Self-Partitioning Adaptive Widening Networks

Anonymous Authors[1]

Abstract

The design of neural architectures is commonly treated as a process separate from parameter optimization, which necessitates expensive retraining whenever the architecture is modified and frequently results in over-parameterized, memory-intensive models. To overcome these limitations, we introduce the Self-Partitioning Adaptive Widening Network (SPAWN), a novel framework that jointly learns both its architecture and its parameters by dynamically increasing its representational capacity during training. SPAWN starts from a single shallow predictor and incrementally builds a soft-gated mixture of experts by partitioning the input space where additional complexity is needed. In contrast to prior approaches that expand networks by adding neurons or layers, SPAWN increases capacity by recursively subdividing the input space, yielding a differentiable tree of affine experts trained end-to-end. This expansion process is governed by data-driven criteria that determine when to grow, where to split, and how to preserve the model's outputs, enabling seamless optimization without retraining from scratch. Empirical evaluations on standard regression and classification benchmarks demonstrate that SPAWN automatically discovers compact architectures that match or surpass the predictive performance of substantially larger, fixed-capacity models, while using markedly fewer parameters.

1. Introduction

The architecture of a learning model is a structural decision that fundamentally dictates its capacity to learn, generalize, and perform on a given task. Traditionally, the choice of this architecture relies on human knowledge and heuristics rather than systematic principles, often resulting in over-parameterized models that are computationally expensive to train and deploy. While automated methods such as Neural Architecture Search (NAS, Elsken et al., 2019), network pruning (Liu et al., 2019c), and evolutionary algorithms (Stanley & Miikkulainen, 2002) have been proposed to mitigate the need for manual tuning, they introduce significant challenges of their own. These approaches typically treat architecture search as an explicit optimization over a discrete or relaxed space, isolated from parameter training. Consequently, they are often statistically and computationally prohibitive, and can still yield architectures that are needlessly complex for the task at hand.

An alternative to architecture search is to allow the model to incrementally increase its capacity during the training process. This concept has proven effective in several settings, including continual learning (Rusu et al., 2016), optimization (Caccia et al., 2022), and reinforcement learning (Berner et al., 2019; Tedeschi et al., 2025). Building on this intuition, a promising line of research focuses on dynamically growing neural networks by adding neurons or layers one at a time, guided by gradient-derived signals (Chen et al., 2016; Evci et al., 2022; Wu et al., 2020). However, while modern NAS and dynamic growth methodologies have produced impressive results, they mostly focus on deep neural networks. In these frameworks, the unit of growth is a scalar parameter or a layer, leading to architectures that inherit the complex inductive biases of deep networks. Conversely, traditional methods that dynamically grow shallow architectures, such as decision trees (Kontschieder et al., 2015) or gradient boosting, typically rely on discrete, non-differentiable splits that cannot be optimized end-to-end via gradient descent. There remains a gap in the literature for methods that combine end-to-end differentiable neural network optimization with localized, data-driven growth of shallow, space-partitioning models.

This motivates our research question: *Can we shape the optimal architecture during the optimization process, starting from a minimal state, and expanding without exceeding the necessary dimensionality or relying on prior domain knowledge?*

We present *Self-Partitioning Adaptive Widening Network* (SPAWN), a novel neural architecture that steps away from the standard deep learning growth paradigm. Instead of

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

expanding a deep network by adding hidden units or layers, we dynamically grow a differentiable, shallow tree of experts. Crucially, our fundamental unit of growth is a region of the input space, rather than an isolated parameter. By dynamically partitioning the data space based on task feedback, we ensure the network expands its capacity only where local data complexity demands it.

Starting as a single shallow predictor, SPAWN progressively builds capacity by splitting individual experts into children, each learning a distinct affine map for its newly generated sub-region. Because the model's global nonlinear expressivity resides in this partition, the individual leaves can be instantiated with any generic estimator, ranging from inherently interpretable linear models to more complex functional forms, depending on the task. Rather than relying on heuristics, the structural evolution of SPAWN is governed by three data-driven principles: *when* to expand is triggered only after the current architecture has reached its optimal configuration; *where* to expand targets the specific input regions exhibiting the highest error variance; and *how* to expand guarantees strict output invariance, allowing gradient-based training to resume seamlessly after every split. The resulting process produces architectures whose capacity is set by the data rather than by the practitioner.

Original Contributions. In this work, we introduce *Self-Partitioning Adaptive Widening Network* (SPAWN), a new growing neural architecture that shapes its own structure during the training phase in a data-driven fashion. In Section 2, we formalize the core concepts of our architecture, while in Section 3 we explain and justify *when*, *where*, and *how* the expansion is performed. Then, in Section 4, we evaluate SPAWN architectures on regression and classification tasks over standard benchmarks, showing how our models converge to architectures that match the predictive accuracy of standard fixed-size MLP baselines and adaptive tree-based approaches, but with fewer parameters.

2. Architecture

SPAWN learns a predictor $f_\theta : \mathbb{R}^d \to \mathbb{R}^p$ for supervised problems, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathbb{R}^d$ denoting the input features and y_i taking values in $\mathbb{R}^p$ for regression or in a finite label set for classification.[1] We construct f_θ as a mixture of E linear experts, partitioned across the input space by $G = E-1$ soft gates arranged in a binary tree, so that capacity is allocated locally: each expert is responsible for a region of the input space, and the global nonlinearity of f_θ resides in the partition rather than in the experts themselves. Let $\mathcal{E} = \{1, \dots, E\}$ and $\mathcal{G} = \{1, \dots, G\}$ denote the expert and gate index sets, respectively. For each expert $h \in \mathcal{E}$, let $g_h(x) \in [0,1]$ denote the gating weight assigned by the model, with $\sum_{h\in\mathcal{E}} g_h(x) = 1$, and let $e_h(x) = \mathbf{W}_h x + b_h \in \mathbb{R}^p$ denote the affine expert output. The model prediction is then

$$f_\theta(x) \;=\; \sum_{h\in\mathcal{E}} g_h(x)\, e_h(x), \tag{1}$$

Section 2.1 details how the gating weights $g_h(x)$ are computed from the tree structure, and Section 2.2 introduces the regularizers that keep the routing well-behaved during training. Throughout this section, the number of experts E is held fixed; Section 3 then introduces the adaptive procedure that grows E from a single linear predictor.

2.1. Soft-Gated Tree of Linear Experts

The predictor f_θ combines E linear experts with a soft gating mechanism that computes their gating weights $g_h(x)$, structured as a binary tree whose internal nodes are sigmoidal gates and whose leaves are the experts. The tree structure localizes specialization: each new sub-tree repartitions only its parent's region, so expansion is naturally contained and preserves the routing of existing experts. This follows the hierarchical MoE tradition of (Jordan & Jacobs, 1994) and stands in contrast to flat softmax routing of modern large-scale MoE models (Shazeer et al., 2017; Fedus et al., 2022), where experts compete globally for gating weight and adding a new expert perturbs all existing assignments.

Gate. A single gate $k \in \mathcal{G}$ is a sigmoidal linear classifier parameterized by a weight vector $\omega_k \in \mathbb{R}^d$ and a bias $\beta_k \in \mathbb{R}$:

$$z_k(x) = \omega_k^\top x + \beta_k, \qquad p_k(x) = \mathrm{sigmoid}(z_k(x)) \in [0,1]. \tag{2}$$

The quantity $p_k(x)$ can be interpreted as the degree to which input x is assigned to the positive side of gate k; correspondingly, $1 - p_k(x) = \mathrm{sigmoid}(-z_k(x))$ represents the degree to which x is assigned to the negative side. The hyperplane $\{x : z_k(x) = 0\}$ is the decision boundary of the gate.

Tree structure. We arrange the G gates as internal nodes of a binary tree (Figure 1) whose $E = G + 1$ leaves are the experts. Each expert is therefore associated with a unique root-to-leaf path, and its gating weight $g_h(x)$ is the soft path weight obtained as a product of $p_k(x)$ factors for gates crossed on the positive side, and $1 - p_k(x)$ factors for gates crossed on the negative side.

Formally, we encode the tree topology in a *link matrix* $\mathbf{L} \in \{-1, 0, +1\}^{E\times G}$ whose entries are read off the tree as in

[1] Inputs are whitened using training-set statistics $\mu, \sigma \in \mathbb{R}^d$: $\tilde{x} = (x - \mu)/\sigma$ element-wise. This is standard practice in MLP training; we drop the tilde throughout.

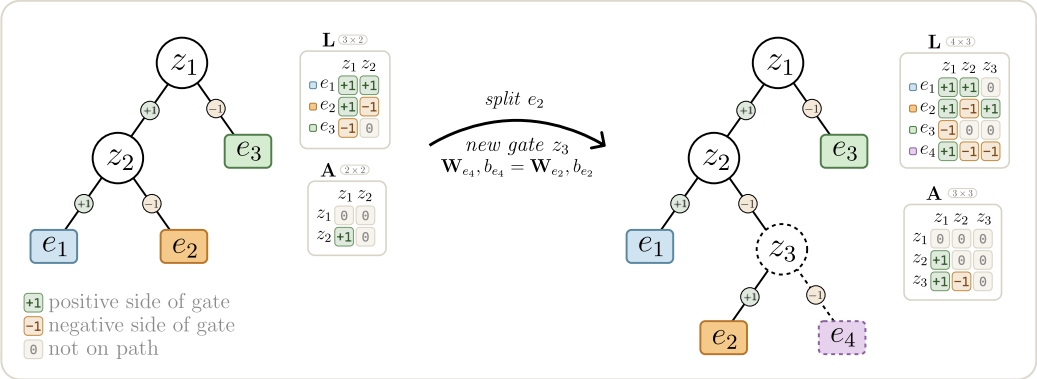

Figure 1. Expansion of SPAWN soft-gated binary tree (left to right).

Figure 1, with

$$\mathbf{L}_{hk} = \begin{cases} +1 & \text{if expert } h \text{ is on the positive side of gate } k, \\ -1 & \text{if expert } h \text{ is on the negative side of gate } k, \\ 0 & \text{if gate } k \text{ is not on the path to expert } h. \end{cases} \tag{3}$$

Splitting $\mathbf{L}$ into positive and negative parts $\mathbf{L}^+ = \max(\mathbf{L}, 0)$ and $\mathbf{L}^- = \max(-\mathbf{L}, 0)$, the log gating weight of each expert has a closed-form expression via a single matrix multiplication in log space:

$$\log g_h(x) \;=\; \sum_{k \in \mathcal{G}} \mathbf{L}^+_{hk} \log p_k(x) \;+\; \mathbf{L}^-_{hk} \log\big(1 - p_k(x)\big). \tag{4}$$

Evaluating the sum in log space is numerically stable under deep paths and reduces the tree evaluation to a fixed tensor contraction with no per-sample branching.

Experts. Each expert is an affine predictor $e_h(x) = \mathbf{W}_h x + b_h$, with $\mathbf{W}_h \in \mathbb{R}^{p \times d}$ and $b_h \in \mathbb{R}^p$, following the original mixture-of-experts formulation (Jacobs et al., 1991). Under soft routing, f_θ is a smooth non-linear function of x; in the hard-routing limit, it becomes piecewise affine over a polyhedral partition of the input space. In both regimes, the nonlinearity resides in the gating structure, while the predictors remain interpretable affine maps over an explicit region.

Mixture output. Combining the two components, the forward pass computes f_θ as in Equation 1; the full computation is summarized in Figure 2. The trainable parameters are $\theta = \{\omega_k, \beta_k\}_{k \in \mathcal{G}} \cup \{\mathbf{W}_h, b_h\}_{h \in \mathcal{E}}$.

2.2. Regularization Losses

Training a mixture of experts solely with a task loss $\mathcal{L}_t(\theta)$ often suffers from two failure modes: *fractional routing*, in which every sample is ambiguously assigned to many experts at once, and *dead experts*, in which a few experts dominate while others receive negligible gating weight (Shazeer et al., 2017; Fedus et al., 2022). We counter both with entropy-based regularizers acting on complementary aspects of the routing distribution: one at the sample level, discouraging ambiguity, and one at the gate level, encouraging balanced partitioning. The full training objective is

$$\mathcal{L}(\theta) = \mathcal{L}_t(\theta) + \lambda_s \mathcal{L}_s(\theta) + \lambda_m \mathcal{L}_m(\theta), \tag{5}$$

with scalar coefficients $\lambda_s, \lambda_m \geq 0$. Both regularizers are normalized to lie in $[0, 1]$, so the two coefficients operate on a common scale.

Sample Entropy. For each sample x in a batch $\mathcal{B}$, the gating-weight vector $g(x) = (g_h(x))_{h \in \mathcal{E}}$ lies on the $(E-1)$-simplex. Its Shannon entropy $H_s(x) = -\sum_{h \in \mathcal{E}} g_h(x) \log g_h(x) \in [0, \log E]$, measures how ambiguously the sample is routed: it is zero when one expert receives all the gating weight, and $\log E$ when the gating weight is uniform across experts. Penalizing this quantity encourages each sample to commit to a single expert, in the spirit of minimum-entropy regularization (Grandvalet & Bengio, 2004; Pereyra et al., 2017). We minimize the batch-averaged entropy, normalized by its maximum value:

$$\mathcal{L}_s(\theta) = \frac{1}{|\mathcal{B}| \log E} \sum_{x \in \mathcal{B}} H_s(x) \;\in\; [0, 1]. \tag{6}$$

The normalization by $\log E$ is essential for stability across expansions, as it keeps the regularizer scale invariant to the current model size.

Marginal Entropy. While sample entropy acts on individual samples, the marginal regularizer acts on the average routing behavior of each gate. Let $m_k(x) \in [0, 1]$ denote the path weight of sample x reaching gate k, defined as the product of $p_j(x)$ and $1 - p_j(x)$ factors along the path from the root to k. The effective split ratio of gate k is then the path-weighted average of its positive assignments,

$$\bar{p}_k = \frac{\sum_{x \in \mathcal{B}} m_k(x)\, p_k(x)}{\sum_{x \in \mathcal{B}} m_k(x)}, \tag{7}$$

where a balanced gate has $\bar{p}_k \approx \frac{1}{2}$ and a collapsed gate has $\bar{p}_k \in \{0, 1\}$. We enforce balance by penalizing the deviation

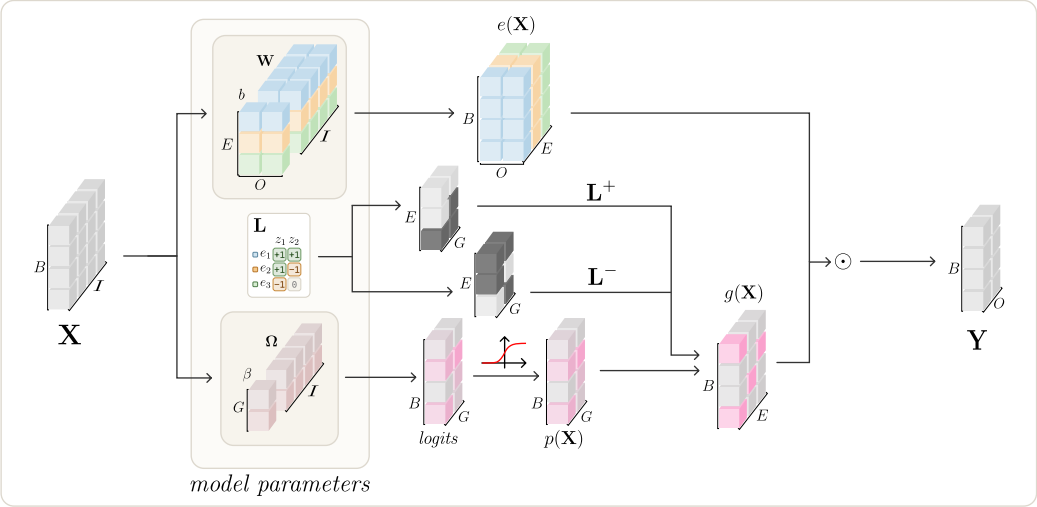

Figure 2. Forward pass for soft-tree structure.

of the per-gate binary entropy $H_b(p) = -p \log p - (1-p)\log(1-p)$ from its maximum $\log 2$, averaged across gates:

$$\mathcal{L}_m(\theta) = 1 - \frac{1}{G \log 2} \sum_{k \in \mathcal{G}} H_b(\bar{p}_k) \in [0, 1]. \quad (8)$$

This penalty is related to that of Frosst & Hinton (2017), but our per-gate formulation encodes tree geometry directly rather than via an explicit depth-decay factor, and the $\log 2$ normalization makes the regularizer invariant to the number of experts.

Evaluating marginal entropy *per gate* rather than over the full E-way gating distribution applies local soft pressure: each gate is pushed toward balanced partitioning of its own region, while still allowing the task loss to favor asymmetric splits when they materially improve prediction. The per-gate formulation also avoids conflating shallow gates, which partition large regions, with deep gates, which partition small sub-regions. Together with the matched $[0, 1]$ normalization of both regularizers, this preserves the relative weighting of λ_s and λ_m as E grows along the expansion trajectory. We refer the reader to Appendix B for an extended ablation on the regularization losses.

3. Adaptive Expansion

The architecture in Section 2 assumes a fixed number of experts E, but SPAWN does not: it starts from $E = 1$, a single linear predictor, and incrementally adds experts during training, splitting under-fit regions using only signals from the forward pass. This is efficient because of the hierarchical gating: a leaf can be replaced by a small subtree without changing the closed-form gating, and new affine experts inherit their parent's parameters so that each expansion is a continuous deformation, not a structural reset. Each expansion involves three decisions: *when* to expand (Section 3.1), *where* to expand, i.e., which expert to split (Section 3.2), and *how* to split so training continues smoothly (Section 3.3). Section 3.4 combines these into the outer expansion loop that yields the final model.

3.1. When to Expand: Plateau Detection

We trigger expansion once the training loss has plateaued, since adding capacity is only useful after the model has extracted as much signal as its current size allows. Expanding earlier would conflate the optimization residual with structural underfitting, biasing the expert-selection criterion of Section 3.2 toward regions that have not yet finished training. Plateau is detected by maintaining a length-$2w$ buffer of recent epoch losses and comparing the means of its older and newer halves: when the relative improvement (clipped at zero) falls below a threshold η, the criterion triggers. We use training rather than validation loss here so that the trigger reflects parameter convergence, not capacity sufficiency; capacity is judged by the outer cycle-level stopper, which retains the validation-best checkpoint across all cycles and epochs as the final model.

3.2. Where to Expand: Within-Region Heterogeneity

The natural choice of average per-expert loss is unsuitable as an expansion signal: a region whose residuals reflect irreducible noise can carry uniformly high loss that no further partitioning will reduce. The signature of structural under-fitting is instead within-region heteroskedasticity, the variation in loss that a single affine predictor cannot accommodate. We aggregate this variation as a sum rather than an average so that, all else equal, populous regions are refined before sparsely sampled ones.

We operationalize this intuition with a gating-weighted mean absolute deviation. For each expert $h \in \mathcal{E}$, let $\ell(x)$ be the per-sample task loss. The gating-weighted mean loss in region h, computed over each mini-batch $\mathcal{B}$, is

$$\bar{\ell}_h(\mathcal{B}) \;=\; \frac{\sum_{x\in\mathcal{B}} g_h(x)\,\ell(x)}{\sum_{x\in\mathcal{B}} g_h(x)}, \tag{9}$$

and the expansion score for expert h accumulates the gating-weighted absolute deviation from this batch-local mean across all mini-batches in the epoch,[2]

$$s_h \;=\; \sum_{\mathcal{B}} \sum_{x\in\mathcal{B}} g_h(x)\,\big|\ell(x) - \bar{\ell}_h(\mathcal{B})\big|. \tag{10}$$

We select $h^\star = \arg\max_{h\in\mathcal{E}} s_h$ as the expert to expand.

3.3. How to Expand: Output-Preserving Inheritance

Once the expert $h^\star$ has been selected, we introduce a new gate that partitions its region into two children and initialize all new parameters so that the model's output at the moment of expansion is exactly preserved. Training resumes from the same functional state, with the regularizers of Section 2.2 guiding the subsequent differentiation between children. The new gate k' has a weight vector ω' sampled uniformly from a small hypercube, since splitting a homogeneous region admits no principled *direction* of separation, and bias β' chosen so that the decision hyperplane passes through the gating-weighted centroid $\mu_{h^\star}$ of samples routed to $h^\star$. This fresh gate is appended as a new column of $\mathbf{L}$ and a new row and column of $\mathbf{A}$. The initialization produces a near-balanced split for typical input densities, with the exact balance set by ω'. Let h^+ and h^- denote the positive and negative children of $h^\star$ under the new gate k'. The two children inherit the parent's expert parameters exactly, $\mathbf{W}_{h^+} = \mathbf{W}_{h^-} = \mathbf{W}_{h^\star}$ and likewise for the biases. Since their predictors are identical at expansion, their joint contribution to the model output is

$$g_{h^+}(x)\,e_{h^+}(x) + g_{h^-}(x)\,e_{h^-}(x) \;=\; \big(g_{h^+}(x) + g_{h^-}(x)\big)\,e_{h^\star}(x), \tag{11}$$

and the tree structure forces the bracketed sum to equal $g_{h^\star}(x)$, preserving the model's output exactly. The link matrix $\mathbf{L}$ is updated accordingly (Figure 1).

3.4. The Expansion Cycle

The three decisions above fit into a simple outer loop: the model alternates between training phases and expansion steps, with the optimizer state and learning rate schedule reinitialized at the start of each cycle. After every cycle, we evaluate the model on a held-out validation set and retain the best-seen parameters. Expansion stops when validation loss fails to improve over a configured number of consecutive cycles, with a hard cap as a second safeguard.

[2] Absolute rather than squared deviation prevents a fourth-power dependence on residuals under MSE regression that would be dominated by outliers; gating-weighting rather than hard assignment keeps the score smooth in the model parameters.

4. Experiments

We evaluate SPAWN on a selection of datasets, both artificial and real-world, targeting both classification and regression tasks of increasing complexity. In this evaluation, we highlight the capacity of SPAWN to autonomously devise efficient architectures that achieve state-of-the-art performance. We compare SPAWN against a range of fixed-size and adaptive baselines, including, MLPs (Rumelhart et al., 1986), Random Forest (Breiman, 2001), XGBoost (Chen & Guestrin, 2016) and Soft Trees (Irsoy et al., 2012). To denote specific model configurations, runs are identified by a compact labeling. We adopt wW_d for MLPs with hidden width W and depth d, tT_d for ensembles of T trees with maximum depth d, and dD for Soft Trees of fixed depth D. The training analysis is intentionally one-sided: each baseline family is swept over a small capacity grid, and the validation-best run is reported per dataset, while SPAWN runs once, learning its own structural model. Section 4.1 reports the best-of-family comparison, Section 4.2 isolates the parameter-efficiency claim against fixed MLPs, and Section 4.3 relaxes the all-linear setting along the gate and expert axes. Datasets, baseline families, hardware, and reproducibility details are presented in Appendix C.

All adaptive runs share the same configuration on every dataset and every variant: the entropy-regularization coefficients of Equation 5 are fixed at $\lambda_s = \lambda_m = 10^{-2}$, with Adam (Kingma & Ba, 2014) at learning rate 10^{-2}, a per-fit budget of up to $C_{\max} = 800$ expansion cycles, and a maximum of 2000 epochs per fit. MLPs use AdamW (Loshchilov & Hutter, 2017) with learning rate 5×10^{-3} and weight decay 10^{-4}. Reported scores are the mean over 20 seeds with $\pm 95\%$ Confidence Intervals, with **bold** marking the best displayed mean in each row and the non-best entries whose difference from it is not statistically significant under a two-sided Welch t-test (Welch, 1947) at $\alpha = 0.05$. The complete hyperparameter table is reported in Appendix D.

Growth patience. The growing procedure of Section 3 stops when validation loss no longer improves over a number of consecutive expansion cycles (the *growth patience*). This hyperparameter determines how aggressively the outer loop commits to a final architecture: a long growth patience continues to expand even when validation has temporarily stalled, recovering models that grow further before stopping; a short growth patience terminates earlier and returns smaller, faster models whose accuracy sits at a slightly earlier point along the size-accuracy curve. We hold growth patience fixed at the conservative value reported in Appendix D across the entire benchmark, deliberately favoring

		Baselines				SPAWN
	Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Classification	Spirals	**100.0** ±0.0 527.9k ($w512_3$) / 5.7s	98.8 ±0.1 88.8k ($t300_{16}$) / 251ms	98.6 ±0.1 40.4k ($t1k_8$) / 89ms	76.3 ±1.3 1.3k (d8) / 30.5s	99.3 ±0.4 456 / 1.3m
	3-Spirals	**100.0** ±0.0 791.0k ($w512_4$) / 13.6s	97.0 ±0.2 86.5k ($t300_{16}$) / 384ms	97.6 ±0.1 143.9k ($t1k_8$) / 335ms	91.5 ±1.5 1.5k (d8) / 51.9s	98.9 ±0.2 645 / 1.6m
	Pinwheel	94.4 ±0.2 1.1M ($w1k_2$) / 3.4s	**95.2** ±0.1 30.0k ($t200_{12}$) / 106ms	95.0 ±0.0 22.6k ($t300_4$) / 55ms	94.5 ±0.2 2.0k (d8) / 28.9s	**95.0** ±0.2 287 / 20.1s
	Checker	89.8 ±0.3 8.6k ($w64_3$) / 4.9s	89.6 ±0.1 296.6k ($t300_{16}$) / 368ms	**91.0** ±0.1 60.4k ($t1k_8$) / 72ms	85.6 ±0.5 1.3k (d8) / 1.9m	90.3 ±0.2 1.2k / 4.4m
	BreastCnc	95.7 ±0.3 338 ($w8_2$) / 258ms	95.3 ±0.2 969 ($t50_4$) / 23ms	94.1 ±0.2 2.0k ($t300_4$) / 42ms	**97.0** ±0.5 8.4k (d8) / 3.6s	95.0 ±0.6 155 / 19.4s
Regression	Sin	0.299 ±0.001 526.8k ($w512_3$) / 1.7s	0.314 ±0.000 28.3k ($t100_8$) / 38ms	0.307 ±0.000 6.0k ($t300_4$) / 10ms	0.319 ±0.013 766 (d8) / 20.7s	**0.296** ±0.001 192 / 23.7s
	Sin^2	0.295 ±0.001 526.8k ($w512_3$) / 5.2s	0.337 ±0.001 156.4k ($t200_{12}$) / 92ms	0.304 ±0.000 7.6k ($t300_4$) / 14ms	0.582 ±0.003 766 (d8) / 25.1s	**0.290** ±0.001 2.1k / 4.3m
	Rastrigin	0.272 ±0.009 132.6k ($w256_3$) / 8.2s	0.390 ±0.001 487.7k ($t300_{16}$) / 333ms	**0.168** ±0.001 59.3k ($t600_6$) / 100ms	0.651 ±0.001 61 (d4) / 1.8s	0.352 ±0.063 3.6k / 13.4m
	Diabetes	**0.669** ±0.009 2.1M ($w1k_3$) / 1.2s	0.698 ±0.003 64.0k ($t200_{12}$) / 123ms	0.696 ±0.006 5.1k ($t300_4$) / 11ms	**0.667** ±0.011 37 (d2) / 340ms	**0.670** ±0.009 34 / 12.9s
	CalHousing	0.456 ±0.003 10.2k ($w1k_1$) / 18.8s	0.444 ±0.000 3.1M ($t300_{16}$) / 14.2s	**0.387** ±0.001 350.8k ($t1k_8$) / 1.2s	0.468 ±0.003 2.6k (d8) / 1.8m	0.470 ±0.010 715 / 7.3m
	kin8nm	**0.255** ±0.003 134.1k ($w256_3$) / 4.2s	0.538 ±0.001 1.9M ($t300_{16}$) / 5.2s	0.438 ±0.002 65.8k ($t600_6$) / 447ms	0.267 ±0.002 2.6k (d8) / 42.6s	0.260 ±0.003 677 / 2.4m

Table 1. Best-of-family comparison on classification and regression benchmarks. Classification reports held-out test accuracy in percent; regression reports held-out test RMSE. Top entry: mean ± 95% CI over seeds. Bottom entry: model size (red; parameters for neural models, nodes for tree models) and mean wall-clock training time (blue).

the higher-accuracy end of this trade-off to probe the procedure's robustness rather than its tunability.

4.1. Best-of-Family Comparison

Table 1 reports a comparison between SPAWN and the validation-optimal member of each baseline family, where the latter is selected via a per-dataset sweep over model capacities. We notice how SPAWN often remains close to the best-performing member of each baseline family under row-wise Welch tests on the displayed held-out scores, while learning an architecture whose parameters are typically one to four orders of magnitude smaller than those of the corresponding MLP baselines, and comparable to or lower than those of the soft-tree counterparts.

On the datasets where SPAWN's mean loss deviates from the best baseline, the per-seed standard deviation is large enough that the two distributions overlap substantially, indicating that the gap is driven by a small number of runs whose growth was cut short by the cycle stopper rather than by a systematic shortfall. This behavior follows directly from running the same fixed patience across every dataset: the values reported in Appendix D are tuned to be conservative on the longest-growing benchmarks, so on faster-converging datasets, the same settings can occasionally accept a suboptimal cycle before the validation curve has fully stabilized. We hold patience fixed across the benchmark to show that the adaptive procedure remains competitive even without per-dataset tuning.

Furthermore, we provide a detailed view of the training dynamics in Figure 3 and 4. The training loss curve (Figure 3) shows a staircase pattern, characterized by performance plateaus that are resolved upon network expansion. Each time a new expert is added, the network gains expressive power, driving the loss to a lower level. Conversely, Figure 4 illustrates the held-out loss. Here, the loss initially descends as useful capacity is introduced (indicated by red dotted lines), but eventually increases when subsequent splits induce overfitting. This inflection point provides the crucial feedback signal utilized by the outer stopping mechanism to terminate structural growth.

Growth patience tuning. The wall-clock training times reported alongside the parameter counts measure an upper bound on the cost of producing the reported model, not an intrinsic cost of the method. The growth trajectories (Tables 13 and 16 of Appendix F) make this concrete: every intermediate checkpoint along an adaptive run is itself a fully trained, optimized model, and on most datasets a checkpoint a few cycles before the selected one already attains accuracy within a small margin of the final score at a fraction of the

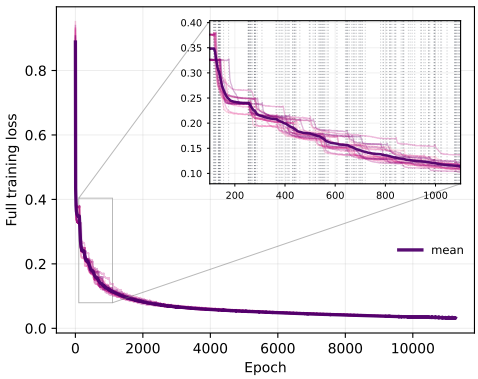

Figure 3. Training loss across the full adaptive run, with a zoom inset on a few expansion cycles.

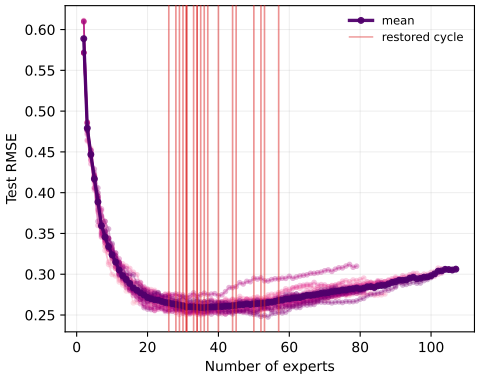

Figure 4. Test loss versus expansion cycle, with the best validation cycle restored by early stopping marked.

parameters and the wall-clock time. Thus, a simple growth-patience tuning can achieve comparable performance while reducing training time.

4.2. Parameter Efficiency Against Fixed MLPs

The best-of-family table compares against the validation-best member of each family. We now hold the family fixed to MLPs and ask how large a fixed MLP must be before it becomes statistically equivalent to the validation-selected SPAWN run.

For each dataset and MLP run, we apply a paired-sample equivalence test on per-seed validation scores when the seeds align and Welch's two-sample test otherwise, with TOST (Schuirmann, 1987) at $\alpha = 0.05$ and an equivalence margin of half the adaptive run's validation-score standard deviation. The smallest MLP run classified as equivalent or superior is reported in the *smallest equiv.* column, and a dash means that no MLP configuration met this criterion. Full procedural details, including the same-complexity run selection rule and the convention used in marginal cases, are in Appendix G.

The same-budget columns measure how effectively a fixed MLP uses a comparable parameter budget; on the geometric and high-frequency synthetic tasks, it usually cannot match SPAWN, since the MLP spends parameters globally while SPAWN spends them where residual structure forces a split. Dash entries in the smallest-equivalent column are meaningful: the largest evaluated MLP still did not reach the equivalence criterion against the selected adaptive run. Overall, we observe that SPAWN in every setting learned a minimal model representation strong enough to achieve comparable rather than superior performance w.r.t. remarkably higher MLP models.

4.3. Beyond the Linear Limit: Gate and Expert Ablations

The expansion procedure of SPAWN is independent of the function class used in each gate and expert. We choose quadratic gates as the minimal expressivity bump beyond the linear baseline: they admit curved decision boundaries while preserving the closed-form gating of Section 2.1, which isolates the effect of richer routing from confounders such as deeper or non-linear leaves. We leverage this to a relaxation of the all-linear setting, **SPAWN-LL**. **SPAWN-QL** uses quadratic gates with linear leaves. We choose this as a minimal probe; however, the formulation also allows richer structures (e.g., MLPs, CNNs, or more sophisticated functional classes). Based on the results in Table 3, introducing quadratic gating (**SPAWN-QL**) while maintaining linear leaves significantly improves the performance-efficiency trade-off of SPAWN. By enabling richer expressiveness in the routing mechanism, **SPAWN-QL** consistently achieves predictive performance comparable to or superior to the all-linear baseline (**SPAWN-LL**), often with a fraction of the parameters. For instance, on highly non-linear tasks such as Rastrigin and 3-Spirals, the quadratic gates substantially improve test scores. Furthermore, this enhanced expressivity directly translates into computational efficiency: on Sin^2 and 3-Spirals, **SPAWN-QL** drastically reduces the model size and cuts mean training time by an order of magnitude. This ablation confirms that selectively increasing the functional complexity of the gates yields a more compact and capable network.

5. Conclusion

We introduced the Self-Partitioning Adaptive Widening Network (SPAWN), a novel neural architecture that unifies parameter optimization with structural architectural design by dynamically growing its capacity during training. Through extensive evaluation, we demonstrated that SPAWN can devise efficient model structures that match or exceed the performance of substantially larger, fixed-capacity baselines across standard regression and classification tasks. Despite

		SPAWN	MLP		
	Dataset	SPAWN	same size	smallest equiv.	best observed
Classification	Spirals	99.3 ±0.4 456 / 1.3m	**97.2** ±2.8 626 ($w16_3$) / 5.6s	**100.0** ±0.0 8.6k ($w64_3$) / 6.2s	**100.0** ±0.0 527.9k ($w512_3$) / 5.7s
	3-Spirals	98.9 ±0.2 645 / 1.6m	**97.7** ±2.4 643 ($w16_3$) / 7.0s	99.9 ±0.1 2.3k ($w32_3$) / 5.1s	**100.0** ±0.0 791.0k ($w512_4$) / 13.6s
	Pinwheel	**95.0** ±0.2 287 / 20.1s	**95.0** ±0.3 285 ($w8_4$) / 2.0s	94.6 ±0.1 1.0k ($w128_1$) / 3.1s	94.4 ±0.2 1.1M ($w1k_2$) / 3.4s
	Checker	**90.3** ±0.2 1.2k / 4.4m	87.8 ±0.8 1.2k ($w32_2$) / 16.4s	– –	89.8 ±0.3 8.6k ($w64_3$) / 4.9s
	BreastCnc	**95.0** ±0.6 155 / 19.4s	**95.5** ±0.4 154 ($w4_2$) / 424ms	**95.7** ±0.5 8.4k ($w256_1$) / 185ms	**95.7** ±0.3 338 ($w8_2$) / 258ms
Regression	Sin	**0.296** ±0.001 192 / 23.7s	0.326 ±0.016 193 ($w64_1$) / 3.4s	– –	0.299 ±0.001 526.8k ($w512_3$) / 1.7s
	Sin^2	**0.290** ±0.001 2.1k / 4.3m	0.436 ±0.046 2.2k ($w32_3$) / 3.3s	– –	0.295 ±0.001 526.8k ($w512_3$) / 5.2s
	Rastrigin	0.352 ±0.063 3.6k / 13.4m	0.607 ±0.006 4.1k ($w1k_1$) / 6.1s	0.326 ±0.017 66.8k ($w256_2$) / 4.4m	**0.272** ±0.009 132.6k ($w256_3$) / 8.2s
	Diabetes	**0.670** ±0.009 34 / 12.9s	0.727 ±0.043 37 ($w2_3$) / 356ms	**0.675** ±0.015 51.1k ($w128_4$) / 195ms	**0.669** ±0.009 2.1M ($w1k_3$) / 1.2s
	CalHousing	0.470 ±0.010 715 / 7.3m	**0.447** ±0.004 705 ($w16_3$) / 27.9s	**0.449** ±0.003 433 ($w16_2$) / 33.7s	0.456 ±0.003 10.2k ($w1k_1$) / 18.8s
	kin8nm	0.260 ±0.003 677 / 2.4m	0.281 ±0.004 705 ($w16_3$) / 10.8s	– –	**0.255** ±0.003 134.1k ($w256_3$) / 4.2s

Table 2. MLP parameter-efficiency comparison on classification and regression benchmarks. Displayed scores are test accuracy (percent) for classification and RMSE for regression. The row below each score reports model size in red, with selected MLP model IDs in parentheses, followed by mean wall-clock training time in blue.

	Regression (RMSE)				Classification (Acc %)			
Variant	Sin^2	Ackley	Rastrigin	kin8nm	Spirals	3-Spirals	Checker	BreastCnc
SPAWN-LL	0.290 ±0.001 2.1k / 4.3m	0.143 ±0.012 3.1k / 7.6m	0.352 ±0.063 3.6k / 13.4m	**0.260** ±0.003 677 / 2.4m	99.3 ±0.4 456 / 1.3m	98.9 ±0.2 645 / 1.6m	**90.3** ±0.2 1.2k / 4.4m	**95.0** ±0.6 155 / 19.4s
SPAWN-QL	**0.288** ±0.001 469 / 41.2s	**0.105** ±0.005 2.4k / 4.0m	**0.153** ±0.012 3.8k / 14.5m	0.267 ±0.003 629 / 1.8m	**99.9** ±0.1 545 / 1.4m	**100.0** ±0.1 183 / 1.0m	89.7 ±0.2 809 / 3.1m	**95.5** ±0.5 185 / 18.1s

Table 3. Results of SPAWN fully linear and with quadratic gates. Top entry: test score (RMSE for regression, accuracy in percent for classification). Bottom entry: model size (red) and mean wall-clock training time (blue).

its parameter efficiency, SPAWN introduces computational overhead during its expansion phase, as a newly introduced predictor must be optimized before performing another expansion. However, hierarchical mixing of experts allows concurrent evaluation of underfit regions, thereby reducing the time burden even further. Finally, the expansion procedure is entirely orthogonal to the function class used within each gate and expert, allowing for further structural enhancements. Future work will explore integrating richer functional forms into the leaf predictors, such as MLPs or CNNs.

References

Ackley, D. H. *A Connectionist Machine for Genetic Hill-climbing*, volume 28 of *The Springer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, MA, 1987. doi: 10.1007/978-1-4613-1997-9.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001. doi: 10.1023/A:1010933404324.

Budden, D., Marblestone, A., Sezener, E., Lattimore, T.,

Wayne, G., and Veness, J. Gaussian gated linear networks. *Advances in Neural Information Processing Systems*, 33: 16508–16519, 2020.

Caccia, L., Xu, J., Ott, M., Ranzato, M., and Denoyer, L. On anytime learning at macroscale. In *Conference on Lifelong Learning Agents*, pp. 165–182. PMLR, 2022.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785.

Chen, T., Goodfellow, I., and Shlens, J. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004. doi: 10.1214/009053604000000067.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Evci, U., van Merrienboer, B., Unterthiner, T., Pedregosa, F., and Vladymyrov, M. GradMax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022.

Fahlman, S. E. and Lebiere, C. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2, pp. 524–532. Morgan Kaufmann, 1990.

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

Frosst, N. and Hinton, G. E. Distilling a neural network into a soft decision tree. Workshop on Interpreting, Explaining and Visualizing Deep Learning – Now what?, NIPS 2017, 2017.

Grandvalet, Y. and Bengio, Y. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, volume 17, pp. 529–536. MIT Press, 2004.

Irsoy, O. T., Yildiz, O. T., and Alpaydın, E. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, pp. 1819–1822. IEEE, 2012.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Johnson, M. J., Duvenaud, D. K., Wiltschko, A. B., Adams, R. P., and Datta, S. R. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, volume 29, pp. 2954–2962, 2016.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6 (2):181–214, 1994. doi: 10.1162/neco.1994.6.2.181.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulò, S. R. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1467–1475. IEEE, 2015. doi: 10.1109/ICCV.2015.172.

Li, Q. and Sompolinsky, H. Globally gated deep linear networks. *Advances in Neural Information Processing Systems*, 35:34789–34801, 2022.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a.

Liu, Q., Wu, L., and Wang, D. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, 2019b.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019c.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Neal, R. M. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996. doi: 10.1007/978-1-4612-0745-0.

Pace, R. K. and Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997. doi: 10.1016/S0167-7152(96)00140-X.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., and Hinton, G. E. Regularizing neural networks by penalizing confident output distributions. ICLR 2017 Workshop, 2017.

Rosenbrock, H. H. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960. doi: 10.1093/comjnl/3.3.175.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. arXiv preprint arXiv:1606.04671, 2016.

Schuirmann, D. J. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of pharmacokinetics and biopharmaceutics*, 15(6):657–680, 1987.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Tanno, R., Arulkumaran, K., Alexander, D. C., Criminisi, A., and Nori, A. Adaptive neural trees. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6166–6175. PMLR, 2019.

Tedeschi, G., Papini, M., Metelli, A. M., and Restelli, M. Search or split: Policy gradient with adaptive policy space. *Machine Learning*, 114:186, 2025. doi: 10.1007/s10994-025-06820-2.

Törn, A. and Žilinskas, A. (eds.). *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, 1989. doi: 10.1007/3-540-50871-6.

Veness, J., Lattimore, T., Budden, D., Bhoopchand, A., Mattern, C., Grabska-Barwinska, A., Sezener, E., Wang, J., Toth, P., Schmitt, S., and Hutter, M. Gated linear networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10015–10023. AAAI Press, 2021. doi: 10.1609/aaai.v35i11.17202.

Welch, B. L. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947. doi: 10.1093/biomet/34.1-2.28.

Wolberg, W., Mangasarian, O., Street, N., and Street, W. Breast cancer wisconsin (diagnostic). UCI Machine Learning Repository, 1993.

Wu, L., Liu, B., Stone, P., and Liu, Q. Firefly neural architecture descent: a general approach for growing neural networks. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A. Related Work

In this appendix, we discuss the literature relevant to this work. We first review dynamic architectures and network growth, in contrast to standard architecture search. We then examine tree-based partitioning models, and finally discuss mixture-of-experts approaches.

Dynamic Architectures and Network Growth. Automated architecture discovery is typically addressed via Neural Architecture Search (Zoph & Le, 2017) and differentiable variants like DARTS (Liu et al., 2019a). However, these approaches treat architecture discovery as an explicit optimization over a specific, *fixed* search space, which is often decoupled from the training procedure itself. Conversely, SPAWN does not search for an optimal architecture within a predefined space; rather, it grows its own optimal model while simultaneously optimizing its parameters. A paradigm closer to ours lies in the literature on growing neural networks. Classical methods like Cascade Correlation (Fahlman & Lebiere, 1990) and Net2Net (Chen et al., 2016) incrementally expand the capacity of the model, while recent gradient-driven approaches, such as Splitting Steepest Descent (Liu et al., 2019b), GradMax (Evci et al., 2022), and Firefly (Wu et al., 2020), add neurons or layers based on observed training signals. The main difference between this family of methods and our work lies in the unit of growth. Existing approaches expand deep networks by adding scalar parameters or hidden units, which inherently fragment capacity. In contrast, SPAWN grows by partitioning regions of the input space via learned hyperplanes, allowing the model to retain a single, coherent affine predictor per partition. Philosophically, this *starting small* intuition is similar to GAPS (Tedeschi et al., 2025), an adaptive-space policy search algorithm that grows the policy when needed and associates specific policies with different state partitions. However, whereas GAPS uses flat, hard axis-aligned cuts driven by gradient angles, we adapt this capacity curriculum to supervised learning using a hierarchical soft tree, generalizing the algorithm to a broader spectrum of tasks.

Tree-Based and Partitioning Models. A parallel line of work embeds tree structure into differentiable models, including soft decision trees (Irsoy et al., 2012) and Deep Neural Decision Forests (Kontschieder et al., 2015). While models like those of Frosst & Hinton (2017) use similarly to SPAWN soft routing gates to determine the policy associated to the leaves, their tree depth is fixed in advance, and their leaves emit static class distributions rather than input-dependent predictions. Most closely related to our growth mechanism are Adaptive Neural Trees (ANTs, Tanno et al., 2019), which also expand during training. We differ from ANTs on three major points: (i) ANT routers and leaves can be deep neural networks, whereas our gates and experts also admit linear solutions, locating the non-linear capacity entirely in the partition. (ii) ANT growth is driven by validation-based search, whereas ours relies purely on training-time signals. (iii) ANT trains new modules locally, while we train the full model jointly at every cycle, achieving smooth continuation via output-preserving parameter inheritance. Closer in inductive bias to our architecture is the family of Gated Linear Networks (GLNs, Veness et al., 2021; Budden et al., 2020; Li & Sompolinsky, 2022). A GLN is a feed-forward stack of neurons in which each neuron stores a bank of linear weight vectors and selects among them via a halfspace context function, with predictions combined across layers by gated geometric mixing and trained locally via online convex optimization at each neuron. Yet, unlike GLNs, which rely on a (deep) fixed stack of randomly sampled gates trained locally, our architecture discovers a shallow, learned binary tree optimized globally end-to-end.

Mixtures of experts and Regularization. Finally, SPAWN can be seen as a dynamically growing Mixture of Experts (MoE). Traditional MoE framewoks (Jacobs et al., 1991; Jordan & Jacobs, 1994), including modern sparsely-gated variations scaled to massive regimes (Shazeer et al., 2017; Fedus et al., 2022) treat the number of experts as a fixed hyperparameter chosen in advance. Our contribution diverges significantly as the number of experts is adjusted during training to account for within-region heterogeneity. The dynamical growth for the network, and thus in the number of experts, requires adequate regularization. In our work, while we draw on minimum-entropy regularization (Grandvalet & Bengio, 2004; Pereyra et al., 2017) and load-balancing auxiliary losses (Shazeer et al., 2017; Fedus et al., 2022), we uniquely normalize these terms so their relative weighting remains invariant as the number of experts grows. This scale-invariance is crucial for continuous expansion and represents a requirement previously unaddressed by fixed-size MoE architectures.

B. Regularization Losses: Extended Derivation

This section provides additional details on the entropy-based regularizers introduced in Section 2.2. We first formalize the closed-form computation of the per-gate path weights $m_k(x)$ via a gate-ancestry matrix, and then expand on the comparison with related formulations from the soft-tree and sparsely-gated MoE literature.

B.1. Closed-Form Path Weights via the Gate-Ancestry Matrix

Recall from Section 2.2 that $m_k(x) \in [0,1]$ denotes the soft path weight of sample x reaching gate k, defined as the product of $p_j(x)$ and $1 - p_j(x)$ factors along the root-to-k path. We encode this path structure in a *gate-ancestry matrix* $\mathbf{A} \in \{-1, 0, +1\}^{G \times G}$, analogous to the link matrix $\mathbf{L}$ of Section 2 and read off the tree in the same way (Figure 1):

$$\mathbf{A}_{kj} = \begin{cases} +1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its positive side,} \\ -1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its negative side,} \\ 0 & \text{otherwise (including } j = k\text{).} \end{cases} \tag{12}$$

Splitting $\mathbf{A}$ into its positive and negative parts $\mathbf{A}^+ = \max(\mathbf{A}, 0)$ and $\mathbf{A}^- = \max(-\mathbf{A}, 0)$, the log path weight admits a closed-form expression analogous to Equation 4:

$$\log m_k(x) = \sum_{j \in \mathcal{G}} \mathbf{A}^+_{kj} \log p_j(x) + \mathbf{A}^-_{kj} \log\big(1 - p_j(x)\big), \tag{13}$$

with the root gate corresponding to the all-zero row of $\mathbf{A}$, so $m_{\text{root}}(x) = 1$ as expected. As with the link matrix $\mathbf{L}$, evaluating this sum in log space is numerically stable under deep paths and reduces the computation of all per-gate path weights to a single tensor contraction with no per-sample branching.

B.2. Comparison with Related Formulations

Soft decision trees. The per-gate balance penalty of Equation 8 is closely related to the regularizer of Frosst & Hinton (2017), which penalizes the cross-entropy of the path-weighted split ratio α_i against the uniform target $(0.5, 0.5)$. Their α_i corresponds exactly to our $\bar{p}_k$, and their cross-entropy penalty equals our per-gate binary entropy $H_b(\bar{p}_k)$ up to an additive constant. The two formulations differ in how they handle tree depth. In Frosst & Hinton (2017) each gate's contribution is being multiplied by an explicit 2^{-d} factor that down-weights deeper gates a priori, while our formulation encodes tree geometry directly through the path weights $m_k(x)$ in the definition of $\bar{p}_k$ (Equation 7). Each gate is thus weighted by the actual fraction of the data it routes, rather than by a fixed function of its depth, which adapts naturally to the unbalanced expansion trajectories produced by the data-driven splitting criterion of Section 3, where deeper sub-trees may still receive substantial mass.

Sparsely gated MoE. A related load-balancing objective appears in the auxiliary losses of sparsely gated MoE architectures (Shazeer et al., 2017; Fedus et al., 2022), where the goal is to prevent a small number of experts from dominating routing under a flat softmax. There, balance is enforced globally over the full E-way expert distribution. Our per-gate formulation differs in two ways. (i) It is *local*, i.e., it applies soft pressure to each binary split independently. (ii) It is *hierarchical*, i.e., it exploits the tree structure to evaluate balance at the level of individual partitions rather than over the leaf set. As argued in Section 2.2, evaluating balance per gate avoids conflating shallow gates, which partition large regions, with deep gates, which partition small sub-regions, while still allowing the task loss to override the balance bias when an asymmetric split materially improves prediction.

B.3. Normalization and Invariance to Expansion

Both regularizers are normalized to lie in $[0, 1]$, which is essential for the adaptive procedure of Section 3. As the model grows, the number of experts E and the number of gates G both increase: an unnormalized sample entropy $H_s(x)$ has range $[0, \log E]$ that widens with growth, while an unnormalized sum $\sum_k H_b(\bar{p}_k)$ scales linearly with G. Without normalization, the relative weighting of λ_s and λ_m against the task loss would drift along the expansion trajectory, requiring per-cycle retuning. Dividing by $\log E$ and $G \log 2$ respectively keeps both penalties on a common $[0, 1]$ scale, so a single fixed pair (λ_s, λ_m) remains valid across the entire growth procedure.

B.4. Structural Necessity of the Regularizers at Expansion

The output-preserving inheritance of Section 3.3, formalized by Equation 11, motivates even more our regularizers choice: at every expansion, the task-loss gradient on the parameters of the newly inserted gate is identically zero, leaving $\mathcal{L}_s$ and $\mathcal{L}_m$ as the only possible first-order source of routing differentiation. We make this precise below.

Adopting the notation of Section 3.3, let k' be the new gate with parameters (ω', β') and h^+, h^- its two children inheriting $\mathbf{W}_{h^\star}, b_{h^\star}$. Given $\theta' = \{\omega', \beta', \mathbf{W}_{h^+}, b_{h^+}, \mathbf{W}_{h^-}, b_{h^-}\}$ and $\cdot|_0$ for evaluation at this initialization.

For every input x and every task loss $\mathcal{L}_t$ differentiable in $f_\theta(x)$,

$$\nabla_{\omega'}\mathcal{L}_t\big|_0 = 0, \qquad \nabla_{\beta'}\mathcal{L}_t\big|_0 = 0,$$

whereas $\nabla_{\mathbf{W}_{h^\pm}}\mathcal{L}_t|_0$ and $\nabla_{b_{h^\pm}}\mathcal{L}_t|_0$ are in general non-zero.

Proof. By the chain rule, it suffices to analyze $\nabla_{\theta'} f_\theta(x)$. Among the E summands of Equation 1, only those indexed by h^+, h^- depend on θ', and their joint contribution is

$$g_{h^+}(x)\, e_{h^+}(x) + g_{h^-}(x)\, e_{h^-}(x). \tag{14}$$

Since $g_{h^+}(x) + g_{h^-}(x) = g_{h^\star}(x)$ by Equation 11 and $g_{h^\star}(x)$ is independent of (ω', β'), differentiating yields

$$\frac{\partial g_{h^+}(x)}{\partial \omega'} = -\frac{\partial g_{h^-}(x)}{\partial \omega'}, \qquad \frac{\partial g_{h^+}(x)}{\partial \beta'} = -\frac{\partial g_{h^-}(x)}{\partial \beta'}.$$

Differentiating Equation 14 with respect to ω', and using $e_{h^+}(x) = e_{h^-}(x) = e_{h^\star}(x)$ at initialization,

$$\frac{\partial}{\partial \omega'}[g_{h^+} e_{h^+} + g_{h^-} e_{h^-}]\Big|_0 = \frac{\partial g_{h^+}}{\partial \omega'}\Big|_0 \big(e_{h^+}(x) - e_{h^-}(x)\big)\big|_0 = 0,$$

and identically for β'. For the leaf parameters, $\partial f_\theta / \partial \mathbf{W}_{h^+} = g_{h^+}(x)\, x^\top$ and $\partial f_\theta / \partial b_{h^+} = g_{h^+}(x)$ are generically non-zero, and likewise for h^-. □

The regularizers $\mathcal{L}_s$ (Equation 6) and $\mathcal{L}_m$ (Equation 8) depend on (ω', β') through the split weights assigned to h^+ and h^-, so they provide a gradient signal available to the new gate at the first post-split optimizer step. Without them, the gate is updated only indirectly: the justification above shows that its task-loss gradient is proportional to $e_{h^+}(x) - e_{h^-}(x)$, which starts at zero and grows only as the children diverge under their own non-zero leaf gradients. Dropping the regularizers, removes a useful first-order signal that drives routing differentiation at the moment of expansion.

C. Experimental Details

In This section we go through dataset descriptions, baseline definitions, and execution details for the experimental campaign of Section 4.

C.1. Datasets

The reported campaign covers 15 datasets, partitioned into a synthetic regression block, a synthetic classification block, and a real-world tabular block. The dimensions and split sizes are reported in Table 4. Inputs are whitened using training-set statistics as described in Section 2. Regression targets are kept on their raw scale before metric computation, while classification targets are integer class labels and are evaluated by held-out accuracy.

Synthetic regression. The synthetic regression block contains *Sin*, *Sin*^{2}, *Ackley*, *Rastrigin*, and *Rosenbrock*. Sin and Sin^2 test high-frequency one-dimensional structure. The latter denotes the sine-of-squared-input target generated by the companion code. Ackley, Rastrigin, and Rosenbrock are optimization-derived functions repurposed as regression stress tests. Ackley mixes local oscillations with a slowly varying basin (Ackley, 1987); Rastrigin combines a quadratic envelope with a dense periodic lattice (Törn & Žilinskas, 1989); Rosenbrock concentrates error along an anisotropic curved valley (Rosenbrock, 1960). The two sinusoidal tasks are one-dimensional scalar regressions with 896/224/280 train/validation/test samples, while the three optimization-derived tasks are two-dimensional scalar regressions with 1408/352/440 samples.

Synthetic classification. The synthetic classification block contains *Moons*, *Spirals*, *3-Spirals*, *Pinwheel*, and *Checker*, which test piecewise decision logic, nested manifolds, and low-dimensional geometric support. Moons follows the standard two-dimensional two-class benchmark implemented in scikit-learn (Pedregosa et al., 2011); Pinwheel follows the rotating-arm construction used in structured generative-model benchmarks (Johnson et al., 2016). Spirals and 3-Spirals are two and three-class spiral manifolds generated synthetically, and Checker is a two-class checkerboard rule on the plane. All five tasks are two-dimensional; their class counts and split sizes are listed in Table 4.

Real-world tabular. The real-world block contains *Diabetes*, *California Housing*, *kin8nm*, *Breast Cancer*, and the cleaned *Sarcos* robotics regression. Diabetes is the 10-feature regression benchmark popularized with least-angle regression (Efron et al., 2004); California Housing is the 8-feature census block regression dataset derived from the spatial autoregression study of Pace & Barry (1997); kin8nm is the 8-feature robot-arm regression task distributed with the DELVE benchmark collection (Neal, 1996); and Breast Cancer is the 30-feature Wisconsin diagnostic binary classification dataset (Wolberg et al., 1993). Sarcos is a 21-input, 7-output inverse-dynamics regression task. The Sarcos dataset is structured over two different loader functions. The canonical loader preserves the original train/test files, while the cleaned loader first pools the original training and test matrices and then draws a fresh 80/10/10 train/validation/test split with the common split seed. This is the version denoted Sarcos* in the result tables. It avoids inheriting the predefined split and evaluates all Sarcos runs under the same split protocol used for the rest of the benchmark.

Dataset	Task / target	d_{in}	d_{out} / classes	Train / val / test	Total
Sin	regression, scalar	1	1	896 / 224 / 280	1,400
Sin^2	regression, scalar	1	1	896 / 224 / 280	1,400
Ackley	regression, scalar	2	1	1,408 / 352 / 440	2,200
Rastrigin	regression, scalar	2	1	1,408 / 352 / 440	2,200
Rosenbrock	regression, scalar	2	1	1,408 / 352 / 440	2,200
Moons	classification	2	2	1,024 / 256 / 320	1,600
Spirals	classification	2	2	1,536 / 384 / 480	2,400
3-Spirals	classification	2	3	1,728 / 432 / 540	2,700
Pinwheel	classification	2	5	1,024 / 256 / 320	1,600
Checker	classification	2	2	4,096 / 1,024 / 1,280	6,400
Diabetes	regression, scalar	10	1	282 / 71 / 89	442
CalHousing	regression, scalar	8	1	16,512 / 2,064 / 2,064	20,640
kin8nm	regression, scalar	8	1	6,552 / 820 / 820	8,192
BreastCnc	classification	30	2	364 / 91 / 114	569
Sarcos*	regression, vector	21	7	39,146 / 4,893 / 4,894	48,933

Table 4. Dataset dimensions and split sizes for the reported 15-dataset campaign.

C.2. Baselines

We compare SPAWN against a range of fixed-size and adaptive baselines, including MLPs, tree families architectures and soft-tree architecture.

Multi-layer perceptron (MLP). Standard fully-connected feed-forward network with ReLU activations, trained by back-propagation (Rumelhart et al., 1986) and swept over hidden width and depth.

Random forest (RForest). Bagged ensemble of axis-aligned decision trees (Breiman, 2001), swept over the number of trees and maximum depth.

Gradient-boosted trees (XGBoost). Stage-wise additive ensemble of regression trees with second-order gradient updates and shrinkage (Chen & Guestrin, 2016), swept over the number of trees and maximum depth.

Soft binary decision tree (SoftTr). Fixed-depth differentiable soft tree (Irsoy et al., 2012; Frosst & Hinton, 2017), included as the closest fixed-depth analogue of our soft-gated tree and a natural reference for the gating mechanism, swept over tree depth.

The capacity ladders swept for each family are listed in Appendix D (Table 5).

C.3. Hardware and Reproducibility

The full benchmark campaign was executed on CPU workers using an Intel Xeon Gold 6238R processor at 2.20 GHz, without GPU acceleration, with multiple seed and configuration runs scheduled in parallel. The wall-clock training times reported alongside the parameter counts in the main-text tables are not measured under this parallel scheduling, however, because contention for shared resources would inflate them in a way that is sensitive to cluster load rather than to the cost of the method itself. Instead, each reported training time is obtained from a separate timing campaign in which 3 seeds per model are executed sequentially on a single Apple M4 Pro with 12 CPU cores, using the timing script provided in the

repository.[3]

D. Baseline Ladders and Adaptive Hyperparameters

This section lists the swept capacity ladders for every reported baseline family used in the main-text comparisons of Sections 4.1 and 4.2, together with the fixed hyperparameter values used by the adaptive procedure across the entire benchmark.

All reported MLP results are drawn from the dense sweep. For readability, the best-of-family and per-model tables display the subset of dense-sweep MLPs listed in Table 5, while the parameter-efficiency comparison of Section 4.2 uses the full dense grid to identify same-budget, smallest-equivalent, and validation-best MLPs.

Family	Ladder
MLP (displayed subset of dense sweep)	$\mathrm{w32}_1$, $\mathrm{w64}_1$, $\mathrm{w128}_2$, $\mathrm{w256}_2$, $\mathrm{w512}_3$, $\mathrm{w512}_4$, $\mathrm{w1k}_4$
MLP (full dense sweep)	widths $\in \{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$, depths $\in \{1, 2, 3, 4\}$
RForest	$(T, d) \in \{(50, 4), (100, 8), (200, 12), (300, 16)\}$
XGBoost	$(T, d) \in \{(100, 3), (300, 4), (600, 6), (1000, 8)\}$
SoftTr	depth $\in \{2, 4, 8\}$

Table 5. Capacity ladders swept for each reported baseline family.

Adaptive hyperparameters. The adaptive procedure runs with the single fixed configuration of Table 6 on every dataset. Three groups of knobs determine the compute budget that the practitioner controls: an inner training-loss plateau detector, validation-monitored cycle and growth stoppers, and the hard cap on the number of expansion cycles. All other values, in particular the optimizer learning rate, the entropy-regularization coefficients λ_s and λ_m, and gradient clipping, are kept at the same defaults across the entire benchmark.

Group	Hyperparameter	Value
Inner stopper	rolling-difference window k	50
	rolling-difference threshold	10^{-2}
Cycle / growth stoppers	cycle patience	50
	validation patience	2500
	final-validation patience	200
	hard cycle cap $C_{\max}$	800
Optimization	adaptive-phase learning rate	10^{-2}
	fine-tuning learning rate	10^{-3}
	gradient-norm clip	1.0
Regularization	entropy weight $\lambda_s = \lambda_m$	10^{-2}
Splitting	expansion score	gate-weighted MAD (Eq. 10)

Table 6. Adaptive hyperparameter values used across the entire benchmark and across all reported variants (SPAWN-LL and SPAWN-QL). The inner stopper closes a cycle when local optimization stalls; the cycle and growth stoppers close cycles or terminate growth based on validation score; $C_{\max}$ is the hard upper bound on the number of expansion cycles.

E. Full Main-Text Result Tables

In this section we report all the experimental results obtained which were not present in the main text. Specifically, in Table 7 we report the results for the regression tasks; in Table 8 we report the results for the classification tasks. Finally in Tables 9 and 10 we report the parameter-efficiency results against the MLPs architecture on classification and regression tasks.

[3]The link to the repository is omitted to preserve anonymity and will be restored in the camera-ready version. A copy of the code is included in the supplementary materials.

	Baselines				SPAWN
Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Sin	0.299 ±0.001 526.8k ($w512_3$) / 1.7s	0.314 ±0.000 28.3k ($t100_8$) / 38ms	0.307 ±0.000 6.0k ($t300_4$) / 10ms	0.319 ±0.013 766 (d8) / 20.7s	**0.296 ±0.001** 192 / 23.7s
Sin^2	0.295 ±0.001 526.8k ($w512_3$) / 5.2s	0.337 ±0.001 156.4k ($t200_{12}$) / 92ms	0.304 ±0.000 7.6k ($t300_4$) / 14ms	0.582 ±0.003 766 (d8) / 25.1s	**0.290 ±0.001** 2.1k / 4.3m
Ackley	0.100 ±0.002 132.6k ($w256_3$) / 8.7s	0.128 ±0.000 509.8k ($t300_{16}$) / 320ms	**0.090 ±0.000** 55.7k ($t600_6$) / 96ms	0.212 ±0.001 1.0k (d8) / 39.1s	0.143 ±0.012 3.1k / 7.6m
Rastrigin	0.272 ±0.009 132.6k ($w256_3$) / 8.2s	0.390 ±0.001 487.7k ($t300_{16}$) / 333ms	**0.168 ±0.001** 59.3k ($t600_6$) / 100ms	0.651 ±0.001 61 (d4) / 1.8s	0.352 ±0.063 3.6k / 13.4m
Rosenbrock	**0.005 ±0.000** 4.1k ($w1k_1$) / 6.8s	0.071 ±0.001 388.4k ($t300_{16}$) / 358ms	0.065 ±0.001 143.2k ($t1k_8$) / 225ms	0.012 ±0.001 1.0k (d8) / 31.0s	0.010 ±0.000 512 / 48.3s
Diabetes	**0.669 ±0.009** 2.1M ($w1k_3$) / 1.2s	0.698 ±0.003 64.0k ($t200_{12}$) / 123ms	0.696 ±0.006 5.1k ($t300_4$) / 11ms	**0.667 ±0.011** 37 (d2) / 340ms	**0.670 ±0.009** 34 / 12.9s
CalHousing	0.456 ±0.003 10.2k ($w1k_1$) / 18.8s	0.444 ±0.000 3.1M ($t300_{16}$) / 14.2s	**0.387 ±0.001** 350.8k ($t1k_8$) / 1.2s	0.468 ±0.003 2.6k (d8) / 1.8m	0.470 ±0.010 715 / 7.3m
kin8nm	**0.255 ±0.003** 134.1k ($w256_3$) / 4.2s	0.538 ±0.001 1.9M ($t300_{16}$) / 5.2s	0.438 ±0.002 65.8k ($t600_6$) / 447ms	0.267 ±0.002 2.6k (d8) / 42.6s	0.260 ±0.003 677 / 2.4m
Sarcos*	**0.084 ±0.000** 1.1M ($w1k_2$) / 3.8m	0.158 ±0.000 7.0M ($t300_{16}$) / 1.9m	0.100 ±0.000 2.7M ($t1k_8$) / 42.5s	0.101 ±0.001 7.4k (d8) / 25.7m	**0.087 ±0.002** 40.3k / 2.2h

Table 7. Full best-of-family comparison on regression benchmarks. Top entry: held-out test RMSE mean ± 95% CI over seeds. Bottom entry: model size (red; parameters for neural models, nodes for tree models) and mean wall-clock training time (blue).

F. Per-Model Comparisons and Growth Trajectories

This section expands the reported baseline families across their displayed capacity ladders and traces each SPAWN variant along its growth trajectory. The main-text best-of-family table (Table 1) reports only the validation-selected run per family and dataset. The tables below show the full displayed ladders and adaptive checkpoints.

Tables 11–12 and 14–15 list neural baselines (MLP, SoftTr) and tree-based baselines (RForest, XGBoost) separately for readability. Tables 13 and 16 trace each SPAWN family at checkpoints labeled 0%, 33%, 67%, and 100% of the expansion budget; 0% corresponds to the first expansion cycle (E=2), while 33% and 67% are intermediate checkpoints and 100% is the full-growth checkpoint. Each table reports score (mean ± 95% C.I. over 20 seeds) followed by a row with model size. Bold marks the best displayed mean in each row and non-best entries not significantly different from it under a two-sided Welch test at $\alpha = 0.05$. The adaptive growth-trajectory tables additionally report mean wall-clock training time. The 0% column captures the few-experts regime, the 0% to 100% progression shows how growth refines the partition. Timing measurements are included only in the tables where they support the compute-efficiency discussion, because collecting comparable wall-clock times requires a separate sequential timing pass on an otherwise idle machine, as described in Appendix C.3. The full per-model baseline ladders therefore omit timings and focus on accuracy and model size.

G. Statistical Equivalence Procedure

The parameter-efficiency tables of Section 4.2 report, for each dataset, the smallest MLP run whose accuracy is statistically equivalent or superior to that of the validation-selected SPAWN run, the same-budget MLP run, and the validation-best MLP. This section specifies the procedure used to perform the analysis.

Inputs. For each dataset and each MLP run, the campaign script computes a per-seed validation score (RMSE for regression, accuracy for classification) on the same held-out validation split used for model selection throughout the paper. The same per-seed validation scores are computed for the validation-selected SPAWN run on the same dataset. When the seed pools of the two compared runs coincide, the comparison is paired across seeds, otherwise it falls back to a two-sample comparison.

Equivalence test. For each (dataset, MLP run) pair we apply two-one-sided tests (TOST) at $\alpha = 0.05$ on validation scores. The equivalence margin is set to one half of the standard deviation of the SPAWN run's validation scores on that dataset, so the margin is itself dataset-adaptive and tracks the natural seed-level variability of the adaptive procedure. The

	Baselines				SPAWN
Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Moons	97.3 ±0.1 2.1M (w1k$_3$) / 3.0s	96.6 ±0.1 7.4k (t100$_8$) / 45ms	**97.5 ±0.0** 3.8k (t300$_4$) / 12ms	96.9 ±0.1 77 (d4) / 5.0s	96.9 ±0.0 49 / 15.1s
Spirals	**100.0 ±0.0** 527.9k (w512$_3$) / 5.7s	98.8 ±0.1 88.8k (t300$_{16}$) / 251ms	98.6 ±0.1 40.4k (t1k$_8$) / 89ms	76.3 ±1.3 1.3k (d8) / 30.5s	99.3 ±0.4 456 / 1.3m
3-Spirals	**100.0 ±0.0** 791.0k (w512$_4$) / 13.6s	97.0 ±0.2 86.5k (t300$_{16}$) / 384ms	97.6 ±0.1 143.9k (t1k$_8$) / 335ms	91.5 ±1.5 1.5k (d8) / 51.9s	98.9 ±0.2 645 / 1.6m
Pinwheel	94.4 ±0.2 1.1M (w1k$_2$) / 3.4s	**95.2 ±0.1** 30.0k (t200$_{12}$) / 106ms	95.0 ±0.0 22.6k (t300$_4$) / 55ms	94.5 ±0.2 2.0k (d8) / 28.9s	**95.0 ±0.2** 287 / 20.1s
Checker	89.8 ±0.3 8.6k (w64$_3$) / 4.9s	89.6 ±0.1 296.6k (t300$_{16}$) / 368ms	**91.0 ±0.1** 60.4k (t1k$_8$) / 72ms	85.6 ±0.5 1.3k (d8) / 1.9m	90.3 ±0.2 1.2k / 4.4m
BreastCnc	95.7 ±0.3 338 (w8$_2$) / 258ms	95.3 ±0.2 969 (t50$_4$) / 23ms	94.1 ±0.2 2.0k (t300$_4$) / 42ms	**97.0 ±0.5** 8.4k (d8) / 3.6s	95.0 ±0.6 155 / 19.4s

Table 8. Full best-of-family comparison on classification benchmarks. Top entry: held-out test accuracy in percent mean ± 95% CI over seeds. Bottom entry: model size (red; parameters for neural models, nodes for tree models) and mean wall-clock training time (blue).

pairing rule above selects the per-comparison test: a paired-sample t test when seeds align, and Welch's two-sample t test otherwise.

Classification of an MLP run. A run is classified as *equivalent* if the TOST decision rejects the null of nonequivalence at $\alpha = 0.05$. Independently, the ordinary two-sided difference test is applied at the same level: if the difference test rejects in favor of the MLP, the run is classified as *MLP-superior*; if it rejects in favor of SPAWN, the run is classified as *adaptive-superior*. The smallest-equivalent column reports the minimum-parameter run classified as either equivalent or MLP-superior. A dash entry indicates that no run in the swept dense MLP ladder of Appendix D reached either classification on that dataset.

Same-budget run. The same-budget run is selected as the closest MLP run to the SPAWN parameter count, requiring the relative parameter difference to fall within a 5% window. When no run lies within that window, the procedure falls back to the smallest run whose parameter count exceeds the SPAWN count, so the comparison remains conservative for the adaptive run.

Reported scores. Equivalence and selection decisions all use validation scores. The numbers displayed in Table 2 are the corresponding held-out test scores on the same seeds, following standard practice of separating the model-selection signal from the reported metric.

H. Qualitative Behavior: Implicit Regularization

This section accompanies the main-text claim that adaptive growth produces predictions that follow the data manifold without spending capacity in low-density regions of the input space. We illustrate the effect on the synthetic low-dimensional benchmarks from the regression and classification suites, comparing the validation-selected SPAWN variants against the MLP runs selected by the parameter-efficiency protocol of Section 4.2.

The grids below extend that comparison to all low-dimensional datasets for which prediction geometry can be plotted. Rows follow the same dataset order as the MLP parameter-efficiency table, while figures within each row show the selected SPAWN-LL and SPAWN-QL variants, the same-size MLP, and the smallest-equivalent MLP from Table 2. When no smallest-equivalent MLP exists under the equivalence test of Appendix G, the final figure uses the validation-best MLP instead. Tabular benchmarks are intentionally omitted because their input geometry is not directly visualizable.

Implicit regularization through partition growth. Figures 5 and 6 reflects the architectural prior of Section 3. A leaf is split only when its residuals indicate that one affine predictor is no longer adequate, so the model stays smooth on noisy targets, because local experts are not asked to explain arbitrary high-frequency noise, and stays sharp on geometric targets, because the partition refines where the residual signal is localized. A fixed MLP spreads its parameters uniformly across the input space, while SPAWN allocates them regionally.

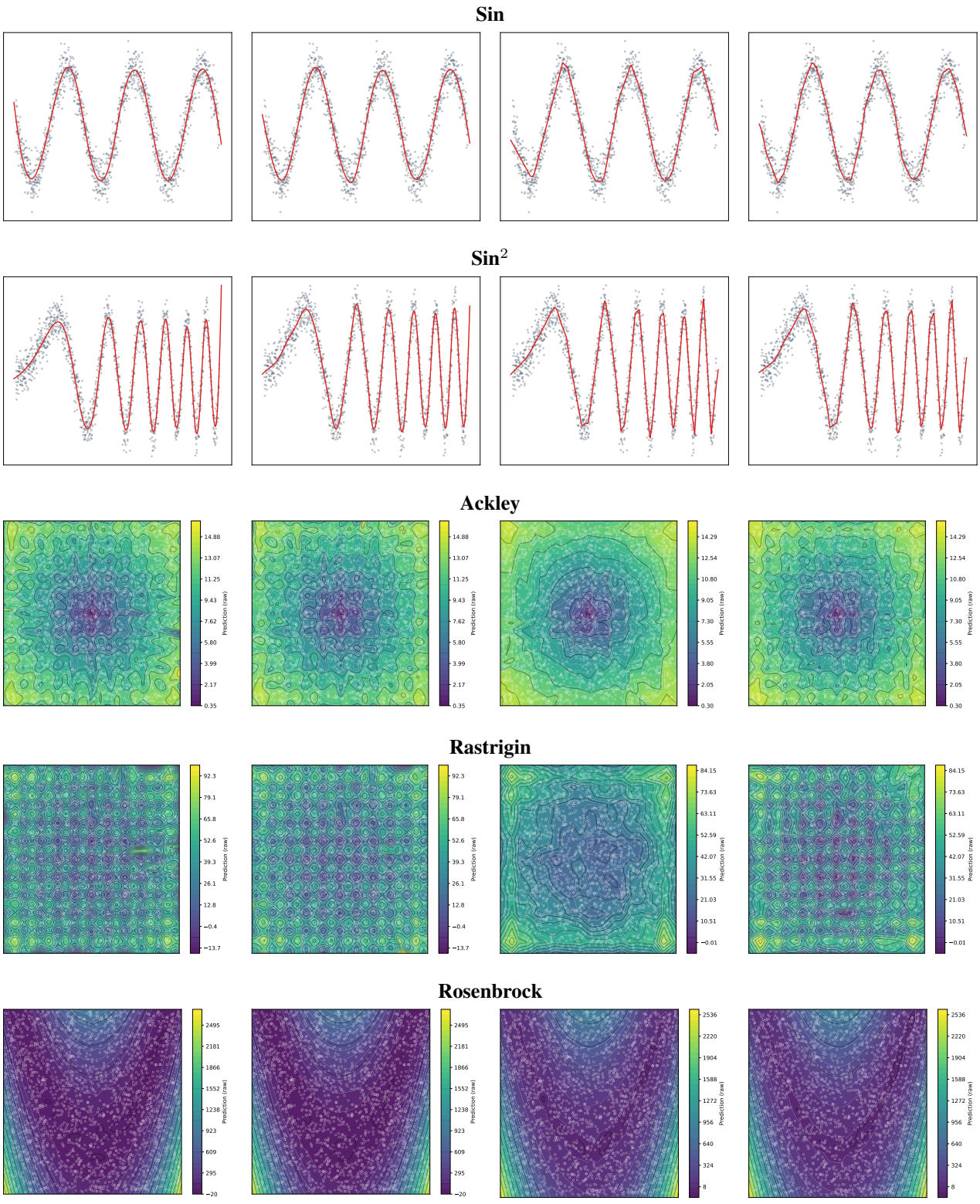

Figure 5. Qualitative prediction shapes on low-dimensional regression benchmarks. Within each dataset row, panels from left to right show: (a) SPAWN-LL, (b) SPAWN-QL, (c) the same-size MLP, and (d) the smallest-equivalent MLP; for Sin and Sin^2, where no smallest-equivalent MLP exists, panel (d) shows the validation-best MLP.

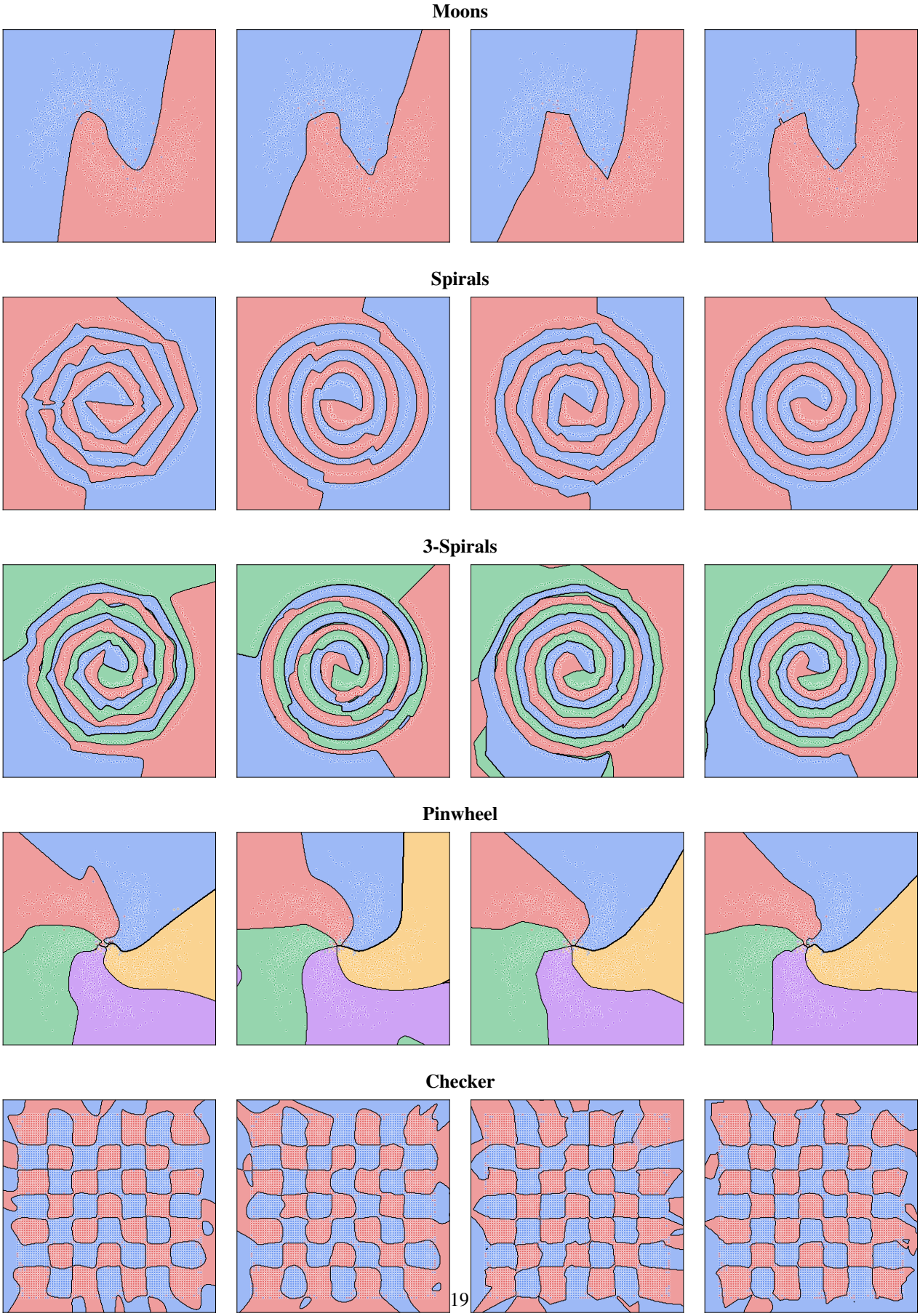

Figure 6. Qualitative decision regions on low-dimensional classification benchmarks. Within each dataset row, panels from left to right show: (a) SPAWN-LL, (b) SPAWN-QL, (c) the same-size MLP, and (d) the smallest-equivalent MLP; for Checker, where no smallest-equivalent MLP exists, panel (d) shows the validation-best MLP.

	SPAWN	**MLP**		
Dataset	SPAWN	same size	smallest equiv.	best observed
Sin	**0.296** ±**0.001** 192 / 23.7s	0.326 ±0.016 193 ($w64_1$) / 3.4s	– –	0.299 ±0.001 526.8k ($w512_3$) / 1.7s
Sin^2	**0.290** ±**0.001** 2.1k / 4.3m	0.436 ±0.046 2.2k ($w32_3$) / 3.3s	– –	0.295 ±0.001 526.8k ($w512_3$) / 5.2s
Ackley	0.143 ±0.012 3.1k / 7.6m	0.194 ±0.004 3.3k ($w32_4$) / 4.9s	0.116 ±0.004 33.5k ($w128_3$) / 6.7s	**0.100** ±**0.002** 132.6k ($w256_3$) / 8.7s
Rastrigin	0.352 ±0.063 3.6k / 13.4m	0.607 ±0.006 4.1k ($w1k_1$) / 6.1s	0.326 ±0.017 66.8k ($w256_2$) / 4.4m	**0.272** ±**0.009** 132.6k ($w256_3$) / 8.2s
Rosenbrock	0.010 ±0.000 512 / 48.3s	0.010 ±0.001 513 ($w128_1$) / 5.6s	0.006 ±0.000 1.0k ($w256_1$) / 5.6s	**0.005** ±**0.000** 4.1k ($w1k_1$) / 6.8s
Diabetes	**0.670** ±**0.009** 34 / 12.9s	0.727 ±0.043 37 ($w2_3$) / 356ms	**0.675** ±**0.015** 51.1k ($w128_4$) / 195ms	**0.669** ±**0.009** 2.1M ($w1k_3$) / 1.2s
CalHousing	0.470 ±0.010 715 / 7.3m	**0.447** ±**0.004** 705 ($w16_3$) / 27.9s	**0.449** ±**0.003** 433 ($w16_2$) / 33.7s	0.456 ±0.003 10.2k ($w1k_1$) / 18.8s
kin8nm	0.260 ±0.003 677 / 2.4m	0.281 ±0.004 705 ($w16_3$) / 10.8s	– –	**0.255** ±**0.003** 134.1k ($w256_3$) / 4.2s
Sarcos*	**0.087** ±**0.002** 40.3k / 2.2h	0.094 ±0.000 53.3k ($w128_4$) / 17.0m	0.085 ±0.000 277.5k ($w512_2$) / 2.4h	**0.084** ±**0.000** 1.1M ($w1k_2$) / 3.8m

Table 9. Full MLP parameter-efficiency comparison on regression benchmarks. Displayed scores are the RMSE. The row below each score reports model size in red, with selected MLP model IDs in parentheses, followed by mean wall-clock training time in blue.

	SPAWN	**MLP**		
Dataset	SPAWN	same size	smallest equiv.	best observed
Moons	96.9 ±0.0 49 / 15.1s	**92.8** ±**5.0** 62 ($w4_3$) / 2.0s	**97.2** ±**0.2** 898 ($w16_4$) / 1.6s	**97.3** ±**0.1** 2.1M ($w1k_3$) / 3.0s
Spirals	99.3 ±0.4 456 / 1.3m	**97.2** ±**2.8** 626 ($w16_3$) / 5.6s	**100.0** ±**0.0** 8.6k ($w64_3$) / 6.2s	**100.0** ±**0.0** 527.9k ($w512_3$) / 5.7s
3-Spirals	98.9 ±0.2 645 / 1.6m	**97.7** ±**2.4** 643 ($w16_3$) / 7.0s	99.9 ±0.1 2.3k ($w32_3$) / 5.1s	**100.0** ±**0.0** 791.0k ($w512_4$) / 13.6s
Pinwheel	**95.0** ±**0.2** 287 / 20.1s	**95.0** ±**0.3** 285 ($w8_4$) / 2.0s	94.6 ±0.1 1.0k ($w128_1$) / 3.1s	94.4 ±0.2 1.1M ($w1k_2$) / 3.4s
Checker	**90.3** ±**0.2** 1.2k / 4.4m	87.8 ±0.8 1.2k ($w32_2$) / 16.4s	– –	89.8 ±0.3 8.6k ($w64_3$) / 4.9s
BreastCnc	**95.0** ±**0.6** 155 / 19.4s	**95.5** ±**0.4** 154 ($w4_2$) / 424ms	**95.7** ±**0.5** 8.4k ($w256_1$) / 185ms	**95.7** ±**0.3** 338 ($w8_2$) / 258ms

Table 10. MLP parameter-efficiency comparison on classification benchmarks. Displayed scores are test accuracy in percent. The row below each score reports model size in red, with selected MLP model IDs in parentheses, followed by mean wall-clock training time in blue.

	MLP							SoftTr		
Dataset	$w32_1$	$w64_1$	$w128_2$	$w256_2$	$w512_3$	$w512_4$	$w1k_4$	d2	d4	d8
Sin	0.356 ±0.021 97	0.326 ±0.016 193	**0.299 ±0.001** 16.9k	**0.298 ±0.001** 66.6k	**0.299 ±0.001** 526.8k	0.301 ±0.001 789.5k	**0.301 ±0.003** 3.2M	0.849 ±0.027 10	0.372 ±0.004 46	0.319 ±0.013 766
Sin^2	0.709 ±0.033 97	0.614 ±0.021 193	0.349 ±0.036 16.9k	0.313 ±0.008 66.6k	**0.295 ±0.001** 526.8k	**0.312 ±0.032** 789.5k	0.389 ±0.064 3.2M	0.964 ±0.002 10	0.920 ±0.051 46	0.582 ±0.003 766
Ackley	0.218 ±0.003 129	0.212 ±0.001 257	0.182 ±0.006 17.0k	0.133 ±0.006 66.8k	**0.125 ±0.004** 527.4k	**0.123 ±0.004** 790.0k	0.132 ±0.005 3.2M	0.742 ±0.006 13	0.214 ±0.001 61	0.212 ±0.001 1.0k
Rastrigin	0.654 ±0.002 129	0.654 ±0.002 257	0.469 ±0.047 17.0k	**0.326 ±0.017** 66.8k	**0.315 ±0.017** 527.4k	**0.325 ±0.017** 790.0k	**0.337 ±0.013** 3.2M	0.811 ±0.002 13	0.651 ±0.001 61	0.655 ±0.002 1.0k
Rosenbrock	0.037 ±0.003 129	0.018 ±0.001 257	**0.007 ±0.000** 17.0k	**0.006 ±0.000** 66.8k	0.007 ±0.000 527.4k	0.008 ±0.001 790.0k	**0.045 ±0.076** 3.2M	0.565 ±0.064 13	0.037 ±0.002 61	0.012 ±0.001 1.0k
Diabetes	0.673 ±0.011 385	**0.672 ±0.013** 769	**0.653 ±0.011** 18.0k	**0.656 ±0.019** 68.9k	**0.660 ±0.014** 531.5k	0.691 ±0.019 794.1k	**0.668 ±0.015** 3.2M	**0.667 ±0.011** 37	**0.664 ±0.006** 181	0.673 ±0.004 3.1k
CalHousing	0.464 ±0.001 321	0.457 ±0.002 641	**0.452 ±0.003** 17.8k	**0.454 ±0.004** 68.4k	**0.455 ±0.004** 530.4k	**0.456 ±0.003** 793.1k	0.461 ±0.003 3.2M	0.526 ±0.004 31	0.470 ±0.002 151	0.468 ±0.003 2.6k
kin8nm	0.321 ±0.003 321	0.296 ±0.004 641	**0.259 ±0.002** 17.8k	**0.256 ±0.002** 68.4k	**0.258 ±0.003** 530.4k	**0.259 ±0.002** 793.1k	0.265 ±0.004 3.2M	0.597 ±0.008 31	0.334 ±0.005 151	0.267 ±0.002 2.6k
Sarcos*	0.182 ±0.001 935	0.154 ±0.001 1.9k	0.099 ±0.000 20.2k	**0.087 ±0.000** 73.2k	0.088 ±0.001 540.2k	0.090 ±0.001 802.8k	**0.171 ±0.157** 3.2M	0.447 ±0.000 94	0.208 ±0.001 442	0.101 ±0.001 7.4k

Table 11. Per-model comparison on regression—neural baselines (MLP and soft decision trees). Each top entry reports held-out test RMSE (mean ± 95% CI); the row below reports model size in red.

	RForest				XGBoost			
Dataset	$t50_4$	$t100_8$	$t200_{12}$	$t300_{16}$	$t100_3$	$t300_4$	$t600_6$	$t1k_8$
Sin	0.551 ±0.006 1.5k	0.314 ±0.000 28.3k	0.336 ±0.000 163.0k	0.349 ±0.000 325.0k	0.323 ±0.000 1.5k	**0.307 ±0.000** 6.0k	0.309 ±0.000 11.2k	0.313 ±0.000 20.6k
Sin^2	0.698 ±0.001 1.6k	0.478 ±0.002 31.6k	0.337 ±0.001 156.4k	0.343 ±0.000 317.9k	0.445 ±0.002 1.5k	**0.304 ±0.000** 7.6k	0.307 ±0.000 15.0k	0.309 ±0.000 22.5k
Ackley	0.379 ±0.003 1.6k	0.180 ±0.001 37.7k	0.134 ±0.000 244.8k	0.128 ±0.000 509.8k	0.192 ±0.001 1.5k	0.107 ±0.001 9.0k	**0.090 ±0.000** 55.7k	0.101 ±0.000 196.1k
Rastrigin	0.627 ±0.001 1.5k	0.481 ±0.001 35.2k	0.408 ±0.001 231.0k	0.390 ±0.001 487.7k	0.477 ±0.002 1.5k	0.198 ±0.001 9.2k	**0.168 ±0.001** 59.3k	0.223 ±0.001 208.8k
Rosenbrock	0.457 ±0.004 1.5k	0.186 ±0.002 29.5k	0.093 ±0.001 165.5k	0.071 ±0.001 388.4k	0.269 ±0.002 1.5k	0.087 ±0.002 8.3k	**0.065 ±0.001** 49.6k	**0.065 ±0.001** 143.2k
Diabetes	0.687 ±0.003 1.5k	0.698 ±0.003 18.8k	0.698 ±0.003 64.0k	0.698 ±0.002 103.7k	**0.679 ±0.004** 1.5k	0.696 ±0.006 5.1k	0.712 ±0.004 12.0k	0.731 ±0.005 19.4k
CalHousing	0.636 ±0.001 1.6k	0.512 ±0.001 46.0k	0.456 ±0.000 686.4k	0.444 ±0.000 3.1M	0.509 ±0.001 1.5k	0.429 ±0.001 9.2k	0.389 ±0.001 69.0k	**0.387 ±0.001** 350.8k
kin8nm	0.755 ±0.001 1.6k	0.614 ±0.001 48.3k	0.553 ±0.001 626.8k	0.538 ±0.001 1.9M	0.686 ±0.001 1.5k	0.564 ±0.002 9.1k	0.438 ±0.002 65.8k	**0.433 ±0.001** 321.8k
Sarcos*	0.598 ±0.000 1.6k	0.367 ±0.000 51.1k	0.218 ±0.000 1.2M	0.158 ±0.000 7.0M	0.305 ±0.000 10.5k	0.190 ±0.000 64.7k	0.121 ±0.000 491.1k	**0.100 ±0.000** 2.7M

Table 12. Per-model comparison on regression—tree-based baselines (random forests and XGBoost). Each top entry reports held-out test RMSE (mean ± 95% CI); the row below reports model size in red.

	SPAWN-LL				SPAWN-QL			
Dataset	0%	33%	67%	100%	0%	33%	67%	100%
Sin	0.937 ±0.018 6 / 518ms	0.298 ±0.001 86 / 10.4s	0.301 ±0.001 398 / 18.7s	0.296 ±0.001 192 / 23.7s	0.857 ±0.001 7 / 811ms	0.298 ±0.000 112 / 7.7s	0.300 ±0.000 217 / 13.7s	**0.295 ±0.000** 125 / 20.1s
Sin^2	0.960 ±0.000 6 / 191ms	0.342 ±0.004 438 / 49.6s	0.289 ±0.001 874 / 2.2m	0.290 ±0.001 2.1k / 4.3m	0.946 ±0.001 7 / 507ms	0.290 ±0.002 177 / 17.1s	**0.291 ±0.000** 497 / 23.7s	**0.288 ±0.001** 469 / 41.2s
Ackley	0.737 ±0.005 9 / 413ms	0.192 ±0.006 639 / 14.8s	0.197 ±0.009 663 / 28.2s	0.143 ±0.012 3.1k / 7.6m	0.248 ±0.000 11 / 1.8s	0.175 ±0.008 795 / 51.5s	**0.147 ±0.006** 1.0k / 2.2m	**0.105 ±0.005** 2.4k / 4.0m
Rastrigin	0.805 ±0.002 9 / 379ms	0.572 ±0.024 993 / 2.6m	0.395 ±0.049 2.0k / 9.1m	0.352 ±0.063 3.6k / 13.4m	0.654 ±0.001 11 / 428ms	0.420 ±0.042 907 / 3.0m	**0.151 ±0.014** 1.2k / 10.2m	**0.153 ±0.012** 3.8k / 14.5m
Rosenbrock	0.745 ±0.009 9 / 860ms	0.016 ±0.001 141 / 27.4s	0.013 ±0.000 279 / 45.4s	**0.010 ±0.000** 512 / 48.3s	0.352 ±0.027 11 / 1.2s	0.018 ±0.002 155 / 22.0s	**0.079 ±0.009** 299 / 35.0s	0.012 ±0.001 235 / 29.5s
Diabetes	0.688 ±0.008 33 / 249ms	2.131 ±0.170 407 / 6.8s	2.148 ±0.147 759 / 9.7s	**0.670 ±0.009** 34 / 12.9s	0.701 ±0.015 43 / 232ms	2.183 ±0.166 587 / 6.2s	2.149 ±0.171 1.1k / 9.1s	**0.679 ±0.009** 43 / 12.3s
CalHousing	0.568 ±0.014 27 / 4.3s	**0.480 ±0.007** 441 / 2.3m	**0.461 ±0.013** 855 / 5.0m	**0.470 ±0.010** 715 / 7.3m	0.527 ±0.004 35 / 4.7s	**0.461 ±0.006** 555 / 2.6m	**0.473 ±0.130** 1.1k / 5.3m	**0.467 ±0.014** 702 / 5.3m
kin8nm	0.589 ±0.009 27 / 1.6s	**0.260 ±0.003** 477 / 51.2s	0.262 ±0.004 909 / 1.6m	**0.260 ±0.003** 677 / 2.4m	0.574 ±0.010 35 / 1.7s	0.266 ±0.003 607 / 53.3s	0.282 ±0.005 1.2k / 1.6m	0.267 ±0.003 629 / 1.8m
Sarcos*	0.262 ±0.001 330 / 8.2s	0.102 ±0.001 19.2k / 22.7m	0.098 ±0.001 23.9k / 1.2h	**0.087 ±0.002** 40.3k / 2.2h	0.259 ±0.002 351 / 8.5s	0.111 ±0.001 11.6k / 13.6m	0.106 ±0.001 23.0k / 38.7m	0.090 ±0.001 38.8k / 52.8m

Table 13. SPAWN growth trajectory on regression benchmarks. Each top entry reports RMSE (mean ± 95% CI); the row below reports model size in red followed by mean wall-clock training time in blue.

	MLP							SoftTr		
Dataset	w32_1	w64_1	w128_2	w256_2	w512_3	w512_4	w1k_4	d2	d4	d8
Moons	97.0 ±0.1 162	97.0 ±0.1 322	**97.3 ±0.2** 17.2k	**97.5 ±0.2** 67.1k	**97.3 ±0.1** 527.9k	97.0 ±0.2 790.5k	97.0 ±0.2 3.2M	94.5 ±1.8 17	96.9 ±0.1 77	97.1 ±0.1 1.3k
Spirals	58.9 ±1.0 162	64.0 ±1.6 322	**100.0 ±0.0** 17.2k	**100.0 ±0.0** 67.1k	**100.0 ±0.0** 527.9k	**100.0 ±0.0** 790.5k	**99.2 ±0.9** 3.2M	53.4 ±0.5 17	57.1 ±0.5 77	76.3 ±1.3 1.3k
3-Spirals	45.1 ±1.8 195	54.2 ±1.6 387	**100.0 ±0.0** 17.3k	**100.0 ±0.0** 67.3k	**100.0 ±0.0** 528.4k	**100.0 ±0.0** 791.0k	**99.7 ±0.5** 3.2M	36.1 ±0.8 21	44.0 ±1.9 93	91.5 ±1.5 1.5k
Pinwheel	**95.1 ±0.1** 261	94.9 ±0.1 517	94.5 ±0.2 17.5k	94.6 ±0.1 67.8k	**94.9 ±0.3** 529.4k	**95.0 ±0.3** 792.1k	94.7 ±0.3 3.2M	90.2 ±0.3 29	**95.0 ±0.2** 125	94.5 ±0.2 2.0k
Checker	68.1 ±1.8 162	78.7 ±1.3 322	**90.3 ±0.3** 17.2k	**89.7 ±0.6** 67.1k	**89.5 ±0.7** 527.9k	**87.9 ±2.2** 790.5k	86.4 ±2.5 3.2M	51.4 ±0.3 17	51.7 ±2.1 77	85.6 ±0.5 1.3k
BreastCnc	95.4 ±0.4 1.1k	95.7 ±0.3 2.1k	95.6 ±0.4 20.7k	95.4 ±0.4 74.2k	95.6 ±0.7 542.2k	95.2 ±0.4 804.9k	95.0 ±0.5 3.2M	95.9 ±0.2 101	**96.3 ±0.3** 497	**97.0 ±0.5** 8.4k

Table 14. Per-model comparison on classification—neural baselines (MLP and soft decision trees). Each top entry reports the held-out accuracy in percent (mean ± 95% CI); the row below reports model size in red.

	RForest				XGBoost			
Dataset	$t50_4$	$t100_8$	$t200_{12}$	$t300_{16}$	$t100_3$	$t300_4$	$t600_6$	$t1k_8$
Moons	93.5 ±0.4 1.1k	96.6 ±0.1 7.4k	96.5 ±0.1 18.8k	96.4 ±0.1 28.5k	**97.8 ±0.0** 1.1k	97.5 ±0.0 3.8k	97.4 ±0.1 4.9k	96.9 ±0.1 5.7k
Spirals	64.3 ±0.9 817	88.3 ±0.6 7.2k	97.8 ±0.2 38.4k	**98.8 ±0.1** 88.8k	78.2 ±0.8 1.1k	91.5 ±0.2 6.1k	98.1 ±0.1 23.3k	98.6 ±0.1 40.4k
3-Spirals	45.9 ±1.3 658	72.5 ±1.0 5.2k	89.9 ±0.4 29.2k	97.0 ±0.2 86.5k	68.4 ±0.6 3.1k	86.2 ±0.3 17.5k	97.2 ±0.1 76.1k	**97.6 ±0.1** 143.9k
Pinwheel	92.9 ±0.4 1.4k	**95.7 ±0.2** 10.8k	95.2 ±0.1 30.0k	95.2 ±0.1 46.3k	**95.5 ±0.1** 5.9k	95.0 ±0.0 22.6k	95.1 ±0.1 27.1k	95.1 ±0.1 30.1k
Checker	61.5 ±1.2 1.5k	83.7 ±0.4 24.6k	89.6 ±0.1 139.2k	89.6 ±0.1 296.6k	71.4 ±0.7 1.5k	85.6 ±0.4 8.4k	**91.3 ±0.1** 38.9k	91.0 ±0.1 60.4k
BreastCnc	**95.3 ±0.2** 969	**95.3 ±0.2** 2.8k	**95.2 ±0.2** 5.7k	**95.1 ±0.2** 8.5k	94.1 ±0.2 1.0k	94.1 ±0.2 2.0k	94.2 ±0.2 2.0k	94.2 ±0.2 2.0k

Table 15. Per-model comparison on classification—tree-based baselines (radonm forests and XGBoost). Each top entry reports the held-out accuracy in percent (mean ± 95% CI); the row below reports model size in red.

	SPAWN-LL				SPAWN-QL			
Dataset	0%	33%	67%	100%	0%	33%	67%	100%
Moons	88.1 ±0.0 15 / 295ms	**96.9 ±0.1** 177 / 5.6s	96.5 ±0.2 330 / 10.1s	**96.9 ±0.0** 49 / 15.1s	91.2 ±2.0 17 / 758ms	96.6 ±0.2 204 / 7.6s	95.9 ±0.2 391 / 17.4s	**97.0 ±0.2** 101 / 21.7s
Spirals	54.3 ±0.9 15 / 411ms	68.6 ±4.0 267 / 14.8s	99.2 ±0.3 528 / 1.1m	**99.3 ±0.4** 456 / 1.3m	51.7 ±0.3 17 / 393ms	**99.9 ±0.1** 226 / 51.0s	**100.0 ±0.1** 424 / 1.3m	**99.9 ±0.1** 545 / 1.4m
3-Spirals	36.2 ±0.9 21 / 388ms	93.0 ±6.6 369 / 42.7s	99.1 ±0.3 705 / 1.3m	98.9 ±0.2 645 / 1.6m	39.3 ±0.3 23 / 461ms	99.7 ±0.1 289 / 34.0s	99.4 ±0.1 555 / 49.1s	**100.0 ±0.1** 183 / 1.0m
Pinwheel	92.9 ±0.3 33 / 2.4s	94.8 ±0.2 357 / 8.7s	94.6 ±0.3 699 / 14.5s	**95.0 ±0.2** 287 / 20.1s	94.3 ±0.3 35 / 3.3s	**95.0 ±0.2** 375 / 8.0s	95.3 ±0.3 695 / 15.8s	**95.4 ±0.3** 158 / 21.5s
Checker	49.3 ±0.4 15 / 804ms	87.2 ±0.4 501 / 1.2m	**89.8 ±0.2** 996 / 2.8m	**90.3 ±0.2** 1.2k / 4.4m	48.3 ±0.4 17 / 873ms	88.9 ±0.4 391 / 1.1m	89.7 ±0.2 776 / 2.0m	89.7 ±0.2 809 / 3.1m
BreastCnc	94.3 ±0.8 155 / 1.7s	94.3 ±0.9 1.7k / 14.4s	94.3 ±0.9 3.2k / 16.6s	**95.0 ±0.6** 155 / 19.4s	93.8 ±0.8 185 / 1.9s	93.6 ±0.7 2.3k / 12.7s	93.6 ±0.7 4.2k / 14.9s	**95.5 ±0.5** 185 / 18.1s

Table 16. SPAWN growth trajectory on classification benchmarks. Each top entry reports accuracy in percent (mean ± 95% CI); the row below reports model size in red followed by mean wall-clock training time in blue.