 1, single B200 GPU). The two panels share the same legend on the right. Solid lines denote methods with elastic capabilities, whereas dashed lines denote non-elastic baselines trained **independently** at each shown capacity. We note that these dashed non-elastic baselines require significantly more training time and therefore, should **not** be used for direct comparison with our method. Instead, they are presented as oracle results for reference. Markers are the data points in Appendix G.1, Table 11; for ES-SSM, the ten K values instantiate exported parameter budgets spanning roughly 330 M to around 1.3 B. The light-pink band highlights the high-BPB degradation region (Test BPB > 1.5). Inline values annotate the headline operating points. For readability, both panels use a piecewise-linear y-axis that stretches the competitive band $[0.90, 1.10]$ and compresses the collapse range. Per-family numerical results are reported in Appendix G.1.

reference budgets $K \in \{2, 8, 16, 32\}$. All runs use one NVIDIA B200 GPU. We omit Mamba-2 from the training suite.

Main comparison. figure 3 visualizes the main SlimPajama table from two complementary views: quality as a function of the runtime spectral budget K, and quality as a function of measured inference throughput. Elastic families are evaluated at all ten runtime budgets, whereas the non-elastic families contribute four independently trained, parameter-matched reference points. Detailed numerical results are reported in Appendix G.1, Table 11; full per-benchmark tables are deferred to Appendix G.

Main result analysis. Figure 3 and table 11 show that ES-SSM produces a smooth, usable budget curve from one full-capacity training run. In figure 3A, increasing K steadily improves ES-SSM test BPB, with the curve moving from the low-budget regime toward the competitive band by moderate budgets and reaching its best operating region at large budgets. Importantly, even the smallest ES-SSM prefixes remain outside the collapse zone, while MatFormer suffers severe low-budget degradation and MatMamba remains on a higher-BPB trajectory before approaching its best value only at large K. Figure 3B shows the same behavior from the deployment perspective: reducing K moves ES-SSM toward substantially higher throughput while preserving a controlled quality degradation, so the spectral budget acts as a practical runtime knob rather than only a parameter-count index. At moderate and large budgets, ES-SSM remains competitive with independently trained Transformer++ / Pythia-style / GLA reference points, while avoiding the severe low-budget collapse seen in MatFormer and the higher-BPB plateau of MatMamba. The main reason is that ES-SSM makes the elastic axis the Hankel spectral-channel prefix: budget dropout trains the same prefixes used at inference, and the input-adaptive gates let each retained channel contribute conditionally rather than acting as a fixed spectral mixture. Thus the reported ES-SSM points should be read as exported compact models from one checkpoint, not as separately trained models for each budget.

5.2 Cross-domain validation

To test whether budgeted inference is specific to byte-level language modeling or persists more broadly, we evaluate ES-SSM on four additional settings: long-range classification, audio, offline decision-making, and a byte-level generality replication at around 330 M parameters / 10 B byte-tokens. Figure 4 highlights the quality and training-cost trade-off on offline decision-making, while figure 5 visualizes the LRA budget sweep. Detailed LRA, Speech Commands, D4RL, and byte-level generality tables are reported in Appendix G.2, Appendix G.3, Appendix G.4, and Appendix G.5, respectively.

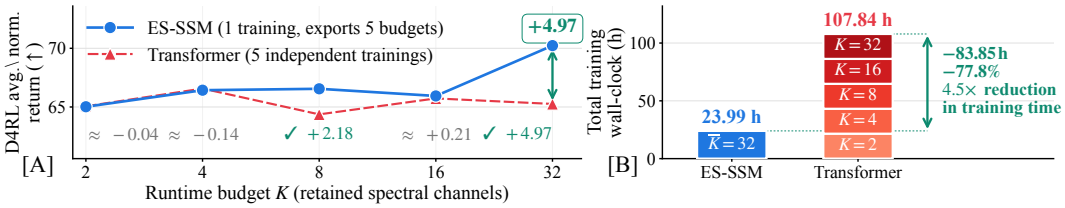

Figure 4: D4RL quality and training-cost comparison. [A] Quality (avg. normalized return, larger means better) vs. runtime budget K. ES-SSM (solid blue) is a single $\overline{K}{=}32$ training exported at five budgets via prefix truncation; Transformer (dashed red) is five *independently trained* models, with the dashed connector serving only as a visual guide. The bottom strip shows the per-K delta (ES-SSM − Transformer); ✓, ≈, × denote win, tie, and loss (tie threshold 0.5 pts). [B] Total training wall-clock on the full D4RL suite (9 datasets × 3 seeds, RTX 3090). The Transformer bar is segmented into its five separate training runs, while ES-SSM trains a single model. The teal arrow reports the savings. Full numerical tables are reported in Appendix G.4.

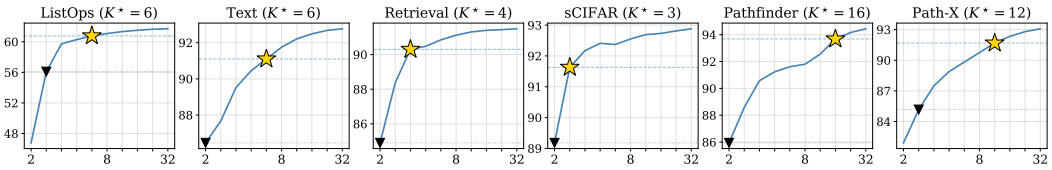

Figure 5: LRA budget sweep at the matched ∼50 M scale. Each curve is obtained by direct prefix truncation from the same $\overline{K}{=}32$ ES-SSM checkpoint. Gold stars mark the smallest K retaining at least 98% of the full-budget score; black triangles mark the smallest tested budget K that retains at least 90% of the full-budget score. Across all six tasks, ES-SSM degrades smoothly as K decreases rather than collapsing abruptly. Full LRA tables are reported in Appendix G.2.

Cross-domain analysis. Across these additional domains, ES-SSM remains competitive while preserving the same deployment rule: low-budget points are exported compact models from a single full-capacity checkpoint, not separately trained models. On LRA, ES-SSM gives the strongest PATH-X result in figure 5 and reaches the 98% sweet spot at $K{=}12$; the full sweep shows gradual degradation rather than abrupt collapse. On Speech Commands V2, ES-SSM is within 0.27 percentage points of Conformer at full capacity, while $K{=}3$ already retains almost all of the accuracy. On D4RL, figure 4A shows that ES-SSM is not merely cheaper to deploy: its exported budgets match the independently trained Transformer models at $K{=}2, 4, 16$ and improve the average normalized return at $K{=}8$ and $K{=}32$, with the largest gain appearing at full capacity. This pattern suggests that the spectral prefix gives useful compact policies at small budgets, while the additional channels at larger budgets provide meaningful refinement for offline control. Figure 4B further separates model quality from training cost: ES-SSM obtains all five D4RL deployment budgets from one training run, reducing total D4RL training wall-clock from 107.84 h to 23.99 h, a saving of 83.85 h (77.8%) and 4.5× reduction in total training time. The byte-level generality replication is a sanity check for the training recipe: ES-SSM reaches the best validation BPB among the three compared architectures at that scale. Together, these results support the central claim that the advantage is not an isolated single-budget win, but a deployable family of compact sequence models obtained by direct prefix truncation.

6 Conclusion

We introduced ES-SSM, a train-once, export-many spectral SSM framework for budgeted sequence modeling. By combining input-adaptive channel-wise gates with budget dropout, ES-SSM makes direct prefix truncation reliable along the Hankel spectral-channel axis, allowing one full-capacity checkpoint to export compact models across runtime budgets. Across byte-level language modeling, LRA, Speech Commands V2, D4RL, and byte-level generality checks, ES-SSM yields smooth quality–cost curves from one training run, avoids severe low-budget collapse in the main scaling

experiment, achieves strong LRA and audio results, and substantially reduces multi-budget training cost in D4RL while remaining competitive with independently trained baselines. Future work includes larger-scale training, hybrid elastic architectures, and further efficiency optimization of the spectral branch; limitations and broader impacts are discussed in Appendix A.

Acknowledgments and Disclosure of Funding

This work was conducted at the Multi-agent Intelligence, Control, and Optimization (MICO) Lab at George Mason University. The MICO Lab is supported in part by the Army Research Office under award W911NF2220242 and the National Science Foundation under awards CMMI-DCSD-2540134 and ECCS-EPCN-2332210. The authors declare no competing interests.

References

[1] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[2] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[3] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[4] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[7] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[8] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of the association for computational linguistics: EMNLP 2020*, pages 4163–4174, 2020.

[9] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.

[10] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35: 30233–30249, 2022.

[11] Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. Matformer: Nested transformer for elastic inference. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 140535–140564. Curran Associates, Inc., 2024. doi: 10.52202/079017-4461. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/fe066022bab2a6c6a3c57032a1623c70-Paper-Conference.pdf`.

[12] Abhinav Shukla, Sai Vemprala, Aditya Kusupati, and Ashish Kapoor. Matmamba: A matryoshka state space model. *arXiv preprint arXiv:2410.06718*, 2024.

[13] Elad Hazan, Karan Singh, and Cyril Zhang. Learning linear dynamical systems via spectral filtering. *Advances in Neural Information Processing Systems*, 30, 2017.

[14] Naman Agarwal, Daniel Suo, Xinyi Chen, and Elad Hazan. Spectral state space models. *arXiv preprint arXiv:2312.06837*, 2023.

[15] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

[16] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[17] Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*, 2024.

[18] Megan Flynn, Alexander Wang, Dean Edward Alvarez, Christopher De Sa, and Anil Damle. Stat: Shrinking transformers after training. *arXiv preprint arXiv:2406.00061*, 2024.

[19] Jiateng Wei, Quan Lu, Ning Jiang, Siqi Li, Jingyang Xiang, Jun Chen, and Yong Liu. Structured optimal brain pruning for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13991–14007, 2024.

[20] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

[21] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

[22] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021.

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

[24] Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in neural information processing systems*, 35:22982–22994, 2022.

[25] Dachuan Song, Li Shen, Duy Duong-Tran, and Xuan Wang. Causality-based subject and task fingerprints using fmri time-series data. In *Proceedings of the 15th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–10, 2024.

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

[27] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International conference on machine learning*, pages 26670–26698. PMLR, 2023.

[28] Elad Hazan, Holden Lee, Karan Singh, Cyril Zhang, and Yi Zhang. Spectral filtering for general linear dynamical systems. *Advances in Neural Information Processing Systems*, 31, 2018.

[29] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

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

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

[32] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

[33] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[34] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[35] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[36] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[37] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pages 35549–35562, 2024.

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

[39] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[40] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[41] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in neural information processing systems*, 31, 2018.

[42] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

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

[44] Sarthak Yadav and Zheng-Hua Tan. Audio mamba: Selective state spaces for self-supervised audio representations. *arXiv preprint arXiv:2406.02178*, 2024.

[45] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

A Limitations and broader Impacts

Limitations. ES-SSM is evaluated up to the around-1.3B-parameter, 30B-byte-token SlimPajama setting and across several long-sequence benchmarks. Extending ES-SSM to larger multi-billion-parameter models and trillion-token training regimes, including standard large language model and hybrid architectures, is a natural next step for future work. The current experiments also consider a fixed set of runtime budgets $\mathcal{K}$: finer-grained budget choices and online budget selection have the potential to improve deployment flexibility. Finally, compared with Transformer attention, which benefits from mature fused implementations and inference-system support, ES-SSM still has room for systems-level efficiency optimization, including FFT batching, memory layout, projection fusion, and backend support.

Broader impacts. ES-SSM aims to reduce deployment cost by exporting multiple compact models from one full-capacity training run, which may lower the compute and energy needed to support heterogeneous hardware budgets. This can make long-sequence modeling more accessible on resource-constrained platforms, but the same efficiency gains can also lower the barrier to deploying sequence models in harmful or poorly governed applications. ES-SSM does not directly address dataset bias, privacy leakage, security risks, or misuse; deployments should inherit the safeguards, data-governance practices, and monitoring appropriate to the downstream application.

B Additional derivations and proofs

B.1 Derivation of the Hankel spectral parameterization

This appendix derives the base spectral parameterization used by ES-SSM: a stable SSM kernel is expanded over fixed Hankel spectral channels as $G(\tau) \approx \sum_{k=1}^{K} M_k \phi_k(\tau)$, yielding the corresponding input-output map $\hat{y}(t) = \sum_{k=1}^{K} M_k(\Phi_k * u)(t)$. The main text presents the key idea: a stable SSM kernel can be decomposed into scalar geometric traces, and these traces can be approximated by a small number of fixed Hankel spectral channels. Here we give the detailed construction and notation.

B.2 From SSM eigenmodes to geometric traces

Consider the causal convolutional kernel induced by a linear state-space model,

$$G(\tau) = CA^{\tau}B, \qquad \tau = 0, \dots, L-1,$$

where

$$A \in \mathbb{R}^{N\times N}, \qquad B \in \mathbb{R}^{N\times d_{\text{in}}}, \qquad C \in \mathbb{R}^{d_{\text{out}}\times N}.$$

Here N is the hidden state dimension and L is the finite convolution horizon.

For clarity, we follow the standard stable symmetric setting used in spectral filtering analyses [13, 14]. Assume that A is real symmetric and has eigenvalues in $[0, 1)$. Then

$$A = U\Lambda U^{\top}, \qquad \Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_N),$$

where U is orthogonal and $\lambda_\ell \in [0, 1)$. Substituting this decomposition into the kernel gives

$$G(\tau) = CU\Lambda^{\tau}U^{\top}B.$$

Expanding over the eigenmodes yields

$$G(\tau) = \sum_{\ell=1}^{N} \lambda_\ell^{\tau}(CU)_{:\ell}(U^{\top}B)_{\ell:}. \tag{6}$$

Define the input-output coefficient matrix associated with the ℓ-th eigenmode as

$$W_\ell \triangleq (CU)_{:\ell}(U^{\top}B)_{\ell:} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}. \tag{7}$$

Then equation (6) can be written compactly as

$$G(\tau) = \sum_{\ell=1}^{N} \lambda_\ell^{\tau} W_\ell. \tag{8}$$

This expression shows that the matrix-valued kernel is a sum of scalar temporal traces multiplied by input-output coefficient matrices. For each eigenvalue λ_ℓ, the time-dependent part is the geometric sequence

$$[1, \lambda_\ell, \lambda_\ell^2, \ldots, \lambda_\ell^{L-1}].$$

Thus approximating the SSM kernel over the horizon L reduces to approximating scalar geometric sequences.

For a generic $\lambda \in [0, 1)$, define the geometric vector

$$\nu(\lambda) \triangleq \left[1, \lambda, \lambda^2, \ldots, \lambda^{L-1}\right]^\top \in \mathbb{R}^L. \tag{9}$$

Its entries satisfy

$$[\nu(\lambda_\ell)]_{\tau+1} = \lambda_\ell^\tau, \qquad \tau = 0, \ldots, L-1.$$

Therefore, although $\nu(\lambda_\ell)$ does not appear explicitly in equation (8), the scalar trace $\{\lambda_\ell^\tau\}_{\tau=0}^{L-1}$ is exactly the entrywise representation of $\nu(\lambda_\ell)$.

B.3 Construction of Hankel spectral channels

Following the spectral filtering construction of Hazan et al. [13], we build a universal basis for stable geometric traces using a reweighted version of $\nu(\lambda)$. Define

$$\mu(\lambda) \triangleq (1-\lambda)\nu(\lambda) = (1-\lambda)\left[1, \lambda, \lambda^2, \ldots, \lambda^{L-1}\right]^\top, \qquad \lambda \in [0, 1]. \tag{10}$$

The endpoint $\lambda = 1$ is included only for the reweighted vector $\mu(\lambda)$, for which $\mu(1) = 0$. The factor $1 - \lambda$ keeps the family $\{\mu(\lambda)\}$ uniformly bounded near the stability boundary.

The Hankel matrix used to construct the spectral channels is

$$Z \triangleq \int_0^1 \mu(\rho)\mu(\rho)^\top d\rho \in \mathbb{R}^{L \times L}, \tag{11}$$

where ρ is a dummy integration variable. Equivalently, its entries are

$$Z_{ij} = \int_0^1 (1-\rho)^2 \rho^{i+j-2} d\rho, \qquad i, j = 1, \ldots, L.$$

Thus Z is a positive semidefinite Hankel matrix.

Let

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_L \geq 0$$

denote the eigenvalues of Z, and let

$$\phi_1, \phi_2, \ldots, \phi_L \in \mathbb{R}^L$$

denote the corresponding normalized eigenvectors. We refer to ϕ_k as the k-th Hankel spectral channel. These channels depend only on the sequence length L, not on the hidden dimension N or on a particular realization of A, B, C.

Using the first K channels gives the top-K projection

$$\tilde{\mu}_K(\lambda) \triangleq \sum_{k=1}^{K} \langle \mu(\lambda), \phi_k \rangle \phi_k \in \mathbb{R}^L. \tag{12}$$

Lemma 4.1 and the spectral decay bound in Appendix E of Hazan et al. [13] imply that, for $L \geq 10$, $1 \leq K \leq L$, and a universal constant $c_{\mathrm{H}} > 0$,

$$\left\|\mu(\lambda) - \tilde{\mu}_K(\lambda)\right\|_2 \leq c_{\mathrm{H}} (\log L)^{1/4} \exp\left(-\frac{\pi^2 K}{16 \log L}\right), \qquad \lambda \in [0, 1]. \tag{13}$$

The cited reconstruction result bounds the squared norm by $\left(6 \sum_{j>K} \sigma_j\right)^{1/2}$; taking its square root yields the norm bound above. Thus the leading Hankel channels provide an exponentially improving approximation as K increases.

B.4 From reweighted traces to geometric traces

The bound in equation (13) is stated for the reweighted vector $\mu(\lambda) = (1-\lambda)\nu(\lambda)$. To relate it to the original geometric vector $\nu(\lambda)$, define

$$\tilde{\nu}_K(\lambda) \triangleq \sum_{k=1}^{K} \langle \nu(\lambda), \phi_k \rangle \phi_k, \qquad \lambda \in [0,1). \tag{14}$$

Because projection onto the span of $\{\phi_1, \dots, \phi_K\}$ is linear and

$$\mu(\lambda) = (1-\lambda)\nu(\lambda),$$

we have

$$\tilde{\mu}_K(\lambda) = (1-\lambda)\tilde{\nu}_K(\lambda). \tag{15}$$

Therefore,

$$\left\| \nu(\lambda) - \tilde{\nu}_K(\lambda) \right\|_2 = \frac{1}{1-\lambda} \left\| \mu(\lambda) - \tilde{\mu}_K(\lambda) \right\|_2. \tag{16}$$

Consequently, the approximation guarantee for $\mu(\lambda)$ transfers to $\nu(\lambda)$ on any interval $[0, 1-\delta]$ bounded away from the stability boundary. In particular, for $\lambda \le 1-\delta$,

$$\|\nu(\lambda) - \tilde{\nu}_K(\lambda)\|_2 \le \delta^{-1} c_{\mathrm{H}} (\log L)^{1/4} \exp\left(-\frac{\pi^2 K}{16 \log L} \right).$$

The factor $1/(1-\lambda)$ reflects the fact that unweighted geometric traces become harder to approximate uniformly as $\lambda \to 1$.

B.5 Substitution into the SSM kernel

We now substitute the spectral approximation of each scalar geometric trace into the SSM kernel. From equation (14), for each eigenvalue λ_ℓ,

$$\nu(\lambda_\ell) \approx \sum_{k=1}^{K} \langle \nu(\lambda_\ell), \phi_k \rangle \phi_k.$$

Taking the $(\tau+1)$-th entry gives

$$\lambda_\ell^\tau = [\nu(\lambda_\ell)]_{\tau+1} \approx \sum_{k=1}^{K} \langle \nu(\lambda_\ell), \phi_k \rangle [\phi_k]_{\tau+1}. \tag{17}$$

For notational convenience, define

$$\phi_k(\tau) \triangleq [\phi_k]_{\tau+1}, \qquad \tau = 0, \dots, L-1.$$

Then

$$\lambda_\ell^\tau \approx \sum_{k=1}^{K} \langle \nu(\lambda_\ell), \phi_k \rangle \phi_k(\tau). \tag{18}$$

Substituting equation (18) into equation (8) yields

$$G(\tau) = \sum_{\ell=1}^{N} \lambda_\ell^\tau W_\ell \approx \sum_{\ell=1}^{N} \sum_{k=1}^{K} \langle \nu(\lambda_\ell), \phi_k \rangle \phi_k(\tau) W_\ell.$$

Exchanging the order of summation gives

$$G(\tau) \approx \sum_{k=1}^{K} \left(\sum_{\ell=1}^{N} \langle \nu(\lambda_\ell), \phi_k \rangle W_\ell \right) \phi_k(\tau). \tag{19}$$

Define

$$M_k \triangleq \sum_{\ell=1}^{N} \langle \nu(\lambda_\ell), \phi_k \rangle W_\ell \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}. \tag{20}$$

Then the SSM kernel admits the spectral approximation

$$G(\tau) \approx \sum_{k=1}^{K} M_k \phi_k(\tau), \qquad \tau = 0, \dots, L-1. \tag{21}$$

Finally, viewing

$$\Phi_k = \{\phi_k(\tau)\}_{\tau=0}^{L-1}$$

as a causal convolution filter, the corresponding input-output map is

$$\hat{y}(t) = \sum_{k=1}^{K} M_k (\Phi_k * u)(t), \tag{22}$$

where $*$ denotes causal convolution.

In neural sequence modeling, we do not explicitly identify A, B, C, U, or the eigenvalues $\{\lambda_\ell\}_{\ell=1}^{N}$. Instead, the Hankel channels $\{\Phi_k\}_{k=1}^{K}$ are fixed in advance, and the matrices $\{M_k\}_{k=1}^{K}$ are learned directly from data. Thus the hidden dimension N of the classical SSM realization is absorbed into the learned channel projections, while the spectral budget K becomes the controllable inference budget.

B.6 Prefix consistency of exported compact models

We formalize the mechanical consistency property used by direct prefix truncation. The statement is per spectral sublayer and conditional on the same sublayer input sequence $u_{1:L}$; across a full network, different runtime budgets may of course produce different intermediate representations after the first budgeted layer.

Remark 1 (Prefix consistency)**.** *Consider a full-capacity ES-SSM spectral sublayer with maximum budget $\overline{K}$. Let the budget-K exported compact sublayer be obtained by retaining $\{\Phi_k, M_k\}_{k=1}^{K}$, the first K rows of $\mathbf{W}_{g,2}$, and the first K entries of $\mathbf{b}_{g,2}$, while keeping the shared gate parameters $\mathbf{W}_{g,1}, \mathbf{b}_{g,1}$ unchanged. For any retained channel $k \le K$ and any token position t, the exported sublayer computes the same gate logit $s_k(t)$, the same spectral feature $(\Phi_k * u)(t)$, and uses the same projection matrix M_k as the full-capacity sublayer. Consequently, the budget-K sublayer output is exactly*

$$[\mathcal{S}_K(u)](t) = \sum_{k=1}^{K} \frac{1}{\sqrt{K}} \operatorname{sigmoid}(s_k(t))\, M_k (\Phi_k * u)(t),$$

with no dependence on any removed channel $j > K$.

Justification. Write the shared hidden gate activation as

$$h_g(t) = \mathrm{GELU}(\mathbf{W}_{g,1} u(t) + \mathbf{b}_{g,1}).$$

In the full-capacity sublayer, the k-th gate logit is

$$s_k(t) = (\mathbf{W}_{g,2})_{k:} h_g(t) + (\mathbf{b}_{g,2})_k.$$

For an exported budget-K sublayer with $k \le K$, row k of $\mathbf{W}_{g,2}$ and entry k of $\mathbf{b}_{g,2}$ are retained, and $h_g(t)$ is computed from the unchanged shared parameters. Hence the value of $s_k(t)$ is unchanged by removing rows $j > K$. The spectral filter Φ_k and projection matrix M_k are also retained unchanged, so $(\Phi_k * u)(t)$ and $M_k(\Phi_k * u)(t)$ are identical for the retained channel. Substituting these unchanged retained-channel quantities into the definition of equation (4) gives the displayed expression. Since all terms with $j > K$ are absent from the exported sum and no retained logit or projection depends on those removed channel-indexed tensors, the exported computation has no dependence on removed channels.

B.7 Gradient allocation under budget dropout

We consider the default step-level sampling schedule. The training schedule also has a simple consequence for which channel-specific parameters enter the forward pass and can receive task gradients. Let

$$\Theta_k^{\mathrm{ch}} \;=\; \{M_k, (\mathbf{W}_{g,2})_{k:}, (\mathbf{b}_{g,2})_k\}$$

denote the trainable parameters associated specifically with spectral channel k in one ES-SSM sublayer. The fixed Hankel filter Φ_k is not included because it is a non-learnable buffer.

Algorithm 1: ES-SSM inference

Input: input $\mathbf{u}$; trained full-capacity parameters θ; runtime budget $K \le \overline{K}$
Output: output $\hat{\mathbf{y}}^{(K)}$

Export $\theta^{(K)}$ by direct prefix truncation in every ES-SSM spectral sublayer: retain $\{\Phi_k, M_k\}_{k=1}^{K}$, the first K rows of $\mathbf{W}_{g,2}$, and the first K entries of $\mathbf{b}_{g,2}$; remove channels $k > K$
Keep all shared parameters unchanged and disable DropPath for deterministic inference
$\mathbf{r} \leftarrow \mathrm{Embed}(\mathbf{u})$
for *each ES-SSM block* ℓ *in* $\theta^{(K)}$ **do**
    $\mathbf{v} \leftarrow \mathrm{LN}_1^{(\ell)}(\mathbf{r})$
    $\mathbf{o}^{(\ell)} \leftarrow \mathcal{S}_K^{(\ell)}(\mathbf{v})$ `// spectral sublayer`
    $\mathbf{r}_{\mathrm{mid}} \leftarrow \mathbf{r} + \mathbf{o}^{(\ell)}$
    $\mathbf{r} \leftarrow \mathbf{r}_{\mathrm{mid}} + \mathrm{FFN}^{(\ell)}(\mathrm{LN}_2^{(\ell)}(\mathbf{r}_{\mathrm{mid}}))$
end
$\hat{\mathbf{y}}^{(K)} \leftarrow \mathrm{OutputHead}(\mathbf{r})$
return $\hat{\mathbf{y}}^{(K)}$

Remark 2 (Budget-dropout update allocation)**.** *At a training step with sampled budget K_{train}, the parameters Θ_k^{ch} enter the computational graph only if $k \le K_{\mathrm{train}}$. Therefore, channel-specific parameters for lower-index channels have gradient paths at least as often as those for higher-index channels. After warmup, if the full-budget anchor cadence n_{full} is finite, every channel-specific parameter group Θ_k^{ch} is included in the forward pass on at least one out of every n_{full} consecutive optimization steps.*

Justification. For a sampled budget K_{train}, each spectral sublayer evaluates

$$[\mathcal{S}_{K_{\mathrm{train}}}(u)](t) = \sum_{j=1}^{K_{\mathrm{train}}} g_j^{(K_{\mathrm{train}})}(t)\, M_j(\Phi_j * u)(t).$$

Only terms with $j \le K_{\mathrm{train}}$ appear in the forward computation. Thus for any $k > K_{\mathrm{train}}$, the tensors in Θ_k^{ch} are not used to compute the loss at that step, so their task gradient from that step is zero. Conversely, when $k \le K_{\mathrm{train}}$, the corresponding projection, gate-output row, and gate-output bias entry participate in the retained-channel computation and have the usual backpropagation path.

Let $p_k = \Pr(K_{\mathrm{train}} \ge k)$ denote the post-warmup probability that channel k is included. Since the events $\{K_{\mathrm{train}} \ge k\}$ are nested, $p_i \ge p_j$ whenever $i < j$, so lower-index channel-specific parameters have gradient paths at least as often in expectation. If q denotes the distribution over budgets on non-anchor steps, then for finite n_{full} the long-run inclusion probability is

$$p_k = \frac{1}{n_{\mathrm{full}}} + \left(1 - \frac{1}{n_{\mathrm{full}}}\right) \sum_{\kappa \in \mathcal{K}:\, \kappa \ge k} q(\kappa).$$

In particular, the anchor rule sets $K_{\mathrm{train}} = \overline{K}$ whenever $j \bmod n_{\mathrm{full}} = 0$. Hence every block of n_{full} consecutive post-warmup optimization steps contains a full-budget step, which includes every channel $k \le \overline{K}$. This gives each channel-specific parameter group an inclusion-frequency lower bound of $1/n_{\mathrm{full}}$ from the anchor updates alone.

C Efficient implementation

This section describes the implementation choices used for the ES-SSM results reported in the main text.

Algorithm 2: ES-SSM training

Input: data $\mathcal{D}$; runtime-budget set $\mathcal{K}$; $\overline{K}$; E_{warmup}; n_{full}
Output: trained parameters θ

$K_{\min} \leftarrow \min \mathcal{K}$
for *each minibatch* $(\mathbf{u}, \mathbf{y}) \sim \mathcal{D}$ *at epoch* e *and step* j **do**
  if $e < E_{\text{warmup}} \vee (n_{\text{full}} < \infty \wedge j \bmod n_{\text{full}} = 0)$ **then**
    $K_{\text{train}} \leftarrow \overline{K}$ `// warmup / anchor`
  else
    draw $\xi \sim \text{Unif}[0, 1]$
    $K_{\text{target}} \leftarrow K_{\min}(\overline{K}/K_{\min})^{\xi}$
    $K_{\text{train}} \leftarrow \arg\min_{k \in \mathcal{K}} |k - K_{\text{target}}|$
  end
  $\hat{\mathbf{y}}^{(K_{\text{train}})} \leftarrow F_\theta(\mathbf{u}; K_{\text{train}})$
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{task}}(\hat{\mathbf{y}}^{(K_{\text{train}})}, \mathbf{y})$
  update θ with AdamW, $\nabla_\theta \mathcal{L}$, and global ℓ_2-norm gradient clipping
end
return θ

C.1 Exported inference and budget-dropout training pseudocode

C.2 FFT-based spectral convolution

The spectral features $\{(\Phi_k * u)(t)\}_{k=1}^{\overline{K}}$ are computed jointly by a single length-n real FFT with $n = 2L$, which is the standard zero-padded length required for a *causal* linear convolution on sequences of length L.[3]

Throughout this appendix we use the main-text setting $d_{\text{in}} = d_{\text{out}} = d$. Let $\mathcal{U} \in \mathbb{R}^{B_{\text{sz}} \times d \times L}$ be the layer input with the time axis last (batch size B_{sz}, feature dimension d), and let $\widehat{\mathcal{U}} = \text{rFFT}(\mathcal{U}, n = 2L) \in \mathbb{C}^{B_{\text{sz}} \times d \times (n/2+1)}$. The filter transforms $\{\widehat{\Phi}_k\}_{k=1}^{\overline{K}} \in \mathbb{C}^{\overline{K} \times (n/2+1)}$ are precomputed once at model construction (Appendix C.6) and stored as a non-learnable buffer. Writing c for the feature-channel index to avoid collision with the dimension symbol d, the sublayer evaluates

$$\mathcal{F}_{b,t,k,c} \;=\; \text{irFFT}\big(\widehat{\mathcal{U}}_{b,c} \odot \widehat{\Phi}_k,\; n = 2L\big)_t, \qquad t = 1, \ldots, L,\; c = 1, \ldots, d, \tag{23}$$

so that $\mathcal{F} \in \mathbb{R}^{B_{\text{sz}} \times L \times \overline{K} \times d}$, with the inverse FFT output truncated to its first L samples to recover the causal convolution tail. All $\overline{K}$ channels share the same $\widehat{\mathcal{U}}$, so the full spectral feature bank is produced by one forward rFFT, one broadcast complex multiply of shapes $(B_{\text{sz}}, d, 1, n/2 + 1) \times (1, 1, \overline{K}, n/2 + 1)$, and one inverse rFFT followed by a transpose to the layout above.

For an exported compact model at runtime budget $K \le \overline{K}$, the broadcast multiply is built from the first K filter transforms $\{\widehat{\Phi}_k\}_{k=1}^{K}$. Every downstream tensor ($\mathcal{F}$, the gate $g^{(K)}$, and the projection stack $\{M_k\}$) is likewise sliced along the channel axis. No recomputation or re-fitting is required.

C.3 Precision policy

Design. The bulk of the network (embeddings, attention/SSM matmuls, FFN, LayerNorms) runs in `bf16` under PyTorch AMP. The spectral branch in equation (23), however, runs in `float32` with autocast explicitly disabled. The Hankel eigenvalues span several orders of magnitude, so low-precision spectral arithmetic can bias the smaller-magnitude channels. Keeping the FFT, complex multiply, and inverse FFT in `fp32` removes this effect at negligible cost, while `bf16` remains the default for the rest of the network because it is stable without loss scaling.

Empirical validation. We validate the policy with a controlled four-config ablation at 160 M parameters on SlimPajama (d=512, n_{layers}=15, $\overline{K}$=32, L=2048, 1.5 B byte-tokens). The four configurations differ only in (i) the AMP main-network dtype and (ii) whether the spectral branch

[3] Using $n = L$ would alias the tail of the convolution and is incorrect for causal filtering.

Config	main	spec.	Val BPB$_{K=2}$	Val BPB$_{K=8}$	Val BPB$_{K=\overline{K}}$	$\Delta_{K=\overline{K}}$
B. naive bf16	`bf16`	low	1.7826	1.6064	1.6089	$+0.0063$
C. naive fp16+scaler	`fp16`	low	1.7841	1.6088	1.6114	$+0.0087$
D. `bf16+fp32` spectral (**ours**)	`bf16`	`fp32`	1.7786	1.6002	1.6027	—
E. `fp16+fp32` spectral	`fp16`	`fp32`	1.7778	1.5997	1.6023	-0.0004

Table 1: Precision-policy ablation at 160 M parameters on SlimPajama, 1.5 B byte-tokens, $\overline{K}$=32. All four configurations share architecture, optimizer, schedule, data, and random seed; only the AMP main-network dtype and whether the spectral branch is forced to `fp32` differ. "low" in the spec. column means the spectral multiply runs in the AMP main dtype (i.e. the FFT itself is auto-cast to `fp32` by PyTorch's AMP fallback for FFT, but the complex multiply runs in `bf16`/`fp16`). $\Delta_{K=\overline{K}}$ is val BPB minus our policy (D) at K=$\overline{K}$=32.

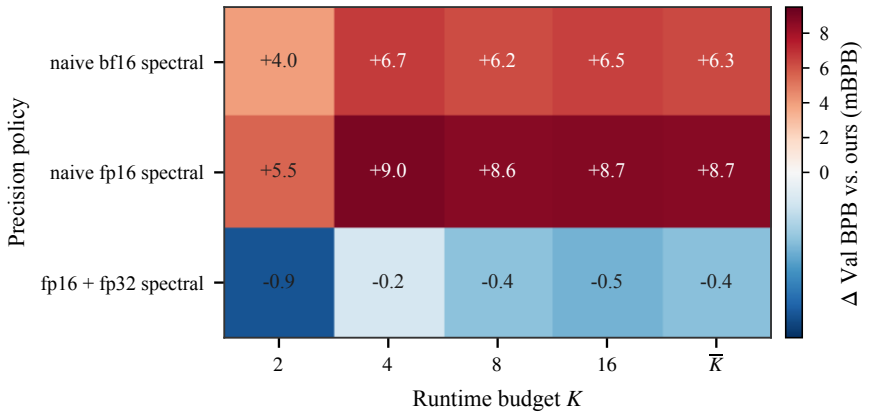

Figure 6: Δ Val BPB relative to our policy (row D in table 1) at runtime budget $K \in \{2, 4, 8, 16, \overline{K}\}$. Positive entries (red) are worse than ours; negative (blue) are better. Both naive low-precision rows (B, C) are uniformly positive ($10/10$ cells), and the fp16-naive row C is darker than the bf16-naive row B at every K. The forced-`fp32` spectral row (E) is statistically indistinguishable from D ($|\Delta_K| < 0.001$ BPB at every K).

is forced to `fp32`; all other hyperparameters and the random seed are identical. Table 1 reports validation BPB at three runtime budgets and the gap to our policy (Δ) at K=$\overline{K}$. Figures 6 to 8 visualise the same data three ways.

The ablation supports our precision policy as a practical accuracy–efficiency choice in this regime. Both naive low-precision spectral configurations are worse at every runtime budget: the full-capacity penalty is $+0.006$ BPB for bf16 spectral arithmetic and $+0.009$ BPB for fp16 spectral arithmetic. By contrast, once the spectral branch is forced to `fp32`, the bf16-main and fp16-main runs are indistinguishable within 0.001 BPB at every K. The training curves show the same pattern throughout optimization rather than only at the final checkpoint. We therefore use `bf16` for the main network and `fp32` for the spectral branch.

Cost. Using `fp32` in the spectral branch does not add measurable wall-time overhead in this experiment: the two `fp32`-spectral runs finish within measurement noise of each other. The naive low-precision spectral emulation is actually slower, because complex half-precision multiplication must be manually decomposed into real and imaginary components under PyTorch AMP.

Limitations. This is a single-seed ablation at 1.5 B tokens. The observed low-precision spectral penalties are larger than nearby validation noise, but we treat the experiment as evidence for the precision policy rather than as a full scaling law for numerical precision.

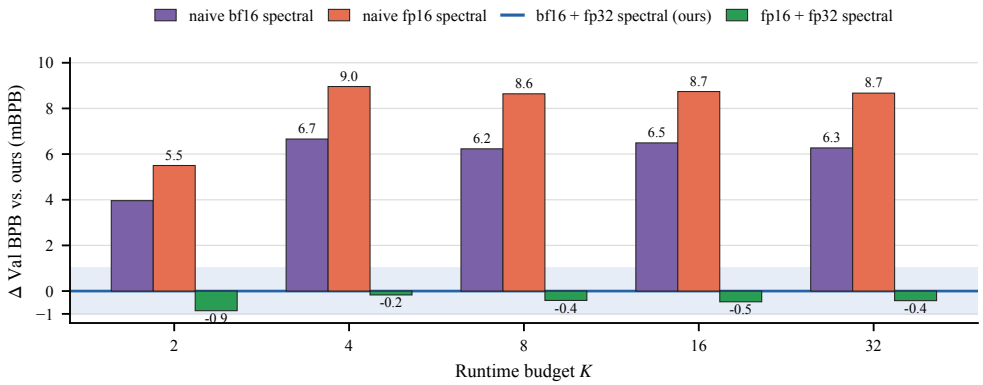

Figure 7: Per-K validation-BPB difference relative to our precision policy (D) after 1.5 B SlimPajama tokens. Positive values are worse than our policy. The naive low-precision spectral configurations are worse at every runtime budget, while the fp16 main-network run with an fp32 spectral branch stays within the ± 1 mBPB band.

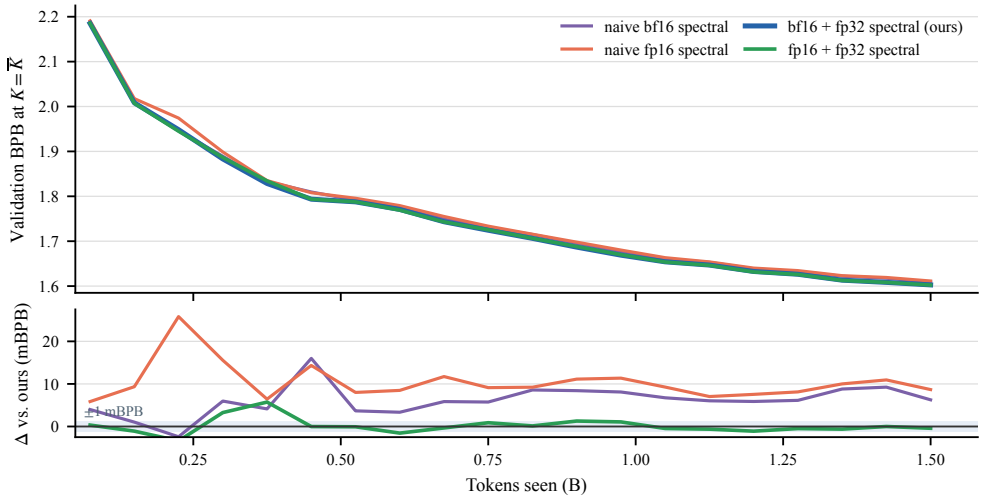

Figure 8: Training dynamics for the precision-policy ablation. Top: validation BPB at $K = \overline{K} = 32$ over training. Bottom: validation-BPB difference relative to our policy (D), in mBPB. The two fp32-spectral configurations track each other within a small band, while the naive low-precision spectral configurations keep a positive offset after early training.

C.4 Gate-before-projection ordering

Given the spectral feature tensor $\mathcal{F} \in \mathbb{R}^{B_{\text{sz}} \times L \times \overline{K} \times d}$ from equation (23) and the gate coefficients $g^{(K)} \in \mathbb{R}^{B_{\text{sz}} \times L \times K}$, two algebraically equivalent orderings are possible for equation (4):

$$\text{(a) gate-first:} \quad \left[g^{(K)}_{b,t,k}\, \mathcal{F}_{b,t,k,\cdot}\right] M_k^\top,$$
$$\text{(b) project-first:} \quad g^{(K)}_{b,t,k} \left[\mathcal{F}_{b,t,k,\cdot}\, M_k^\top\right].$$

We use ordering (a) and combine projection with channel summation in a single contraction, avoiding an explicit (B_{sz}, L, K, d) per-channel projected-output tensor. For $d_{\text{in}} = d_{\text{out}} = d$, both orderings require $B_{\text{sz}} LKd$ scalar gate multiplications. Concretely, the kernel implementing ordering (a) is a single `einsum` of the form `"blkd, kde → ble"` applied to the gated features and the stacked projection tensor $\widetilde{M} \in \mathbb{R}^{\overline{K} \times d \times d}$ defined by $\widetilde{M}[k, c, e] = (M_k)_{e,c}$ for input and output feature indices $c, e \in \{1, \ldots, d\}$ (i.e. $M_k^\top$ stacked along the channel axis), prefix-sliced to $\widetilde{M}[:K]$ at runtime.

C.5 Parameter initialization and shared modules

The channel projections $\{M_k\}_{k=1}^{\overline{K}}$ are initialized elementwise i.i.d. from $\mathcal{N}(0,\, 10^{-6})$, i.e. with standard deviation 10^{-3}. At step 0 the spectral branch output therefore has near-zero magnitude; combined with the near-closed gates described below, this means the network begins training as a pure residual plus FFN path, and the spectral branch must be earned during optimization. The gating MLP uses default PyTorch `nn.Linear` initialization for both layers, with one exception: the output-layer bias $\mathbf{b}_{g,2}$ is overridden to the constant -2, giving $\mathrm{sigmoid}(s_k(t)) \approx 0.12$ for every k, t at initialization. The token embedding is drawn from $\mathcal{N}(0,\, 0.02^2)$ with padding index 0 and is tied to the output head (standard weight tying). DropPath follows a layerwise linear schedule from 0 at the first layer to the nominal maximum (e.g. 0.1) at the top layer, and dropout with rate 0.1 is applied to the embedding output and within the FFN.

C.6 Hankel filter bank construction

The filter bank $\{\Phi_k\}_{k=1}^{\overline{K}}$ is the top-$\overline{K}$ eigensystem of the Hankel matrix Z defined in equations (10) and (11). The entries of Z admit the closed form

$$Z_{i,j} = \int_0^1 (\rho - 1)^2 \rho^{i+j-2}\, d\rho = \frac{2}{(i+j)^3 - (i+j)}, \qquad i, j = 1, \ldots, L, \tag{24}$$

so Z is a Hankel matrix generated by the scalar sequence $h_q = 2/(q^3 - q)$, $q = 2, \ldots, 2L$, where ρ is the integration variable. We compute the leading $\overline{K}$ eigenpairs (σ_k, ϕ_k) once per sequence length L and cache them.

We never materialize Z as a full $L \times L$ matrix. Instead, any Hankel matrix–vector product Zv is computed in $\mathcal{O}(L \log L)$ by embedding Z into a circulant of size $2L$: one forward rFFT on the zero-padded v, one elementwise complex multiply with the precomputed generating-vector transform $\widehat{h} = \mathrm{rFFT}(\tilde{h})$, where $\tilde{h}$ is the circulant embedding of the generating sequence $\{h_q\}$, and one inverse rFFT to read off the relevant entries of the result. With oversampling margin $r_{\mathrm{os}} = 16$, a Rayleigh–Ritz iteration applies Z to the $\overline{K} + r_{\mathrm{os}}$ columns of Q as a single batched matvec, so each iteration requires two batched FFTs in total.

We extract $\{(\sigma_k, \phi_k)\}_{k \le \overline{K}}$ via subspace iteration with periodic Rayleigh–Ritz restarts: (i) initialize $Q \in \mathbb{R}^{L \times (\overline{K} + r_{\mathrm{os}})}$ with Gaussian entries and orthonormalize via QR; (ii) repeat for $T_{\max} = 60$ iterations: $Q \leftarrow \mathrm{QR}(ZQ)$, and every $\lfloor T_{\max}/4 \rfloor$ iterations perform a Rayleigh–Ritz step ($R = Q^\top ZQ$; eigendecompose R; sort descending; rotate $Q \leftarrow QV_R$, where V_R contains the Ritz vectors) and check the per-eigenpair residual $\mathrm{res}_k = \|ZQ_{:,k} - \sigma_k Q_{:,k}\|_2 / |\sigma_k|$; (iii) stop early when $\max_{k \le \overline{K}} \mathrm{res}_k < 10^{-6}$. The resulting top-$\overline{K}$ eigenvectors form $\{\phi_k\}_{k=1}^{\overline{K}}$, and their rFFTs (padded to length $2L$) form the filter-transform buffer used in Appendix C.2.

Because $\overline{K}$ and L are fixed for a given experiment, the filter bank is computed once on CPU and cached to disk; subsequent runs load the cached tensor. This precomputation takes under a minute for $L = 2048$, $\overline{K} = 32$ and is not counted in training wall-clock.

C.7 Full training recipe

Table 2 lists the optimization hyperparameters used in all main-text experiments unless stated otherwise.

D Budget dropout sampler

This appendix specifies the sampler referred to in the main text (section 4.2, equation (5)). It implements three design constraints: (i) early training should resemble full-capacity training so that every per-channel parameter is given a chance to initialize meaningfully; (ii) throughout training, the tail channels k close to $\overline{K}$ must continue to receive gradient signal, or they will drift and the full-capacity model will degrade; and (iii) the mid-training distribution over K_{train} should cover $[K_{\min}, \overline{K}]$ approximately uniformly on the log-scale so that every runtime budget in the set is exercised at comparable frequency.

Table 2: Training recipe.

Hyperparameter	Value
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$, $\varepsilon = 10^{-8}$
Weight decay	0.1 on non-LayerNorm, non-embedding, non-bias parameters
Peak learning rate	3×10^{-4}
Warmup	Linear, first 2% of total optimization steps
Decay	Cosine to 10% of peak at the final step
Gradient clipping	Global ℓ_2 norm clipped to 1.0
Effective batch size	Scale-dependent; see section 5
Mixed precision	See Appendix C.3
Activation checkpointing	Enabled for large SlimPajama-scale ES-SSM runs; off for small ablations
Early stopping	Val. BPB at $K = \overline{K}$; patience 8 epochs

Algorithm 3: SAMPLEBUDGET: budget dropout sampler used during training

Input: runtime-budget set $\mathcal{K}$ with $\overline{K} = \max \mathcal{K}$; epoch e; global optimization step j; warmup epochs E_{warmup}; full-budget anchor cadence n_{full}; random generator RNG

Output: sampled training budget $K_{\text{train}} \in \mathcal{K}$

$K_{\min} \leftarrow \min \mathcal{K}$
if $e < E_{\text{warmup}}$ **then**
  $K_{\text{train}} \leftarrow \overline{K}$ `// full-budget warmup`
else
  if $n_{\text{full}} < \infty$ *and* $j \bmod n_{\text{full}} = 0$ **then**
    $K_{\text{train}} \leftarrow \overline{K}$ `// full-budget anchor update`
  else
    draw $\xi \sim \text{Unif}[0, 1]$ using RNG
    $K_{\text{target}} \leftarrow K_{\min}(\overline{K}/K_{\min})^{\xi}$ `// log-scale target`
    $K_{\text{train}} \leftarrow \arg\min_{k \in \mathcal{K}} |k - K_{\text{target}}|$ `// snap to nearest allowed budget`
  end
end
return K_{train}

Default hyperparameters. Unless a benchmark-specific setup states otherwise, we use $E_{\text{warmup}} = 8$ epochs, $n_{\text{full}} = 8$ steps, $K_{\min} = 2$, $\overline{K} = 32$, and $\mathcal{K} = \{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$. The RNG is seeded deterministically so that two runs with the same seed and sampling protocol receive identical K_{train} sequences.

Two choices in algorithm 3 control gradient allocation across spectral channels. The warmup phase keeps $K_{\text{train}} = \overline{K}$ while the small initialized projections and near-closed gates settle into a stable regime. After warmup, the full-budget anchor update guarantees every channel a minimum update rate of $1/n_{\text{full}}$, preventing tail channels from being trained only through rare sampled full-budget steps.

Empirical validation of the full-budget anchor cadence. The default $n_{\text{full}} = 8$ lies in the empirical sweet spot of a more detailed cadence sweep. Let $p_{\max}$ be the probability that the log-scale sampler selects $\overline{K}$ before the full-budget anchor rule is applied; for our runtime-budget set, $p_{\max} \approx 0.048$. The per-step probability of training at $K = \overline{K}$ is

$$P(K_{\text{train}} = \overline{K}) \;=\; \frac{1}{n_{\text{full}}} \;+\; \Big(1 - \frac{1}{n_{\text{full}}}\Big) \cdot p_{\max} \;=\; \frac{1}{n_{\text{full}}} \;+\; \frac{n_{\text{full}} - 1}{n_{\text{full}}}\, p_{\max}. \tag{25}$$

For $n_{\text{full}} = 8$, this is approximately 0.167, so the maximum-budget path receives a full-capacity update roughly once every six optimizer steps. Smaller cadences over-emphasize the full-budget path, while larger cadences leave tail channels too dependent on the sampled log-scale distribution.

The sweep in figure 9 and table 3 tests $n_{\text{full}} \in \{1, 2, 3, 4, 6, 8, 12, 16, 24, \infty\}$ with three seeds per cadence. The coarser table in Appendix H.4 is retained only to show that the endpoint $n_{\text{full}} =$

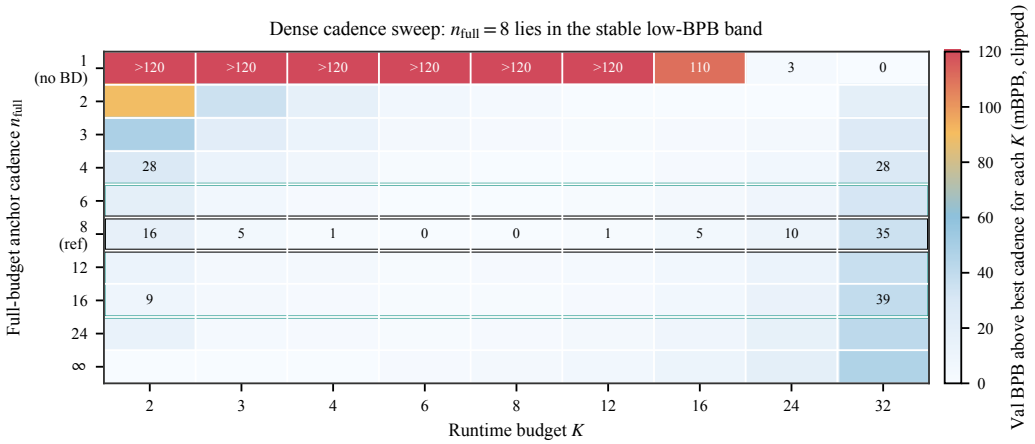

Figure 9: Full-budget anchor cadence sweep. Each cell shows validation BPB above the best cadence for that runtime budget, in milli-BPB and clipped at 120 for readability. The black row marks the default $n_{\text{full}} = 8$, which forces $K_{\text{train}} = \overline{K}$ once every eight post-warmup optimizer steps; the green box marks the empirically stable band $n_{\text{full}} \in \{4, 6, 8, 12, 16\}$. The endpoint $n_{\text{full}} = 1$ degenerates to full-budget training and collapses small-K prefixes, while very large cadences undertrain the tail channels. Exact means and standard deviations are reported in table 3.

Table 3: Effect of the full-budget anchor cadence n_{full} on validation BPB across runtime budgets, mean $\pm$ std over 3 random seeds using a 287.1 M-parameter model trained for 80 M tokens per run (≈ 0.28 tokens per parameter). Two competing effects make the curve U-shaped: as $n_{\text{full}} \downarrow$ the $K = \overline{K}$ channel gets more gradient and $K = \overline{K}$ BPB falls monotonically; as $n_{\text{full}} \downarrow$ the small-K operating points get less probability mass and small-K BPB rises. The reference choice $n_{\text{full}} = 8$ sits in a wide flat optimum spanning $n_{\text{full}} \in [4, 16]$ at every K, inside which the trade-off is empirically balanced — $K = 2$ val BPB lies in $[2.447, 2.466]$ and $K = \overline{K}$ val BPB in $[2.215, 2.226]$, all within two standard errors of one another. The catastrophic collapse at $n_{\text{full}} = 1$ ($K = 2$ val BPB 4.245 ± 0.190) recovers the no-budget-dropout failure mode of table 23 as a degenerate limit. The sweep does not identify a single best n_{full} — neither 4 nor 16 is statistically distinguishable from 8 on this seed set — but it rules out both endpoints and licenses 8 as a defensible round-number default rather than a hand-tuned one.

n_{full}	$K=2$	$K=3$	$K=4$	$K=6$	$K=8$	$K=12$	$K=16$	$K=24$	$K=32$
1 (≡ no_bd)	4.245 ± 0.190	4.072 ± 0.109	4.049 ± 0.069	3.485 ± 0.060	3.005 ± 0.046	2.506 ± 0.065	2.316 ± 0.049	2.208 ± 0.032	2.187 ± 0.027
2	2.527 ± 0.032	2.331 ± 0.020	2.272 ± 0.021	2.227 ± 0.024	2.216 ± 0.024	2.209 ± 0.025	2.206 ± 0.025	2.205 ± 0.025	2.206 ± 0.025
3	2.486 ± 0.037	2.317 ± 0.023	2.267 ± 0.024	2.225 ± 0.023	2.215 ± 0.024	2.210 ± 0.024	2.208 ± 0.023	2.209 ± 0.024	2.213 ± 0.024
4	2.466 ± 0.038	2.308 ± 0.021	2.262 ± 0.022	2.222 ± 0.022	2.213 ± 0.023	2.209 ± 0.023	2.208 ± 0.023	2.211 ± 0.024	2.215 ± 0.024
6	2.457 ± 0.035	2.302 ± 0.021	2.257 ± 0.021	2.221 ± 0.022	2.213 ± 0.023	2.210 ± 0.023	2.210 ± 0.024	2.213 ± 0.024	2.218 ± 0.025
8 (reference)	2.454 ± 0.034	2.301 ± 0.021	2.256 ± 0.022	2.221 ± 0.022	2.213 ± 0.021	2.210 ± 0.022	2.211 ± 0.022	2.215 ± 0.022	2.222 ± 0.023
12	2.450 ± 0.038	2.300 ± 0.024	2.259 ± 0.026	2.222 ± 0.025	2.215 ± 0.025	2.212 ± 0.025	2.212 ± 0.025	2.217 ± 0.025	2.224 ± 0.025
16	2.447 ± 0.037	2.300 ± 0.024	2.257 ± 0.024	2.222 ± 0.024	2.214 ± 0.024	2.211 ± 0.024	2.212 ± 0.024	2.218 ± 0.024	2.226 ± 0.024
24	2.452 ± 0.038	2.299 ± 0.026	2.257 ± 0.026	2.222 ± 0.025	2.215 ± 0.025	2.213 ± 0.025	2.214 ± 0.025	2.220 ± 0.026	2.228 ± 0.027
∞	2.438 ± 0.033	2.296 ± 0.022	2.255 ± 0.022	2.223 ± 0.024	2.217 ± 0.024	2.215 ± 0.025	2.217 ± 0.026	2.224 ± 0.027	2.233 ± 0.028

1 recovers the no-budget-dropout failure mode, while reasonable non-endpoint cadences behave similarly.

D.1 Sensitivity to the runtime-budget set

We examine whether it is necessary to sample every integer runtime budget during training. We compare the default ten-point budget set $\mathcal{K}_{\text{Paper-10}} = \{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$ against the dense set $\mathcal{K}_{\text{Dense-31}} = \{2, 3, \ldots, 32\}$. Both models contain 287.1M parameters and are trained for 80.019M byte-tokens with seed 87. We keep the data, initialization, optimizer, learning-rate schedule, batch configuration, full-budget warmup, and evaluation examples fixed. The mean sampled budgets are also nearly identical (22.6388 versus 22.6450). Both checkpoints are evaluated at every integer budget $K \in \{2, \ldots, 32\}$.

Table 4: Sensitivity to the set of budgets sampled during training. Both checkpoints are evaluated on all 31 integer runtime budgets. Lower BPB is better.

Training budget set	Mean val. BPB	Max. gap	$K = 2$	$K = 32$
Dense-31	2.2307	–	2.4838	2.2244
Paper-10	2.2319	0.0030	2.4848	2.2253

Table 5: Validation BPB under a 40M-token full-budget warmup and zero warmup.

Runtime budget K	40M-token warmup	Zero warmup	Absolute difference
2	1.9552	1.9465	0.0087
8	1.8221	1.8182	0.0039
32	1.8205	1.8204	0.0001

Across all 31 evaluation budgets, the mean absolute validation-BPB difference between Paper-10 and Dense-31 is 0.0012, and the maximum difference is 0.0030 at $K = 7$. Paper-10 directly samples only ten budgets during training; among the remaining 21 unseen integer budgets, the maximum linear-interpolation residual is 0.0032 BPB, and no residual exceeds 0.01 BPB. These results indicate that the default ten-point set provides sufficient coverage of the low-, intermediate-, and full-budget regimes in this controlled setting, without requiring every deployable budget to be sampled during training.

This experiment is a single-seed, 287.1M-parameter, 80M-token diagnostic. It supports the adequacy of the default budget set in this setting, but does not establish that the set is minimal or optimal, nor that changing the endpoint range $[2, 32]$ would have no effect.

D.2 Full-budget warmup sensitivity

We tested model performance under the default full-budget warmup and under zero warmup, with all other conditions being the same, in a separate 400.03 M-byte-token run. Across all ten standard budgets, the mean and maximum absolute differences are small. We provide detailed numbers for three representative settings in table 5.

This provides preliminary evidence that the impact of warmup on the final result might not be significant. Nevertheless, we retain the warmup as a precaution during initialization.

D.3 Training cost at matched full-budget quality

We compare the total training resources required to reach matched full-budget validation quality. The always-full control defines the target at a validation BPB of 1.839901, while the first measured ES-SSM checkpoint that meets this target achieves a BPB of 1.834882. As shown in table 6, ES-SSM processes 17.3% more byte-tokens and optimization steps, but requires 30.2% less cumulative analytic compute and 39.4% less measured wall-clock time. Thus, ES-SSM reaches matched full-budget quality using more updates but less total computation and substantially less elapsed training time.

Analytic compute estimates the number of operations associated with the channels active under each sampled training budget. On a per-update basis, ES-SSM uses 40.5% less analytic active compute and 48.4% less measured wall-clock time than always-full training. These per-update savings more than offset the 17.3% increase in optimization steps. The preceding ES-SSM checkpoint has a BPB of 1.840758, while the following checkpoint remains below the target at 1.837340, confirming the matched-quality crossing.

E Complexity and parameter count

This appendix makes precise the "exported compact model is physically smaller" claim from section 4. We give closed-form expressions for the parameter count and per-token FLOPs of an ES-SSM layer at runtime budget K, identify the fixed and K-scalable components, and summarize the resulting

Table 6: Training resources required to reach matched full-budget validation quality. Resource columns are normalized to the always-full control (100%); lower values indicate lower total cost. The ES-SSM row corresponds to the first measured checkpoint that meets the target quality.

Method	Full-budget val. BPB ↓	Steps and byte-tokens	Analytic compute	Measured wall-clock
Always-full control	1.839901	100.0%	100.0%	100.0%
ES-SSM	**1.834882**	117.3%	**69.8%**	**60.6%**

Table 7: Per-layer parameter ledger. "Buffer" entries are cached spectral quantities and are not learnable parameters.

Component	Expression	Scales with K?
Channel projections $\{M_k\}_{k\le K}$	$K\,d^2$	**yes**
Gate MLP output row $\mathbf{W}_{g,2}[:K,:]$	$K\,(d/2)$	**yes**
Gate MLP output bias $\mathbf{b}_{g,2}[:K]$	K	**yes**
Gate MLP hidden layer $\mathbf{W}_{g,1}, \mathbf{b}_{g,1}$	$d\,(d/2) + d/2$	no
FFN (widening 4)	$8\,d^2 + 5\,d$	no
2× LayerNorm (pre-spectral, pre-FFN)	$4\,d$	no
Filter transforms $\{\widehat{\Phi}_k\}_{k\le K}$ *(buffer)*	$K\,(L+1)$ complex	(buffer, not counted)

compute and memory trajectories for generic budgets $K \le \overline{K}$. All counts below are *per layer* unless stated otherwise; for an n_{layers}-deep model, multiply by n_{layers} and add the shared embedding/output-head contribution $V\,d$ (tied), where V is the vocabulary size.

For clarity we assume $d_{\text{in}} = d_{\text{out}} = d$, which is the setting used in all main-text experiments; the generalization is immediate.

E.1 Parameter count at runtime budget K

Summing the learnable rows of table 7,

$$P_{\text{layer}}(K) \;=\; \underbrace{K\big(d^2 + d/2 + 1\big)}_{K\text{-scalable}} \;+\; \underbrace{\tfrac{17}{2}\,d^2 + \mathrm{O}(d)}_{\text{fixed floor}}. \tag{26}$$

Equation (26) has two practical consequences. First, the *marginal* cost of adding one spectral channel is exactly $d^2 + d/2 + 1$ parameters — an entire $d \times d$ projection plus one row of the gating MLP. Second, the fixed floor is large: even at $K = 0$ the layer would retain $\frac{17}{2}d^2 + \mathrm{O}(d)$ parameters (gate hidden layer, FFN, LayerNorms). Hence the prefix-K model is physically smaller than the full model, but *not* by a factor of $K/\overline{K}$: the reduction is $K/\overline{K}$ only on the channel-scalable part.

Concretely, ignoring $\mathrm{O}(d)$ terms, the ratio of prefix-K to full parameter counts is

$$\frac{P_{\text{layer}}(K)}{P_{\text{layer}}(\overline{K})} \;=\; \frac{K + \frac{17}{2}}{\overline{K} + \frac{17}{2}} \quad \xrightarrow[\overline{K}=32]{} \quad \frac{K + 8.5}{40.5}. \tag{27}$$

At $\overline{K} = 32$, prefix-K models occupy roughly $26\%,\ 41\%,\ 60\%$ of full-model parameters at $K \in \{2, 8, 16\}$ respectively — a meaningful reduction, but substantially less aggressive than the $K/\overline{K}$ ratio alone would suggest. We report the exact measured counts in table 11.

E.2 Per-token FLOPs at runtime budget K

We count multiply–accumulate operations (FLOPs) per input token, ignoring LayerNorm / activation elementwise work which is subdominant.

Summing the rows of table 8,

$$C_{\text{layer}}(K) \;=\; \underbrace{K\big(d^2 + \tfrac{d}{2} + \mathrm{O}(d\log L)\big)}_{K\text{-scalable}} \;+\; \underbrace{\tfrac{17}{2}\,d^2 + \mathrm{O}(d\log L)}_{\text{fixed floor}}. \tag{28}$$

Table 8: Per-token FLOPs ledger. The spectral FFT is amortized across the sequence; we report per-token cost by dividing the $\mathrm{O}(Ld \log L)$ sequence-level FFT cost by L.

Stage	FLOPs / token	Scales with K?
Gate MLP hidden	$d^2/2$	no
Gate MLP output (sliced)	$K\,d/2$	**yes**
Spectral rFFT (input)	$\mathrm{O}(d \log L)$	no
Spectral complex multiply	$\mathrm{O}(K\,d)$	**yes**
Spectral irFFT (output)	$\mathrm{O}(K\,d \log L)$	**yes**
Channel projection einsum	$K\,d^2$	**yes**
FFN (widening 4)	$8\,d^2$	no

Apart from the $d \log L$ term produced by the FFT, the FLOPs and parameter expressions are structurally identical. The channel projection Kd^2 dominates at large K, while the FFN $8d^2$ forms the compute floor at small K.

Remark 3 (Motivation for the $1/\sqrt{K}$ scaling)**.** *The factor $1/\sqrt{K}$ is a variance-preserving normalization under an idealized model of the* gated *channel contributions. Write $a_k(t) = \mathrm{sigmoid}(s_k(t))M_k(\Phi_k * u)(t)$, so that $[\mathcal{S}_K(u)](t) = K^{-1/2}\sum_{k=1}^{K} a_k(t)$. For a fixed sublayer input distribution, if $\mathbb{E}a_k(t) = 0$, $\mathbb{E}\langle a_i(t), a_j(t)\rangle = 0$ for $i \neq j$, and $\mathbb{E}\|a_k(t)\|_2^2 \leq C_a^2$ uniformly in k, then*

$$\mathbb{E}\|[\mathcal{S}_K(u)](t)\|_2^2 = \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\|a_k(t)\|_2^2 \leq C_a^2. \tag{29}$$

This calculation motivates the fixed $1/\sqrt{K}$ normalization used across training and inference budgets.

F Experimental setup

This appendix collects the per-experiment setup details that the main text deferred. Subsections F.1–F.4 cover datasets, baselines, training protocols, and hardware in turn. Per-experiment hyperparameter tables follow in subsections F.5–F.9.

F.1 Datasets and tasks

SlimPajama-627B (byte level). Our main language-modeling benchmark. We stream the SlimPajama-627B corpus [29] and tokenize raw UTF-8 bytes with a 258-symbol vocabulary (256 bytes plus PAD$=0$ and BOS$=1$). Sequences are sampled at length L=2048. Validation and test caches contain 1000 deterministic segments each, built once from a fixed seed and shared bit-for-bit across families.

Long Range Arena (LRA). We use six LRA-style long-sequence classification tasks [33]: LISTOPS (L=2,000), TEXT (IMDb bytes, L=4,096), RETRIEVAL (paired AAN byte sequences, L=4,000×2), SCIFAR (flattened grayscale images, L=1,024), PATHFINDER (L=1,024), and PATH-X (L=16,384). Reported metric is Top-1 accuracy.

Speech Commands V2 (SC-10). The ten-class subset of Google Speech Commands V2 [34]. Waveforms at 16 kHz are converted to 40-bin log-mel spectrograms with a 25 ms window and 10 ms hop, yielding L=100 frames per clip. Reported metric is Top-1 accuracy.

D4RL MuJoCo (offline RL). The nine canonical D4RL $v2$ datasets [35] across three locomotion environments: HalfCheetah, Hopper, Walker2d, each in MEDIUM/MEDIUM-REPLAY/MEDIUM-EXPERT regimes. Returns are normalized with the standard D4RL score, $\mathrm{norm}(G) = (G - G_{\mathrm{rand}})/(G_{\mathrm{expert}} - G_{\mathrm{rand}}) \times 100$, using the benchmark's random-policy and expert-policy reference returns for each environment.

F.2 Models and baselines

We compare ES-SSM against three families of baselines, organized by what each comparison isolates.

Non-elastic baselines. *Transformer++* follows the LLaMA recipe [30]: pre-norm RMSNorm, SwiGLU FFN, RoPE positional encoding, no biases, tied input/output embeddings. *Pythia-style* [31]

follows the GPT-NeoX/Pythia-style recipe: pre-norm LayerNorm, GELU FFN with widening factor 4, parallel residual, RoPE applied to 25% of head dimensions, untied embeddings, biases everywhere, with widths and depths resized for our parameter-matched budgets. *GLA* [32] is gated linear attention with low-rank gate projection and SwiGLU FFN. These three families serve as architecture-axis controls: at every reference budget $K \in \{2, 8, 16, 32\}$ we train an independent fixed-size model whose parameter count matches the corresponding ES-SSM exported compact model within $\pm 5\%$.

Elastic baselines. *MatFormer* [11] is the prior elastic Transformer; we use FFN-width nesting on top of the LLaMA backbone, with nested capacity targets matched to the ES-SSM runtime-budget set reported in table 11. *MatMamba* [12] is the prior elastic SSM with head-dimension nesting on Mamba-2. Both baselines train once under a capacity-sampling schedule over their nested capacity targets, matching the train-once budgeted-inference setting of ES-SSM.

Static spectral baselines. Where a fixed-Hankel-basis control is informative, we include a static spectral baseline that removes ES-SSM's input-adaptive gate and budget dropout. On SC-10 this is AR-STU [14]; on LRA this is the non-autoregressive STU baseline in Appendix G.2. These rows isolate the contribution of the two ES-SSM mechanisms over a fixed spectral basis.

Cited rows. The DT-paper baseline numbers [36] on D4RL MuJoCo are also reported as external references; the paired D4RL comparison uses DT (ours), trained and evaluated under the same protocol as ES-SSM.

F.3 Training protocols

Optimizer and schedule (language and classification). For SlimPajama, LRA, and SC-10, we use AdamW with $(\beta_1, \beta_2)=(0.9, 0.95)$, cosine LR decay after a short linear warmup, global ℓ_2 gradient clipping at 1.0, and `bf16` mixed precision. The ES-SSM spectral branch runs in `float32` as described in Appendix C.3. D4RL uses the separate Decision Transformer recipe in Appendix F.8.

Capacity-sampling schedule (elastic families). ES-SSM uses budget dropout as defined in algorithm 3. MatFormer and MatMamba use analogous sampling over their nested width/head targets. In the SlimPajama scaling experiment, all three elastic families are evaluated on the same ten runtime budgets, with MatFormer / MatMamba capacities indexed by the matched ES-SSM budget in table 11.

Exported compact models. After training, each elastic family exports a compact artifact for every evaluated capacity. ES-SSM slices the channel-indexed spectral tensors, including $\{\widehat{\Phi}_k\}$, $\{M_k\}$, rows of $\mathbf{W}_{g,2}$, and entries of $\mathbf{b}_{g,2}$. MatFormer and MatMamba slice their nested FFN-width or head-dimension tensors. Size, latency, and peak memory are measured on these exported compact models, not on the full-capacity training checkpoint.

F.4 Hardware

The main SlimPajama scaling experiment runs on a single NVIDIA B200 (180 GB HBM3e). The byte-level generality replication uses A100 (80 GB) GPUs. Ablations run on A100 (40 GB) cards. The D4RL MuJoCo experiments fit on RTX 3090 (24 GB) cards. All models are implemented in PyTorch 2.4+. Inference-cost comparisons use the same timing harness, batch size, sequence length, precision policy, and hardware within each benchmark. The ES-SSM spectral branch uses the batched FFT path of Appendix C.2.

F.5 SlimPajama scaling: hyperparameters

Table 9 lists the SlimPajama scaling-experiment hyperparameters, and table 10 lists the full-budget ES-SSM architecture used for the main run. The measured per-K parameter counts used in the main comparison are reported in table 11.

F.6 Long Range Arena: hyperparameters

The LRA setting is specified in the LRA results appendix rather than repeated here. Appendix G.2 gives the matched ~ 50 M architecture configurations, baseline roster, dataset variants, per-task training recipe, and seed policy; see tables 12 and 13.

Table 9: SlimPajama scaling: training hyperparameters. Identical across all six families except for the capacity schedule: elastic families use a capacity-sampling schedule over their runtime-budget set, while non-elastic families train at a single fixed K.

Knob	Value
Sequence length L	2048
Vocabulary	258 (bytes + PAD + BOS)
Token budget	30 B byte-tokens
Effective batch	16 sequences $\times$ 2 grad accum. $\times$ 2048 = 65,536 tokens / step
Optimizer	AdamW, β=(0.9, 0.95), ε=10^{-8}
Weight decay	0.1 on tensor parameters; 0 otherwise
Peak LR	3×10^{-4}
LR warmup	Linear, first 1% of steps
LR decay	Cosine to 10% of peak
Gradient clip	Global ℓ_2-norm 1.0
Mixed precision	`bf16` (B200 native), spectral branch in `fp32`
Dropout	0.1
Stochastic depth	$p_{\max} = 0.1$, linearly scaled across layers
ES-SSM budget dropout	algorithm 3
Budget warmup	First 1 B byte-tokens at full budget
Validation cadence	Every 1 B byte-tokens
Hardware	1$\times$ NVIDIA B200 (180 GB) per run

Table 10: SlimPajama main ES-SSM architecture. This is the full-budget configuration used to train the ES-SSM rows in table 11; smaller ES-SSM entries are exported compact models obtained by slicing this checkpoint along the spectral-channel axis.

Knob	Value
d_{model}	1262
n_{layers}	20
Maximum spectral budget $\overline{K}$	32
Runtime-budget set $\mathcal{K}$	$\{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$
Gate MLP hidden width	$d_{\text{model}}/2 = 631$
Gate output bias init.	-2.0
FFN	GELU MLP, widening factor 4
Dropout / stochastic depth	0.1 / linearly scaled to 0.1
Embedding / output head	Tied, vocabulary size 258
Full-budget parameters	around 1.3 B

F.7 Speech Commands V2: hyperparameters

We train on $\sim$25 K-clip SC-10 with the standard 80/10/10 split. Each 1-second clip yields a L=100 frame log-mel sequence with 40 mel bins. ES-SSM uses d=256, n_{layers}=4, $\overline{K}$=32. The classification head is a temporal mean pool followed by a single linear projection. Models train for 40 epochs with batch size 128, peak LR 3×10^{-4}, 5 seeds per cell.

F.8 D4RL: hyperparameters

D4RL [35] uses the offline-RL recipe of Decision Transformer [36]: AdamW with lr=10^{-4}, weight decay 10^{-4}, batch size 64, context length K_{ctx}=20, and 150,000 optimization steps. Final evaluation uses 100 episodes per (model, dataset, seed) at the DT-paper high target return, with three seeds per cell. ES-SSM trains once at $\overline{K}$=32 with the same budget-dropout sampler used elsewhere and exports compact models for the reported budgets; matched Transformer baselines are independently trained at the corresponding parameter budgets.

F.9 Byte-level generality replication: hyperparameters

We train *Transformer++*, MatFormer, and ES-SSM at around 330 M parameters and 10 B byte-tokens on the same SlimPajama byte stream. This replication checks that the byte-level data pipeline and

full-capacity validation protocol behave consistently before the main SlimPajama scaling study. Tokens-per-step are held at $65{,}536$ for all architectures, with AdamW, cosine decay, gradient clipping, and `bf16` AMP matched across runs. Validation BPB is logged every $200\,\mathrm{M}$ tokens on a fixed 1000-segment cache.

G Additional main results

This appendix gives the full tables behind the main result figures, including per-K sweeps, training trajectories, and per-task breakdowns.

G.1 SlimPajama: full per-K results

Table 11 gives the full paired quality–cost table behind figure 3. It covers the complete runtime-budget set for the elastic families and the independently trained reference points for the non-elastic baselines.

All ES-SSM rows over K are exported compact models from one trained $\overline{K} = 32$ checkpoint. Different parameter budgets for Transformer++, Pythia-style, and GLA are independently trained models, so their training cost is budget-specific.

G.2 Long Range Arena

Setup. We evaluate the six LRA tasks at the $\sim 50\,\mathrm{M}$ parameter scale. We use 3 seeds per cell on all six tasks, including PATH-X with its $16{,}384$-token sequence length. ES-SSM trains once at $\overline{K}{=}32$ and exports compact models for $K \in \mathcal{K}{=}\{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$. All four models are matched at the $\sim 50\,\mathrm{M}$ scale with $d_{\text{model}}{=}512$ (table 12).

Baselines. We compare against Transformer with FlashAttention-2 [37], Mamba-2 [38], and STU [14]. STU is the closest mechanism baseline: it uses the same fixed Hankel top-K basis but lacks the input-adaptive sigmoid gate and budget-dropout training.

Datasets. We use the standard LRA task definitions: LISTOPS, byte-level IMDb TEXT, AAN RETRIEVAL, sCIFAR, PATHFINDER, and PATH-X [39–41, 33]. Sequence lengths are listed in table 13; PATH-X is the longest task at $16{,}384$ tokens.

Training recipe. All four models use AdamW ($\beta{=}(0.9, 0.95)$, weight decay 0.05, gradient clipping 1.0), bf16 mixed precision, gradient checkpointing, and a cosine LR schedule with learning-rate warmup over the first 10% of optimization steps for LISTOPS, TEXT, RETRIEVAL, and PATH-X, and 5% for sCIFAR and PATHFINDER. Per-task batch sizes, training epochs, and peak learning rates are listed in table 13.

LRA budget schedule. ES-SSM uses no initial full-budget warmup ($E_{\text{warmup}} = 0$). With a zero-based optimization-step counter j, all micro-batches in steps satisfying $j \bmod 8 = 0$ use $K_{\text{train}} = \overline{K} = 32$. At other steps, each micro-batch draws its own budget using the log-scale sampler in equation (5) over $\mathcal{K} = \{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$. Thus the full-budget anchor cadence is $n_{\text{full}} = 8$ optimization steps; this schedule is separate from learning-rate warmup.

Environment. We use PyTorch $2.5.1$ with CUDA 12.4. Mamba-2 uses its triton SSD kernel, Transformer attention uses PyTorch's scaled-dot-product attention, and STU / ES-SSM use our implementation.

Full-capacity comparison. table 14 reports Top-1 test accuracy at the full-capacity $\overline{K}{=}32$ across all six tasks. ES-SSM gives the best result on every task in this matched-scale comparison. The strongest baseline is the static spectral STU model, but ES-SSM still improves it consistently, with the largest margins on TEXT, RETRIEVAL, and PATH-X; these results support the role of the adaptive gate and budget-dropout training beyond a fixed Hankel spectral basis.

Budgeted inference and sweet spots. Table 15 and figure 5 report ES-SSM's exported compact models from the same $\overline{K}{=}32$ training run at all ten budgets in $\mathcal{K}$ (mean over 3 seeds for every task). Under a 98%-of-$\overline{K}$ retention threshold the sweet-spot budget $K^\star$ varies by task; below the 90%-retention collapse boundary accuracy degrades much faster, consistent with the budget-dropout training pressure described in section 4. On these completed sweeps, the 98% sweet spots are $K^\star{=}6$ for LISTOPS, 6 for TEXT, 4 for RETRIEVAL, 3 for SCIFAR, 16 for PATHFINDER, and 12 for PATH-X.

Under the 90% threshold, only LISTOPS and PATH-X cross the collapse boundary at $K{=}2$; the other tasks remain above it across the evaluated budget range.

G.3 Speech Commands V2 (audio classification)

Setup. SC-10 is the standard 10-keyword subset of Google Speech Commands V2 [34]. We convert each 1-s waveform to a length-100 sequence of 40-bin log-mel frames and use the official train, validation, and test splits. All sequence models share the same linear audio front-end and mean-pooling classification head, and all reported numbers use the same augmentation and optimizer recipe over 5 seeds.

Baselines and protocol. We compare ES-SSM against Transformer++, Conformer [42], Mamba [43], Audio-Mamba [44], and AR-STU. All models are matched to within $\pm 5\%$ of 10 M trainable parameters. ES-SSM trains once at $\overline{K}{=}32$ with the same budget dropout schedule used elsewhere, and exports compact models for the ten runtime budgets $\mathcal{K}{=}\{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$.

Results. table 16 shows that ES-SSM reaches 97.40% full-capacity accuracy, within 0.27 pp of Conformer and 0.19 pp of the bidirectional Audio-Mamba baseline, while improving the static spectral AR-STU baseline by 2.03 pp. This is the mechanism-relevant comparison because AR-STU uses the same Hankel spectral channels but lacks the input-adaptive gate and budget dropout. ES-SSM also matches the best calibration in the table and remains close to AR-STU's FFT-based latency, while running substantially faster than the Conformer and selective-scan baselines.

Budgeted inference. table 17 reports exported compact ES-SSM(K) classifiers obtained by direct prefix truncation. The main point is that most of the full-capacity audio accuracy is already present at very small budgets: $K{=}3$ retains 99.55% of the $\overline{K}{=}32$ score, and $K{=}8$ is the best operating point in this sweep. Even $K{=}2$ exceeds the full-capacity AR-STU baseline, showing that the exported compact models are usable well below the full spectral budget.

G.4 D4RL offline reinforcement learning

Setup. We replace the Transformer block in Decision Transformer [36] with ES-SSM and keep the standard interleaved (R_t, s_t, a_t) token layout. Models are evaluated by target-return-conditioned rollouts on the nine D4RL MuJoCo datasets [35], using normalized return over 3 seeds and 100 episodes per cell. ES-SSM trains once at $\overline{K}{=}32$ and exports compact models at $K \in \{2, 4, 8, 16, 32\}$ by direct prefix truncation; the matched Transformer baselines are separate models trained independently at the corresponding parameter budgets. Therefore, covering the same set of D4RL operating points requires one ES-SSM training run but multiple Transformer training runs, with the latter incurring substantially larger total training time and compute; table 18 summarizes this accounting.

This accounting is central to the D4RL comparison. A matched Transformer can be competitive at an isolated parameter budget, but covering the full five-budget deployment set requires 5 independently trained models per dataset–seed cell. ES-SSM obtains the same five operating points from one trained checkpoint, reducing total D4RL training wall-clock from 107.84 h to 23.99 h.

Full-capacity results. table 19 shows that ES-SSM at $K{=}32$ improves the average normalized return over our matched DT baseline by $+4.97$ points. The gain is not uniform across every dataset: it is driven mainly by HOPPER-MEDIUM-REPLAY, WALKER2D-MEDIUM-REPLAY, and HOPPER-MEDIUM-EXPERT, while the remaining tasks are close to the DT baseline. The important takeaway is narrower but useful: the budget-dropout training recipe does not create a full-capacity penalty on this control benchmark.

Exported compact models. table 20 compares the exported compact ES-SSM models with separately trained Transformers at matched parameter counts. ES-SSM is ahead for $K \geq 8$, and the smallest two budgets are within 0.15 normalized-return points of the independently trained Transformers. This makes the deployment claim precise: one ES-SSM training gives several useful operating points on the parameter–quality curve, while matching those points with Transformers requires separate trainings.

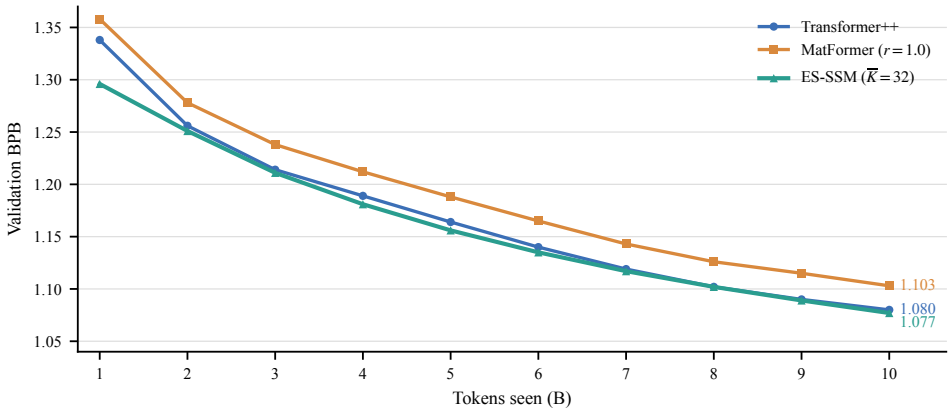

Figure 10: Byte-level generality training trajectories. Full-capacity validation BPB versus tokens seen for the three architectures at the around 330 M-parameter / 10 B-byte-token replication scale. Same data as table 22; lower is better.

G.5 Byte-level generality replication

Setup. We run a cross-architecture replication of the byte-level training protocol at around 330 M parameters / 10 B byte-tokens. Three architectures (*Transformer++*, MatFormer, ES-SSM) are trained on the same SlimPajama byte stream with identical optimizer, schedule, effective batch (tokens-per-step = 65,536), seed, validation cache, and gradient-clip; only the architecture differs. The role of this study is to verify that the shared framework reproduces a stable, monotonically converging full-capacity BPB curve for every architecture, and that the relative ordering at full capacity behaves as expected before the main SlimPajama scaling study.

Result. Table 21 reports the final validation BPB at full capacity ($\overline{K}$=32 for ES-SSM, full FFN width for MatFormer). Table 22 traces the full-capacity validation BPB at every tenth of the token budget, and figure 10 shows the same trajectories.

Key observations. tables 21 and 22 show three basic checks.

(*i*) *Matryoshka cost is small.* MatFormer reaches 1.103 BPB versus 1.080 for the same-recipe Transformer++, a +23 mBPB cost for supporting four nested FFN ratios.

(*ii*) *ES-SSM is competitive at full capacity.* ES-SSM at $\overline{K}$=32 reaches 1.077 BPB, essentially tied with Transformer++ at 1.080 and ahead of MatFormer at 1.103. Because this replication is single-seed and ES-SSM has a modestly larger parameter count, we read the Transformer++ comparison as parity rather than a large win.

(*iii*) *The shared framework reproduces.* All three validation trajectories descend monotonically, supporting the use of the same byte-level pipeline in the larger study.

H Ablations and mechanism diagnostics

This section isolates the contribution of each design choice in the ES-SSM spectral sublayer (section 4.1) and the budget dropout training schedule (section 4.2). The first three subsections are one-variable ablations: each row changes a single knob relative to the reference recipe and holds everything else fixed. The final two subsections are mechanism diagnostics that measure how the trained model uses its spectral channels, rather than the downstream effect of removing one.

H.1 Protocol

All ablations use the same ES-SSM diagnostic preset: $d = 1024$, $n_{\text{layers}} = 18$, $\overline{K} = 32$, sequence length $L = 2048$, and approximately 770 M trainable parameters. Each run trains on SlimPajama for 400 M

tokens with AdamW, `bf16` AMP, the same effective batch, and seed 86; remaining hyperparameters follow Appendix C.7. Each ablation differs from the reference in exactly one setting.

We evaluate validation BPB at $K \in \{2, 3, 4, 5, 6, 8, 12, 16, 24, 32\}$ so that each row shows both full-capacity and low-budget behavior. Values are next-byte BPB on the SlimPajama validation cache, taken at the best-val checkpoint. The ablations are single-seed; pilot runs put the reference variance near 0.005 BPB at $K = \overline{K}$, and captions flag differences at or below that noise floor.

H.2 Budget dropout

Budget dropout is the training-time mechanism that exposes the model to multiple runtime budgets. We compare the reference recipe against training the same architecture only at $K_{\text{train}} = \overline{K}$, followed by direct prefix truncation at evaluation.

H.3 Adaptive channel gating

The sigmoid gate $g_k^{(K)}(t) = K^{-1/2}\,\text{sigmoid}(s_k(t))$ makes channel contributions input-dependent. Two degenerate limits test whether input adaptivity is the operative property: a uniform gate $g_k^{(K)} = 1/\sqrt{K}$, constant across channels and across t, and a per-channel scalar gate that learns one trainable weight per channel but remains input-independent.

H.4 Full-budget anchor cadence n_{full}

This subsection gives a coarse one-seed ablation of the full-budget anchor cadence; the finer validation justifying the default is in Appendix D. After warmup, forcing $K_{\text{train}} = \overline{K}$ every n_{full} steps gives each channel an update-rate floor of $1/n_{\text{full}}$. Table 25 checks the endpoints and several non-endpoint cadences.

The remaining two subsections describe properties of the trained reference model rather than the effect of a removed component, anchoring the interpretive claims made above.

H.5 Gate statistics

For a fixed sublayer input, direct prefix truncation preserves the gate value of every retained channel: for any retained channel $k \le K$, $\text{sigmoid}(s_k(t))$ is identical to the value computed in the full-capacity model because slicing away later gate-output rows does not affect row k. The only budget-dependent term applied to a retained channel is the deterministic scale $1/\sqrt{K}$. Together with slicing the channel-indexed tensors $\{M_k\}$ and $\{\Phi_k\}$, this lets each exported compact model reuse the same learned prefix without channel search or budget-specific reordering. In this diagnostic subsection, we write the unscaled gate activity as $\gamma_k(t) \triangleq \text{sigmoid}(s_k(t))$ and its held-out mean as $\bar{\gamma}_k$; this is distinct from the budgeted gate coefficient $g_k^{(K)}(t) = K^{-1/2}\gamma_k(t)$ used in the model forward pass.

H.6 Effective gate support

We quantify how concentrated the gate activity is across spectral channels with two complementary measures. Both are computed from the per-channel mean unscaled gate activities $\bar{\gamma}_k$ measured on the same held-out cache as table 26, using all spectral channels for each gate variant:

$$K_{\text{eff}}^{\text{PR}} \triangleq \Big(\sum_k \bar{\gamma}_k\Big)^2 \Big/ \sum_k \bar{\gamma}_k^2, \qquad \text{(participation ratio)}$$

$$K_{\text{eff}}^{(\tau)} \triangleq \#\{k : \bar{\gamma}_k \ge \tau\}. \qquad \text{(threshold count at } \tau\text{)}$$

The participation ratio equals the number of channels included in the sum when the gate distribution is uniform and 1 when a single channel dominates; $K_{\text{eff}}^{(0.05)}$ counts channels whose mean unscaled gate activity is at least 0.05 on the held-out cache – a coarser but more interpretable proxy.

Table 11: Byte-level SlimPajama main scaling results (1.3B family, 30B byte-token training). Elastic families (ES-SSM, MatFormer, MatMamba) train once at $\overline{K} = 32$ and are evaluated at all ten runtime budgets in $\mathcal{K}$; non-elastic families (Transformer++, Pythia-style, GLA) contribute four independently trained reference budgets. These non-elastic rows are high-resource reference baselines rather than equal-training-cost competitors, since supporting multiple budgets requires multiple separate training runs. Quality is bits-per-byte (lower is better); inference cost is measured at sequence L=2048, batch 1, on a single B200 GPU. For ES-SSM, the ten K values instantiate ten exported parameter budgets, spanning roughly 330 M to around 1.3 B. A small black downward triangle marks a collapsed test result whose Test BPB is more than 90% higher than the best Test BPB within the same family. All BPB values are rounded to two decimal places.

		Quality			**Inference cost**		
Family	K	Params (M)	Val BPB	Test BPB	Lat. (ms)	Tput (kt/s)	Mem (GB)
ES-SSM (ours)	2	335.00	1.40	1.41	6.18	331.40	0.87
ES-SSM (ours)	3	366.90	1.22	1.23	7.50	273.10	0.93
ES-SSM (ours)	4	398.70	1.10	1.10	8.65	236.80	1.00
ES-SSM (ours)	5	430.60	1.05	1.06	9.98	205.20	1.06
ES-SSM (ours)	6	462.50	1.02	1.03	11.34	180.60	1.13
ES-SSM (ours)	8	526.20	0.99	0.99	14.13	144.90	1.25
ES-SSM (ours)	12	653.70	0.98	0.98	19.48	105.10	1.51
ES-SSM (ours)	16	781.10	0.96	0.97	28.60	71.60	1.76
ES-SSM (ours)	24	1036.00	0.95	0.95	39.42	51.90	2.27
ES-SSM (ours)	32	1291.00	0.95	0.96	59.83	34.20	2.78
MatFormer	2	341.80	5.32	5.35 ▼	4.52	453.10	0.84
MatFormer	3	379.50	5.04	5.06 ▼	5.01	408.80	0.93
MatFormer	4	419.40	4.89	4.93 ▼	5.34	383.50	1.02
MatFormer	5	460.00	4.63	4.68 ▼	5.88	348.30	1.12
MatFormer	6	502.10	4.33	4.37 ▼	6.21	329.80	1.22
MatFormer	8	577.40	3.76	3.78 ▼	7.19	284.80	1.40
MatFormer	12	718.90	3.00	3.03 ▼	8.46	242.10	1.74
MatFormer	16	855.90	2.15	2.17 ▼	10.03	204.20	2.05
MatFormer	24	1078.60	1.24	1.27	12.03	170.20	2.59
MatFormer	32	1310.70	0.98	0.99	14.32	143.00	3.20
MatMamba	2	262.00	1.63	1.65	3.02	678.10	0.58
MatMamba	3	333.00	1.29	1.30	3.71	552.00	0.74
MatMamba	4	404.00	1.10	1.12	4.34	471.90	0.89
MatMamba	5	475.00	1.06	1.08	4.99	410.40	1.05
MatMamba	6	547.00	1.04	1.05	5.81	352.50	1.21
MatMamba	8	689.00	1.02	1.04	7.27	281.70	1.52
MatMamba	12	792.00	1.01	1.02	8.09	253.20	1.75
MatMamba	16	894.00	1.00	1.01	8.81	232.50	1.97
MatMamba	24	1099.00	1.00	1.01	10.28	199.20	2.43
MatMamba	32	1310.00	1.00	1.00	11.96	171.20	2.89
Transformer++	2	354.40	1.04	1.06	5.11	400.60	0.75
Transformer++	8	510.30	1.00	1.01	5.74	356.90	1.03
Transformer++	16	762.10	0.96	0.96	7.15	286.30	1.52
Transformer++	32	1323.80	0.93	0.94	11.49	178.30	2.63
Pythia-style	2	341.80	1.06	1.07	4.56	449.30	0.81
Pythia-style	8	521.30	1.01	1.02	5.64	362.90	1.15
Pythia-style	16	798.80	0.97	0.99	6.58	311.50	1.68
Pythia-style	32	1286.40	0.94	0.94	9.57	214.10	2.61
GLA	2	335.00	1.07	1.08	3.94	519.80	0.62
GLA	8	526.00	1.02	1.02	4.46	459.20	0.94
GLA	16	781.00	0.98	1.00	5.31	385.70	1.41
GLA	32	1300.00	0.96	0.97	8.27	247.60	2.39

Table 12: Per-architecture configurations at the matched ~ 50 M parameter scale. All models share $d_{\text{model}}=512$. Layer counts are tuned so each backbone stays in the same effective parameter range; embedding and classifier head are tiny in comparison.

Model	d_{model}	n_{layers}	Specific	Params
Transformer (FA-2)	512	16	$h=8$, RoPE	50.8 M
Mamba-2	512	13	$d_{\text{state}}=128$, headdim=64	50.1 M
STU	512	5	$K=32$, fixed Hankel basis	52.8 M
ES-SSM (ours)	512	5	$\overline{K}=32$, adaptive gate, budget dropout	53.5 M

Table 13: Per-task training recipe. Effective global batch is `batch` $\times$ `grad_accum`.

Task	Seq. len.	Batch	Grad accum	Epochs	LR (peak)
LISTOPS	2,000	32	4	6	3×10^{-4}
TEXT	4,096	16	4	5	2×10^{-4}
RETRIEVAL	$4{,}000\times2$	8	4	4	2×10^{-4}
SCIFAR	1,024	64	4	10	4×10^{-4}
PATHFINDER	1,024	64	4	4	3×10^{-4}
PATH-X	16,384	4	8	2	1×10^{-4}

Table 14: LRA full-capacity results at the matched ~ 50 M parameter scale ($\overline{K}=32$). Top-1 test accuracy (%), mean$\pm$std over 3 seeds for every task. Bold marks the per-task best.

Model	LISTOPS	TEXT	RETRIEVAL	SCIFAR	PATHFINDER	PATH-X
Transformer (FA-2)	48.70±0.50	76.38±0.92	70.84±0.42	61.21±0.56	84.12±0.30	FAIL
Mamba-2	49.60±0.41	85.05±0.59	81.31±1.10	78.08±0.66	81.03±0.52	74.02±0.53
STU	60.10±0.03	89.25±1.15	89.68±0.26	91.24±0.14	94.28±0.27	88.01±0.32
ES-SSM (ours)	**61.70±0.68**	**92.77±0.60**	**91.49±0.72**	**92.88±0.24**	**94.44±0.20**	**93.08±0.13**

Table 15: ES-SSM budget sweep on a single $\overline{K}=32$ training run, ~ 50 M scale. Test top-1 accuracy (%), mean over 3 seeds for every task. Underlined entries mark the smallest K that retains $\geq 98\%$ of the $\overline{K}$ score (sweet spot $K^\star$).

Task	2	3	4	5	6	8	12	16	24	32
LISTOPS	46.77	56.09	59.73	60.28	<u>60.77</u>	61.08	61.33	61.51	61.64	61.70
TEXT	86.46	87.67	89.52	90.45	<u>91.10</u>	91.75	92.21	92.49	92.68	92.77
RETRIEVAL	84.90	88.38	<u>90.30</u>	90.48	90.85	91.12	91.31	91.40	91.44	91.49
SCIFAR	89.18	<u>91.63</u>	92.17	92.41	92.37	92.55	92.69	92.73	92.81	92.88
PATHFINDER	85.94	88.58	90.57	91.23	91.61	91.80	92.55	<u>93.68</u>	94.16	94.44
PATH-X	81.91	85.17	87.50	88.89	89.82	90.75	<u>91.68</u>	92.34	92.80	93.08

Table 16: SC-10 full-capacity accuracy, inference cost, and calibration. Top-1 accuracy and Expected Calibration Error [45] use mean$\pm$std over 5 seeds. Per-sample latency is measured at batch 1 on an A100 80GB with bfloat16, using the median over 30 warmed-up runs. All models are matched at $\sim$ 10 M parameters. Bold marks the best mean in each column; the ES-SSM row is the single full-capacity training run from which table 17 is exported.

Model	Params	Top-1 (%)	Latency (ms)	ECE
Conformer	10.31 M	$\mathbf{97.67 \pm 0.26}$	13.82 ± 0.19	0.067 ± 0.014
Transformer++	10.64 M	91.92 ± 1.71	4.23 ± 0.07	0.084 ± 0.014
Mamba (causal)	10.20 M	97.19 ± 0.52	9.15 ± 0.10	0.086 ± 0.025
Audio-Mamba (bidir.)	10.20 M	97.59 ± 0.26	9.34 ± 0.18	$\mathbf{0.059 \pm 0.005}$
AR-STU	10.51 M	95.37 ± 0.07	$\mathbf{2.34 \pm 0.04}$	0.072 ± 0.006
ES-SSM (ours, $\overline{K}=32$)	10.66 M	97.40 ± 0.29	2.97 ± 0.02	$\mathbf{0.059 \pm 0.003}$

Table 17: SC-10 budget sweep on a single $\overline{K}$=32 ES-SSM training run. Compact ES-SSM(K) models are exported by direct prefix truncation of the channel projection stack $\{M_k\}$ and Hankel spectral channels to the prefix K, so the Params column reflects a deployable standalone model. Top-1 test accuracy is mean±std over 5 seeds; retention is relative to K=$\overline{K}$=32. Bold marks the peak accuracy across all K.

K	Params	Top-1 Acc. (%)	Retention (%)	Budget (%)	Latency (ms)
2	2.78 M	96.25 ± 0.35	98.81	6.25	2.95
3	3.04 M	96.96 ± 0.24	99.55	9.38	2.94
4	3.30 M	97.44 ± 0.28	100.05	12.50	2.97
5	3.56 M	97.61 ± 0.20	100.22	15.62	2.97
6	3.83 M	97.66 ± 0.25	100.28	18.75	2.96
8	4.35 M	$\mathbf{97.73 \pm 0.15}$	**100.34**	25.00	2.98
12	5.40 M	97.72 ± 0.10	100.33	37.50	2.95
16	6.45 M	97.61 ± 0.16	100.22	50.00	2.95
24	8.56 M	97.52 ± 0.20	100.13	75.00	3.00
32	10.66 M	97.40 ± 0.29	100.00	100.00	2.95

Table 18: D4RL training-cost accounting. A cell denotes one dataset–seed pair; the full suite contains 9 datasets and 3 seeds (27 cells). ES-SSM trains one full-capacity model per cell at $\overline{K}$=32 and exports all five runtime budgets by direct prefix truncation. The matched Transformer suite trains one independent model for each budget $K \in \{2, 4, 8, 16, 32\}$. Times are single-RTX-3090 training wall-clock; rollout evaluation time is excluded.

Protocol	Trainings / cell	Time / training	Suite time / cell	Total time	Relative total
ES-SSM exported suite	1	53.3 min	53.3 min	23.99 h	1.00×
Transformer 5-K suite	5	47.9 min	239.6 min	107.84 h	4.50×
ES-SSM reduction	5× fewer	–	186.3 min saved	83.85 h saved	77.8% lower

Table 19: D4RL MuJoCo: normalized returns. Higher is better. Mean ± std over 3 seeds, 100 rollout episodes per cell. "DT-paper" is the originally reported number [36]; "DT (ours)" is the same architecture trained and evaluated under our recipe (150 k steps, 100-episode rollout) for paired comparison. ES-SSM at K=32 matches DT in parameter count (711 k vs. 730 k). Bold ES-SSM cells: ES-SSM $\geq$ DT(ours). Bold Δ: $|\Delta| > 5$.

Dataset	DT-paper	DT (ours)	**ES-SSM** (K=32)	Δ vs. DT (ours)
halfcheetah-medium	42.6	42.22±0.23	**42.30±0.09**	+0.08
halfcheetah-medium-replay	36.6	34.64±0.80	33.64±2.37	−1.00
halfcheetah-medium-expert	86.8	91.12±0.94	**91.31±0.77**	+0.19
hopper-medium	67.6	50.91±2.51	**52.59±2.74**	+1.68
hopper-medium-replay	82.7	47.96±19.93	**79.50±4.02**	**+31.54**
hopper-medium-expert	107.6	82.83±17.59	**88.06±9.74**	**+5.23**
walker2d-medium	74.0	72.86±2.20	71.41±3.04	−1.45
walker2d-medium-replay	66.6	61.04±6.01	**69.17±2.50**	**+8.13**
walker2d-medium-expert	108.1	103.72±1.14	**104.09±8.73**	+0.37
Average (9 datasets)	74.7	65.26	**70.23**	+4.97

Table 20: D4RL MuJoCo inference table. ES-SSM rows are exported compact models from a single $\overline{K}=32$ training run; Transformer rows are independently trained baselines (one row = one training). Norm. return averaged over 9 datasets, 3 seeds, 100 rollout episodes. Inference cost measured at batch 1, $K_{\mathrm{ctx}}=20$ on RTX 3090. Bold marks the higher normalized return within each matched budget pair.

Family	K	Params	Norm. return	Lat. (ms)	Tput (kt/s)	Mem (MB)
ES-SSM exported compact models (one $\overline{K}=32$ training)						
ES-SSM	32	711 k	**70.23**	2.75	21.9	16
ES-SSM	16	461 k	**65.94**	2.74	21.9	13
ES-SSM	8	335 k	**66.55**	2.73	22.0	12
ES-SSM	4	273 k	66.43	2.69	22.3	11
ES-SSM	2	241 k	65.02	2.74	21.9	10
Transformer baseline (one independent training per row)						
Transformer (DT)	32	729 k	65.26	1.61	37.5	12
Transformer	16	436 k	65.73	1.58	38.3	11
Transformer	8	317 k	64.37	1.58	38.2	11
Transformer	4	265 k	**66.57**	1.56	38.5	10
Transformer	2	265 k	**65.06**	1.55	38.7	10

Table 21: Byte-level generality replication at around 330 M parameters and 10 B byte-tokens on SlimPajama. Validation BPB on the shared 1000-segment cache; lower is better. All three runs reach the full 10 B-token budget cleanly and write a final `best.pt` at the last validation gate.

Family	Params	Effective batch	**Final val. BPB** (↓)	Wall
Transformer++ (full)	304 M	65,536 tokens/step	1.0800	∼62 h
MatFormer (full width)	304 M	65,536 tokens/step	1.1026	∼56 h
ES-SSM ($\overline{K}=32$)	335 M	65,536 tokens/step	**1.0771**	∼66 h

Table 22: Full-capacity validation BPB versus tokens seen. Each row is one architecture; columns step the byte-token budget from 1 B to 10 B in 1 B increments. All three trajectories descend monotonically with no observed instability or recovery spike.

Tokens (B)	1	2	3	4	5	6	7	8	9	10
Transformer++	1.338	1.256	1.214	1.189	1.164	1.140	1.119	1.102	1.090	1.080
MatFormer (full width)	1.358	1.278	1.238	1.212	1.188	1.165	1.143	1.126	1.115	1.103
ES-SSM ($\overline{K}=32$)	1.296	1.251	1.211	1.181	1.156	1.135	1.117	1.102	1.089	**1.077**

Table 23: Without budget dropout, the exported compact model at budget K is a post hoc direct prefix truncation of a full-capacity model and is not optimised for any operating point below $\overline{K}$. At full capacity, the reference scores 1.826 BPB versus 1.796 without budget dropout, a 0.030 BPB penalty in this single-seed comparison. In exchange, budget dropout improves BPB by 0.975 at $K=8$, 2.710 at $K=3$, and a peak 2.749 at $K=4$. These low-budget gains are substantially larger than the full-capacity penalty and the reported seed noise of this protocol. The non-monotonicity of the no-budget-dropout row, where $K=3$ scores worse than $K=2$ (4.583 vs. 3.974 BPB) and $K=4$ even worse still, makes the same point structurally: without budget dropout, channel ordering under direct prefix truncation is not a controlled accuracy–compute knob, and adding one more channel at a small runtime budget can hurt because that channel was never trained to be used in isolation.

Training regime	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=8$	$K=12$	$K=16$	$K=24$	$K=32$
ES-SSM (reference)	1.965	1.873	1.847	1.833	1.827	1.822	1.821	1.822	1.824	1.826
No budget dropout ($\overline{K}$)	3.974	4.583	4.596	4.432	3.555	2.797	2.079	1.895	1.810	1.796

Table 24: Both degeneracies cost BPB at every K. At $K = \overline{K}$ the uniform gate costs 0.226 BPB and the per-channel scalar 0.114 BPB relative to the input-adaptive reference; the corresponding gaps at $K = 2$ are 0.281 and 0.139 BPB. All four main gaps are an order of magnitude above the seed noise floor. The per-channel scalar variant sits between the uniform gate and the input-adaptive reference at every K and accounts for 0.112 of the 0.226 BPB total gating gain at $K = \overline{K}$ — almost exactly half — suggesting that learned per-channel reweighting is mildly useful in isolation but that the dominant share of the gain comes from conditioning on $u(t)$. The split should be read with the caveat that the adaptive and per-channel variants differ in capacity (a two-layer MLP versus $K = 32$ scalars), so the comparison changes both the gate's functional form and its parameter count; a parameter-matched test would require a wider architectural search beyond this ablation. Direct evidence for input-conditionality of the reference gate appears in table 26, and the order-of-magnitude difference in effective gate support between adaptive and non-adaptive variants appears in table 27.

Gating variant	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=8$	$K=12$	$K=16$	$K=24$	$K=32$
Adaptive sigmoid gate (reference)	1.965	1.873	1.847	1.833	1.827	1.822	1.821	1.822	1.824	1.826
Uniform $g_k^{(K)} = 1/\sqrt{K}$	2.246	2.131	2.091	2.068	2.058	2.049	2.043	2.041	2.045	2.052
Per-channel scalar, input-independent	2.104	2.000	1.965	1.947	1.939	1.933	1.929	1.929	1.933	1.940

Table 25: Coarse full-budget anchor cadence ablation. The endpoint $n_{\text{full}} = 1$ collapses budget dropout to pure full-capacity training and shows the same direct-truncation failure mode as table 23, as expected. Among the non-endpoint budget-dropout settings ($\infty, 16, 8, 4$), the curve is flat to within 0.02 BPB at every K, so this sparse one-seed table should not be used to rank these cadences. The default $n_{\text{full}} = 8$ is justified by the finer three-seed validation in Appendix D; this table only shows that reasonable non-endpoint cadences preserve the budget dropout behavior while $n_{\text{full}} = 1$ does not.

n_{full} (steps)	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=8$	$K=12$	$K=16$	$K=24$	$K=32$
∞ (never force $\overline{K}$)	1.955	1.869	1.845	1.832	1.826	1.823	1.823	1.825	1.829	1.834
16	1.960	1.871	1.847	1.834	1.828	1.825	1.824	1.825	1.828	1.831
8 (reference)	1.965	1.873	1.847	1.833	1.827	1.822	1.821	1.822	1.824	1.826
4	1.974	1.877	1.851	1.836	1.828	1.823	1.821	1.821	1.822	1.824
1 (no budget dropout)	3.974	4.583	4.596	4.432	3.555	2.797	2.079	1.895	1.810	1.796

Table 26: Selected-channel sigmoid gate statistics, averaged over all positions and held-out sequences and across layers. The trained model assigns substantial gate mass to the low-index channels ($\bar{\gamma}_k \approx 0.33$ at $k = 2$ and $\bar{\gamma}_k \approx 0.15$ at $k = 3$), and the per-channel standard deviation matches or exceeds the per-channel mean at every listed channel (most visibly 0.231 vs. 0.150 at $k = 3$), so the gate is strongly input-conditional rather than saturating at a constant. Several high-index channels that are nearly closed on average ($k \geq 8$, $\bar{\gamma}_k \leq 0.01$ in our seq-2048 measurements) still reach maximum gate values from about 0.4 to nearly 1.0 on specific token contexts; the gate is opportunistically opening these channels rather than learning a hard mask. This pattern of input-conditional, channel-specific use is the load-bearing property table 24 identifies.

Channel index k	Mean γ_k	Std γ_k	Min γ_k	Max γ_k
2	0.326	0.334	0.000	1.000
3	0.150	0.231	0.000	0.999
4	0.068	0.142	0.000	0.995
5	0.032	0.091	0.000	0.987
6	0.024	0.061	0.000	0.969
8	0.007	0.029	0.000	0.957
12	0.001	0.001	0.000	0.075
16	0.001	0.004	0.000	0.443
24	0.001	0.004	0.000	0.665
32	0.001	0.007	0.000	0.550

Table 27: Effective gate support for the gate variants, computed from the per-channel mean unscaled gate activities measured on the held-out validation cache. The adaptive gate concentrates gate activity into a small number of channels: the reference model has participation ratio 3.84, and only four channels exceed the 0.05 mean-activity threshold. Removing budget dropout changes this count only modestly, while the non-adaptive variants fail to sparsify: the uniform gate uses all channels by construction, and the per-channel scalar remains broadly spread across the channel set. This supports the interpretation that input-adaptive gating, not merely a static channel weight, concentrates gate activity onto a compact subset of channels. This diagnostic measures gate activity only, rather than the full channel contribution $g_k^{(K)}(t)M_k(\Phi_k * u)(t)$; low mean-activity channels may still provide input-specific high-budget refinements.

Variant	$K_{\text{eff}}^{\text{PR}}$	$K_{\text{eff}}^{(0.05)}$	$K_{\text{eff}}^{(0.01)}$
Adaptive sigmoid gate (reference)	3.84	4	7
Adaptive, no budget dropout (NO_BD)	4.35	4	7
Per-channel scalar (GATE_STATIC)	28.25	32	32
Uniform $1/\sqrt{K}$ (GATE_UNIFORM)	32.00	32	32

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the train-once, export-many goal, the ES-SSM mechanisms, and the experimental scope. The paper also states the main limitations of scale and implementation speed.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a limitations paragraph discussing the largest reported scale, training-token budget, and current latency/throughput limitations. Additional single-seed or diagnostic limitations are identified in the appendix where applicable.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The assumptions and derivations for the Hankel spectral parameterization are stated in the preliminaries and Appendix B. Empirical claims are separated from these derivations.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The method, algorithms, runtime-budget set, training schedule, datasets, baselines, metrics, and hyperparameters are specified in Methods, Experiments, and Appendix F.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and documentation are publicly available at `https://github.com/GMU-MICO/ES-SSM`. All datasets used are public datasets, with sources cited in the paper and appendix.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text gives the evaluation protocol, and the appendix reports dataset construction, splits, model configurations, optimizer settings, precision policy, and benchmark-specific hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Tables report mean and standard deviation over multiple seeds where repeated runs are performed. Single-seed settings, such as selected diagnostic ablations, are explicitly marked and justified by compute cost.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports hardware, latency, throughput, memory, and training-cost accounting in the experiments and appendix, including per-budget inference cost and D4RL/SlimPajama compute tables.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public datasets and benchmark protocols, does not collect human-subject data, and focuses on model architecture and deployment efficiency. We are not aware of any deviation from the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix A includes a broader impacts paragraph discussing the potential compute-efficiency benefit and the risk that more efficient sequence models may also make harmful downstream deployment easier.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk pretrained generative model, scraped dataset, or human-subject dataset. Released assets are code and experiment configurations for public benchmarks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Existing datasets and baselines are credited through citations and dataset descriptions. The code is released under the Apache License 2.0 and includes instructions for accessing public datasets under their original terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The ES-SSM code and documentation are publicly available at `https://github.com/GMU-MICO/ES-SSM` under the Apache License 2.0.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or newly collected human-subject data.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject research, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: No LLM is used as an important, original, or non-standard component of the method, experiments, or evaluation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.