



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Adaptive Self-Growing Neural Networks through Differentiable Input-Space Partitioning

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA INFORMATICA

Federico Lolli, 10720843

Advisor:

Prof. Marcello Restelli

Co-advisors:

Dott. Gianmarco Tedeschi

Prof. Matteo Papini

Academic year:

2025-2026

Abstract: The design of neural architectures is commonly treated as a process separate from parameter optimization, typically relying on human expertise or automated search. Still, manual design requires domain knowledge that does not transfer across tasks, while automated search methods often remain computationally prohibitive. 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.

Key-words: Supervised Learning, Machine Learning, Growing Network, Adaptive Methods, Neural Architecture, MLP

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, 11], network pruning [26], and evolutionary algorithms [37] 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 adjust its capacity during the training process. This concept has proven effective in several settings, including continual learning [34], optimization [7], and reinforcement learning [2, 39]. 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 [9, 12, 45].

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 [22] or gradient boosting [8], 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 the central research question: *Can the optimal architecture be shaped during the optimization process, starting from a minimal configuration, and expanding without exceeding the necessary dimensionality or relying on prior domain knowledge?*

This work presents *Self-Partitioning Adaptive Widening Network* (SPAWN), a novel neural architecture that steps away from the standard deep learning growth paradigm. Instead of expanding a neural network by adding hidden units or layers, SPAWN dynamically grows a differentiable, shallow tree of experts. Crucially, the fundamental unit of growth is a partition of the input space, rather than an isolated parameter. By dynamically partitioning the data space based on task feedback, 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 function over its newly generated sub-region. Because the model’s global non-linear 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 iteration. The resulting process produces architectures whose capacity is set by the data rather than by the practitioner.

Original Contributions. This work introduces *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. The main contributions are: (i) a differentiable tree-of-experts architecture whose capacity is determined by the data rather than by the practitioner; (ii) a principled expansion procedure that decides *when*, *where*, and *how* to grow, while preserving the network output at every iteration; and (iii) an empirical evaluation showing that the resulting architectures match the predictive accuracy of fixed-size MLP baselines and adaptive tree-based approaches while using substantially fewer parameters.

Outline. The remainder of this thesis is organized as follows. Section 2 starts by formalizing the soft-gated tree architecture and its training objective while Section 3 develops the adaptive expansion procedure, detailing the data-driven criteria that guide the growth of the network. An empirical evaluation against both fixed-capacity and growing-network baselines on regression and classification benchmarks follows in Section 4. Section 5 frames the method within the literature of neural architecture search, network growing, and mixture-of-experts models. Ending with Section 6, drawing conclusions and outlines directions for future work.

2. Architecture

SPAWN learns a predictor $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^p$ 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 features¹ and y_i taking values in $\mathbb{R}^p$ for regression or in a finite label set for classification. The model is constructed as a mixture of E experts, partitioned across the input space by $G = E - 1$ soft gates arranged in a binary tree. Each expert is responsible for a region of the input space, so that capacity is allocated locally rather than spread uniformly.

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

¹Inputs 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, and the tilde is dropped throughout.

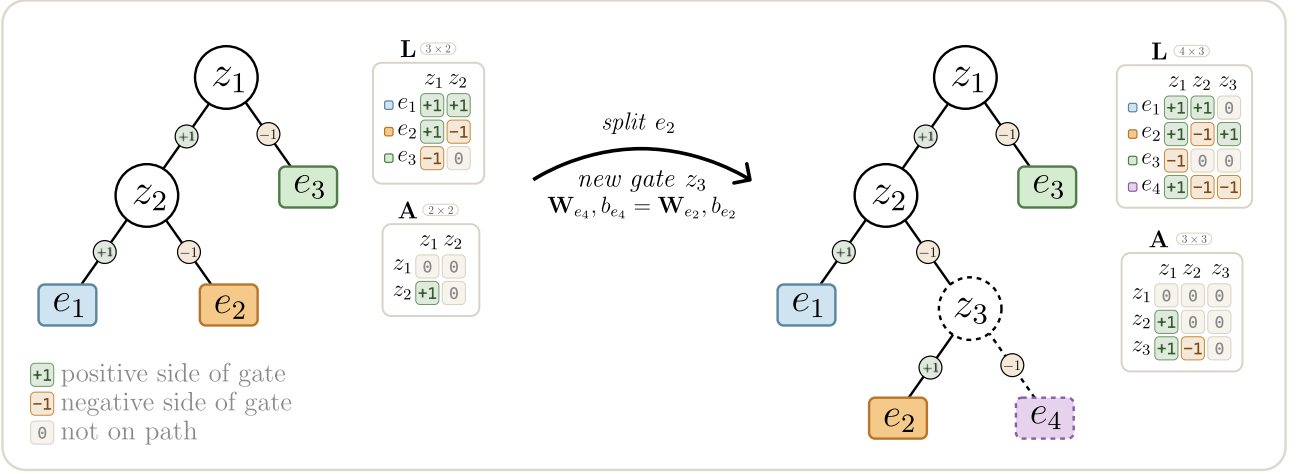


Figure 1: Expansion of SPAWN soft-gated binary tree (left to right).

$e_h(x) \in \mathbb{R}^p$ denote the expert output. The model prediction is the gating-weighted combination of all expert outputs:

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

The architecture is presented in a fixed- E setting for clarity. While each component works for any fixed number of experts, the design is intentionally tailored to support the adaptive growth procedure introduced in Section 3. Every structural choice is conceived so that new experts can be inserted without disrupting existing routing or changing the model output. Section 2.1 details the soft-gated tree that partitions the input space and delegates to the experts, and Section 2.2 introduces the entropy-based regularizers that keep routing well-behaved during training.

2.1. Soft-Gated Tree of Experts

The predictor f_θ combines E 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 re-partitions only its parent’s region, so expansion is naturally contained and preserves the routing of existing experts. This follows the hierarchical MoE tradition of [20] and stands in contrast to flat softmax routing of modern large-scale MoE models [14, 36], 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 $\beta_k \in \mathbb{R}$:

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

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, $1 - p_k(x) = \text{sigmoid}(-z_k(x))$ 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. The gate logit z_k is presented as a linear form for concreteness, but the parameterization is a free design choice within the framework. The experiments evaluate two different instantiations side by side, the linear-gate variant SPAWN-LL and the quadratic-gate variant SPAWN-QL (Section 4.5). The formulation that follows depends only on the gate output $p_k(x)$, not on the specific parameterization of z_k , and therefore applies to both variants unchanged.

Tree structure. The G gates are arranged 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, the tree topology is encoded in a *link matrix* $\mathbf{L} \in \{-1, 0, +1\}^{E \times G}$ whose entries are read off the tree as in Figure 1, with

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

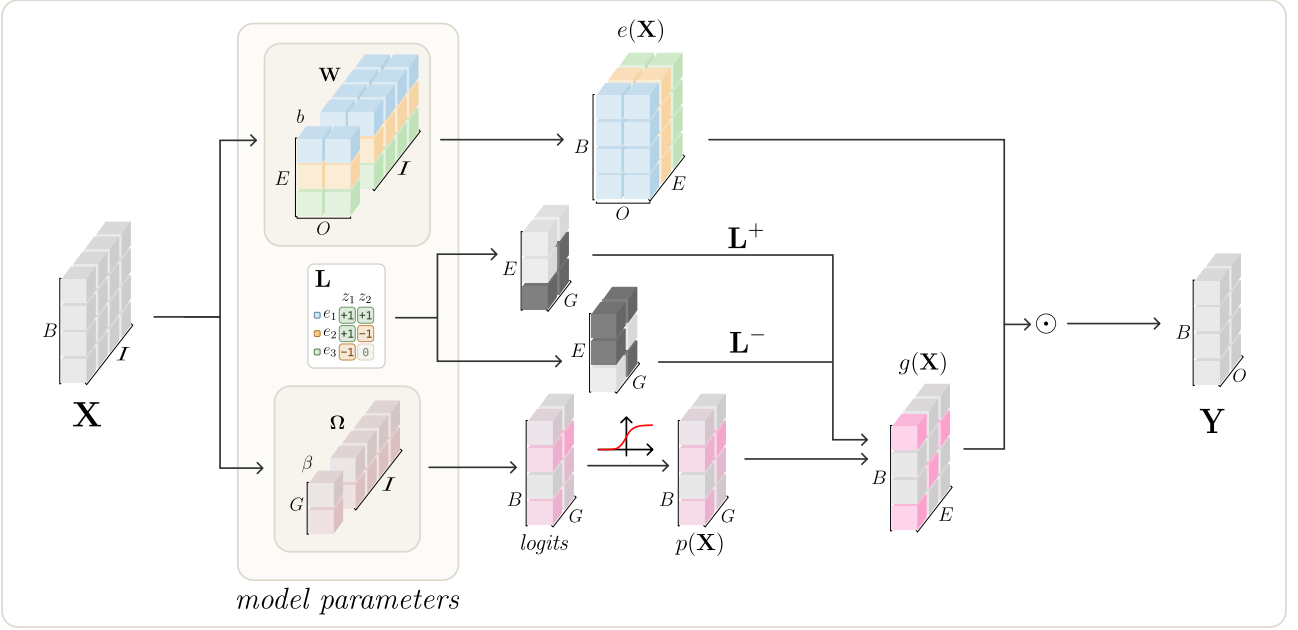


Figure 2: Forward pass for soft-tree structure.

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:

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

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. The framework places no restriction on the functional form of the experts: any differentiable predictor $e_h: \mathbb{R}^d \rightarrow \mathbb{R}^p$ is admissible. For simplicity, this work adopts affine experts $e_h(x) = \mathbf{W}_h x + b_h$, with $\mathbf{W}_h \in \mathbb{R}^{p \times d}$ and $b_h \in \mathbb{R}^p$, following the original mixture-of-experts formulation [18]. 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.

Together, the gate activations and expert outputs fully determine the model prediction through Equation 1. At inference, the input is broadcast to all gates and experts in parallel: the gating weights are recovered in closed form via the log-space contraction of Equation 4, and the prediction is the weighted sum of expert outputs. Figure 2 illustrates the complete forward pass. The trainable parameter set is $\theta = \{\omega_k, \beta_k\}_{k \in \mathcal{G}} \cup \{\mathbf{W}_h, b_h\}_{h \in \mathcal{E}}$.

2.2. Regularization Losses

Training a mixture of experts with only a task loss invites two well-known failure modes [14, 36]: *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. Both are countered with entropy-based regularizers that act 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

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

where $\mathcal{L}_t$ is the task loss (mean squared error for regression, cross-entropy for classification), $\mathcal{L}_s$ is the sample-entropy regularizer, and $\mathcal{L}_m$ is the marginal-entropy regularizer, both defined in the following. The coefficients $\lambda_s, \lambda_m \geq 0$ are scalar regularization coefficients. Both regularizers are normalized to lie in $[0, 1]$, so that 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

$$H_s(x) = - \sum_{h \in \mathcal{E}} g_h(x) \log g_h(x) \in [0, \log E] \quad (6)$$

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 [16, 31]. The objective minimizes the batch-averaged entropy, normalized by its maximum value:

$$\mathcal{L}_s(\theta) = \frac{1}{|\mathcal{B}| \log E} \sum_{x \in \mathcal{B}} H_s(x) \in [0, 1]. \quad (7)$$

The normalization by $\log E$ is essential for stability across expansions and is discussed together with the marginal regularizer below.

Marginal entropy. While sample entropy acts on individual samples, the marginal regularizer acts on the average routing behavior of each gate. The first step is to formalize the soft assignment weight of a sample reaching a given 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 gate k . This path structure is encoded 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), with

$$\mathbf{A}_{kj} = \begin{cases} +1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its positive side,} \\ -1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its negative side,} \\ 0 & \text{otherwise (including } j = k). \end{cases} \quad (8)$$

Splitting $\mathbf{A}$ into $\mathbf{A}^+ = \max(\mathbf{A}, 0)$ and $\mathbf{A}^- = \max(-\mathbf{A}, 0)$, the log marginal admits a closed-form expression analogous to Equation 4,

$$\log m_k(x) = \sum_{j \in \mathcal{G}} \mathbf{A}_{kj}^+ \log p_j(x) + \mathbf{A}_{kj}^- \log(1 - p_j(x)), \quad (9)$$

with the root gate given by the all-zero row of $\mathbf{A}$, so $m_{\text{root}}(x) = 1$. Given a batch, the effective split ratio of gate k , computed as the path-weighted average of its positive assignments across samples, is

$$\bar{p}_k = \frac{\sum_{x \in \mathcal{B}} m_k(x) p_k(x)}{\sum_{x \in \mathcal{B}} m_k(x)}. \quad (10)$$

A balanced gate divides its domain roughly in half, so $\bar{p}_k \approx \frac{1}{2}$; a collapsed gate has $\bar{p}_k \in \{0, 1\}$, meaning one of its two children receives negligible assignment and is effectively unused.²

Balance is enforced by maximizing the per-gate binary entropy of the split ratio, averaged across gates, or equivalently by penalizing its deviation from its maximum value $\log 2$:

$$\mathcal{L}_m(\theta) = 1 - \frac{1}{G \log 2} \sum_{k \in \mathcal{G}} H_b(\bar{p}_k) \in [0, 1], \quad H_b(p) = -p \log p - (1 - p) \log(1 - p). \quad (11)$$

The loss is zero when each gate splits its domain uniformly and approaches one when all gates collapse. Evaluating marginal entropy *per gate* rather than over the full E -way gating-weight distribution applies local soft pressure: each gate is pushed toward balanced partitioning of its own region, but the task loss can override this bias when an asymmetric split materially improves prediction. The per-gate formulation also avoids conflating shallow gates, which partition large regions, with deep gates, which partition small sub-regions. The matched $[0, 1]$ normalization of both regularizers preserves the relative weighting of λ_s and λ_m as E grows along the expansion trajectory.

²Divisions involving sums of path weights are guarded against zero denominators in the implementation; such numerical safeguards are omitted from the notation throughout.

Relation to prior work. The per-gate balance penalty (Equation 11) is closely related to the soft-decision-tree regularizer of Frosst and Hinton [15]: their path-weighted split ratio α_i is exactly the $\bar{p}_k$ defined above, and their cross-entropy penalty equals the per-gate binary entropy $H_b(\bar{p}_k)$ up to an additive constant. The two differ in how they treat depth. Frosst and Hinton [15] scaled down deeper gates by a fixed 2^{-d} factor, whereas the path weights $m_k(x)$ used here score each gate by the fraction of data it actually routes (Equation 10), which better fits the unbalanced trees produced by the data-driven splitting of Section 3, where deep sub-trees may still carry substantial mass. A comparable load-balancing term appears in the auxiliary losses of sparsely gated MoE [14, 36], but it enforces balance globally over the flat E experts, whereas the penalty proposed here acts locally and hierarchically, on each binary split and at the level of individual partitions.

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.

Structural necessity of the regularizers at expansion. The output-preserving inheritance of Section 3.3, formalized by Equation 15, further motivates the regularizer design: 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. The following makes this precise.

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

Proposition 2.1. *For every input x and every task loss $\mathcal{L}_t$ differentiable in $f_\theta(x)$,*

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

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

$$g_{h^+}(x) e_{h^+}(x) + g_{h^-}(x) e_{h^-}(x). \quad (12)$$

Since $g_{h^+}(x) + g_{h^-}(x) = g_{h^*}(x)$ by Equation 15 and $g_{h^*}(x)$ is independent of (ω', β') , differentiating yields

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

Differentiating Equation 12 with respect to ω' , and using $e_{h^+}(x) = e_{h^-}(x) = e_{h^*}(x)$ at initialization,

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

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^- . $\square$

The regularizers $\mathcal{L}_s$ (Equation 7) and $\mathcal{L}_m$ (Equation 11) 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. Section 4.5 provides the empirical evidence that supports this thesis, confirming that each penalty removed in isolation produces a distinct gating pathology, so that only the full objective avoids both.

3. Adaptive Expansion

The architecture described in Section 2 assumes a fixed number of experts E . In practice, the appropriate capacity of the model is not known in advance, as it depends on the complexity of the task and on how the input space is best partitioned. The Self-Partitioning Adaptive Widening Network (SPAWN) addresses this by starting from a single expert and growing the tree incrementally during training, adding capacity only where the data signals that the current partition is structurally under-fitting.

This growth is efficient because of the hierarchical tree structure: splitting a leaf into two children is a local operation that does not perturb the routing of existing experts, and the new experts inherit their parent’s parameters so that each expansion is a continuous deformation of the model rather than a structural reset. Each expansion involves three decisions: *when* to expand, that is, detecting that the current architecture has plateaued (Section 3.1), *where* to expand, that is, selecting the expert whose region is most under-fit (Section 3.2), and *how* to split so that training continues smoothly from the same functional state (Section 3.3). Section 3.4 combines these three components into the outer training loop that yields the final model.

3.1. When to Expand: Plateau Detection

Expansion is triggered 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. In practice, the plateau is detected from two rolling windows of the training-loss curve, each spanning w epochs. Expansion triggers when the relative drop in mean loss from the lagged window to the current one falls below a threshold η . Training rather than validation loss is used here so that the trigger reflects parameter convergence, not capacity sufficiency, and 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 expert cannot accommodate. This variation is aggregated as a sum rather than an average so that, all else equal, populous regions are refined before sparsely sampled ones.

This intuition is formalized 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:

$$\bar{\ell}_h(\mathcal{B}) = \frac{\sum_{x \in \mathcal{B}} g_h(x) \ell(x)}{\sum_{x \in \mathcal{B}} g_h(x)}, \quad (13)$$

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,³

$$s_h = \sum_{\mathcal{B}} \sum_{x \in \mathcal{B}} g_h(x) |\ell(x) - \bar{\ell}_h(\mathcal{B})|. \quad (14)$$

The expert selected for expansion is $h^* = \arg \max_{h \in \mathcal{E}} s_h$.

3.3. How to Expand: Output-Preserving Inheritance

Once the expert h^* has been selected, a new gate is introduced that partitions its region into two children, and all new parameters are initialized 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 μ_{h^*} of samples routed to

³Absolute 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.

Algorithm 1 SPAWN training procedure.

Require: Dataset $\mathcal{D}$, plateau window w , threshold η , max cycles C , patience P

Ensure: Trained model f_{θ^*}

```
1: Initialize  $E \leftarrow 1$  (single expert),  $\theta^* \leftarrow \theta$ ,  $p \leftarrow 0$ 
2: for  $c = 1, \dots, C$  do
3:   Create fresh optimizer and learning-rate schedule  $\triangleright$  avoid stale momentum
4:   repeat
5:     Train one epoch with  $\mathcal{L} = \mathcal{L}_t + \lambda_s \mathcal{L}_s + \lambda_m \mathcal{L}_m$   $\triangleright$  Eq. 5
6:   until plateau: relative drop over  $w$ -epoch windows  $< \eta$   $\triangleright$  Section 3.1
7:   if validation loss improved then
8:      $\theta^* \leftarrow \theta$ ,  $p \leftarrow 0$   $\triangleright$  retain best checkpoint
9:   else
10:     $p \leftarrow p + 1$   $\triangleright$  capacity early stopping
11:    if  $p \geq P$  then
12:      break
13:    end if
14:  end if
15:  Compute  $s_h$  for all  $h \in \mathcal{E}$   $\triangleright$  Eq. 14, Section 3.2
16:  Split  $h^* = \arg \max_h s_h$   $\triangleright$  Eq. 15, Section 3.3
17:   $E \leftarrow E + 1$ 
18: end for
19: Restore  $\theta^*$  and fine-tune with fresh optimizer
20: return  $f_{\theta^*}$ 
```

h^* . This fresh gate is appended as a new column of $\mathbf{L}$ and a new row and column of $\mathbf{A}$. Since the hyperplane passes through the centroid, the split is statistically balanced the more symmetric the distribution of samples around it. Following Equation 10, each child receives half of the parent’s routed mass in expectation, $\bar{p}_{k'} \approx \frac{1}{2}$. Let h^+ and h^- denote the positive and negative children of h^* under the new gate k' . The two children inherit the parent’s expert parameters exactly, $\mathbf{W}_{h^+} = \mathbf{W}_{h^-} = \mathbf{W}_{h^*}$ and likewise for the biases. Since their predictors are identical at expansion, their joint contribution to the model output is

$$g_{h^+}(x) e_{h^+}(x) + g_{h^-}(x) e_{h^-}(x) = (g_{h^+}(x) + g_{h^-}(x)) e_{h^*}(x), \quad (15)$$

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

3.4. The Expansion Cycle

The three components described above combine into the outer training loop summarized in Algorithm 1. SPAWN starts from a single expert and alternates between a *training phase*, which optimizes the current architecture until the plateau detector of Section 3.1 fires, and an *expansion step*, which selects the most heterogeneous expert (Section 3.2) and splits it via output-preserving inheritance (Section 3.3). A fresh optimizer and learning-rate schedule are created at the start of each cycle so that the newly added parameters are not disadvantaged by stale momentum.

After each training phase, the model is evaluated on a held-out validation set and the checkpoint with the best validation score seen so far is retained. The loop terminates when validation has not improved for P consecutive cycles or a maximum cycle count C is reached. Because each successive cycle operates on a strictly more capacious architecture, this checkpoint retention acts as *early stopping over model capacity*: it returns the smallest grown architecture at which held-out performance peaked, discarding any subsequent expansions that only served to overfit. As discussed in Section 4.6, on small or noisy datasets this capacity-level early stop is the component that prevents the final model from degrading. Once the loop ends, the best checkpoint is restored and fine-tuned with a dedicated optimizer pass.

4. Experiments

This section presents a systematic empirical evaluation of SPAWN along two complementary axes: comparison against fixed-architecture families whose capacity is externally tuned, and comparison against other growing-network methods that expand their own capacity during training. After describing the evaluation protocol, datasets, and baselines (Sections 4.1–4.3), Section 4.4 reports the main results, Section 4.5 isolates the contribution of individual design choices through targeted ablation studies, and Section 4.6 draws the final assessment. Unless otherwise noted, the SPAWN entries in the main comparison tables refer to the fully-linear variant SPAWN-LL, in which both gates and experts are linear. The effect of different formulations have Section 4.5 dedicated.

4.1. Experimental Setup

The evaluation covers a suite of regression and classification benchmarks of increasing complexity along two complementary axes. The first axis compares SPAWN under one fixed growth schedule (same hyperparameters) against several fixed-architecture baseline families, each swept over a capacity grid (varying hyperparameters) for each dataset, to assess whether adaptive growth can recover competitive accuracy at a fraction of the parameters. The second axis compares its tree-structured growth strategy against alternative neural expansion mechanisms under matched stopping conditions, to isolate the effect of the growth method itself.

The evaluation protocol is intentionally one-sided in favor of the baselines: each fixed-architecture family is swept over a per-dataset capacity grid (refer to Appendix A for further details) and the validation-best member is reported per dataset, whereas SPAWN runs once under a single fixed configuration on every dataset and reaches its final size through growth alone. The capacity ladder is swept, rather than fixing one architecture, precisely so that no baseline is penalized by an arbitrary choice of hyperparameters on a given task. Committing to one configuration in advance may bias the comparison, possibly leading to pathological training scenarios (underfit/overfit), while reporting the best-of-family member gives each fixed-capacity competitor its strongest candidate before it is compared against a method that selects its own capacity. A baseline’s headline number is therefore not the cost of one model but the validation-best member of a per-dataset capacity sweep, found only by training the entire ladder across all seeds; SPAWN instead pays its structural-search cost once, inside the single run that simultaneously decides and trains the architecture.

All adaptive runs share the same configuration on every dataset and every variant: the entropy-regularization coefficients of Equation 5 are fixed at $\lambda_s = \lambda_m = 10^{-2}$, parameter optimization is done with Adam [21] 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 [27] with learning rate 5×10^{-3} and weight decay 10^{-4} . Reported scores are the mean over 10 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 [43] at $\alpha = 0.05$. The complete hyperparameter table is reported in Appendix A.

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, while a short growth patience terminates earlier and returns smaller, faster models whose accuracy sits at a slightly earlier point along the size-accuracy curve. Growth patience is held fixed at the conservative value reported in Appendix A across the entire benchmark, deliberately favoring the higher-accuracy end of this trade-off to probe the procedure’s robustness rather than its tunability.

Hardware and reproducibility. The full benchmark campaign was executed on a single Apple M4 Pro with 12 CPU cores, without GPU acceleration. All wall-clock training times reported alongside the parameter counts in the main-text tables are measured by running 3 seeds per model sequentially on the same machine.

4.2. Datasets

The reported campaign covers 19 datasets, partitioned into a synthetic regression block, a synthetic classification block, a real-world tabular block, and a set of larger, higher-dimensional benchmarks. The dimensions and split sizes for the core 15 datasets are reported in Table 1; the four larger benchmarks are summarized in Table 2. 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. The complete source code, training scripts, and dataset-generation routines are available in a companion repository.⁴

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 [1]; *Rastrigin* combines a quadratic envelope with a dense periodic lattice [40]; *Rosenbrock* concentrates error along an anisotropic curved valley [32]. 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 [30]; *Pinwheel* follows the rotating-arm construction used in structured generative-model benchmarks [19]. *Spirals* and *3-Spirals* are two and three-class spiral manifolds generated synthetically. All four are produced using standard scikit-learn utilities [30], while *Checker* is a custom two-class checkerboard rule on the plane. All five tasks are two-dimensional; their class counts and split sizes are listed in Table 1.

Real-world tabular. The real-world block contains *Diabetes*, *California Housing*, *kin8nm*, *Breast Cancer*, and the cleaned *Sarcos* robotics regression [42].⁵ *Diabetes* is the 10-feature regression benchmark popularized with least-angle regression [10]; *California Housing* is the 8-feature census block regression dataset derived from the spatial autoregression study of Pace and Barry [29]; *kin8nm* is the 8-feature robot-arm regression task distributed with the DELVE benchmark collection [28]; and *Breast Cancer* is the 30-feature Wisconsin diagnostic binary classification dataset [44]. *Sarcos* is a 21-input, 7-output inverse-dynamics regression task.

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

Table 1: Dataset dimensions and split sizes for the core 15-dataset campaign.

Larger-scale benchmarks. To probe whether SPAWN’s input-space partitioning scales beyond the low-dimensional, small-sample regime of the core benchmark, the evaluation is extended to four additional datasets spanning both axes (Table 2). They include a large multi-class tabular problem (Covertypes [4], 54 features, 7 classes; a 50k stratified subsample is used for the multi-seed sweep), a large-scale tabular regression task (YearPredictionMSD [3], 90 features), and two ten-dimensional synthetic regression functions (*Rastrigin*-10D, *Ackley*-10D) that raise the input dimensionality of the synthetic family by an order of magnitude. The same single fixed SPAWN configuration is used as on the core benchmark, with the per-cycle and growth-cycle budgets reduced for tractability on the larger datasets (Appendix A). One protocol caveat: the largest datasets use the 50k stratified subsample for tractability on CPU.

⁴Access to the repository can be requested by contacting the author at federico.lolli@mail.polimi.it.

⁵The cleaned variant pools the original training and test matrices and draws a fresh 80/10/10 split with the common seed, avoiding the predefined partition. This version is denoted *Sarcos** in the result tables.

Dataset	Task	n	d	Notes
Covertypes	clf (7)	581k	54	50k stratified subsample for the sweep
YearPredictionMSD	reg	515k	90	official artist-disjoint split (full builder)
Rastrigin-10D	reg	30k	10	synthetic, 10-dimensional
Ackley-10D	reg	30k	10	synthetic, 10-dimensional

Table 2: Additional larger and higher-dimensional datasets. Subsampled variants are used only for the multi-seed feasibility sweep; full-data builders are retained for benchmark-faithful reporting.

4.3. Baselines

The two evaluation axes outlined in Section 4.1 call for two groups of baselines. The first consists of conventional model families (neural networks, tree ensembles, differentiable trees) whose architecture is fixed before training and whose capacity is tuned via a per-dataset grid sweep; these test whether adaptive growth can recover the accuracy of the best externally tuned configuration. The second consists of growing-network methods that, like SPAWN, increase their capacity during training; these test whether tree-structured, input-space-partitioning growth offers advantages over alternative neural expansion mechanisms. To denote specific model configurations, we adopt the notation 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 .

Fixed-architecture baselines. Each family is swept over a per-dataset capacity grid (Table 12) and the validation-best member is reported.

Multi-layer perceptron (MLP). Standard fully-connected feed-forward network with ReLU activations, trained by back-propagation [33] and swept over a dense grid of hidden widths $\{2, 4, \dots, 1024\}$ and depths $\{1, 2, 3, 4\}$. The grid is deliberately wide on both axes so that the best-of-family selection is never restricted by an arbitrary capacity ceiling: on every dataset the reported MLP is the validation-best member of this grid, including very small architectures where they suffice. The optimizer and regularization settings (AdamW, learning rate 5×10^{-3} , weight decay 10^{-4}) are held fixed across the entire grid and every dataset, so the ladder varies capacity only, isolating the architecture-size axis the comparison is about.

Random forest (RForest) [5]. Ensemble of independently trained decision trees, each fit on a bootstrap sample of the data with random feature subsampling at each split. Swept over four configurations of ensemble size and maximum depth, $(T, d) \in \{(50, 4), (100, 8), (200, 12), (300, 16)\}$, covering ensembles from shallow and compact to deep and large.

Gradient-boosted trees (XGBoost) [8]. Sequential ensemble of shallow decision trees, where each tree fits the residual of the previous ensemble. Swept over four configurations of boosting rounds and maximum depth, $(T, d) \in \{(100, 3), (300, 4), (600, 6), (1000, 8)\}$.

Soft binary decision tree (SoftTr). Fixed-depth differentiable soft tree [15, 17], included as the closest fixed-depth analog of SPAWN’s soft-gated tree and a natural reference for the gating mechanism, swept over tree depth.

Growing-network baselines. The following methods, like SPAWN, expand their capacity during training. Both are based on the authors’ implementations⁶ and adapted to the current evaluation protocol, run under SPAWN’s own stopping criterion (growth patience 50, 2000 epochs per phase, no parameter caps).

GradMax [12]. Grows width by adding neurons into existing hidden layers, initialising the incoming weights to zero (so the new unit contributes nothing at insertion, preserving the function) and the outgoing weights to the closed-form SVD solution that maximises the gradient norm of the new unit.

Firefly [45]. Grows by local architecture descent: at each phase it builds an over-grown candidate network in a functional neighbourhood of the current one, combining splits of existing neurons (each halved onto two copies, preserving the function) with brand-new neurons, briefly optimises the candidates, and keeps the top few by an integrated-gradient importance score. The adopted protocol grows a single hidden layer in width, as depth growth is out of scope.

4.4. Results

Best-of-family comparison. Tables 3 and 4 report 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.

⁶GradMax: <https://github.com/google-research/growneuron>. Firefly: <https://github.com/klightz/Firefly>.

Baselines					SPAWN
Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Moons	97.3 $\pm$ 0.1	96.6 $\pm$ 0.1	97.5 $\pm$ 0.0	96.9 $\pm$ 0.1	96.9 $\pm$ 0.0
	2.1M (w1k ₃) / 3.0s	7.4k (t100 ₈) / 45ms	3.8k (t300 ₄) / 12ms	77 (d4) / 5.0s	49 / 15.1s
Spirals	100.0 $\pm$ 0.0	98.8 $\pm$ 0.1	98.6 $\pm$ 0.1	76.3 $\pm$ 1.3	99.3 $\pm$ 0.4
	527.9k (w512 ₃) / 5.7s	88.8k (t300 ₁₆) / 251ms	40.4k (t1k ₈) / 89ms	1.3k (d8) / 30.5s	456 / 1.3m
3-Spirals	100.0 $\pm$ 0.0	97.0 $\pm$ 0.2	97.6 $\pm$ 0.1	91.5 $\pm$ 1.5	98.9 $\pm$ 0.2
	791.0k (w512 ₄) / 13.6s	86.5k (t300 ₁₆) / 384ms	143.9k (t1k ₈) / 335ms	1.5k (d8) / 51.9s	645 / 1.6m
Pinwheel	94.4 $\pm$ 0.2	95.2 $\pm$ 0.1	95.0 $\pm$ 0.0	94.5 $\pm$ 0.2	95.0 $\pm$ 0.2
	1.1M (w1k ₂) / 3.4s	30.0k (t200 ₁₂) / 106ms	22.6k (t300 ₄) / 55ms	2.0k (d8) / 28.9s	287 / 20.1s
Checker	89.8 $\pm$ 0.3	89.6 $\pm$ 0.1	91.0 $\pm$ 0.1	85.6 $\pm$ 0.5	90.3 $\pm$ 0.2
	8.6k (w64 ₃) / 4.9s	296.6k (t300 ₁₆) / 368ms	60.4k (t1k ₈) / 72ms	1.3k (d8) / 1.9m	1.2k / 4.4m
BreastCnc	95.7 $\pm$ 0.3	95.3 $\pm$ 0.2	94.1 $\pm$ 0.2	97.0 $\pm$ 0.5	95.0 $\pm$ 0.6
	338 (w8 ₂) / 258ms	969 (t50 ₄) / 23ms	2.0k (t300 ₄) / 42ms	8.4k (d8) / 3.6s	155 / 19.4s

Table 3: Best-of-family comparison on classification benchmarks. Top: test accuracy (% , mean $\pm$ 95% CI). Bottom: model size (red) and training time (blue).

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 parameter count is comparable to or lower than those of the soft-tree counterparts and markedly smaller than those of the MLP baselines. On the hardest targets (Rastrigin, Ackley), individual runs can exhibit elevated variance because the inter-cycle validation loss does not always follow a smooth elbow, occasionally triggering the early-stopping criterion before the architecture has fully converged. The fixed patience is retained across all datasets for fairness.

Larger-scale benchmarks. Table 5 reports SPAWN alongside the tuned best-of-family baselines on the larger, higher-dimensional datasets introduced in Section 4.2. On the synthetic 10-dimensional functions SPAWN is competitive with the dense MLP probe: it reaches $R^2 = 0.929$ on Ackley-10D with 983 parameters, against 0.931 for a 930k-parameter MLP, and matches it on Rastrigin-10D. The method also trains stably at the 50k-sample, 54–90-feature scale, showing no evident degradation when scaling beyond the core benchmark. On these larger datasets gradient-boosted trees are the strongest family overall, particularly on Covertype where the gap is substantial. The tree-based advantage is expected in this regime and the method is not designed to compete with specialized tabular learners at this scale. These results are included for completeness and to verify that the adaptive procedure remains stable as input dimensionality and sample count increase. Against the dense MLP, the parameter-frugality pattern of the core benchmark persists: SPAWN matches the MLP on the synthetic functions while using three orders of magnitude fewer parameters.

Growth trajectories. Tables 3, 4 and 5 report the final model selected by each adaptive run. To illustrate how the growth procedure arrives at that model, Figure 3 overlays all seeds on kin8nm. The training loss (Figure 3a) follows a staircase pattern across seeds, plateauing as the current architecture saturates and dropping sharply when a new expert is added. The held-out loss (Figure 3b) initially descends as useful capacity is introduced but eventually rises when further splits induce overfitting. The vertical red lines mark the validation-best expansion cycle for each seed, namely, the checkpoint restored as the final model, which typically occurs well before the growth loop terminates.

Growing-network comparison. Table 6 reports held-out performance and per-run wall-clock times. Under matched budgets, SPAWN attains the best score on 10 of 15 datasets, with its advantage concentrated on the regression benchmarks and on the geometric classification tasks. GradMax leads on Spirals, 3-Spirals, and Breast Cancer, and Firefly on Sin, though in each case the margins are narrow except on Breast Cancer. Across the board, SPAWN reaches competitive or superior accuracy while using a substantially smaller fraction of the parameters: on the datasets where GradMax matches or surpasses SPAWN’s accuracy, it does so at parameter counts one to two orders of magnitude larger (e.g. 57.5k and 105.2k on Spirals and 3-Spirals against SPAWN’s 456 and 645).

The two baselines also differ in character. GradMax grows genuine width with function-preserving SVD initialisation and is the strongest competitor overall, winning four of the five datasets where the growers surpass SPAWN. Firefly grows a single hidden layer in width, which limits its expressivity on depth-requiring tasks, though under matched budgets it still reaches 99.6%/98.0% on Spirals/3-Spirals. On timing, SPAWN is gener-

Baselines					SPAWN
Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Sin	0.299 $\pm$ 0.001 526.8k (w512 ₃) / 1.7s	0.314 $\pm$ 0.000 28.3k (t100 ₈) / 38ms	0.307 $\pm$ 0.000 6.0k (t300 ₄) / 10ms	0.319 $\pm$ 0.013 766 (d8) / 20.7s	0.296 $\pm$ 0.001 192 / 23.7s
Sin ²	0.295 $\pm$ 0.001 526.8k (w512 ₃) / 5.2s	0.337 $\pm$ 0.001 156.4k (t200 ₁₂) / 92ms	0.304 $\pm$ 0.000 7.6k (t300 ₄) / 14ms	0.582 $\pm$ 0.003 766 (d8) / 25.1s	0.290 $\pm$ 0.001 2.1k / 4.3m
Ackley	0.100 $\pm$ 0.002 132.6k (w256 ₃) / 8.7s	0.128 $\pm$ 0.000 509.8k (t300 ₁₆) / 320ms	0.090 $\pm$ 0.000 55.7k (t600 ₆) / 96ms	0.212 $\pm$ 0.001 1.0k (d8) / 39.1s	0.143 $\pm$ 0.012 3.1k / 7.6m
Rastrigin	0.272 $\pm$ 0.009 132.6k (w256 ₃) / 8.2s	0.390 $\pm$ 0.001 487.7k (t300 ₁₆) / 333ms	0.168 $\pm$ 0.001 59.3k (t600 ₆) / 100ms	0.651 $\pm$ 0.001 61 (d4) / 1.8s	0.352 $\pm$ 0.063 3.6k / 13.4m
Rosenbrock	0.005 $\pm$ 0.000 4.1k (w1k ₁) / 6.8s	0.071 $\pm$ 0.001 388.4k (t300 ₁₆) / 358ms	0.065 $\pm$ 0.001 143.2k (t1k ₈) / 225ms	0.012 $\pm$ 0.001 1.0k (d8) / 31.0s	0.010 $\pm$ 0.000 512 / 48.3s
Diabetes	0.669 $\pm$ 0.009 2.1M (w1k ₃) / 1.2s	0.698 $\pm$ 0.003 64.0k (t200 ₁₂) / 123ms	0.696 $\pm$ 0.006 5.1k (t300 ₄) / 11ms	0.667 $\pm$ 0.011 37 (d2) / 340ms	0.670 $\pm$ 0.009 34 / 12.9s
CalHousing	0.456 $\pm$ 0.003 10.2k (w1k ₁) / 18.8s	0.444 $\pm$ 0.000 3.1M (t300 ₁₆) / 14.2s	0.387 $\pm$ 0.001 350.8k (t1k ₈) / 1.2s	0.468 $\pm$ 0.003 2.6k (d8) / 1.8m	0.470 $\pm$ 0.010 715 / 7.3m
kin8nm	0.255 $\pm$ 0.003 134.1k (w256 ₃) / 4.2s	0.538 $\pm$ 0.001 1.9M (t300 ₁₆) / 5.2s	0.438 $\pm$ 0.002 65.8k (t600 ₆) / 447ms	0.267 $\pm$ 0.002 2.6k (d8) / 42.6s	0.260 $\pm$ 0.003 677 / 2.4m
Sarcos*	0.084 $\pm$ 0.000 1.1M (w1k ₂) / 3.8m	0.158 $\pm$ 0.000 7.0M (t300 ₁₆) / 1.9m	0.100 $\pm$ 0.000 2.7M (t1k ₈) / 42.5s	0.101 $\pm$ 0.001 7.4k (d8) / 25.7m	0.087 $\pm$ 0.002 40.3k / 2.2h

Table 4: Best-of-family comparison on regression benchmarks. Top: test RMSE (mean $\pm$ 95% CI). Bottom: model size (red) and training time (blue).

	Baselines					SPAWN
	Dataset	MLP	RForest	XGBoost	SoftTr	SPAWN
Regression	Rastrigin-10D	0.489 $\pm$ 0.004 930.3k	0.496 $\pm$ 0.002 2.9M	0.895 $\pm$ 0.001 74.6k	0.476 $\pm$ 0.005 3.1k	0.503 $\pm$ 0.004 287
	Ackley-10D	0.931 $\pm$ 0.001 930.3k	0.752 $\pm$ 0.003 3.1M	0.970 $\pm$ 0.000 74.2k	0.929 $\pm$ 0.001 3.1k	0.929 $\pm$ 0.001 983
	YearPredMSD	0.315 $\pm$ 0.007 473.6k	0.249 $\pm$ 0.004 3.4M	0.308 $\pm$ 0.004 39.7k	0.318 $\pm$ 0.005 23.5k	0.295 $\pm$ 0.007 455
Clf	Covertime	85.0 $\pm$ 0.5 455.9k	80.9 $\pm$ 0.3 1.1M	88.3 $\pm$ 0.2 779.9k	82.9 $\pm$ 0.4 15.8k	79.8 $\pm$ 0.4 5.4k

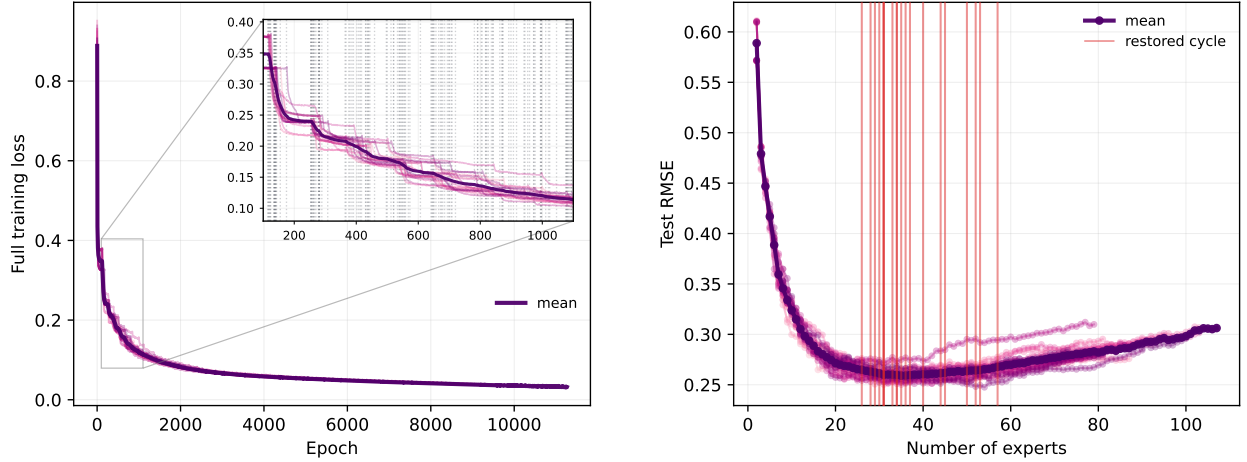
Table 5: Best-of-family comparison on the larger, higher-dimensional datasets. Regression: test R^2 ; classification: test accuracy (%); higher is better for both. Top: mean $\pm$ 95% CI. Bottom: model size (red).

ally slower than the two growing baselines, though the differences are moderate on most datasets and significant only on the heaviest ones (e.g. Sarcos).

4.5. Ablation Studies

Gate parameterizations: linear and quadratic. The expansion procedure of SPAWN is independent of the function class used in each gate and expert, so the framework is naturally instantiated by a choice of gate parameterization. This work considers two such instantiations side by side: **SPAWN-LL**, which uses linear gates and linear experts, and **SPAWN-QL**, which uses quadratic gates with linear experts. Quadratic gates are the smallest expressivity increment that admits curved decision boundaries while preserving the closed-form gating of Section 2.1, so the two variants differ *only* in the routing function and isolate the effect of curved partitioning from confounders such as deeper or non-linear leaves. Both are naturally supported by the growing framework, richer gate or expert classes (e.g. MLPs, CNNs, or other functional forms) are, indeed, equally admissible and left to future work.

Table 7 compares the two. Curved routing changes the accuracy-efficiency trade-off rather than uniformly dominating: on highly non-linear tasks such as Rastrigin and 3-Spirals, SPAWN-QL substantially improves test scores, and on Sin² and 3-Spirals it reaches comparable accuracy while drastically reducing model size and cutting mean training time by an order of magnitude, since a single curved gate replaces several linear splits. On the remaining tasks the two variants are statistically comparable, with SPAWN-LL retaining a slight edge



(a) Training loss across the full adaptive run, with a zoom inset on a few expansion cycles. (b) Test loss versus expansion cycle, with the best validation cycle restored by early stopping marked.

Figure 3: Training and test loss curves on the kin8nm dataset.

on kin8nm and Checker. The two parameterizations thus offer complementary operating points within one framework: linear gates give the most interpretable, axis-of-a-hyperplane partition, while quadratic gates buy a more compact partition where the data geometry is curved.

Regularizer ablation. The two entropy penalties of Section 2.2 play a structural role in the expansion procedure: at the instant of expansion the task-loss gradient on a newly inserted gate is identically zero (proposition 2.1), so the entropy penalties are the only first-order force that can differentiate a fresh gate. The ablation in this section is the empirical counterpart, showing that each penalty, removed in isolation, produces a distinct and opposite gating pathology, so that only the full objective avoids both.

The ablation covers the four on/off combinations of (λ_s, λ_m) on four datasets spanning the easy-classification and hard-regression regimes. The active penalty is set to 10^{-2} in every arm, so the arms are *full* (both penalties on), *none* (both off), and the two single-penalty arms *sample only* (λ_s on) and *marginal only* (λ_m on). Table 8 reports the held-out metric and the resulting expert count, Table 9 the gating-health signals. On the classification tasks the decision boundary is simple enough that test accuracy is essentially unchanged across all arms. The key effect of the regularizers is on the organization of capacity: the collapse-prone *sample only* arm reaches the same accuracy with up to $2.5\times$ the experts of the full objective (Checkerboard, 311 vs. 124). Utilization, specialization, and termination are exactly the properties a growing method must control, and they surface the moment the task is hard.

The two penalties fail in opposite directions. Removing the marginal penalty (sample-only, $\lambda_m=0$) collapses routing onto a few experts: pooled marginal balance falls from 0.92 to 0.36 and the dead-expert fraction rises from 0.09 to 0.44, so roughly 45% of experts never win a sample, a state *worse* than removing both penalties. Removing the sample penalty (marginal-only, $\lambda_s=0$) keeps the experts balanced but never lets per-sample routing commit, and on the hardest task (Rastrigin) growth then fails to terminate: 13 of 88 runs were still expanding at 200–575 cycles when stopped. The full objective ($\lambda_s=\lambda_m=10^{-2}$) exhibits neither pathology and terminates on every run. The mechanism is symmetric: λ_s sharpens per-sample routing so a new gate commits to a region, while λ_m spreads marginal usage so no expert is starved; each one alone over-rotates the gates toward its own objective, and only together do they realize the differentiation pressure of section 2.2 without degeneracy. This is why the default $\lambda_s=\lambda_m=10^{-2}$ is a principled operating point rather than an arbitrary pick. This ablation establishes that *both penalties are necessary*. A strength sweep that would establish 10^{-2} as the best magnitude is a weaker, separate claim left to future work. It is worth stressing, consistently with Section 4.6, that these routing-shaping penalties do not, and are not intended to, prevent the capacity-driven overfitting that affects the smallest datasets.

Parameter efficiency against fixed MLPs. The best-of-family comparison reports the validation-best member of each family. Here the family is fixed to MLPs, asking how large a fixed MLP must be before it becomes statistically equivalent to the validation-selected SPAWN run.

Growing			SPAWN
Dataset	GradMax	Firefly	SPAWN
Sin	0.293 $\pm$ 0.010 <i>4.0k / 11.5s</i>	0.284 $\pm$ 0.009 <i>1.1k / 11.6s</i>	0.296 <i>192 / 23.7s</i>
Sin ²	0.483 $\pm$ 0.081 <i>29.4k / 21.4s</i>	0.361 $\pm$ 0.019 <i>2.4k / 20.9s</i>	0.290 <i>2.1k / 4.3m</i>
Ackley	0.231 $\pm$ 0.004 <i>11.4k / 17.5s</i>	0.220 $\pm$ 0.004 <i>2.8k / 24.4s</i>	0.143 <i>3.1k / 7.6m</i>
Rastrigin	0.681 $\pm$ 0.027 <i>35.7k / 26.9s</i>	0.481 $\pm$ 0.040 <i>3.2k / 26.3s</i>	0.352 <i>3.6k / 13.4m</i>
Rosenbrock	0.032 $\pm$ 0.004 <i>21.3k / 25.2s</i>	0.026 $\pm$ 0.003 <i>1.1k / 14.2s</i>	0.010 <i>512 / 48.3s</i>
Diabetes	0.755 $\pm$ 0.046 <i>76 / 4.0s</i>	0.700 $\pm$ 0.024 <i>82 / 7.0s</i>	0.670 <i>34 / 12.9s</i>
CalHousing	0.448 $\pm$ 0.006 <i>10.1k / 2.7m</i>	0.470 $\pm$ 0.053 <i>5.4k / 3.8m</i>	0.470 <i>715 / 7.3m</i>
kin8nm	0.283 $\pm$ 0.004 <i>20.1k / 83.4s</i>	0.268 $\pm$ 0.004 <i>5.7k / 83.8s</i>	0.260 <i>677 / 2.4m</i>
Sarcos	0.125 $\pm$ 0.003 [†] <i>681.7k / 59.8m</i>	0.132 $\pm$ 0.001 <i>15.5k / 8.5m</i>	0.087 <i>40.3k / 2.2h</i>

(a) Regression (RMSE, lower better).

Growing			SPAWN
Dataset	GradMax	Firefly	SPAWN
Moons	96.66 $\pm$ 0.64 <i>1.5k / 7.6s</i>	96.81 $\pm$ 0.71 <i>284 / 6.6s</i>	96.90 <i>49 / 15.1s</i>
Spirals	99.75 $\pm$ 0.31 <i>57.5k / 6.4m</i>	99.56 $\pm$ 0.39 <i>3.9k / 29.2s</i>	99.30 <i>456 / 78.0s</i>
3-Spirals	99.39 $\pm$ 0.36 <i>105.2k / 5.1m</i>	98.00 $\pm$ 0.46 <i>4.8k / 31.7s</i>	98.90 <i>645 / 1.6m</i>
Pinwheel	94.12 $\pm$ 0.94 <i>1.4k / 8.7s</i>	93.78 $\pm$ 0.77 <i>1.8k / 9.8s</i>	95.00 <i>287 / 20.1s</i>
Checker	86.05 $\pm$ 1.61 <i>239.8k / 4.9m</i>	89.38 $\pm$ 0.26 <i>3.7k / 64.5s</i>	90.30 <i>1.2k / 4.4m</i>
BreastCnc	97.89 $\pm$ 0.99 <i>232 / 5.0s</i>	97.72 $\pm$ 1.07 <i>2.6k / 4.4s</i>	95.00 <i>155 / 19.4s</i>

(b) Classification (Acc. %, higher better).

Table 6: Growing-network baselines versus SPAWN. Top entry: held-out test metric mean $\pm$ 95% CI over 10 seeds. Bottom entry: model size (red; parameters) and mean wall-clock training time (blue). †: 3/10 Sarcos seeds exceeded the compute budget; that cell is over the 7 completed seeds.

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

Table 7: SPAWN-LL versus SPAWN-QL on a representative subset of the benchmark. Top: test score (mean $\pm$ 95% CI). Bottom: model size (red) and training time (blue).

The *smallest equiv.* column reports, for each dataset, the smallest MLP run whose validation score is statistically equivalent or superior to the validation-selected SPAWN run under a two-one-sided test [TOST, 35], namely a statistical test involving two simultaneous one-sided t -tests verifying that the difference in means falls within a symmetric equivalence margin. A dash means no MLP configuration in the swept ladder met this criterion. The full procedure, including the equivalence margin, the paired-versus-two-sample rule, the same-complexity run selection, and the convention used in marginal cases, is specified in Section 4.5.

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. A dash in the smallest-equivalent column means that no MLP in the swept ladder was classified equivalent-or-superior *on validation* under the test of Section 4.5.⁷

It is also worth identifying where this efficiency advantage is narrowest. On California Housing a small fixed MLP is both smaller than and significantly better than SPAWN: the w_{16_2} MLP (433 parameters) and the w_{16_3} MLP (705 parameters) both attain a lower RMSE than the 715-parameter SPAWN run. This is expected behavior rather than a breakdown of the method. SPAWN’s parameter count scales with the number of experts times the number of input features ($E \cdot F$), so on tabular datasets with many features each expert is comparatively expensive and even a few of them erase the parameter-savings margin against a compact MLP, whereas on the low-dimensional synthetic tasks each expert is cheap. This is reinforced by the growth dynamics:

⁷On kin8nm, for instance, the dash coincides with a *best observed* MLP that is in fact slightly test-superior (0.255 versus 0.260), so dashes should be read as inconclusive.

Regularizer arm	Classification (test acc. [%], $\uparrow$)			Regression (RMSE, $\downarrow$)
	Spirals	Pinwheel	Checker	Rastrigin
full	99.3 $\pm$ 0.6 57 E	93.9 $\pm$ 0.9 14 E	91.7 $\pm$ 0.6 124 E	0.389 $\pm$ 0.113 558 E
none	99.4 $\pm$ 0.8 56 E	94.1 $\pm$ 1.1 15 E	92.0 $\pm$ 0.5 143 E	0.483 $\pm$ 0.188 303 E
sample only	99.4 $\pm$ 0.3 90 E	93.9 $\pm$ 0.9 12 E	90.0 $\pm$ 1.1 311 E	0.672 $\pm$ 0.032 54 E
marginal only	99.5 $\pm$ 0.3 88 E	93.9 $\pm$ 0.8 14 E	92.1 $\pm$ 0.4 145 E	0.688 $\pm$ 0.015 [†] 39 E

Table 8: Regularizer ablation across four datasets (11 seeds, SPAWN-LL). Top entry: held-out test metric (mean $\pm$ 95% CI). Bottom entry: mean expert count (red, E). [†]: some seeds did not terminate; cell over completed seeds only.

Regularizer arm	marg. balance $\uparrow$	dead-expert frac. $\downarrow$	sample entropy	non-term. seeds
full	0.92	0.09	0.33	0
none	0.84	0.08	0.47	0
sample only	0.36	0.44	0.29	0
marginal only	0.96	0.01	0.55	13

Table 9: End-of-training gating health, pooled over all four datasets. *Marg. balance*: expert utilization uniformity ($\rightarrow 1$ = even); *dead-expert frac.*: share never winning a sample; *sample entropy*: per-sample routing commitment (low = committed); *non-term.*: runs (of 88) that never converged.

because each cycle is driven to overfit the current residual while generalization is optimized only *across* cycles, a single cycle can already overfit on these tasks, leaving the inter-cycle dynamics little room to act. The same tendency is visible across the regression tasks, with Diabetes and California Housing the clearest cases. On Spirals a compact-grid MLP (w_{64_3} , 8.6k parameters) significantly beats SPAWN on accuracy, although SPAWN remains 18.9 $\times$ smaller, a case where SPAWN trades a small accuracy gap for a large parameter saving.

Overall, SPAWN learns in every setting a markedly more compact representation that is statistically competitive with much larger MLP models, with few explicit exceptions on tabular datasets with many features.

Statistical equivalence procedure. The parameter-efficiency tables above 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 paragraph 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, two-one-sided tests are applied (TOST) [35] 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 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 A reached either classification on that dataset.

	SPAWN	MLP		
Dataset	SPAWN	same size	smallest equiv.	best observed
Moons	96.9 $\pm$ 0.0 49 / 15.1s	92.8 $\pm$ 5.0 62 (w4 ₃) / 2.0s	97.2 $\pm$ 0.2 898 (w16 ₄) / 1.6s	97.3 $\pm$ 0.1 2.1M (w1k ₃) / 3.0s
Spirals	99.3 $\pm$ 0.4 456 / 1.3m	97.2 $\pm$ 2.8 626 (w16 ₃) / 5.6s	100.0 $\pm$ 0.0 8.6k (w64 ₃) / 6.2s	100.0 $\pm$ 0.0 527.9k (w512 ₃) / 5.7s
3-Spirals	98.9 $\pm$ 0.2 645 / 1.6m	97.7 $\pm$ 2.4 643 (w16 ₃) / 7.0s	99.9 $\pm$ 0.1 2.3k (w32 ₃) / 5.1s	100.0 $\pm$ 0.0 791.0k (w512 ₄) / 13.6s
Pinwheel	95.0 $\pm$ 0.2 287 / 20.1s	95.0 $\pm$ 0.3 285 (w8 ₄) / 2.0s	94.6 $\pm$ 0.1 1.0k (w128 ₁) / 3.1s	94.4 $\pm$ 0.2 1.1M (w1k ₂) / 3.4s
Checker	90.3 $\pm$ 0.2 1.2k / 4.4m	87.8 $\pm$ 0.8 1.2k (w32 ₂) / 16.4s	– –	89.8 $\pm$ 0.3 8.6k (w64 ₃) / 4.9s
BreastCnc	95.0 $\pm$ 0.6 155 / 19.4s	95.5 $\pm$ 0.4 154 (w4 ₂) / 424ms	95.7 $\pm$ 0.5 8.4k (w256 ₁) / 185ms	95.7 $\pm$ 0.3 338 (w8 ₂) / 258ms

Table 10: MLP parameter-efficiency comparison on classification benchmarks. Top: test accuracy (% , mean $\pm$ 95% CI). Bottom: model size (red) and training time (blue).

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 Tables 10 and 11 are the corresponding held-out test scores on the same seeds, following standard practice of separating the model-selection signal from the reported metric.

4.6. Summary and Limitations

Taken together, the experiments show that under one fixed configuration SPAWN grows models whose accuracy is competitive with the best per-dataset-tuned member of each baseline family while using one to three orders of magnitude fewer parameters. This architecture is found in a single self-sizing run rather than through a capacity sweep. The parameter-efficiency comparison against fixed MLPs is the sharpest form of this result, and the two ablations confirm that the compactness stems from design choices that are individually essential. Curved gates improve both accuracy and parameter efficiency where the geometry is intrinsically non-linear, and each entropy penalty suppresses a distinct gating pathology.

The current evaluation is confined to small-to-mid tabular and synthetic problems. On the larger, higher-dimensional benchmarks gradient-boosted trees remain the strongest family, and among the growing methods SPAWN is the slowest, though the gap is moderate on most datasets.

5. Related Work

This section reviews the literature relevant to this work. It first covers dynamic architectures and network growth, in contrast to standard architecture search, then examines tree-based partitioning models, and finally discusses mixture-of-experts approaches.

Dynamic Architectures and Network Growth. Automated architecture discovery is typically addressed via Neural Architecture Search [46] and differentiable variants like DARTS [24]. 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 SPAWN lies in the literature on growing neural networks. Classical methods like Cascade Correlation [13] and Net2Net [9] incrementally expand the capacity of the model, while recent gradient-driven approaches, such as Splitting Steepest Descent [25], GradMax [12], and Firefly [45], add neurons or layers based on observed training signals. The main difference between this family of methods and SPAWN lies in the unit of growth. Existing approaches expand neural networks by adding scalar parameters or hidden units, which inherently fragment capacity. In contrast, SPAWN grows by partitioning regions of

	SPAWN	MLP		
Dataset	SPAWN	same size	smallest equiv.	best observed
Sin	0.296 ± 0.001 <i>192 / 23.7s</i>	0.326 ± 0.016 <i>193 (w64₁) / 3.4s</i>	–	0.299 ± 0.001 <i>526.8k (w512₃) / 1.7s</i>
Sin ²	0.290 ± 0.001 <i>2.1k / 4.3m</i>	0.436 ± 0.046 <i>2.2k (w32₃) / 3.3s</i>	–	0.295 ± 0.001 <i>526.8k (w512₃) / 5.2s</i>
Ackley	0.143 ± 0.012 <i>3.1k / 7.6m</i>	0.194 ± 0.004 <i>3.3k (w32₄) / 4.9s</i>	0.116 ± 0.004 <i>33.5k (w128₃) / 6.7s</i>	0.100 ± 0.002 <i>132.6k (w256₃) / 8.7s</i>
Rastrigin	0.352 ± 0.063 <i>3.6k / 13.4m</i>	0.607 ± 0.006 <i>4.1k (w1k₁) / 6.1s</i>	0.326 ± 0.017 <i>66.8k (w256₂) / 4.4m</i>	0.272 ± 0.009 <i>132.6k (w256₃) / 8.2s</i>
Rosenbrock	0.010 ± 0.000 <i>512 / 48.3s</i>	0.010 ± 0.001 <i>513 (w128₁) / 5.6s</i>	0.006 ± 0.000 <i>1.0k (w256₁) / 5.6s</i>	0.005 ± 0.000 <i>4.1k (w1k₁) / 6.8s</i>
Diabetes	0.670 ± 0.009 <i>34 / 12.9s</i>	0.727 ± 0.043 <i>37 (w2₃) / 356ms</i>	0.675 ± 0.015 <i>51.1k (w128₄) / 195ms</i>	0.669 ± 0.009 <i>2.1M (w1k₃) / 1.2s</i>
CalHousing	0.470 ± 0.010 <i>715 / 7.3m</i>	0.447 ± 0.004 <i>705 (w16₃) / 27.9s</i>	0.449 ± 0.003 <i>433 (w16₂) / 33.7s</i>	0.456 ± 0.003 <i>10.2k (w1k₁) / 18.8s</i>
kin8nm	0.260 ± 0.003 <i>677 / 2.4m</i>	0.281 ± 0.004 <i>705 (w16₃) / 10.8s</i>	–	0.255 ± 0.003 <i>134.1k (w256₃) / 4.2s</i>
Sarcos*	0.087 ± 0.002 <i>40.3k / 2.2h</i>	0.094 ± 0.000 <i>53.3k (w128₄) / 17.0m</i>	0.085 ± 0.000 <i>277.5k (w512₂) / 2.4h</i>	0.084 ± 0.000 <i>1.1M (w1k₂) / 3.8m</i>

Table 11: MLP parameter-efficiency comparison on regression benchmarks. Top: test RMSE (mean $\pm$ 95% CI). Bottom: model size (red) and training time (blue).

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 [39], 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, SPAWN adapts this capacity curriculum to supervised learning using a hierarchical soft tree, generalizing the approach 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 [17] and Deep Neural Decision Forests [22]. While models like those of Frosst and Hinton [15] 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 the proposed growth mechanism are Adaptive Neural Trees [ANTs, 38], which also expand during training. SPAWN differs from ANTs on three major points: (i) ANT routers and leaves can be deep neural networks, whereas the gates and experts of SPAWN also admit linear solutions, locating the non-linear capacity entirely in the partition. (ii) ANT growth is driven by validation-based search, whereas SPAWN’s relies purely on training-time signals. (iii) ANT trains new modules locally, while SPAWN trains the full model jointly at every cycle, achieving smooth continuation via output-preserving parameter inheritance. Closer in inductive bias to the proposed architecture is the family of Gated Linear Networks [GLNs, 6, 23, 41]. 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, SPAWN 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 frameworks [18, 20], including modern sparsely-gated variations scaled to massive regimes [14, 36] treat the number of experts as a fixed hyperparameter chosen in advance. The proposed approach diverges significantly, as the number of experts is adjusted during training to account for within-region heterogeneity. The dynamic growth of the network, and thus of the number of experts, requires adequate regularization. While SPAWN draws on minimum-entropy regularization [16, 31] and load-balancing auxiliary losses [14, 36], it uniquely normalizes 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.

6. Conclusion

This work 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, it has been shown 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 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, and application on broader classes of tasks.

References

- [1] David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*, 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.
- [2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [3] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 591–596, 2011.
- [4] Jock A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 1999. doi: 10.1016/S0168-1699(99)00046-0.
- [5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.
- [6] David Budden, Adam Marblestone, Eren Sezener, Tor Lattimore, Gregory Wayne, and Joel Veness. Gaussian gated linear networks. *Advances in Neural Information Processing Systems*, 33:16508–16519, 2020.
- [7] Lucas Caccia, Jing Xu, Myle Ott, Marcaurelio Ranzato, and Ludovic Denoyer. On anytime learning at macroscale. In *Conference on Lifelong Learning Agents*, pages 165–182. PMLR, 2022.
- [8] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016. doi: 10.1145/2939672.2939785.
- [9] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016.
- [10] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004. doi: 10.1214/009053604000000067.
- [11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.
- [12] Utku Evci, Bart van Merriënboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. GradMax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022.
- [13] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, 1990.
- [14] 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.

- [15] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. Workshop on Interpreting, Explaining and Visualizing Deep Learning – Now what?, NIPS 2017, 2017.
- [16] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, volume 17, pages 529–536. MIT Press, 2004.
- [17] Olcay T. Irsoy, Ozan T. Yildiz, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, pages 1819–1822. IEEE, 2012.
- [18] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.
- [19] Matthew J. Johnson, David K. Duvenaud, Alexander B. Wiltschko, Ryan P. Adams, and Sandeep R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, volume 29, pages 2954–2962, 2016.
- [20] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994. doi: 10.1162/neco.1994.6.2.181.
- [21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [22] Peter Kotschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475. IEEE, 2015. doi: 10.1109/ICCV.2015.172.
- [23] Qianyi Li and Haim Sompolinsky. Globally gated deep linear networks. *Advances in Neural Information Processing Systems*, 35:34789–34801, 2022.
- [24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.
- [25] Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, 2019.
- [26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.
- [27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [28] Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996. doi: 10.1007/978-1-4612-0745-0.
- [29] R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3): 291–297, 1997. doi: 10.1016/S0167-7152(96)00140-X.
- [30] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.
- [31] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. ICLR 2017 Workshop, 2017.
- [32] H. H. Rosenbrock. 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.
- [33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.
- [34] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

- [35] Donald J Schuirmann. 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.
- [36] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.
- [37] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [38] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6166–6175. PMLR, 2019.
- [39] Gianmarco Tedeschi, Matteo Papini, Alberto Maria Metelli, and Marcello Restelli. Search or split: Policy gradient with adaptive policy space. *Machine Learning*, 114:186, 2025. doi: 10.1007/s10994-025-06820-2.
- [40] Aimo Törn and Antanas Žilinskas, editors. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, 1989. doi: 10.1007/3-540-50871-6.
- [41] Joel Veness, Tor Lattimore, David Budden, Avishkar Bhoopchand, Christopher Mattern, Agnieszka Grabska-Barwinska, Eren Sezener, Jianan Wang, Peter Toth, Simon Schmitt, and Marcus Hutter. Gated linear networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10015–10023. AAAI Press, 2021. doi: 10.1609/aaai.v35i11.17202.
- [42] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An $O(n)$ algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 1079–1086, 2000.
- [43] B. L. Welch. 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.
- [44] William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast cancer wisconsin (diagnostic). UCI Machine Learning Repository, 1993.
- [45] Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, 2020.
- [46] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A. 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 4.4 and 4.5, 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 12, while the parameter-efficiency comparison of Section 4.5 uses the full dense grid to identify same-budget, smallest-equivalent, and validation-best MLPs.

Adaptive hyperparameters. The adaptive procedure runs with the single fixed configuration of Table 13 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.

Reduced budgets for larger-scale benchmarks. The four larger-scale datasets of Table 2 use a reduced hard cycle cap to keep the campaign tractable on CPU. The ten-dimensional synthetic benchmarks (Rastrigin-10D, Ackley-10D) use $C_{\max} = 60$, while the large tabular benchmarks (Covertime, YearPrediction-MSD) use $C_{\max} = 40$. All other hyperparameters remain identical to Table 13.

Family	Ladder
MLP (displayed subset of dense sweep)	w32 ₁ , w64 ₁ , w128 ₂ , w256 ₂ , w512 ₃ , w512 ₄ , 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 12: Capacity ladders swept for each reported baseline family.

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. 14)

Table 13: 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.

B. 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 tables (Tables 3 and 4) report only the validation-selected run per family and dataset. The tables below show the full displayed ladders and adaptive checkpoints.

Tables 14–15 and 17–18 list neural baselines (MLP, SoftTr) and tree-based baselines (RForest, XGBoost) separately for readability. Tables 16 and 19 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 $\pm$ 95% C.I. over 10 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 Section 4.1. The full per-model baseline ladders therefore omit timings and focus on accuracy and model size.

C. 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. The effect is illustrated 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 4.5.

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 Tables 10 and 11. When no smallest-equivalent MLP exists under the equivalence test of Section 4.5, the final figure uses the validation-best MLP instead. Tabular benchmarks are intentionally omitted because

Dataset	MLP							Soft Tr		
	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 14: 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.

Dataset	RForest					XGBoost		
	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 15: 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.

their input geometry is not directly visualizable.

Implicit regularization through partition growth. Figures 4 and 5 reflects the architectural prior of Section 3. A leaf is split only when its residuals indicate that one 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.

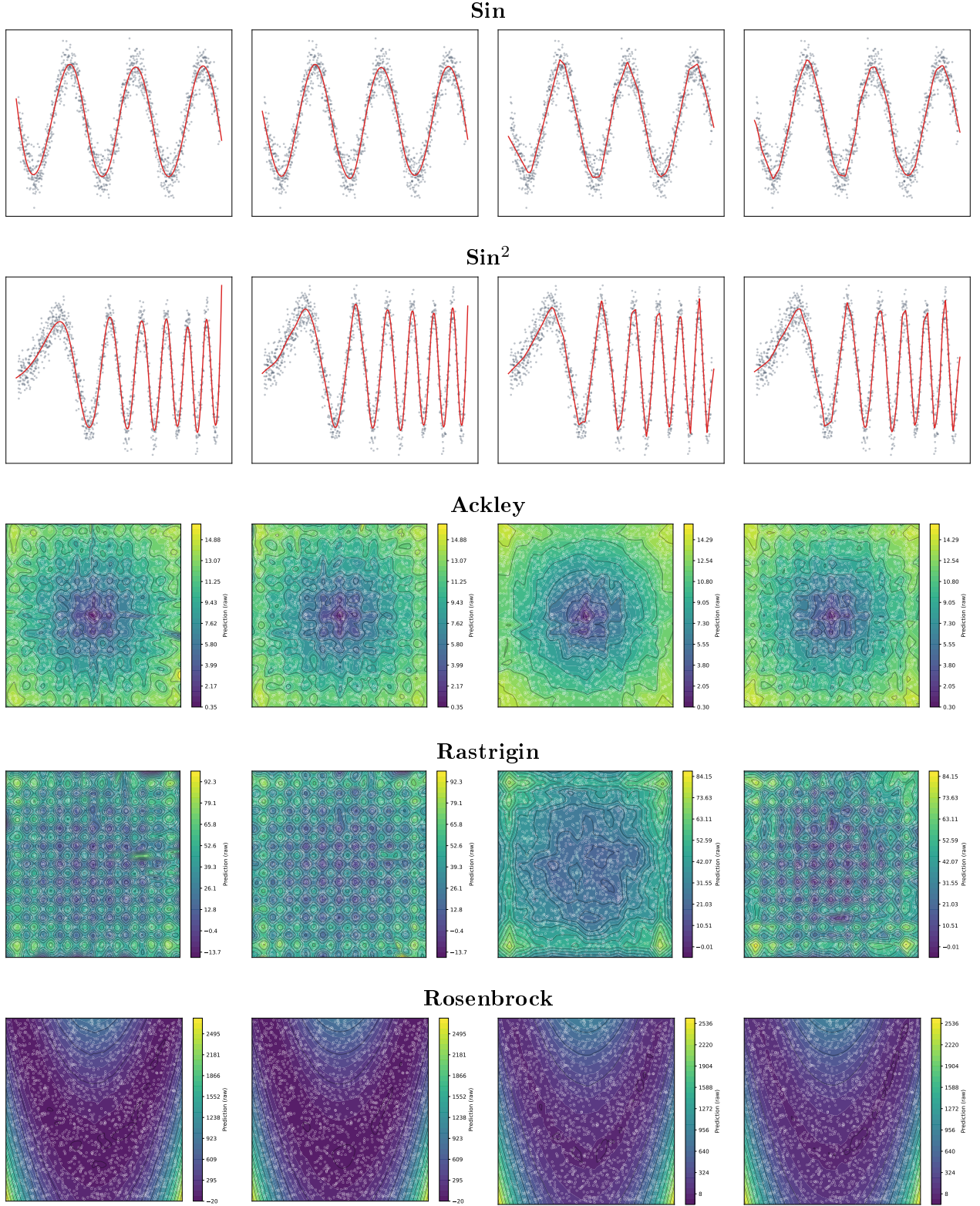


Figure 4: Qualitative prediction shapes on low-dimensional regression benchmarks. Within each dataset row, panels from left to right show: (a) SPAWN-LL, (b) SPAWN-QL, (c) the same-size MLP, and (d) the smallest-equivalent MLP; for Sin and Sin², where no smallest-equivalent MLP exists, panel (d) shows the validation-best MLP.

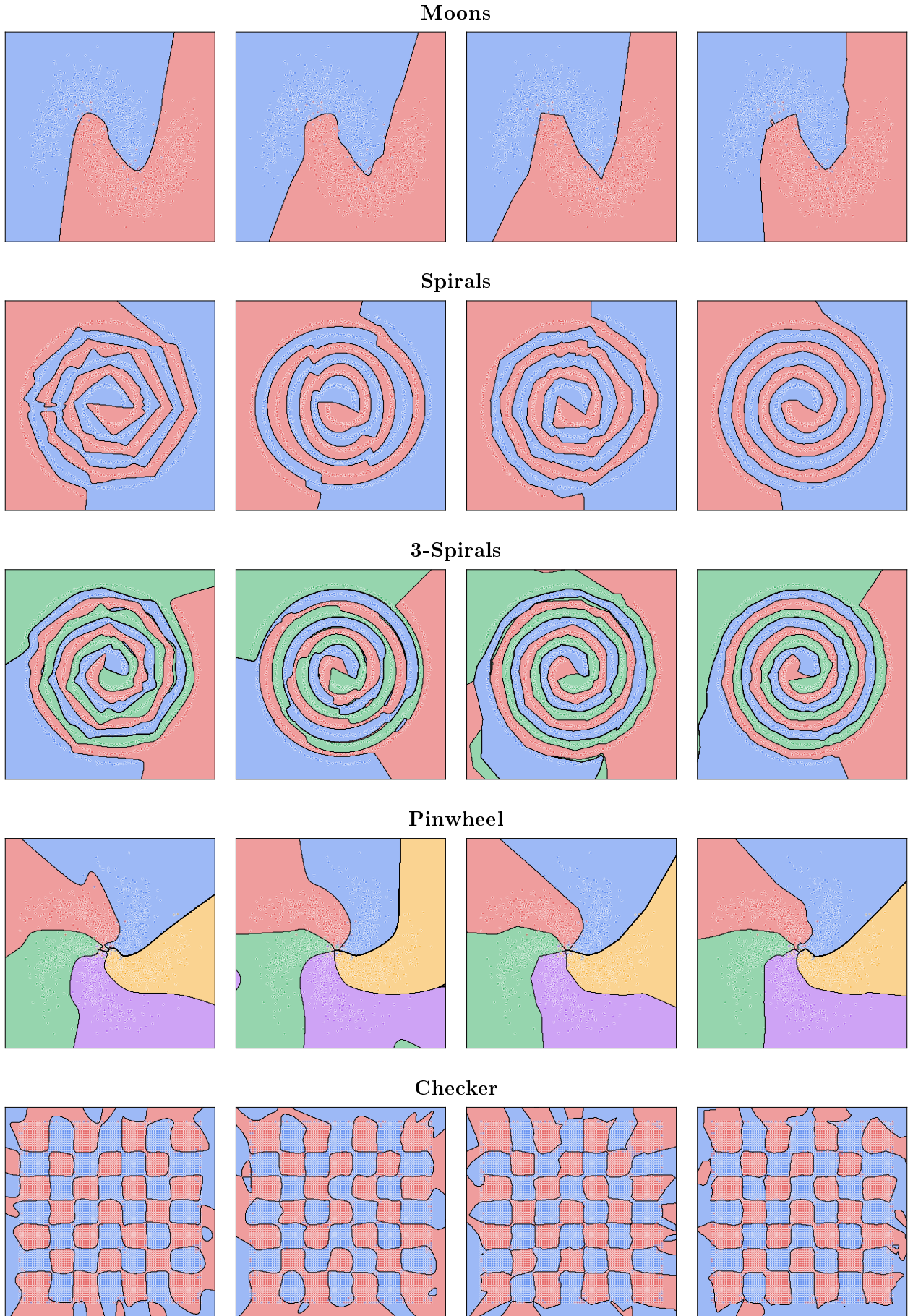


Figure 5: Qualitative decision regions on low-dimensional classification benchmarks. Within each dataset row, panels from left to right show: (a) SPAWN-LL, (b) SPAWN-QL, (c) the same-size MLP, and (d) the smallest-equivalent MLP; for Checker, where no smallest-equivalent MLP exists, panel (d) shows the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

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

Table 16: SPAWN growth trajectory on regression benchmarks. Each top entry reports RMSE (mean $\pm$ 95% CI); the row below reports model size in red followed by mean wall-clock training time in blue.

MLP					Soft Tr					
Dataset	w32 ₁	w64 ₁	w128 ₂	w256 ₂	w512 ₃	w512 ₄	w1k ₄	d2	d4	d8
Moons	97.0 $\pm$ 0.1 162	97.0 $\pm$ 0.1 322	97.3 $\pm$ 0.2 17.2k	97.5 $\pm$ 0.2 67.1k	97.3 $\pm$ 0.1 527.9k	97.0 $\pm$ 0.2 790.5k	97.0 $\pm$ 0.2 3.2M	94.5 $\pm$ 1.8 17	96.9 $\pm$ 0.1 77	97.1 $\pm$ 0.1 1.3k
Spirals	58.9 $\pm$ 1.0 162	64.0 $\pm$ 1.6 322	100.0 $\pm$ 0.0 17.2k	100.0 $\pm$ 0.0 67.1k	100.0 $\pm$ 0.0 527.9k	100.0 $\pm$ 0.0 790.5k	99.2 $\pm$ 0.9 3.2M	53.4 $\pm$ 0.5 17	57.1 $\pm$ 0.5 77	76.3 $\pm$ 1.3 1.3k
3-Spirals	45.1 $\pm$ 1.8 195	54.2 $\pm$ 1.6 387	100.0 $\pm$ 0.0 17.3k	100.0 $\pm$ 0.0 67.3k	100.0 $\pm$ 0.0 528.4k	100.0 $\pm$ 0.0 791.0k	99.7 $\pm$ 0.5 3.2M	36.1 $\pm$ 0.8 21	44.0 $\pm$ 1.9 93	91.5 $\pm$ 1.5 1.5k
Pinwheel	95.1 $\pm$ 0.1 261	94.9 $\pm$ 0.1 517	94.5 $\pm$ 0.2 17.5k	94.6 $\pm$ 0.1 67.8k	94.9 $\pm$ 0.3 529.4k	95.0 $\pm$ 0.3 792.1k	94.7 $\pm$ 0.3 3.2M	90.2 $\pm$ 0.3 29	95.0 $\pm$ 0.2 125	94.5 $\pm$ 0.2 2.0k
Checker	68.1 $\pm$ 1.8 162	78.7 $\pm$ 1.3 322	90.3 $\pm$ 0.3 17.2k	89.7 $\pm$ 0.6 67.1k	89.5 $\pm$ 0.7 527.9k	87.9 $\pm$ 2.2 790.5k	86.4 $\pm$ 2.5 3.2M	51.4 $\pm$ 0.3 17	51.7 $\pm$ 2.1 77	85.6 $\pm$ 0.5 1.3k
BreastCnc	95.4 $\pm$ 0.4 1.1k	95.7 $\pm$ 0.3 2.1k	95.6 $\pm$ 0.4 20.7k	95.4 $\pm$ 0.4 74.2k	95.6 $\pm$ 0.7 542.2k	95.2 $\pm$ 0.4 804.9k	95.0 $\pm$ 0.5 3.2M	95.9 $\pm$ 0.2 101	96.3 $\pm$ 0.3 497	97.0 $\pm$ 0.5 8.4k

Table 17: Per-model comparison on classification—neural baselines (MLP and soft decision trees). Each top entry reports the held-out accuracy in percent (mean $\pm$ 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
Breast Cnc	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 18: Per-model comparison on classification—tree-based baselines (random forests and XGBoost). Each top entry reports the held-out accuracy in percent (mean $\pm$ 95% CI); the row below reports model size in red.

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

Table 19: SPAWN growth trajectory on classification benchmarks. Each top entry reports accuracy in percent (mean $\pm$ 95% CI); the row below reports model size in **red** followed by mean wall-clock training time in **blue**.

Abstract in lingua italiana

La progettazione delle architetture neurali è comunemente trattata come un processo separato dall'ottimizzazione dei parametri, affidandosi tipicamente all'esperienza umana o a metodi di ricerca automatica. La progettazione manuale, tuttavia, richiede competenze fortemente legate al dominio applicativo e difficilmente trasferibili, mentre i metodi di ricerca automatica comportano spesso costi computazionali proibitivi. Per superare queste limitazioni, introduciamo la Self-Partitioning Adaptive Widening Network (SPAWN), un framework che apprende congiuntamente architettura e parametri aumentando dinamicamente la propria capacità rappresentativa durante l'addestramento. SPAWN parte da un singolo predittore elementare e costruisce incrementalmente una miscela di esperti con gating differenziabile, partizionando lo spazio degli input là dove è necessaria maggiore complessità. A differenza degli approcci che espandono la rete aggiungendo neuroni o strati, SPAWN accresce la capacità suddividendo ricorsivamente lo spazio degli input e producendo un albero differenziabile di esperti affini addestrato end-to-end. Criteri basati sui dati governano quando crescere, dove suddividere e come preservare le uscite del modello, consentendo un'ottimizzazione continua senza necessità di ri-addestramento. Le valutazioni empiriche su benchmark standard di regressione e classificazione dimostrano che SPAWN individua automaticamente architetture compatte in grado di eguagliare o superare le prestazioni predittive di modelli a capacità fissa significativamente più grandi, impiegando un numero notevolmente inferiore di parametri.

Parole chiave: Apprendimento Supervisionato, Apprendimento Automatico, Reti ad Espansione Adattiva, Metodi Adattativi, Architettura Neurale, MLP

Acknowledgements

Vorrei dedicare un ringraziamento speciale alla mia famiglia. A mia madre, per il suo sostegno incondizionato e l'energia inesauribile che mi ha donato durante tutto il mio percorso di studi. Ai miei nonni, che mi hanno incoraggiato ad ambire in grande e a non arrendermi mai, a valutare ogni fallimento come un'opportunità, una consapevolezza essenziale nella ricerca scientifica. A mio padre per avermi insegnato a ragionare in modo critico e per avermi avvicinato al mondo dell'informatica fin da piccolo. Ai miei fratelli per avermi sempre regalato momenti di svago e spensieratezza quando più ne avevo bisogno.

Inoltre un posto speciale va alla mia fidanzata, che ha preservato la mia salute mentale e la mia serenità, ricordandomi quanto sono importanti nella vita l'equilibrio e le persone oltre il loro lavoro.

Infine, un caloroso ringraziamento va al mio relatore Prof. Marcello Restelli e ai co-relatori Dott. Gianmarco Tedeschi e Prof. Matteo Papini per avermi dato l'opportunità di sperimentare la bellezza della ricerca scientifica in un contesto stimolante e collaborativo. Questo lavoro non sarebbe stato possibile senza il loro supporto e la loro fiducia nelle mie capacità.