DS-RNNs: Conditional Computation in Recurrent Models via Input-Dependent Sparse Gating

Anonymous authors
Paper under double-blind review

Abstract

Recurrent neural networks (RNNs) and state-space models (SSMs) typically execute the same dense computation for every input, coupling inference cost to parameter count and exacerbating interference in multi-task or heterogeneous regimes. We introduce Dynamic-Sparse RNNs (DS-RNNs), a framework for conditional computation via learnable, input-dependent sparse gating. In DS-RNNs, a small router network predicts sparse masks over input and hidden-state channels, effectively routing each input to a specialized sparse subnetwork. Unlike prior adaptive methods such as DeltaRNNs or static sparse training like RigL, our approach is fully learned and budget-controlled, and utilizes structured masking that translates into practical FLOP savings on commodity hardware by reducing sparse matrix multiplications to dense operations on active submatrices. Empirically, DS-RNNs maintain most of the dense model performance at 90% sparsity (sometimes even exceeding it) across diverse architectures, including LSTMs, GRUs, LTCs, and S4 models. They match or outperform RigL and DeltaRNNs on most tasks, while remaining stable in regimes where these baselines fail. We further show that DS-RNNs naturally induce subnetwork specialization without explicit supervision: masks for different classes (single-task) or tasks (multi-task) exhibit low overlap and are predictive of class identity. Our theoretical analysis provides intuition for this behavior, linking it to a formal bound on gradient interference. As a result, DS-RNNs improve robustness: DS-LSTM scales better with task count in multi-task regression and consistently reduces forgetting across benchmarks when combined with standard continual learning methods (e.g., GEM, replay).

1 Introduction

Recurrent neural networks (RNNs) and, more recently, state space-models (SSMs) have become core building blocks for sequential learning, powering applications that range from speech processing and time-series forecasting to language modeling and embodied control. As datasets and tasks grow in diversity, however, the conventional design choice of executing the same dense computation for every input becomes increasingly problematic: different inputs may require different features, different memory horizons, and different parts of the parameter space. The inefficiency is especially acute in recurrent models, where the same weight matrices are applied repeatedly across time, coupling inference cost tightly to parameter count and amplifying interference when a single parameter set must serve heterogeneous inputs or multiple tasks simultaneously.

A broad line of research addresses these issues through *conditional computation*, i.e. activating only a subset of parameters as a function of the input. At the module level, Mixture-of-Experts (MoE) (Shazeer et al., 2017) routes each token to a small subset of expert networks, enabling large parameter counts with bounded per-example computation, but at the cost of routing overhead, load-balancing instability, and auxiliary losses (Wang et al., 2024). At the weight level, *hypernetworks* (Ha et al., 2016) generate full weight matrices on-the-fly from a context vector, enabling highly adaptive dynamics but at an overhead that can match or exceed the forward pass itself. Within recurrent models specifically, adaptive computation has been approached at two granularities: at the *step level*, where Skim-RNN (Seo et al., 2017) learns a binary policy to decide which fraction of the hidden state must be updated at each timestep; and at the *channel level*, where computation is modulated by selectively gating which input or state dimensions contribute to the update. Our work belongs

to the latter family, whose most direct prior instantiation are *Delta Networks* (or DeltaRNNs) (Neil et al., 2017). These models zero out input channels whose activation has not changed sufficiently since the previous timestep, excluding them from the recurrent update. DeltaRNNs establish that selective channel gating is a viable route to recurrent conditional computation, but have three concrete limitations: the gating threshold is fixed and hand-tuned rather than learned, there is no mechanism to enforce a target computational budget, and the approach relies on temporal autocorrelation between consecutive inputs, an assumption that may not hold in all tasks. A separate family of methods, including dynamic sparse training techniques like RigL Evci et al. (2020), enforces static sparsity by training a single sparse subnetwork shared across all inputs at inference. These methods are orthogonal to conditional computation: they cannot express that different inputs benefit from different subnetworks, and their irregular connectivity patterns are poorly supported by current GPU architectures.

Here we introduce *Dynamic-Sparse RNNs (DS-RNNs)*, which close the gap between DeltaRNNs and a fully learnable, budget-controlled conditional computation framework for recurrent models. In DS-RNNs, a lightweight router network predicts sparse gating masks over input and hidden-state channels as a function of the current input, routing each input to a specialized sparse subnetwork. An overview of the method is shown in fig. 1. Unlike Delta Networks, the gating criterion is fully learned and budget-controlled, requiring no assumptions about temporal autocorrelation. Unlike RigL and related static methods, different inputs activate different parameter subsets at inference time. Critically, the masking is structured: entire input or hidden channels are zeroed, reducing sparse matrix multiplications to dense operations on active submatrices, which maps directly onto standard GEMM kernels and achieves FLOP reductions on commodity hardware without specialized sparse primitives.

This per-example subnetwork selection pursues two complementary objectives. First, *specialization*: different inputs activate different parameter subsets, better leveraging available capacity across heterogeneous data. Second, *interference mitigation*: when tasks or classes activate largely disjoint subnetworks, gradient updates are confined to approximately disjoint subspaces, reducing interference between different inputs. We provide a theoretical analysis formalizing this second property, showing that gradient interference is provably bounded by the overlap between router representations of the input distributions.

Empirically, DS-RNNs retain most dense-model classification performance at 90% sparsity across LSTM, GRU, and LTC architectures, and in some cases exceed it (e.g. on Gesture and Ozone datasets). At equal sparsity, DS matches or outperforms RigL on LSTM and GRU, and dramatically outperforms it on LTC, where static sparsity causes near-total optimization failure. Compared with DeltaRNNs, DS achieve clear gains on several tasks, particularly regression, while matching performance on the rest. A summary of the comparisons is shown in fig. 2-(a). We further extend input-dependent sparse gating to S4 and Liquid-S4, where DS remains competitive with dense baselines on Long Range Arena (Tay et al., 2020) tasks, and exceeds them on the ListOps and IMDb tasks.

Mechanistic analysis confirms that DS-RNNs spontaneously route different tasks and classes to largely disjoint parameter subsets (mean off-diagonal task IoU below 0.3, versus approximately 0.7 for dense LSTM, fig. 2-(d)) and that the resulting masks are predictive of class identity, without any explicit disentanglement objective. This phenomenon has downstream benefits in multi-task and continual learning (CL) settings. In multi-task regression, DS-LSTM degrades significantly less than dense LSTM as task count grows (fig. 2-(b)). In CL, DS-RNN is complementary to existing methods: standard CL methods such as replay and GEM protect the router from forgetting, while DS-RNN reduces the interference that makes forgetting likely in the first place by routing tasks to approximately disjoint parameter subspaces. Used in combination, DS consistently reduces forgetting and improves stream accuracy across all four benchmarks (fig. 2-(c)).

Our main contributions are:

- We propose DS-RNNs, a framework for learnable input-dependent sparse gating in recurrent models, and extend it to SSMs such as S4. Unlike Delta Networks, the gating criterion is fully learned and budget-controlled, and unlike static sparse training methods such as RigL, different inputs activate different parameter subsets at inference time. At 90% sparsity, DS-RNNs largely match dense classification performance and outperform RigL and DeltaRNNs on most tasks. For SSMs, DS-S4

at 80-90% sparsity is competitive with or surpasses the dense S4 baseline on Long Range Arena benchmarks.

- We provide a theoretical analysis showing that sparse masking confines gradient updates to input-specific subspaces, with interference bounded by the routing separation between task distributions.
- We structurally verify subnetwork specialization via Fuzzy IoU analysis and UMAP projections of learned masks, demonstrating that DS-RNNs route different tasks and classes to largely distinguishable parameter subsets; this provides a mechanism for interference reduction, which is complementary to existing CL methods and consistently improves forgetting metrics when the two are combined.

The remainder of the paper is organized as follows. In section 2, we review prior work on conditional computation, adaptive recurrent models, and sparse training methods, situating DS-RNNs within these lines of research. In section 3, we introduce the DS-RNN framework, detailing the input-dependent routing mechanism, the structured sparsity formulation, and its extension to state space models. Section 3.2 presents a theoretical analysis of the approach, showing how sparse masking induces input-specific subspaces and bounds gradient interference. In section 4, we describe the experimental setup and evaluation protocol across recurrent and state space architectures. Section 5 reports empirical results, including comparisons to dense and sparse baselines (section 5.1, 5.2), as well as analyses of multi-task (section 5.3) and continual learning settings (section 5.5). Finally, section 5.4 provides mechanistic insights into subnetwork specialization and routing behavior, and section 6 concludes with a discussion of limitations and future directions.

2 Related Work

Conditional computation at the module level Conditional computation activates only a fraction of the network per input, amortizing large parameter counts without a proportional increase in FLOPs. The Mixture-of-Experts (MoE) framework (Shazeer et al., 2017) is the canonical example: a learned gating function routes each token to a subset of expert networks, enabling trillion-parameter models with bounded per-example computation (Fedus et al., 2022). Mixture-of-Depths (Raposo et al., 2024) extends this principle to the depth axis, skipping entire transformer blocks for tokens deemed easy. These methods operate at module granularity and leave intra-layer weight redundancy unaddressed; they also introduce non-trivial routing overhead like dispatch, load balancing, and auxiliary losses (Wang et al., 2024), that DS-RNN avoids by operating within the module.

Adaptive computation in recurrent models Several methods reduce recurrent computation by adapting *how much* or *what* to compute at each step. Along the temporal axis, Adaptive Computation Time (Graves, 2016) learns to halt early on easy inputs. Skim-RNNs (Seo et al., 2017) learn a binary policy that decides whether to run the full recurrent update or a cheaper small RNN at each step. Most directly related to DS-RNN are Delta Networks (Neil et al., 2017), which also select which input channels contribute to the recurrent update: channels whose activation has not changed sufficiently since the previous step are zeroed out and do not contribute to the update. The key differences are that Delta Networks use a fixed threshold rather than a learned policy, do not enforce a computational budget, and rely on temporal autocorrelation that may not be present in all tasks. Hypernetworks (Ha et al., 2016) achieve full input-dependent weight adaptation at the cost of generating entire weight matrices on-the-fly, an overhead that can match or exceed the forward pass itself. DS-RNN targets the regime between these extremes: a learned, budget-controlled channel selection policy without weight generation overhead.

Static and dynamic weight-level sparsity A separate line of work maintains a fixed sparsity level throughout training by periodically rewiring connections. SET (Mocanu et al., 2018) removes low-magnitude weights and randomly regrows connections; RigL (Evci et al., 2020) improves on this by regrowing along high-gradient directions, discovering subnetworks that rival the dense baseline at 90-95% sparsity. SNIP (Lee et al., 2018) identifies critical connections at initialization via saliency metrics. All of these methods produce a *single* sparse topology shared across all inputs at inference. This is precisely what DS-RNN is not: the value

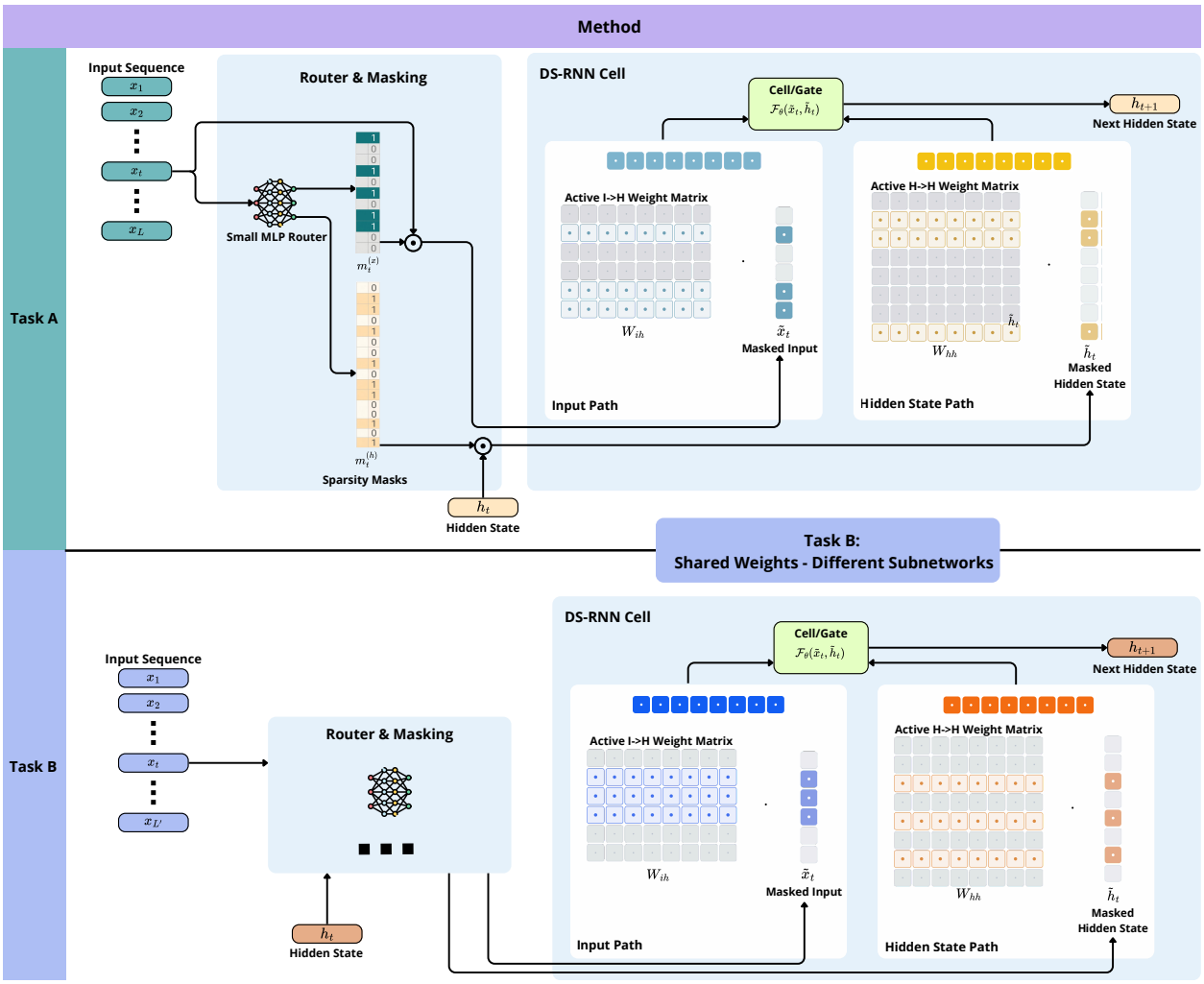

Figure 1: **DS-RNN Method Overview.** At each timestep t, a small MLP router takes the current input x_t and produces two sparse masks $m_t^{(x)} \in [0,1]^D$ and $m_t^{(h)} \in [0,1]^H$ over input and hidden-state channels respectively. The masks are zero everywhere except at the top-k positions, where values are continuous outputs of a scaled sigmoid; k is set by the target sparsity budget. The resulting masked vectors $\tilde{x}_t$ and $\tilde{h}_t$ are then fed to the DS-RNN cell, which computes h_{t+1} using only the active columns of W_{ih} and W_{hh}, reducing recurrent matrix multiplications to dense operations on smaller submatrices. Each task (Task A (top) and Task B (bottom) in this example) shares the same weight matrices, but the router selects different active channels for each, carving out task-specific subnetworks from the shared parameter space.

Figure 2: **DS-RNN Main Results Overview. (a)** Bubble chart comparing DS-RNN against a dense RNN, RigL, and DeltaRNN (ΔRNN) at 90% sparsity across five classification (left, Δpp↑) and three regression tasks (right, ΔMAE↓), averaged across the architectures. Bubble size encodes advantage magnitude. DS-RNN consistently outperforms both sparse baselines and remains competitive with the dense model. **(b)** Average MAE (↓) vs. number of jointly trained tasks for dense LSTM and DS-LSTM at two sparsity levels. Dense LSTM error grows substantially with task count; DS-LSTM degrades more gracefully, reflecting reduced inter-task interference. **(c)** Accuracy gain (↑) and forgetting reduction (↓) when combining DS-LSTM with standard continual learning methods across four benchmarks, relative to the dense baseline. DS-RNN is complementary to existing CL strategies, consistently improving both metrics. **(d)** Cross-task subnetwork overlap for dense LSTM and DS-LSTM in the 5-task setting. DS-LSTM produces substantially lower off-diagonal overlap, showing that the router routes different tasks to different parameter subsets without any explicit task-separation objective.

of per-example subnetwork specialization (activating different sparse subnetworks for different inputs) is not captured by any static-at-inference method, regardless of how well the topology is optimized. Furthermore, the unstructured connectivity patterns produced by RigL and related methods are poorly supported by current GPU architectures and rarely translate training-time sparsity into wall-clock gains. DS-RNN's structured channel masking avoids this problem entirely, as discussed in section 3. N:M structured sparsity (NVIDIA Corporation, 2021) and MaskLLM (Yang et al., 2024) address the hardware support issue via semi-structured patterns, but again commit to a fixed topology at inference.

Continual learning Continual learning methods address catastrophic forgetting through several distinct mechanisms. DS-RNN is not directly a continual learning method but is designed to be used in combination with existing ones: it reduces the interference that makes forgetting likely by routing different inputs to (approximately) disjoint parameter subspaces, complementing the mechanisms of existing CL approaches while reducing the FLOP count. Regularization-based approaches such as EWC (Kirkpatrick et al., 2017) and SI (Zenke et al., 2017) penalize changes to parameters important for previous tasks. Replay-based methods (Rolnick et al., 2019; Kim et al., 2020) and gradient episodic memory (GEM) (Lopez-Paz & Ranzato, 2017) constrain future updates via a buffer of past experiences. Parameter isolation methods such as PackNet (Mallya & Lazebnik, 2018) and HAT (Serra et al., 2018) assign disjoint parameter subsets to tasks via binary masks. They are the structurally closest family to DS-RNN, but relying on static per-task mask assignment and task identity at inference, neither of which DS-RNN requires.

Input-dependent state space models Since we extend DS-RNN to S4 (Gu et al., 2022) and Liquid-S4 (Hasani et al., 2022), we note recent SSM variants that also introduce input-dependence. S5 (Smith et al., 2022) introduces input-dependent transition matrices via parallel scans. Mamba (Gu & Dao, 2024) makes the SSM parameters functions of the input at every token, achieving strong performance on language tasks. These methods introduce input-dependence in the *dynamics* of the state space operator. DS-S4 is complementary: it introduces input-dependence in the *channel selection*, which hidden dimensions are active for a given sequence while leaving the SSM dynamics unchanged. The two axes of input-dependence are orthogonal and could in principle be combined.

3 Method

3.1 Dynamic-Sparse RNNs

To address the optimization and interference issues discussed above, we introduce DS-RNNs, which perform *input-dependent sparsification of the recurrent computation*. Unlike selective-update approaches (e.g. Skim-RNNs by Seo et al. (2017)) that decide which hidden units are updated, our mechanism selects which components of x_t and h_{t-1} are allowed to contribute to the update at time t. As a result, different inputs can reuse the same recurrent cell with different active input/state pathways, offering a mechanism for specialization while keeping the underlying recurrent parameters shared.

Let $\mathcal{F}_\theta$ denote the recurrent transition of a base cell (RNN/GRU/LSTM),

$$h_t = \mathcal{F}_\theta(h_{t-1}, x_t). \tag{1}$$

DS-RNN augments this transition with continous masks on the incoming state and input channels:

$$\tilde{h}_{t-1} = m_t^{(h)} \odot h_{t-1}, \qquad \tilde{x}_t = m_t^{(x)} \odot x_t, \tag{2}$$

$$h_t = \mathcal{F}_\theta(\tilde{h}_{t-1}, \tilde{x}_t), \tag{3}$$

where $m_t^{(h)} \in [0,1]^H$ and $m_t^{(x)} \in [0,1]^D$ are input-dependent masks.

For a vanilla RNN cell, Eq. eq. (3) becomes

$$h_t = \phi\Big(W_{hh}(m_t^{(h)} \odot h_{t-1}) + W_{ih}(m_t^{(x)} \odot x_t) + b\Big), \tag{4}$$

which is equivalent to using masked effective matrices $W_{hh}\,\mathrm{diag}(m_t^{(h)})$ and $W_{ih}\,\mathrm{diag}(m_t^{(x)})$ (equivalently, row/column masking depending on tensor convention). The same masking principle is applied to all affine terms in GRU/LSTM gates. For example, in a DS-GRU model:

$$\tilde{h}_{t-1} = m_t^{(h)} \odot h_{t-1}, \qquad \tilde{x}_t = m_t^{(x)} \odot x_t, \tag{5}$$

$$z_t = \sigma\big(W_z\tilde{x}_t + U_z\tilde{h}_{t-1} + b_z\big), \tag{6}$$

$$r_t = \sigma\big(W_r\tilde{x}_t + U_r\tilde{h}_{t-1} + b_r\big), \tag{7}$$

$$\hat{h}_t = \tanh\big(W_h\tilde{x}_t + U_h\left(r_t \odot \tilde{h}_{t-1}\right) + b_h\big), \tag{8}$$

$$h_t = (1 - z_t) \odot \tilde{h}_{t-1} + z_t \odot \hat{h}_t. \tag{9}$$

where the masks $m_t^{(x)}, m_t^{(h)}$ are thus shared across the gates.

Input/State-Component Selection Given a context vector $c_t = x_t$, a lightweight router r_ψ predicts mask logits:

$$a_t^{(h)} = r_\psi^{(h)}(c_t), \qquad a_t^{(x)} = r_\psi^{(x)}(c_t). \tag{10}$$

We then obtain sparse masks via thresholding or top-k projection, and bring them in the $[0, 1]$ range with a scaled sigmoid function:

$$m_t^{(h)} = \mathcal{S}(\sigma(\tau_h\,(a_t^{(h)} - \mu_h))), \qquad m_t^{(x)} = \mathcal{S}(\tau_x\,(a_t^{(x)} - \mu_x)), \tag{11}$$

where $\mathcal{S}(\cdot)$ enforces a target sparsity budget, and $\tau_h, \tau_x, \mu_h, \mu_x \in \mathbb{R}$ are learnable constants (respectively initialized at $\tau_h = 1, \tau_x = 1, \mu_h = 0, \mu_x = 0$). In practice, we use $\mathcal{S}(\cdot) = \mathrm{top}{-}k(\cdot)$ in the forward pass to realize conditional computation, and set k based on the computational budget (i.e. hidden size H, sparsity $s \implies k = \lceil s\,H \rceil$). End-to-end differentiability is preserved as we can backpropagate through the selected mask indexes. The router can be instantiated as a linear map or a small MLP.

Computational cost For a vanilla recurrent update with keep ratios $\rho_h, \rho_x \in [0, 1]$ for hidden-state and input components, respectively, the FLOPs saved by sparse recurrent multiplication relative to the dense baseline are:

$$\Delta\mathrm{FLOPs}_{\mathrm{saved}} \;\approx\; 2H\big((1 - \rho_h)H + (1 - \rho_x)D\big). \tag{12}$$

A necessary condition for net FLOP reduction is that the router overhead $\mathrm{FLOPs}_{\mathrm{router}} \approx 4R(H + D)$ satisfies $\mathrm{FLOPs}_{\mathrm{router}} \ll \Delta\mathrm{FLOPs}_{\mathrm{saved}}$, which motivates the low-dimensional projection $c_t = W_{\mathrm{proj}}x_t \in \mathbb{R}^R$ with $R \ll H$ as the router input. In most experiments we use $R = 4$, or $R = 16$ for larger models.

Critically, because DS-RNN sparsity is *structured* (the mask zeroes entire input channels of the weight matrix rather than arbitrary individual weights) the sparse matrix multiplication can be implemented efficiently without any sparse linear algebra primitives. At each time step, the active components of $\tilde{h}_{t-1}$ and $\tilde{x}_t$ are gathered by index, and only the corresponding columns of W_h and W_x participate in the computation. This reduces the recurrent matrix multiplication to a dense operation on a $H \times \lfloor \rho_h H \rfloor$ submatrix, which maps directly onto standard batched GEMM kernels and achieves practical speedups on commodity hardware. This stands in contrast to unstructured sparsity methods such as RigL, whose irregular connectivity patterns are poorly supported by current GPU architectures. It is also worth noting that, when the same mask is shared across multiple gates as in DS-GRU and DS-LSTM, $\Delta\mathrm{FLOPs}_{\mathrm{saved}}$ is multiplied by a factor of 3 and 4, respectively, further amplifying the efficiency advantage.

3.1.1 Dynamic Sparse Routing for DS-S4 and DS-Liquid-S4

In order to evaluate its scalability, we extend our method to S4 (Gu et al., 2022), one of the most prominent examples of State-Space Model (SSM) architectures. Our DS-S4 layer augments the standard S4 block

with an input-dependent lightweight router that selects which *hidden dimensions* are active for each input sequence.

Let $u \in \mathbb{R}^{B\times H\times L}$ be the S4 input (batch size B, model dimension H, sequence length L). To avoid token-wise routing overhead, we compute a sequence-level context vector by temporal mean pooling:

$$\bar{u} = \frac{1}{L}\sum_{t=1}^{L} u_{:,t,:} \in \mathbb{R}^{B\times H}. \tag{13}$$

where ":" denotes taking all the elements of the tensor along the corresponding dimension. The router maps $\bar{u}$ to a (soft) hidden-dimension mask $m \in [0,1]^{B\times H}$,

$$m = \sigma(f_\theta(\bar{u}))\,, \tag{14}$$

where $f_\theta : \mathbb{R}^H \to \mathbb{R}^H$ is a linear layer or a low-dimensional MLP and $\sigma(\cdot)$ is the logistic sigmoid. In practice, we sparsify m (e.g., via top-k) to match a target sparsity budget (e.g. 90%).

Given the (static) S4 convolution kernel $k \in \mathbb{R}^{C\times L_k\times H}$, we apply the mask by broadcasting across S4 heads and time:

$$\tilde{k}_{b,c,\ell,h} = m_{b,h}\, k_{c,\ell,h,}, \tag{15}$$

which yields an input-conditioned kernel $\tilde{k} \in \mathbb{R}^{B\times C\times L_k\times H}$, with reduced number of non-zero hidden channels depending on the sparsity of m. Note that the removed channels differ for each sequence in the batch. The S4 convolution is then computed as usual via FFT and an einsum-style contraction, later adding the "skip-connection" of the D term in the SSM equation (Gu et al., 2022). However, only the selected hidden dimensions contribute to the output, enabling conditional computation and a reduced number of FLOPs (with an appropriate implementation). It is worth noting that the oberhead caused by the router is mostly negligible, as it gets amortized sharing the sparsity mask being shared throughout the input sequence.

We apply the same routing mechanism to Liquid-S4, which extends S4 by incorporating higher-order input-dependent ("liquid") terms (Hasani et al., 2022). Since Liquid-S4 reuses the same channelized S4 kernels in both its linear and nonlinear components, masking hidden dimensions via m consistently sparsifies the full Liquid-S4 computation. Depending on the implementation, the mask can be applied either in the time domain (k) or directly in the frequency domain ($\mathcal{F}(k)$). In our implementation, masks are applied in the frequency domain.

In all our experiments, we wrap all S4-based models in a standard scalable architecture (following Hasani et al. (2022)), alternating SSM layers (either S4, Liquid-S4, DS-S4, or DS-Liquid-S4) with feed-forward layers, normalizations, and residual connections. Specifically, given a batch of input sequences $u \in \mathbb{R}^{(B\times L\times H)}$, the output of each block is computed as follows:

$$\begin{aligned} y_0 &= \mathrm{SSM}(\mathrm{Norm}(u)) \\ y_1 &= \mathrm{FFN}(y_0) + u \end{aligned}$$

where SSM is the SSM layer, Norm is either 1D batch normalization or layer normalization, and FFN is a token-wise feed-forward network with 2 layers and GLU activation (Dauphin et al., 2017).

3.2 Theoretical Analysis

3.2.1 Router Selection as Priority Signals

The DS-RNN router introduces an input-dependent modulatory signal that determines which hidden units participate in the recurrent update at each time step. Beyond its role as a computational gating mechanism, we show that the router gradient has a precise interpretation as a marginal utility signal, and that training drives the router toward selecting units by their causal influence on the loss.

Router gradient as marginal utility Let $a_t \in \mathbb{R}^H$ denote the router logits at time t, produced by a router network $r_\psi(x_t)$, and let $m_t = \text{top}{-}k(\sigma(a_t)) \in [0,1]^H$ be the corresponding sparsity mask. The recurrent update operates on the gated state $\tilde{h}_{t-1} = m_t \odot h_{t-1}$. Applying the chain rule, we define the *marginal utility* of activating unit i as:

$$\mathcal{U}_{t,i} := \frac{\partial \mathcal{L}}{\partial m_{t,i}} \; = \; h_{t-1,i} \cdot \Gamma_{t,i}, \tag{16}$$

where $\Gamma_t := \partial \mathcal{L} / \partial \tilde{h}_{t-1}$ is the full backpropagated gradient through all future timesteps. This quantity can be interpreted as a first-order "usefulness" of the unit i, i.e. the extent to which the unit i has a causal influence on the loss. The gradient of the loss with respect to the router logits is then:

$$\frac{\partial \mathcal{L}}{\partial a_{t,i}} \; = \; \mathcal{U}_{t,i} \cdot \lambda_{t,i} \tag{17}$$

where $\lambda_{t,i}$ is the derivative of the scaled and shifted sigmoid used in the router:

$$\lambda_{t,i} := \tau \sigma(\tau(a_{t,i} - \mu))(1 - \sigma(\tau(a_{t,i} - \mu)))$$

Since $\lambda_{t,i} > 0$, the sign of the gradient with respect to $a_{t,i}$ is determined entirely by $\mathcal{U}_{t,i}$: the logit for unit i is pushed upward when activating it reduces the loss, and downward otherwise. However, the magnitude of the update for unit i is $|\mathcal{U}_{t,i}| \cdot \lambda_{t,i}$, where $\lambda_{t,i}$ is maximized when $a_{t,i} = \mu$ (the decision boundary) and decays toward zero as $|a_{t,i} - \mu|$ grows. This means that the largest gradient updates are concentrated on units near the current selection threshold: units the router has already confidently selected or rejected receive weaker corrective signal even if their true utility is large. The learnable parameter τ controls the sharpness of this boundary: large τ concentrates updates in a narrow margin around μ, while small τ distributes them more broadly. The learnable μ shifts the threshold itself. Taken together, eq. (17) describes a gradient signal that is correct in sign but margin-weighted in magnitude, concentrating learning on the most ambiguous routing decisions at the current operating point of (τ, μ). Note that eq. (16) and 17 hold for any DS-RNN architecture (e.g. DS-GRU and DS-LSTM with the architecture-specific forms of Γ_t given in Appendix A, section A.1).

Empirical validation To verify that the router converges to a causally meaningful ranking, we conduct a perturbation analysis on the router's choices at inference time, following the general logic of counterfactual explanations and perturbation-based evaluation of feature importance (Wachter et al., 2017; Petsiuk et al., 2018; Hooker et al., 2019). If the logits z_t merely act as a gating mechanism, perturbing the continuous priority weights within the selected top-k set is expected to have minimal impact on the task objective. Conversely, if the router has effectively approximated the underlying marginal utility landscape, overriding its ranking should degrade performance in direct proportion to the severity of the perturbation. We evaluate the final task loss on the SeqMNIST dataset, under four conditions at evaluation time: (1) *Baseline*: the unmodified sparse forward pass; (2) *Intra-TopK*: the active top-k units are preserved, but the priority weights ($\sigma(z_{t,i})$) of the highest-ranked and lowest-ranked active units are swapped; (3) *Random Swap*: one top-k unit is replaced by a random unselected unit; (4) *Sabotage*: the highest-ranked top-k unit is forcefully replaced by the absolute lowest-ranked unit in the network. The empirical results show a strict hierarchy of causal dependence: Baseline (0.0937), Intra-TopK (0.1390), Random Swap (0.2441), and Sabotage (4.2271). The 48% loss increase observed in the Intra-TopK condition demonstrates that the network heavily relies on the relative magnitude of the router's continuous output, suggesting that the logits provide a functional priority signal rather than just a selection mask. Furthermore, the 45-fold loss increase in the Sabotage condition confirms that the router actively isolates critical-path variables with high precision, heavily penalizing the removal of high-utility units. These perturbations indicate that gradient descent on eq. (17) aligns the router's selection policy with the utility of the hidden state.

Connection to disinhibitory cortical circuits A related computational motif appears in cortical microcircuits involving vasoactive intestinal peptide (VIP) interneurons. VIP neurons inhibit somatostatin (SOM) interneurons, which normally suppress pyramidal cells, resulting in disinhibition and increased gain

for selected neuronal populations (Pi et al., 2013; Pfeffer et al., 2013). Millman et al. (2020) show that VIP interneurons selectively enhance responses to weak but task-relevant stimuli, effectively implementing an input-dependent gain allocation over pyramidal cell populations. The DS-RNN router plays an analogous role at the computational level: eq. (16) shows that the router signal $\mathcal{U}_{t,i}$ is a first-order estimate of the causal relevance of unit i to the current objective, and eq. (17) shows that training drives the router to allocate computational resources toward high-utility units. The parallel is functional, not mechanistic: we make no claim about the biological implementation.

3.2.2 Gradient Isolation via Sparse Masking

We provide a theoretical motivation for how DS-RNN masks mitigate gradient interference. We show that sparse gating forces weight updates into task-specific subspaces, with interference bounded by the router's ability to distinguish input distributions. Full proofs are provided in Appendix A, section A.2.

Low-rank gradient structure Consider the DS-RNN update $h_t = \phi(W_h(m_t \odot h_{t-1}) + \cdots)$ where the mask $m_t \in [0,1]^H$ satisfies $\|m_t\|_0 \leq k$. The recurrent Jacobian $J_t = \partial h_t / \partial h_{t-1} = D_t W_h \mathrm{diag}(m_t)$ has $\mathrm{rank}(J_t) \leq k$ because the masking operation nullifies all but k columns of the weight matrix W_h before the hidden state transformation. Consequently, the instantaneous gradient $\partial \mathcal{L}/\partial W_h|_t = \delta_t (m_t \odot h_{t-1})^\top$ is a rank-one matrix whose non-zero entries are confined to the k columns indexed by $\mathrm{supp}(m_t)$. This implies that two (idealized) updates with disjoint mask supports are exactly orthogonal, i.e., $\langle G_t^A, G_t^B \rangle = 0$, preventing any direct parameter-level interference.

Interference bound For two tasks with inputs drawn from independent distributions p_A and p_B, the expected gradient interference is bounded by the mean mask overlap (see proposition 2):

$$\mathbb{E}[|\langle G_t^A, G_t^B \rangle|] \leq C_\delta^2 C_h^2 \, \langle \bar{m}_{p_A}, \bar{m}_{p_B} \rangle \tag{18}$$

where C_δ and C_h are almost-sure bounds on the error signal and hidden state norms, respectively, and $\bar{m}_{p,i} = \mathbb{E}_{x \sim p}[m_{t,i}]$ represents the average activation of the i-th routing unit for distribution p. Notably, this bound is independent of temporal correlations within sequences, as the historical dependencies are captured by the uniform bounds C_δ and C_h.

Routing separation To link this bound to the router's latent space, we model inputs as Gaussian distributions. Under mild geometric alignment conditions (see proposition 3), the mean overlap $\langle \bar{m}_{p_A}, \bar{m}_{p_B} \rangle$ is monotonically decreasing in the routing-space separation $\|W_{\mathrm{proj}}(\mu_A - \mu_B)\|_2$. As the distance between task distributions in the router's latent space grows, the expected gradient interference vanishes. This provides a mechanistic account of the DS-RNN's input-dependent subnetworks: whenever the router can distinguish two tasks, it automatically confines their respective learning signals to largely non-overlapping parameter subspaces, thereby mitigating interference and forgetting.

This theoretical result is confirmed by our experiments on input-dependent subnetwork creation (section 5.4), showing how inputs from different classes in a single task, or from different distributions in a multi-task setting, lead to the creation of scarcely overlapping subnetworks.

4 Experiments

We evaluate DS-RNNs along four axes:

Q1. **Performance and generalization.** (a) To what extent does DS-RNN preserve dense model performance at high levels of sparsity? (b) Does it generalize across recurrent architectures with qualitatively different dynamics (LSTM, GRU, LTC), and to more scalable SSMs such as S4?

Q2. **Comparison against other sparse methods.** How does DS-RNN compare to static-at-inference sparse training (RigL) and to non-learned conditional computation (DeltaRNNs), at matched sparsity?

Q3. **Mechanistic analysis.** Do DS-RNNs learn input-informative sparsity masks, and does subnetwork selection vary meaningfully across inputs and tasks?

Q4. **Downstream benefits of subnetwork selection.** Does input-conditioned subnetwork selection reduce interference in multi-task settings and mitigate catastrophic forgetting in continual learning?

4.1 Dynamic Sparsity across Recurrent Architectures

4.1.1 DS-RNNs vs. Dense RNNs

To answer Q1.-(a), i.e. how much dense-model performance DS-RNNs can retain at a fixed sparsity level, we compare standard recurrent architectures to their dynamic-sparse counterparts. Specifically, we consider two widely used discrete-time RNNs, LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014), as well as the continuous-time Liquid Time-Constant (LTC) model Hasani et al. (2021), which has been shown to be highly expressive. Following the experimental protocol of Hasani et al. (2021), we evaluate GRU/LSTM/LTC and DS-GRU/DS-LSTM/DS-LTC on a suite of small-scale sequence benchmarks: SequentialMNIST (Liu et al., 2003); UCI time-series datasets (HAR (Anguita et al., 2013), Ozone, Power, and Traffic (Dua & Graff, 2017)); the Occupancy dataset (Candanedo & Feldheim, 2016) (room-occupancy detection from physical sensor streams); and a HalfCheetah kinematic modeling task built from 25 rollouts of a pre-trained HalfCheetah-v2 controller (Brockman et al., 2016; Todorov et al., 2012). Results are reported in section 5.1.1.

Experimental protocol Unless otherwise stated, we run each experiment with 4 random seeds and keep hyperparameters shared across architectures to ensure a fair comparison, tuning only the learning rate per architecture type (meaning that e.g. LSTM and DS-LSTM share the same learning rate, but not DS-GRU and GRU). All networks use hidden size 64 and are trained for up to 250 epochs with AdamW (Loshchilov & Hutter, 2017) (no weight decay; $\beta_1 = 0.9$, $\beta_2 = 0.999$), batch size 256, and learning rate selected in $[0.005, 0.01]$. We apply ReduceLROnPlateau scheduling strategy with factor 0.5 and patience 10 epochs. For DS variants, we fix the sparsity level to 90%.

4.1.2 Dynamic Sparse SSMs on LRA

To test whether our input-dependent sparsity mechanism transfers beyond classical RNNs to modern long-context sequence models (Q1.-(b)), we evaluate Dynamic Sparse SSMs on the Long Range Arena (LRA) (Tay et al., 2020) suite. LRA comprises tasks specifically designed to stress long-range dependency modeling under tight compute budgets, making it a natural setting to assess whether conditional computation can preserve (or improve) performance while activating only a subset of parameters. In particular, we focus on four representative LRA benchmarks: *AAN* (byte-level retrieval, i.e. "Retrieval"), *sCIFAR* (sequential CIFAR-10, i.e., "Image"), *IMDb* (byte-level sentiment classification, i.e., "Text"), and *ListOps*, an algorithmic task where the correct hierarchical composition of operators requires memory over long dinstances. We consider both dense and dynamic-sparse variants of S4-based architectures. Specifically, we evaluate S4 and Liquid-S4 backbones, together with their dynamic sparse counterparts (DS-S4 and DS-Liquid-S4), which incorporate our input-dependent sparse gating mechanism. Results are described in section 5.1.2.

Experimental protocol In all models, we use a unified configuration across datasets with model width $d_{\text{model}} = 128$ and SSM hidden size $d_{\text{state}} = 64$, following a similar scale to Hasani et al. (2022) but keeping the same hyperparameters across all LRA tasks for a controlled comparison. For Liquid-S4 experiments, we use the liquid kernel parameterization with a polynomial approximation degree $p = 2$ by default. We closely follow the training recipe of Hasani et al. (2022) for the learning rate schedule and number of epochs, with only minor adjustments to ensure stable training across our unified configuration. We therefore use the AdamW optimizer, with learning rate in the $[0.004, 0.01]$ range (tuned based on the dataset) and 0.01 weight decay for all the parameters except the SSM kernel ones, for which we use 0.001 and 0.0, respectively. As for the learning rate scheduling strategy, we employ the cosine annealing scheduler (Loshchilov & Hutter, 2016). We keep the hyperparameters consistent between dense and dynamic-sparse variants so that any observed

differences can be attributed to the routing/gating mechanism rather than changes in the optimizer or other hyperparameters.

4.2 DS-RNN vs. Sparse Methods

To understand what is the contribution of the learned dynamic sparsity in the DS-RNN performance (Q2.), we contextualize DS-RNNs against two sparse training baselines: RigL and Delta Networks.

4.2.1 DS-RNN vs. RigL

RigL (Evci et al., 2020) maintains fixed sparsity throughout training by periodically pruning and regrowing connections based on gradient information. Unlike DS-RNNs, RigL learns a single sparse topology shared across all inputs, making it a direct test of whether *input-conditioned* sparsity provides benefits beyond simply learning a good static sparse subnetwork, i.e., whether per-example subnetwork specialization is worth the overhead. Results are detailed in section 5.2.

Experimental protocol We follow the same experimental protocol used in the DS-RNN vs. dense RNN comparison to ensure a fair and controlled evaluation: we keep the same datasets, optimizers, batch sizes, learning-rate ranges, and training schedules, and we apply the same per-architecture learning-rate tuning strategy. We train RigL models at 90% sparsity, matching the active-parameter budget of DS-RNNs. At each RigL topology update, we prune and regrow a fraction $\Delta = 0.05$ of the weights (i.e., a 5% prune-and-regrowth fraction), following standard RigL practice. Unless otherwise stated, all remaining hyperparameters are kept identical to the corresponding dense/DS baselines.

4.2.2 DS-RNN vs. DeltaRNN

DeltaRNNs (Neil et al., 2017) represent a conceptually distinct route to efficient recurrent computation: rather than sparsifying weights, they sparsify *updates*, skipping hidden state computations at timesteps where the change in input falls below a fixed threshold. Like DS-RNNs, DeltaRNNs perform input-conditioned computation, i.e. the active subnetwork can vary per-example. The key distinction is whether the criterion for skipping computation is *learned* or *fixed*: DeltaRNNs commit to a single scalar threshold on state deltas, while DS-RNNs learn which parts of the network to activate jointly with the weights. A fixed delta threshold implicitly assumes that input smoothness is a reliable proxy for update redundancy; a learned router imposes no such constraint and can in principle recover smoothness-based skipping when it is useful, or learn a different criterion when it is not. The comparison thus probes whatever not relying on this assumption can lead to advantages. Results are reported in section 5.2.

Experimental protocol We evaluate DeltaRNN variants of LSTM and GRU at matched ≈90% sparsity (87-93% depending on task and threshold), using the same datasets, optimizers, and training schedules as the DS-RNN experiments. Sparsity in DeltaRNNs is controlled indirectly through the input and hidden state thresholds ϑ_x and ϑ_h; we select thresholds that achieve the target sparsity range while maintaining reasonable performance, following the procedure of Neil et al. (2017).

4.3 Downstream Benefits of Subnetwork Selection

To answer Q4., we study two downstream settings: multi-task learning and continual learning.

4.3.1 Multi-Task Learning Experiments

We evaluate DS-RNNs in two multi-task regimes, where a single model is trained jointly on heterogeneous tasks. In the *natural dataset* setting, we jointly train on the classification datasets from the single-task experiments (SequentialMNIST, HAR, Gesture, Ozone, Occupancy) and examine whether DS-LSTM learns specialized subnetworks per task (section 4.4) while matching dense model performance. In the *synthetic* setting, we construct synthetic temporal mapping tasks that allow us to evaluate on an arbitrary number of independent tasks. Each task corresponds to a randomly generated input-output mapping defined by

an independent teacher dynamical system with fixed, randomly initialised parameters, where sequences $x_{1:T} \sim \mathcal{N}(0, I)$ are mapped to targets $y_{1:T}$. This follows the teacher-student paradigm used to study interference in controlled settings (Lee et al., 2021; Hiratani, 2024). Outputs are normalized per task to remove scale confounds. We evaluate models under increasing numbers of tasks ($N_{\text{tasks}} \in \{2, 4, 8, 12\}$) to probe when a fixed-parameter recurrent model begins to suffer from interference, and whether input-conditioned sparsity delays this degradation. Results are outlined in section 5.3.

Experimental protocol We use LSTM and DS-LSTM with hidden size 128, sharing the input projection and prediction head across tasks. A 16-dimensional task embedding is provided as additional input to disambiguate task regimes. In the synthetic setting, we compare DS-LSTM at 50% and 80% sparsity against dense LSTM; in the natural dataset setting we use 90% sparsity. All remaining hyperparameters follow the single-task protocol. Models are trained with uncertainty-based task weighting (Kendall et al., 2018) to dynamically balance losses across objectives.

4.3.2 Continual Learning Experiments

Building on the multi-task analysis, we next investigate whether the subnetwork selection induced by DS-RNNs mitigates catastrophic forgetting relative to a standard dense RNN, when both are combined with standard continual learning methods. We consider standard class-incremental "Split" benchmarks, in which a multiclass problem with C classes is decomposed into $C/2$ binary tasks. Specifically, we evaluate on SplitMNIST, SplitHAR, and SplitDigits (Alpaydin & Alimoglu, 1996), comprising 5, 3, and 5 tasks, respectively. We additionally include RotatedMNIST (5 tasks), an augmented-input benchmark in which each task corresponds to a distinct rotation of the original MNIST images. Results are described in section 5.5.

Experimental protocol For robustness, we perform experiments using three continual-learning strategies: replay buffer (Rolnick et al., 2019), reservoir buffer (Kim et al., 2020), and GEM (Lopez-Paz & Ranzato, 2017), and average their results. We evaluate DS-LSTM with hidden size 128 over sparsity levels $s \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. For each task, training is performed for 5 epochs with a batch size of 128. We keep all other hyperparameters consistent with the rest of the experimental setup.

4.4 Mechanistic Analysis of Subnetwork Specialization

To quantify the isolation of task-specific and class-specif logic (Q3.), we analyze the Fuzzy IoU (Intersection over Union) between the subnetwork pathways activated by each task or class. We compare the cross-task and cross-class IoU of the DS-LSTM against the LSTM baseline to determine if the dynamic gating mechanism induces more isolated parameter utilization. Additionally, to quantify how much class information is encoded in the routing dynamics and sparsity masks independently of any hidden state features, we apply the following procedure to the hidden-to-hidden and input-to-hidden masks of a single-task DS-LSTM trained on SeqMNIST. For each input sequence, the per-timestep masks are summarised by fitting $k = 3$ local k-means centroids, producing a compact representation of the temporal routing pattern (i.e. the activated pathways) for that sequence, on which we train a logistic classifier to predict the class label. We also use UMAP projections of the sparsity masks to visualize whether these naturally cluster by class. Results are reported in section 5.4.

5 Results

5.1 Dynamic Sparsity across Recurrent Architectures

5.1.1 DS-RNN Recovers Most of the Dense RNN Performance

Table 1 reports results at 90% sparsity for both standard and masked-input settings. On classification, DS retains 98.6-99.3% of dense performance on HAR, SeqMNIST, and Occupancy across all three architectures, keeping the performance below or equal to 1pp. More interestingly, Gesture and Ozone F1 *improve* relative to dense for LSTM and GRU (by up to 11.3pp and 5.5pp, respectively), and Gesture improves for LTC as well (+14.5pp). Gesture by +4.3pp, +3.3pp, and +8.4pp for LSTM, GRU, and LTC, respectively, and Ozone

Model	HAR (Acc.↑)		SeqMNIST (Acc.↑)		Gesture (F1↑)		Ozone (F1↑)		Occupancy (F1↑)	
	Dense	DS	Dense	DS	Dense	DS	Dense	DS	Dense	DS
LSTM	**94.8**$_{\pm 0.4}$	93.5$_{\pm 0.5}$	**98.7**$_{\pm 0.1}$	97.7$_{\pm 0.3}$	53.1$_{\pm 6.0}$	**57.4**$_{\pm 5.0}$	85.9$_{\pm 3.70}$	**87.8**$_{\pm 2.0}$	**87.6**$_{\pm 2.8}$	86.6$_{\pm 3.1}$
GRU	**95.8**$_{\pm 0.5}$	94.3$_{\pm 1.3}$	**98.6**$_{\pm 0.2}$	97.9$_{\pm 0.2}$	54.2$_{\pm 8.5}$	**57.5**$_{\pm 5.8}$	90.3$_{\pm 1.40}$	**94.3**$_{\pm 1.9}$	**88.4**$_{\pm 1.6}$	87.7$_{\pm 2.0}$
LTC	**94.1**$_{\pm 0.4}$	93.2$_{\pm 0.4}$	**97.4**$_{\pm 0.4}$	96.4$_{\pm 0.8}$	38.2$_{\pm 7.7}$	**46.6**$_{\pm 4.8}$	53.3$_{\pm 22.3}$	**78.3**$_{\pm 8.9}$	**80.3**$_{\pm 2.5}$	79.7$_{\pm 7.7}$

Model	Traffic (MAE↓)		Power (MAE↓)		Cheetah (MAE↓)	
	Dense	DS	Dense	DS	Dense	DS
LSTM	**0.152**$_{\pm 0.010}$	0.185$_{\pm 0.007}$	**0.018**$_{\pm 0.003}$	0.033$_{\pm 0.009}$	**0.681**$_{\pm 0.011}$	0.889$_{\pm 0.083}$
GRU	**0.146**$_{\pm 0.014}$	0.194$_{\pm 0.017}$	**0.017**$_{\pm 0.002}$	0.035$_{\pm 0.008}$	**0.702**$_{\pm 0.012}$	0.839$_{\pm 0.026}$
LTC	**0.130**$_{\pm 0.011}$	0.160$_{\pm 0.008}$	**0.015**$_{\pm 2e-4}$	0.020$_{\pm 0.006}$	**0.797**$_{\pm 0.004}$	0.946$_{\pm 0.070}$

Table 1: Dense vs. Dynamic Sparse (DS) models at 90% sparsity.

Model	HAR (Acc.↑)		SeqMNIST (Acc.↑)		Gesture (F1↑)		Ozone (F1↑)		Occupancy (F1↑)	
	RigL	DS	RigL	DS	RigL	DS	RigL	DS	RigL	DS
LSTM	93.4$_{\pm 1.10}$	**93.5**$_{\pm 0.5}$	96.9$_{\pm 0.6}$	**97.7**$_{\pm 0.3}$	50.4$_{\pm 2.3}$	**57.4**$_{\pm 5.0}$	21.1$_{\pm 14.7}$	**87.8**$_{\pm 2.0}$	77.3$_{\pm 3.0}$	**86.6**$_{\pm 3.1}$
GRU	94.0$_{\pm 1.40}$	**94.3**$_{\pm 1.3}$	96.6$_{\pm 1.1}$	**97.9**$_{\pm 0.2}$	49.5$_{\pm 1.5}$	**57.5**$_{\pm 5.8}$	90.7$_{\pm 1.70}$	**94.3**$_{\pm 1.9}$	71.6$_{\pm 4.0}$	**87.7**$_{\pm 2.0}$
LTC	23.0$_{\pm 15.5}$	**93.2**$_{\pm 0.4}$	10.7$_{\pm 1.6}$	**96.4**$_{\pm 0.8}$	20.0$_{\pm 0.0}$	**46.6**$_{\pm 4.8}$	0.00$_{\pm 0.0}$	**78.3**$_{\pm 8.9}$	0.00$_{\pm 0.0}$	**79.7**$_{\pm 7.7}$

Model	Traffic (MAE↓)		Power (MAE↓)		Cheetah (MAE↓)	
	RigL	DS	RigL	DS	RigL	DS
LSTM	0.200$_{\pm 0.010}$	**0.185**$_{\pm 0.007}$	**0.024**$_{\pm 0.004}$	0.033$_{\pm 0.009}$	**0.740**$_{\pm 0.003}$	0.889$_{\pm 0.083}$
GRU	**0.192**$_{\pm 0.017}$	0.194$_{\pm 0.017}$	**0.019**$_{\pm 0.003}$	0.035$_{\pm 0.008}$	**0.782**$_{\pm 0.003}$	0.839$_{\pm 0.026}$
LTC	0.828$_{\pm 0.159}$	**0.160**$_{\pm 0.008}$	0.723$_{\pm 0.083}$	**0.020**$_{\pm 0.006}$	2.301$_{\pm 0.000}$	**0.946**$_{\pm 0.070}$

Table 2: RigL vs. Dynamic Sparse (DS) at 90% sparsity.

by +1.9pp, +4.0pp, and +25.0pp. The Ozone improvement for LTC is particularly striking given a dense baseline of only 53.3% F1, and suggests that sparse training is especially beneficial when the dense model training can be unstable (coherent with the huge F1 std ±22.3). One might attribute this to a regularisation effect of sparsity, given that these are also the two tasks with the lowest dense baselines (38-54% F1). However, neither RigL nor DeltaRNNs replicate these improvements at matched sparsity (Tables table 2 and table 3), which points to a specific effect of the DS training dynamics rather than sparsity per se.

Regression tasks show more consistent degradation than classification, but the effect is still contained. Power MAE increases $0.018 \rightarrow 0.033$ for LSTM and $0.017 \rightarrow 0.035$ for GRU, representing the largest relative increase, followed Cheetah MAE increases by 0.208 and 0.137 for LSTM and GRU respectively, with milder increases on Traffic by 0.033 and 0.048. LTC instead shows smaller relative degradation across all three tasks ($\approx 1.2\times$), despite already having weaker dense baselines. The consistent direction of this effect suggests that regression

Model	HAR (Acc.↑)		SeqMNIST (Acc.↑)		Gesture (F1↑)		Ozone (F1↑)		Occupancy (F1↑)	
	Δ-RNN	DS	Δ-RNN	DS	Δ-RNN	DS	Δ-RNN	DS	Δ-RNN	DS
LSTM	**94.8**$_{\pm 0.5}$	93.5$_{\pm 0.5}$	**98.0**$_{\pm 0.1}$	97.7$_{\pm 0.3}$	41.4$_{\pm 1.5}$	**57.5**$_{\pm 5.8}$	81.0$_{\pm 2.6}$	**87.8**$_{\pm 2.0}$	**88.4**$_{\pm 5.0}$	86.6$_{\pm 3.1}$
GRU	93.5$_{\pm 1.9}$	**94.3**$_{\pm 1.3}$	**97.9**$_{\pm 0.1}$	**97.9**$_{\pm 0.2}$	40.3$_{\pm 1.4}$	**57.4**$_{\pm 5.0}$	61.5$_{\pm 7.4}$	**94.3**$_{\pm 1.9}$	87.2$_{\pm 1.4}$	**87.7**$_{\pm 2.0}$

Model	Traffic (MAE↓)		Power (MAE↓)		Cheetah (MAE↓)	
	Δ-RNN	DS	Δ-RNN	DS	Δ-RNN	DS
LSTM	0.537$_{\pm 0.005}$	**0.185**$_{\pm 0.007}$	0.388$_{\pm 0.575}$	**0.033**$_{\pm 0.009}$	1.211$_{\pm 0.015}$	**0.889**$_{\pm 0.083}$
GRU	0.562$_{\pm 0.004}$	**0.194**$_{\pm 0.017}$	0.061$_{\pm 0.001}$	**0.035**$_{\pm 0.008}$	1.188$_{\pm 0.010}$	**0.839**$_{\pm 0.026}$

Table 3: DeltaRNN (Δ-RNN) vs. Dynamic Sparse (DS) at ≈ 90% sparsity.

Task	Model	Dense	DS (80%)	DS (90%)
IMDb (Text)	S4	$83.2_{\pm 0.8}$	$84.4_{\pm 0.6}$	$\mathbf{85.0}_{\pm 0.4}$
	Liquid-S4	$83.3_{\pm 0.7}$	$84.2_{\pm 0.7}$	$\mathbf{85.3}_{\pm 0.3}$
AAN (Retrieval)	S4	$\mathbf{87.2}_{\pm 0.9}$	$85.5_{\pm 1.2}$	$84.5_{\pm 0.5}$
	Liquid-S4	$\mathbf{87.1}_{\pm 0.8}$	$85.0_{\pm 1.7}$	$83.7_{\pm 0.8}$
ListOps	S4	$38.0_{\pm 2.2}$	$\mathbf{43.3}_{\pm 0.9}$	$41.1_{\pm 1.6}$
	Liquid-S4	$38.9_{\pm 1.7}$	$42.6_{\pm 0.6}$	$\mathbf{42.6}_{\pm 0.3}$
sCIFAR (Image)	S4	$\mathbf{81.3}_{\pm 0.5}$	$79.4_{\pm 1.2}$	$77.6_{\pm 1.7}$
	Liquid-S4	$\mathbf{81.3}_{\pm 0.4}$	$78.5_{\pm 0.5}$	$76.6_{\pm 1.6}$

Table 4: Dense vs. Dynamic Sparse S4 and Liquid-S4 on LRA benchmarks (accuracy % ↑). DS uses a 3-layer MLP router for ListOps and sCIFAR, and a linear router for IMDb and AAN. Bold indicates best result per row.

objectives are more sensitive to aggressive recurrent sparsification than classification ones, although the degradation is far from catastrophic compared to DeltaRNN, where regression performance collapses more severely (see section 5.2).

5.1.2 Dynamic Sparsity can be Applied to SSMs in Long-Range Tasks

We evaluate whether input-dependent sparse gating transfers to more scalable sequence models by applying DS to S4 and Liquid-S4 on four LRA benchmarks. Table 4 reports results at 80% and 90% sparsity.

Across all four tasks, DS models remain functional at both sparsity levels, and on two tasks they improve over the dense baseline. On IMDb, DS-S4 reaches 85.0% at 90% sparsity against a dense baseline of 83.2% (+1.8pp), and DS-Liquid-S4 reaches 85.3% against 83.3% (+2.0pp). On ListOps, the improvement is larger: DS-S4 reaches 43.3% at 80% sparsity against a dense baseline of 38.0% (+5.3pp), and DS-Liquid-S4 42.6% against 38.9% (+3.7pp). At 90% sparsity, DS-S4 drops to 41.1% while DS-Liquid-S4 holds at 42.6%, suggesting the liquid kernel is more robust to aggressive sparsification on this task.

On AAN and sCIFAR, DS degrades relative to dense, though the models remain functional throughout. At 90% sparsity, DS-S4 drops 2.7pp on AAN (84.5% vs. 87.2%) and 3.7pp on sCIFAR (77.6% vs. 81.3%); at 80% sparsity the gap narrows to 1.7pp and 1.9pp respectively. The pattern across tasks mirrors what was observed for DS-RNNs: DS improves over dense on tasks where the dense baseline is weaker (IMDb, ListOps), and degrades on tasks with stronger dense baselines (AAN, sCIFAR). This suggests the DS mechanism generalises across recurrent and state-space architectures with the similar qualitative behaviour.

5.2 DS-RNN Outperforms or Matches RigL and DeltaRNN

Table 2 and table 3 compare DS against RigL and DeltaRNN at matched 90% sparsity, averaged across seeds.

DS-RNN vs. RigL For LSTM and GRU, differences on HAR, SeqMNIST, and Occupancy are small and consistently favour DS (within 1-2pp), indicating that both methods handle these tasks comparably at this sparsity level. The gap widens substantially on Gesture and Ozone: RigL-LSTM achieves 50.4% and 21.1% respectively, against DS-LSTM's 57.4% and 87.8%. On Gesture, the GRU pattern mirrors LSTM (RigL: 49.5%; DS: 57.5%), but Ozone is an exception: RigL-GRU reaches 90.7%, marginally above the dense baseline of 90.3%, suggesting the Ozone failure is partly architecture-dependent and more severe for LSTM. For LTC, RigL fails across all tasks: 23.0% on HAR, 10.7% on SeqMNIST, 20.0% on Gesture, and 0.0% on both Ozone and Occupancy, while DS-LTC retains near-dense performance throughout.

On continous tasks, DS and RigL are comparable for LSTM and GRU, with small trade-offs depending on the dataset. RigL holds a modest advantage over DS for LSTM and GRU on Power(0.024 vs. 0.033 for LSTM; 0.019 vs. 0.035 for GRU) and Cheetah (0.740 vs. 0.889 for LSTM; 0.782 vs. 0.839 for GRU), while

Model	SeqMNIST (Acc.↑)	HAR (Acc.↑)	Gesture (F1↑)	Ozone (F1↑)	Occupancy (F1↑)
LSTM	$96.7_{\pm 0.5}$	$\mathbf{96.8}_{\pm 0.2}$	$57.1_{\pm 2.1}$	$99.3_{\pm 0.7}$	$71.9_{\pm 2.1}$
DS-LSTM (50%)	$\mathbf{97.2}_{\pm 0.5}$	$96.6_{\pm 0.7}$	$60.4_{\pm 3.0}$	$99.5_{\pm 0.5}$	$\mathbf{72.9}_{\pm 1.4}$
DS-LSTM (80%)	$\mathbf{97.2}_{\pm 0.2}$	$96.3_{\pm 1.0}$	$62.8_{\pm 4.4}$	$\mathbf{99.7}_{\pm 0.3}$	$71.2_{\pm 0.9}$
DS-LSTM (90%)	$95.5_{\pm 1.1}$	$96.7_{\pm 0.9}$	$\mathbf{64.4}_{\pm 2.3}$	$99.6_{\pm 0.3}$	$70.9_{\pm 1.6}$

Table 5: Multi-task classification results (5 tasks, jointly trained) for natural datasets. Bold indicates best result per column.

DS is marginally better on Traffic. For LTC, the regression picture mirrors the classification one: RigL-LTC reaches Power MAE of 0.723 and Cheetah MAE of 2.301, compared to DS-LTC's 0.020 and 0.946.

A way to explain the RigL+LTC failures relies on the key assumption of magnitude-based pruning methods: treating weight magnitude as a proxy for functional importance. For LTC, this assumption is structurally problematic: weight magnitudes directly govern the network's time constants, so pruning them at 90% disrupts the dynamics regardless of their scalar value, an issue that does not arise when the sparse topology is co-adapted with the weights throughout training.

DS-RNN vs. DeltaRNN On HAR, Delta-LSTM marginally exceeds DS-LSTM (94.8% vs. 93.5%), while DS-GRU marginally exceeds Delta-GRU (94.3% vs. 93.5%); SeqMNIST is effectively tied ($\leq$0.3pp difference). On Occupancy, results are mixed: Delta-LSTM has a modest advantage over DS-LSTM by 1.8pp (though with higher variance $\pm$5.0pp vs. $\pm$3.1pp), while DS-GRU and Delta-GRU are within 0.5pp. Conversely, on Gesture and Ozone, DS is consistently ahead by significant margins: Gesture gaps of 16.0pp and 17.2pp, Ozone gaps of 6.8pp and 32.8pp for LSTM and GRU respectively. On regression tasks, DS holds a decisive advantage too: Delta-LSTM reaches Traffic MAE of 0.537 and Power MAE of 0.388, against DS-LSTM's 0.185 and 0.033; Cheetah MAE is 1.211 vs. 0.889 for LSTM and 1.188 vs. 0.839 for GRU.

A pattern emerging from these results is the increased robustness of DS-RNN across tasks: the two methods are competitive on HAR, SeqMNIST, and Occupancy, while DS-RNN has significant margins on Gesture, Ozone, and all regression tasks. One possible explanation is that Delta-RNN's fixed channel-gating threshold is a reliable proxy for update necessity only when input smoothness and redundancy are present. These conditions hold in datasets like SeqMNIST, but may not hold uniformly across the remaining tasks. A learned channel-selection criterion, by contrast, can adapt to the structure of each task independently.

5.3 Multi-Task Learning

Table 5 reports results for the 5-task natural dataset setting. At 50% and 80% sparsity, DS-LSTM matches or exceeds the dense baseline on all tasks: SeqMNIST improves by +0.5pp at both sparsity levels, and Gesture improves by +3.3pp and +5.7pp respectively. At 90% sparsity DS-LSTM stays competitive: the Gesture advantage grows to +7.3pp while SeqMNIST and Occupancy pay a small cost (-1/1.2pp), consistent with the single-task pattern. Ozone is at ceiling for all models and does not differentiate them.

Figure 3 (and table 7 in the Appendix, section B) reports results on the synthetic multi-task regression setting, where models are trained jointly on up to 12 independent dynamical system tasks. At $N_{\text{tasks}} = 2$, LSTM and DS-LSTM at 50% sparsity are comparable (0.099 vs. 0.113 MAE), while DS-LSTM at 80% sparsity is worse (0.198 MAE), reflecting the capacity cost of more aggressive sparsification when interference pressure is low. As the number of tasks increases, this picture reverses: dense LSTM degrades substantially (0.316 MAE at $N_{\text{tasks}} = 12$), while DS-LSTM at 50% sparsity degrades significantly less (0.200 MAE at $N_{\text{tasks}} = 12$, a 37% relative reduction). DS-LSTM at 80% sparsity shows a similar trend but with higher absolute MAE, consistent with the greater capacity cost of more aggressive sparsification. The synthetic results provide direct evidence that input-conditioned subnetwork selection mitigates interference as task count increases, in a controlled setting where task difficulty and similarity are held fixed.

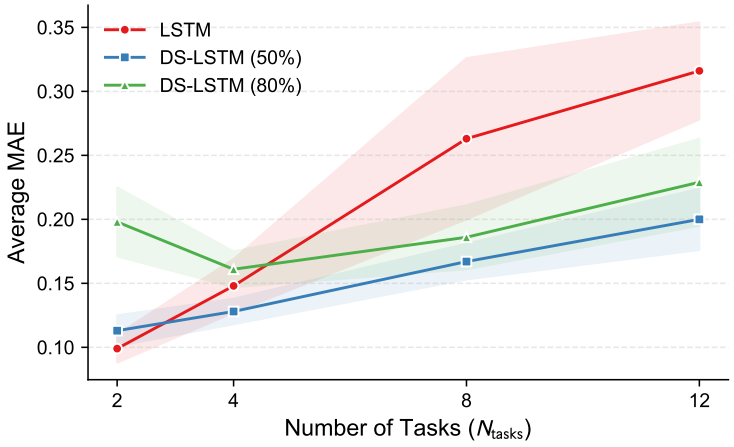

Figure 3: Average MAE on synthetic multi-task regression as a function of the number of jointly trained tasks ($N_{\text{tasks}} \in \{2, 4, 8, 12\}$), for dense LSTM and DS-LSTM at 50% and 80% sparsity. Shaded regions indicate one standard deviation across seeds. Each task corresponds to an independent randomly generated nonlinear dynamical system.

5.4 Subnetwork Specialization

Dynamic Sparsity Masks are Informative of the Input We examine whether DS-LSTM learns input-informative sparsity masks by analysing the Fuzzy IoU between the average masks generated for different inputs. Fuzzy IoU between two masks m_A and m_B is defined as $\text{FIoU}(m_A, m_B) = \frac{\sum_i \min(m_{A,i}, m_{B,i})}{\sum_i \max(m_{A,i}, m_{B,i})}$, measuring the overlap between active parameter subsets. A low off-diagonal Fuzzy IoU indicates that different inputs activate largely disjoint subnetworks.

Cross-task specialization Figure 4-(b) shows the cross-task Fuzzy IoU matrices for LSTM and DS-LSTM in the 5-task setting. For the dense LSTM, all off-diagonal task pairs show high overlap (0.7-0.9), confirming that the dense model uses essentially the same hidden neurons regardless of task identity. For DS-LSTM, off-diagonal task pairs drop to a mean of 0.27, with most pairs in the 0.0-0.3 range and moderate overlap (0.4-0.5) within the gesture-ozone-occupancy cluster. DS-LSTM spontaneously routes different tasks through largely disjoint parameter subsets without any explicit task-isolation objective.

Within-task class specialization Even within a single task, DS-LSTM routes different classes to approximately distinct parameter subsets. fig. 4-(a) shows class-level Fuzzy IoU matrices for HAR (6 classes, mean off-diagonal IoU = 0.343) and SeqMNIST (10 classes, mean off-diagonal IoU = 0.421). Stronger separation is observed for binary classification tasks (Occupancy: 0.069; Ozone: 0.314), where the router concentrates nearly all active parameters in class-specific subsets. The degree of separation decreases as the number of classes grows, consistent with the geometric constraintthat at 90% sparsity each active subnetwork occupies only 10% of the parameter space. It also reflects that some classes share features and therefore could rely on partially overlapping units.

Masks are linearly decodable A logistic classifier trained on the sparsity mask summaries (as described in section 4.4) achieves $\approx 89.4\%$ accuracy on the held-out SeqMNIST test set, and $\approx 76\%$ when the same procedure is applied to the input-to-hidden masks. The result indicates that this compressed representation of the selected neurons within a sequence is predictive of the class label. Per-class F1 scores (fig. 4-(d)) are consistent across most classes, with digit 1 being the most cleanly separated (F1 ≈ 0.96). The 2D UMAP projection of the hidden-to-hidden mask space (fig. 4-(c)) is also consistent with this picture: digit 1, digit 2, and digit 6 form mostly isolated clusters, while the remaining classes occupy partially overlapping regions of the mask manifold, reflecting the graded class separation visible in the Fuzzy IoU matrices.

Figure 4: Subnetwork specialization analysis of DS-LSTM on SeqMNIST and the 5-task multi-task setting. **(a)** Class-level Fuzzy IoU between mean sparsity masks in single-task settings (SeqMNIST, 10 classes; HAR, 6 classes; Occupancy and Ozone, 2 classes). Mean off-diagonal IoU reported above each matrix. **(b)** Cross-task Fuzzy IoU for dense LSTM and DS-LSTM in the 5-task setting. Mean off-diagonal IoU reported above each matrix. **(c)** 2D UMAP projection of hidden-to-hidden sparsity masks, colored by class label (SeqMNIST). **(d)** Logistic classifier trained on per-sequence mask representations (3 local k-means centroids along temporal axis) evaluated on the held-out SeqMNIST test set. Left: accuracy and macro F1 for hidden-to-hidden and input-to-hidden masks. Right: per-class F1 for the hidden-to-hidden classifier.

Together, these results confirm that DS-LSTM learns input-informative routing: different tasks and classes activate largely disjoint parameter subsets, and the masks are quantitatively predictive of class labels. This structural evidence is consistent with the gradient isolation argument of section 3.2.2.

5.5 Dynamic Sparsity Helps Continual Learning Methods

Table 6 reports streaming accuracy and forgetting for DS-LSTM against a dense LSTM baseline, averaged across three CL strategies (replay, reservoir buffer, and GEM), with and without task embeddings. Full results across all sparsity levels $s \in \{0.5, 0.6, 0.7, 0.8\}$ are reported in Appendix table 8.

DS-LSTM reduces forgetting across all four benchmarks while simultaneously improving streaming accuracy in most settings. On SplitDigits ($s = 0.5$), forgetting drops from 0.126 to 0.044 and accuracy improves by 10.5pp. On SplitHAR ($s = 0.6$), forgetting is reduced from 0.175 to 0.064, though the dense LSTM

	SplitMNIST		SplitHAR		SplitDigits		RotMNIST	
Model	Acc.↑	Forg↓	Acc.↑	Forg↓	Acc.↑	Forg↓	Acc.↑	Forg↓
LSTM	$90.6_{\pm 0.4}$	$0.101_{\pm .006}$	$72.9_{\pm 2.8}$	$0.175_{\pm .094}$	$79.8_{\pm 7.0}$	$0.126_{\pm .083}$	$83.1_{\pm 2.0}$	$0.178_{\pm .024}$
DS-LSTM	$\mathbf{92.4}_{\pm 0.4}$	$\mathbf{0.085}_{\pm .004}$	$\mathbf{84.2}_{\pm 2.4}$	$\mathbf{0.064}_{\pm .019}$	$\mathbf{90.3}_{\pm 5.7}$	$\mathbf{0.044}_{\pm .061}$	$\mathbf{85.5}_{\pm 1.5}$	$\mathbf{0.153}_{\pm .020}$

Table 6: Continual learning results averaged across replay, reservoir, and GEM strategies. DS-LSTM sparsity levels: SplitMNIST $s = 0.8$, SplitHAR $s = 0.6$, SplitDigits $s = 0.5$, RotMNIST $s = 0.5$. Full results across $s \in \{0.5, 0.6, 0.7, 0.8\}$ are in Appendix 8.

baseline shows high variance on this benchmark ($\sigma_{\text{forg}} = 0.094$). On SplitMNIST ($s = 0.8$), forgetting is reduced from 0.101 to 0.085 with a 1.8pp accuracy gain. On RotMNIST, improvements are consistent but more modest: forgetting reduces by approximately 0.025 and accuracy improves by 2.4pp. As shown in Appendix table 8, the optimal sparsity level varies across benchmarks and is not monotonically related to performance, although $s = 0.6, 0.7$ is a conservative default that performs competitively across all four benchmarks without task-specific selection.

The benefit of DS-LSTM in the CL setting is consistent with the gradient isolation argument of section 3.2.2 and the subnetwork specialization observed in section 5.4: by routing different tasks to largely disjoint parameter subsets, DS-LSTM reduces the interference that makes forgetting likely, without requiring any modification to the CL strategy itself.

6 Conclusions

We introduced DS-RNN, a simple mechanism for input-dependent sparse computation in recurrent models, where a small router network selects a subset of channels to activate at each timestep. This yields structured sparsity that can be implemented with standard dense operations on smaller submatrices, avoiding the need for specialized sparse kernels.

The central empirical result is that input-dependent routing enables high sparsity without the high loss in performance. At 90% sparsity, DS-RNNs largely match (and in some cases exceed) the performance of dense baselines, while outperforming both static sparse training (RigL) and threshold-based dynamic sparsity (DeltaRNN) on most tasks. These results are consistent with the view that learned, input-dependent routing provides benefits beyond sparsification alone. The same routing mechanism extends to state space models, where DS applied to S4-style architectures remains competitive with dense baselines on Long Range Arena, exceeding them on some tasks (e.g., IMDb, ListOps), though performance degrades on others at high sparsity.

Beyond performance, DS-RNNs exhibit emergent structural specialization. Routing masks for different inputs show substantially reduced overlap (e.g., dropping from 0.7-0.9 in dense LSTM to 0.27 in DS-LSTM), indicating that the model separates inputs into distinct subnetworks without any explicit disentanglement objective. These masks naturally cluster and are also predictive of class labels (e.g., 89.4% accuracy from a logistic classifier trained on mask summaries on SeqMNIST), showing that the induced subnetworks capture semantically meaningful structure. This structural specialization has downstream consequences: in multi-task regression, DS-LSTM's MAE degrades significantly less than dense LSTM as the number of tasks grows from 2 to 12, and in continual learning DS-LSTM reduces forgetting across all four benchmarks when combined with standard methods like GEM or replay buffer.

Our theoretical analysis provides a partial explanation for these effects linking task separation to reduced gradient interference by showing that interference is controlled by the overlap of routing masks: as the overlap between active subnetworks decreases, interference is correspondingly reduced, vanishing in the limit of disjoint routing. This provides a concrete account of how input-dependent routing can mitigate interference by separating tasks into distinct computational pathways. We also interpret the router through a marginal utility view of its gradient, in which the logit of each unit reflects its causal contribution to the loss, with the largest updates concentrated near the top-k selection boundary. Empirically, this is verified by altering the

ranking of selected units, which leads to progressively larger degradations in performance, indicating that the router's decisions track causally important contributions.

Limitations Our experiments are restricted to relatively small models and short sequences, leaving open whether the same benefits hold at the scale of modern sequence models. In addition, DS-RNN does not by itself address continual learning: protecting the router requires external mechanisms (e.g. replay buffer), and the optimal sparsity level varies across settings. Finally, the gradient interference theoretical analysis is limited to first-order effects and relies on simplifying assumptions (e.g. Gaussian input) that may not fully capture the behavior of the full model.

Future directions Extending DS-RNNs to larger-scale models and longer sequences is the most direct next step, both to validate the FLOP savings argument at deployment-relevant scales and to test whether subnetwork specialization continues to provide interference mitigation as model capacity and task diversity grow. Combining DS-RNN's channel-level input-dependence with SSM architectures that introduce input-dependence in the state dynamics (e.g. Mamba) is also a natural extension, as the two axes of input-dependence are orthogonal and could in principle compound.

References

E. Alpaydin and Fevzi. Alimoglu. Pen-Based Recognition of Handwritten Digits. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5MG6K.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, pp. 3–4, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and buildings*, 112: 28–39, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Dheeru Dua and Casey Graff. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, 2017.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952, 2020.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

Albert Gu, Karan Goel, and Christopher R
'e. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7657–7666, 2021.

Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

Naoki Hiratani. Disentangling and mitigating the impact of task similarity for continual learning. *Advances in Neural Information Processing Systems*, 37:3243–3274, 2024.

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

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *Advances in neural information processing systems*, 32, 2019.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Chris Dongjoo Kim, Jinseo Jeong, and Gunhee Kim. Imbalanced continual learning with partitioning reservoir sampling. In *European conference on computer vision*, pp. 411–428. Springer, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Sebastian Lee, Sebastian Goldt, and Andrew Saxe. Continual learning in the teacher-student setup: Impact of task similarity. In *International Conference on Machine Learning*, pp. 6109–6119. PMLR, 2021.

Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, and Hiromichi Fujisawa. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern recognition*, 36(10):2271–2285, 2003.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Daniel J Millman, Gabriel Koch Ocker, Shiella Caldejon, India Kato, Josh D Larkin, Eric Kenji Lee, Jennifer Luviano, Chelsea Nayan, Thuyanh V Nguyen, Kat North, et al. Vip interneurons in mouse primary visual cortex selectively enhance responses to weak but specific stimuli. *Elife*, 9:e55130, 2020.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018.

Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. In *International conference on machine learning*, pp. 2584–2593. PMLR, 2017.

NVIDIA Corporation. Accelerating sparse matrix operations with 2:4 structured sparsity on ampere tensor cores, 2021. URL `https://arxiv.org/pdf/2104.08378`.

Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

Carsten K Pfeffer, Mingshan Xue, Miao He, Z Josh Huang, and Massimo Scanziani. Inhibition of inhibition in visual cortex: the logic of connections between molecularly distinct interneurons. *Nature neuroscience*, 16(8):1068–1076, 2013.

Hyun-Jae Pi, Balázs Hangya, Duda Kvitsiani, Joshua I Sanders, Z Josh Huang, and Adam Kepecs. Cortical interneurons that specialize in disinhibitory control. *Nature*, 503(7477):521–524, 2013.

David Raposo, Sam Ritter, Blake Richards, and et al. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-rnn. *arXiv preprint arXiv:1711.02085*, 2017.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pp. 4548–4557. PMLR, 2018.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V Le, Geoffrey E Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of ICLR*, 2017.

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

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.

Emo Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. IEEE, 2012.

Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

Mingyang Yang, Dan Alistarh, and Elias Frantar. Maskllm: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems*, 2024.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. Pmlr, 2017.

A Appendix

A.1 Architecture-Specific Forms of Masked State Gradients

We give the exact expressions for $\Gamma_t = \partial\mathcal{L}/\partial\tilde{h}_{t-1}$ for each architecture. Let $e_t := \partial\mathcal{L}/\partial h_t$ denote the full upstream gradient at time t. For DS-GRU, we define:

$$\delta_t^z = e_t \odot (\hat{h}_t - \tilde{h}_{t-1}) \odot z_t \odot (1 - z_t)$$
$$\tilde{\delta}_t^{\hat{h}} = z_t \odot e_t \odot (1 - \hat{h}_t^2)$$
$$\delta_t^r = \tilde{h}_{t-1} \odot (U_h^\top \tilde{\delta}_t^{\hat{h}}) \odot r_t \odot (1 - r_t)$$

the gradient decomposes as:

$$\Gamma_t^{\mathrm{GRU}} \;=\; \underbrace{(1-z_t)\odot e_t}_{\text{direct carry}} \;+\; \underbrace{U_z^\top \delta_t^z}_{\text{update gate}} \;+\; \underbrace{r_t \odot (U_h^\top \tilde{\delta}_t^{\hat{h}})}_{\text{candidate}} \;+\; \underbrace{U_r^\top \delta_t^r}_{\text{reset gate}} \;. \tag{19}$$

For DS-LSTM, we define:

$$dc_t := \frac{\partial \mathcal{L}}{\partial c_t} = e_t \odot o_t \odot (1 - \tanh^2(c_t)) + dc_{t+1} \odot f_{t+1}$$

and the gate-weighted error signals:

$$\begin{aligned}
\delta_t^f &= dc_t \odot c_{t-1} \odot f_t(1-f_t) \\
\delta_t^i &= dc_t \odot \hat{c}_t \odot i_t(1-i_t) \\
\delta_t^o &= e_t \odot \tanh(c_t) \odot o_t(1-o_t) \\
\delta_t^c &= dc_t \odot i_t \odot (1-\hat{c}_t^2),
\end{aligned}$$

the gradient is:

$$\Gamma_t^{\mathrm{LSTM}} \;=\; U_f^\top \delta_t^f + U_i^\top \delta_t^i + U_o^\top \delta_t^o + U_c^\top \delta_t^c. \tag{20}$$

A.2 Gradient Isolation Analysis

Let $h_t \in \mathbb{R}^H$ denote the hidden state, $x_t \in \mathbb{R}^D$ the input, and $W_h \in \mathbb{R}^{H\times H}$ the recurrent weight matrix. For simplicity, we consider the standard DS-RNN update, but the analysis can be easily adapted to other models such as DS-LSTM and DS-GRU:

$$h_t = \phi\Big(W_h(m_t \odot h_{t-1}) + W_x(m_t^{(x)} \odot x_t) + b\Big) \tag{21}$$

where $m_t \in [0,1]^H$ is a sparse mask produced by a lightweight router, $M_t = \mathrm{diag}(m_t)$, and $D_t = \mathrm{diag}(\phi'(g_t))$ is the diagonal matrix of activation derivatives at time t. We focus on the recurrent pathway; the input pathway is analogous. The mask has at most k nonzero entries, i.e. $\|\mathrm{m}_t\|_0 \le k$.

Proposition 1 (Low-Rank Jacobian and Gradient Structure)**.** *Under the DS-RNN update eq.* (21)*:*

(i) The recurrent Jacobian $J_t = \partial h_t/\partial h_{t-1} = D_t W_h M_t$ satisfies $\mathrm{rank}(J_t) \le k$.

(ii) The instantaneous gradient of a scalar loss $\mathcal{L}$ with respect to W_h is a rank-one matrix:

$$\left.\frac{\partial \mathcal{L}}{\partial W_h}\right|_t = \delta_t (m_t \odot h_{t-1})^\top, \qquad \delta_t := \frac{\partial \mathcal{L}}{\partial g_t} \in \mathbb{R}^H.$$

(iii) For two updates with masks m_t^A, m_t^B and corresponding gradients G_t^A, G_t^B:

$$\langle G_t^A, G_t^B \rangle = (\delta_t^{A\top} \delta_t^B) \cdot (m_t^A \odot h_{t-1}^A)^\top (m_t^B \odot h_{t-1}^B).$$

In particular, if $\mathrm{supp}(m_t^A) \cap \mathrm{supp}(m_t^B) = \varnothing$ *then* $\langle G_t^A, G_t^B \rangle = 0$.

Proof. (i) $J_t = D_t(W_h M_t)$. Since M_t zeros all but at most k columns of W_h, the matrix $W_h M_t$ has at most k nonzero columns, hence $\mathrm{rank}(W_h M_t) \le k$. Left-multiplication by a diagonal matrix cannot increase rank.

(ii) Since $g_t = W_h(m_t \odot h_{t-1}) + \cdots$, standard matrix calculus gives $\partial \mathcal{L}/\partial W_h|_t = \delta_t (m_t \odot h_{t-1})^\top$, which is an outer product of two vectors, hence rank ≤ 1.

(iii) Using the Frobenius inner product $\langle A, B\rangle = \mathrm{tr}(A^\top B)$ and the outer product form from (ii):

$$\langle G_t^A, G_t^B\rangle = \mathrm{tr}\big((m_t^A \odot h_{t-1}^A)\delta_t^{A\top} \cdot \delta_t^B (m_t^B \odot h_{t-1}^B)^\top\big) = (\delta_t^{A\top}\delta_t^B) \cdot (m_t^A \odot h_{t-1}^A)^\top (m_t^B \odot h_{t-1}^B).$$

If the supports are disjoint then $(m_t^A \odot h_{t-1}^A)^\top (m_t^B \odot h_{t-1}^B) = \sum_i m_{t,i}^A m_{t,i}^B h_{t-1,i}^A h_{t-1,i}^B = 0$ since no index i has $m_{t,i}^A = m_{t,i}^B = 1$.

□

We now derive an explicit upper bound on gradient interference that holds on every sample path and links directly to the routing behavior of the router.

Proposition 2 (Gradient Interference Bound)**.** *Assumption (uniform boundedness): there exist constants $C_\delta, C_h < \infty$ such that $\|\delta_t\|_2 \le C_\delta$ and $\|h_{t-1}\|_\infty \le C_h$ almost surely. The bound $C_h \le 1$ holds deterministically for $\phi = \tanh$.*

Let $x_t^A \sim p_A$ and $x_t^B \sim p_B$ be inputs drawn from two stationary distributions, with streams A and B independent of each other. Under the uniform boundedness assumption:

(i) On every sample path and at every time step:

$$|\langle G_t^A, G_t^B\rangle| \le C_\delta^2\, C_h^2\, \langle m_t^A, m_t^B\rangle.$$

(ii) Taking expectations over both streams:

$$\mathbb{E}[|\langle G_t^A, G_t^B\rangle|] \le C_\delta^2\, C_h^2\, \langle \bar{m}_{p_A}, \bar{m}_{p_B}\rangle$$

where $\bar{m}_{p,i} := \mathbb{E}_{x\sim p}[m_{t,i}]$ is the mean mask activation of distribution p at unit i.

Proof. (i) From proposition 1, apply the Cauchy-Schwarz inequality to the inner product over the active units:

$$\left|\sum_i m_{t,i}^A m_{t,i}^B h_{t-1,i}^A h_{t-1,i}^B\right| \le \|h_{t-1}^A\|_\infty \|h_{t-1}^B\|_\infty \sum_i m_{t,i}^A m_{t,i}^B$$

Applying $|\delta_t^{A\top}\delta_t^B| \le \|\delta_t^A\|_2 \|\delta_t^B\|_2 \le C_\delta^2$ and $\|h_{t-1}\|_\infty \le C_h$ gives

$$|\langle G_t^A, G_t^B\rangle| \le C_\delta^2 C_h^2 \sum_i m_{t,i}^A m_{t,i}^B = C_\delta^2 C_h^2 \langle m_t^A, m_t^B\rangle.$$

(ii) Taking expectations of both sides and using cross-stream independence (since $m_{t,i}^A$ depends only on $x_t^A \sim p_A$ and $m_{t,i}^B$ depends only on $x_t^B \sim p_B$):

$$\mathbb{E}\left[\sum_i m_{t,i}^A m_{t,i}^B\right] = \sum_i \mathbb{E}_{p_A}[m_{t,i}^A]\, \mathbb{E}_{p_B}[m_{t,i}^B] = \langle \bar{m}_{p_A}, \bar{m}_{p_B}\rangle.$$

□

In these propositions, no assumption about temporal correlations within each stream is required. The quantities δ_t and h_{t-1}, which are functions of the full history $x_{1:t}$, are eliminated before taking expectations via the deterministic almost-sure bounds C_δ and C_h. Only cross-stream independence is used in step (ii), which holds naturally since tasks are trained on separate data.

The bound in proposition 2 depends on $\langle \bar{m}_{p_A}, \bar{m}_{p_B}\rangle$. We now show this quantity decays as the two input distributions become more separated in the router's latent space.

Proposition 3 (Routing Separation Implies Small Expected Overlap)**.** *Let $m_{t,i} = \sigma(w_i^\top x_t)$ be the i-th routing mask component with $w_i = W_{\text{proj}}^\top W_{\text{rout},i}^\top$. For $x \sim \mathcal{N}(\mu, \Sigma)$, using the probit approximation, let $\bar{m}_{p,i} \approx \Phi(\nu_{p,i})$ where $\nu_{p,i} = w_i^\top \mu_p / \tilde{\sigma}_i$ and $\tilde{\sigma}_i^2 = 1 + \frac{\pi}{8} w_i^\top \Sigma w_i$. Additionally, assume the following:*

1. *Geometric Alignment: there exists $\gamma > 0$ such that $|w_i^\top \Delta\mu| \geq \gamma \|w_i\|_2 \|\Delta\mu\|_2$, ensuring the router is not perfectly orthogonal to the class separation (this is not too strict as random or learned weights are unlikely to lie exactly in the null space of the class difference. In other words, we are assuming that the router is able to detect the class separation as that separation grows).*
2. *Bounded Midpoint: the class midpoint $\bar{\nu}_i = (\nu_{A,i} + \nu_{B,i})/2$ remains bounded as $\|\Delta\mu\| \to \infty$ (this typically holds due to standard weight decay and data normalization preventing the latent mean from exploding.)*

Then the expected mask overlap satisfies:

$$\langle \bar{m}_{p_A}, \bar{m}_{p_B} \rangle \leq \sum_{i=1}^{H} \Phi\left(\bar{\nu}_i - \frac{|w_i^\top \Delta\mu|}{2\tilde{\sigma}_i}\right) \leq \sum_{i=1}^{H} \Phi\left(\bar{\nu}_i - \frac{\gamma ||w_i||_2 \|\Delta\mu\|_2}{2\tilde{\sigma}_i}\right) \tag{22}$$

where $\Delta\mu = \mu_A - \mu_B$. This bound is monotonically decreasing in $\|\Delta\mu\|_2$ and vanishes as $\|\Delta\mu\|_2 \to \infty$.

Proof. Since $0 \leq \Phi(\cdot) \leq 1$, the product of two probabilities is bounded by their minimum: $\Phi(\nu_i^A)\Phi(\nu_i^B) \leq \min(\Phi(\nu_i^A), \Phi(\nu_i^B))$. By the monotonicity of Φ, this is equivalent to $\Phi(\min(\nu_i^A, \nu_i^B))$. Using the algebraic identity $\min(a, b) = \frac{a+b}{2} - \frac{|a-b|}{2}$ and substituting ν_i^A, ν_i^B, we obtain the first inequality:

$$\min(\nu_i^A, \nu_i^B) = \frac{\nu_i^A + \nu_i^B}{2} - \frac{|\nu_i^A - \nu_i^B|}{2} = \bar{\nu}_i - \frac{|w_i^\top \Delta\mu|}{2\tilde{\sigma}_i}.$$

To establish the second inequality in terms of the magnitude $\|\Delta\mu\|_2$, we apply the Geometric Alignment assumption: $|w_i^\top \Delta\mu| \geq \gamma \|w_i\|_2 \|\Delta\mu\|_2$. Multiplying by -1 reverses the inequality, yielding $-\frac{|w_i^\top \Delta\mu|}{2\tilde{\sigma}_i} \leq -\frac{\gamma \|w_i\|_2 \|\Delta\mu\|_2}{2\tilde{\sigma}_i}$. Applying Φ to both sides preserves this direction. As $\|\Delta\mu\|_2 \to \infty$, the argument of each Φ term approaches $-\infty$ (since $\bar{\nu}_i$ is bounded), causing the sum of H finite terms to vanish. □

Limitations The theoretical analysis has the following limitations:

- *First-order only*: proposition 2 and 3 bound gradient inner products (first-order conflicts). Higher-order effects and optimizer state (e.g. Adam's second moment) can still cause interference.
- *Gradient boundedness*: $C_\delta \leq \infty$ requires a bounded loss gradient, which holds for Lipschitz losses with bounded activations but should be checked per architecture.
- *Recurrent pathway only*: The analysis covers W_h; biases, input weights, normalization parameters, and, importantly, router parameters, also receive gradients and may still interfere.
- *Probit approximation*: The routing separation corollary assumes Gaussian inputs and uses a probit approximation for the sigmoid; for heavy-tailed or multimodal distributions the bound is approximate.
- *Rank-R bottleneck*: Distributional differences invisible to W_{proj} (i.e. in the null space of W_{proj}) produce no separation regardless of how different p_A and p_B are, which motivates monitoring routing-space separability in practice.

B Additional Results

Here we provide additional results for the multitasks and continual learning experiments.

Model	$N_{\text{tasks}} = 2$	$N_{\text{tasks}} = 4$	$N_{\text{tasks}} = 8$	$N_{\text{tasks}} = 12$
LSTM	$\mathbf{0.099}_{\pm 0.011}$	$0.148_{\pm 0.021}$	$0.263_{\pm 0.063}$	$0.316_{\pm 0.038}$
DS-LSTM (50%)	$0.113_{\pm 0.012}$	$\mathbf{0.128}_{\pm 0.010}$	$\mathbf{0.167}_{\pm 0.014}$	$\mathbf{0.200}_{\pm 0.024}$
DS-LSTM (80%)	$0.198_{\pm 0.027}$	$0.161_{\pm 0.014}$	$0.186_{\pm 0.025}$	$0.229_{\pm 0.034}$

Table 7: Multi-task regression results (N_{tasks}, jointly trained) for synthetic dynamical system modeling datasets. Average MAE over all the datasets over which the models are trained is reported. Bold indicates best result per column.

		SplitMNIST		SplitHAR		SplitDigits		RotMNIST	
Model	s	Acc	Forg	Acc	Forg	Acc	Forg	Acc	Forg
LSTM	—	$90.6_{\pm 0.4}$	$0.101_{\pm .006}$	$72.9_{\pm 2.8}$	$0.175_{\pm .094}$	$79.8_{\pm 7.0}$	$0.126_{\pm .083}$	$83.1_{\pm 2.0}$	$0.178_{\pm .024}$
DS-LSTM	0.5	$90.2_{\pm 0.3}$	$0.112_{\pm .004}$	$78.9_{\pm 5.4}$	$\mathbf{0.044}_{\pm .050}$	$\mathbf{90.3}_{\pm 5.7}$	$\mathbf{0.044}_{\pm .061}$	$\mathbf{85.5}_{\pm 1.5}$	$\mathbf{0.153}_{\pm .020}$
DS-LSTM	0.6	$90.0_{\pm 0.5}$	$0.113_{\pm .006}$	$\mathbf{84.2}_{\pm 2.4}$	$0.064_{\pm .019}$	$\mathbf{90.3}_{\pm 4.9}$	$0.049_{\pm .063}$	$84.9_{\pm 2.1}$	$0.160_{\pm .026}$
DS-LSTM	0.7	$92.0_{\pm 0.0}$	$0.086_{\pm .001}$	$83.2_{\pm 1.2}$	$0.082_{\pm .020}$	$88.8_{\pm 8.7}$	$0.069_{\pm .110}$	$84.7_{\pm 2.2}$	$0.160_{\pm .027}$
DS-LSTM	0.8	$\mathbf{92.4}_{\pm 0.4}$	$\mathbf{0.085}_{\pm .004}$	$74.0_{\pm 6.5}$	$0.171_{\pm .016}$	$81.1_{\pm 11.0}$	$0.130_{\pm .086}$	$84.0_{\pm 2.1}$	$0.166_{\pm .025}$

Table 8: Full continual learning results across all sparsity levels, averaged across replay, reservoir, and GEM strategies. Acc ↑ = streaming accuracy; Forg ↓ = catastrophic forgetting. Bold marks the best result for each column per benchmark and task embedding condition.