SELF-PARTITIONING ADAPTIVE WIDENING NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

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

1 INTRODUCTION

The architecture of a learning model is a structural decision that fundamentally dictates its capacity to learn, generalize, and perform on a given task. Traditionally, the choice of this architecture relies on human knowledge and heuristics rather than systematic principles, often resulting in overparameterized models that are computationally expensive to train and deploy. While automated methods such as Neural Architecture Search (NAS, Elsken et al., 2019), network pruning (Liu et al., 2019c), and evolutionary algorithms (Stanley & Miikkulainen, 2002) have been proposed to mitigate the need for manual tuning, they introduce significant challenges of their own. These approaches typically treat architecture search as an explicit optimization over a discrete or relaxed space, isolated from parameter learning. 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 let the model incrementally increase its capacity during the training process. This concept has proven effective in several settings, including continual learning (Rusu et al., 2016), optimization (Caccia et al., 2022), and reinforcement learning (Berner et al., 2019; Tedeschi et al., 2025). Building on this intuition, a promising line of research focuses on dynamically growing neural networks by adding neurons or layers one at a time, guided by gradient-derived signals (Chen et al., 2016; Evci et al., 2022; Wu et al., 2020). However, while modern NAS and dynamic growth methodologies have produced impressive results, they mostly focus on deep neural networks. In these frameworks, the unit of growth is either a scalar parameter or a layer, resulting in architectures that inherit the complex inductive biases of deep networks. Conversely, traditional methods that dynamically grow shallow architectures, such as decision trees (Kontschieder et al., 2015) or gradient boosting (Chen & Guestrin, 2016), 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.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

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

We present *Self-Partitioning Adaptive Widening Network* (SPAWN), a novel neural architecture that departs from the standard deep-learning growth paradigm. Instead of expanding a deep network by adding hidden units or layers, we dynamically grow a differentiable, shallow tree of experts. Crucially, our fundamental unit of growth is a region of the input space, rather than an isolated parameter. By dynamically partitioning the data space based on task feedback, we ensure the network expands its capacity only where local data complexity demands it.

Starting with a simple architecture, SPAWN progressively builds capacity by splitting individual experts into children, each learning a distinct predictor over its newly generated sub-region. The gate and expert function classes can be chosen independently of the expansion procedure. In this work, we use affine experts with linear or quadratic gates, placing the model's non-linear expressivity in the routing structure. The structural evolution of SPAWN is governed by three data-driven criteria: *when* to expand is triggered only after the current architecture's training loss has plateaued; *where* to expand targets the specific input regions exhibiting the greatest within-region loss variation; and *how* to expand guarantees strict output invariance, allowing gradient-based training to resume seamlessly after every split. The resulting process produces architectures whose capacity is determined by the data rather than by the practitioner.

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

2 Architecture

SPAWN learns a predictor $f_\theta : \mathbb{R}^d \to \mathbb{R}^p$ for supervised problems, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathbb{R}^d$ denoting the input features and y_i taking values in $\mathbb{R}^p$ for regression or in a finite label set for classification.[1] We construct f_θ as a mixture of E experts, partitioned across the input space by $G = E - 1$ soft gates arranged in a binary tree, so that each expert contributes according to its responsibility for the input. Let $\mathcal{E} = \{1, \dots, E\}$ and $\mathcal{G} = \{1, \dots, G\}$ denote the expert and gate index sets, respectively. For each expert $h \in \mathcal{E}$, let $g_h(x) \in [0, 1]$ denote the gating weight assigned by the model, with $\sum_{h\in\mathcal{E}} g_h(x) = 1$, and let $e_h(x; \phi_h) \in \mathbb{R}^p$ denote its output, parameterized by ϕ_h. For brevity, we write $e_h(x)$ instead of $e_h(x; \phi_h)$. The model prediction is then

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

For regression, $f_\theta(x)$ is the predicted target, while for classification, the experts output class logits and softmax is applied after their weighted sum. In Section 2.1 we detail how the gating weights $g_h(x)$ are computed from the tree structure. Section 2.2 introduces the gate and expert implementations evaluated in this work and Section 2.3 defines the routing regularizers. Throughout this section, the number of experts E is held fixed. In Section 3 we then introduce the adaptive procedure that grows E during training.

2.1 Soft-Gated Tree of Experts

The predictor f_θ combines E experts using a soft gating mechanism that computes gating weights $g_h(x)$, structured as a binary tree with internal nodes as gates and leaves as experts. The tree structure localizes expansion: each new sub-tree subdivides its parent's routing weight, preserving the weights of the other experts at insertion. This follows the hierarchical Mixture-of-Experts (MoE) tradition of

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

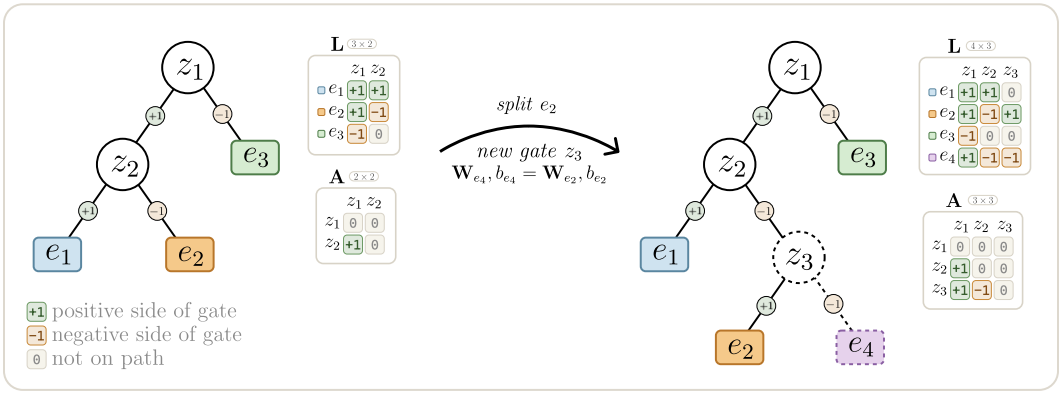


Figure 1: Expansion of the SPAWN soft-gated binary tree from $E = 3$ to $E = 4$ experts (left to right), illustrated with affine experts. Splitting one leaf adds a gate and two children. Each child's gating weight is the product of branch probabilities along its root-to-leaf path (Equation 4).

Jordan & Jacobs (1994) and stands in contrast to flat softmax routing of modern large-scale MoE models (Shazeer et al., 2017; Fedus et al., 2022), where experts compete globally for gating weight and adding a new expert changes the common normalization.

Gate. A single gate $k \in \mathcal{G}$ is a differentiable function parameterized by ψ_k,

$$p_k(\,\cdot\,;\psi_k) : \mathbb{R}^d \to (0,1). \tag{2}$$

The quantity $p_k(x)$ can be interpreted as the degree to which input x is assigned to the positive branch of gate k; correspondingly, $1 - p_k(x)$ represents the degree to which x is assigned to the negative branch. The tree construction depends on these complementary weights, while the gate parameterization determines the shape of the routing boundary.

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

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

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

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

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

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

Experts. Each expert $e_h(x;\phi_h)$ maps the input to the common output space $\mathbb{R}^p$, following the MoE formulation (Jacobs et al., 1991). The experts are trained jointly with the gates by gradient descent. Their function class can be chosen independently of the gate family, provided that they support this optimization and can be copied into two independently trainable children with the same initial output, as required by the output invariance detailed in Section 3.3.

Mixture output. Combining the two components, the forward pass computes f_θ as in Equation 1; Figure 2 illustrates the computation for linear gates and affine experts. The trainable parameters are $\theta = \{\psi_k\}_{k \in \mathcal{G}} \cup \{\phi_h\}_{h \in \mathcal{E}}$.

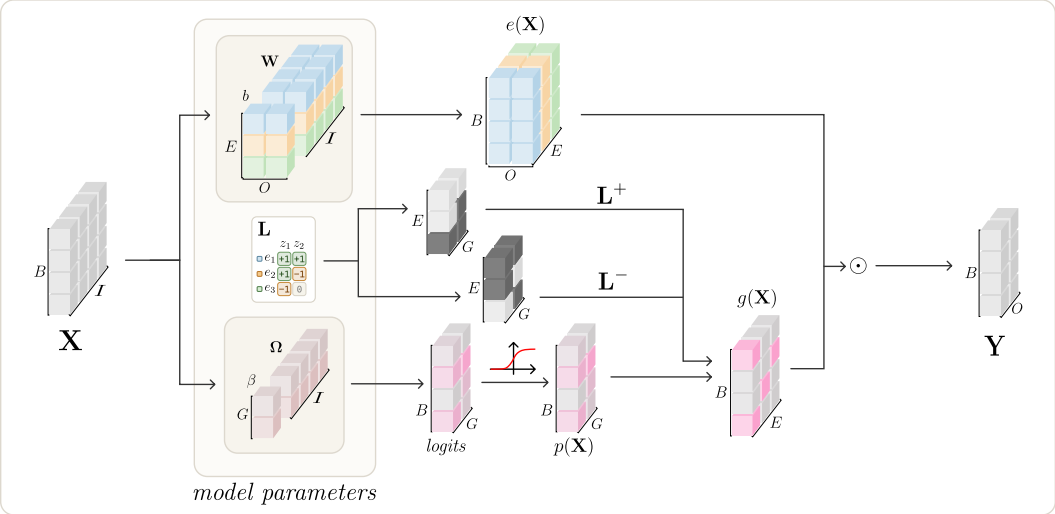


Figure 2: Forward pass for the soft-gated tree with linear gates and affine experts. Gates and experts process the input in parallel, the link matrix combines gate probabilities in log space to obtain $g_h(x)$ (Equation 4), and the prediction is the gating-weighted sum of expert outputs (Equation 1).

2.2 Gate and Expert Implementations

The expansion procedure of SPAWN is independent of the function class used in each gate and expert. In this work, we use affine experts $e_h(x) = \mathbf{W}_h x + b_h$, with $\mathbf{W}_h \in \mathbb{R}^{p\times d}$ and $b_h \in \mathbb{R}^p$, and compare two gate families. Both use $p_k(x) = \mathrm{sigmoid}(z_k(x))$, with either a linear logit,

$$z_k(x) = \omega_k^\top x + \beta_k, \qquad \omega_k \in \mathbb{R}^d, \quad \beta_k \in \mathbb{R}, \tag{5}$$

or a quadratic logit,

$$z_k(x) = \omega_k^\top x + \alpha_k^\top (x \odot x) + \beta_k, \qquad \alpha_k \in \mathbb{R}^d, \tag{6}$$

where $x \odot x$ denotes coordinate-wise squares. The quadratic terms admit curved routing boundaries without introducing cross-coordinate products. We denote these implementations by SPAWN-LL and SPAWN-QL, respectively. Both are featured in the experiments presented in Section 4.2.

With affine experts, the non-linearity of f_θ resides in the routing structure. Under soft routing, both implementations yield smooth non-linear predictors. In the hard-routing limit, the linear-gate variant becomes piecewise affine over a polyhedral partition. In the same limit, quadratic gates change the shape of the regions while retaining affine predictions within each one. Keeping the expert family fixed allows us to study the effect of richer routing on the capacity required by the model.

2.3 Regularization Losses

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

$$\mathcal{L}(\theta) = \mathcal{L}_t(\theta) + \lambda_s \mathcal{L}_s(\theta) + \lambda_m \mathcal{L}_m(\theta), \tag{7}$$

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

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

& Bengio, 2004; Pereyra et al., 2017). We minimize the batch-averaged entropy, normalized by its maximum value:

$$\mathcal{L}_s(\theta) = \frac{1}{|\mathcal{B}| \log E} \sum_{x \in \mathcal{B}} H_s(x) \ \in \ [0, 1]. \tag{8}$$

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

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

$$\bar{p}_k = \frac{\sum_{x \in \mathcal{B}} m_k(x)\, p_k(x)}{\sum_{x \in \mathcal{B}} m_k(x)}, \tag{9}$$

where a balanced gate has $\bar{p}_k \approx \frac{1}{2}$ and a collapsed gate has $\bar{p}_k \in \{0, 1\}$. We enforce balance by penalizing the deviation of the per-gate binary entropy $H_b(p) = -p \log p - (1 - p) \log(1 - p)$ from its maximum $\log 2$, averaged across gates:

$$\mathcal{L}_m(\theta) = 1 - \frac{1}{G \log 2} \sum_{k \in \mathcal{G}} H_b(\bar{p}_k) \ \in \ [0, 1]. \tag{10}$$

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

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

3 Adaptive Expansion

The architecture in Section 2 assumes a fixed number of experts E, but SPAWN incrementally adds experts during training. With $E = 1$ and $g_1(x) = 1$, Equation 1 reduces to a single predictor. Hierarchical gating allows a leaf to be replaced by a small subtree, while the new experts inherit their parent's parameters so that expansion preserves the current prediction. Each expansion involves three decisions: *when* to expand (Section 3.1), *where* to expand, i.e., which expert to split (Section 3.2), and *how* to split so training continues smoothly (Section 3.3). Section 3.4 combines these into the outer expansion loop that yields the final model. We present the algorithm pseudocode in Algorithm 1

Algorithm 1 SPAWN

Input: Dataset $\mathcal{D}$, weights λ_s, λ_m, plateau window k, threshold η, cycle cap $C_{\max}$, patience P
Output: Retained model $f_{\theta^\star}$

```
Initialize f_θ with one gate and two experts
for c = 1, ..., C_max do
    Reset optimizer and learning-rate schedule
    repeat
        Train one epoch on L                                  ▷ Eq. 7
        θ* ← θ if the validation loss is the best so far
    until training loss plateaus                              ▷ Sec. 3.1
    if no validation improvement for P cycles then
        break
    end if
    h* ← arg max_{h∈E} s_h                                    ▷ Eq. 12
    Split h* into two copies                                  ▷ Eq. 13
end for
Restore θ* and fine-tune without splits
return f_θ*
```

3.1 When to Expand: Plateau Detection

We trigger expansion once the training loss has plateaued, since adding capacity is only useful after the model has extracted as much signal as its current size allows. Expanding earlier would add noise to the optimization residual, leading to structural underfitting and biasing the expert-selection criterion in Section 3.2 toward regions that have not yet finished training. Plateau is detected by

maintaining a length-$2k$ buffer of recent epoch losses and comparing the means of its older and newer halves. When the relative improvement (clipped at zero) falls below a threshold η, the criterion triggers. We use training rather than validation loss here so that the trigger reflects parameter convergence, not capacity sufficiency. Capacity is judged by the outer cycle-level stopper, which retains the validation-best checkpoint across all cycles and epochs (Section 3.4).

3.2 Where to Expand: Within-Region Heterogeneity

Average per-expert loss alone does not distinguish regions that would benefit from expansion: a region whose residuals reflect irreducible noise can carry high loss that no further partitioning will reduce. Instead, we use within-region loss variation to prioritize regions where the current predictor fits some samples better than others. This heuristic selects an expert to split, works with either gate family and different expert implementations, and uses a sum rather than an average so that, all else equal, populous regions are refined before sparsely sampled ones.

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

$$\bar{\ell}_h(\mathcal{B}) \;=\; \frac{\sum_{x\in\mathcal{B}} g_h(x)\,\ell(x)}{\sum_{x\in\mathcal{B}} g_h(x)}, \tag{11}$$

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

$$s_h \;=\; \sum_{\mathcal{B}} \sum_{x\in\mathcal{B}} g_h(x)\,\big|\ell(x) - \bar{\ell}_h(\mathcal{B})\big|. \tag{12}$$

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

3.3 How to Expand: Output-Preserving Inheritance

Once the expert $h^\star$ has been selected, we introduce a new gate that partitions its region into two children and initialize all new parameters so that the model's output at the moment of expansion is exactly preserved. Training resumes from the same functional state, with the regularizers in Section 2.3 guiding subsequent differentiation among children. Let h^+ and h^- denote the positive and negative children of $h^\star$ under the new gate k'. The two children inherit the parent's expert parameters exactly, $\phi_{h^+} = \phi_{h^-} = \phi_{h^\star}$, and therefore initially compute the same predictor. Their path weights are $g_{h^+}(x) = g_{h^\star}(x)p_{k'}(x)$ and $g_{h^-}(x) = g_{h^\star}(x)(1-p_{k'}(x))$. Their joint contribution to the model output is then

$$g_{h^+}(x)\,e_{h^+}(x) + g_{h^-}(x)\,e_{h^-}(x) \;=\; g_{h^\star}(x)\,e_{h^\star}(x), \tag{13}$$

since the two branch weights sum to one. This identity holds for any initialization of the new gate and any expert family whose predictor can be copied exactly. For the affine experts evaluated here, inheritance simply copies $\mathbf{W}_{h^\star}$ and $b_{h^\star}$ into both children. After each split, all gates and experts, including the existing ones, are jointly retrained; no existing parameters are frozen.

The initialization of the new gate depends on its function class. For linear gates, ω' is sampled uniformly from a small hypercube and $\beta' = -{\omega'}^\top \mu_{h^\star}$, so that the decision hyperplane passes through the gating-weighted centroid $\mu_{h^\star}$ of samples routed to $h^\star$. For quadratic gates, we initialize both coefficient vectors randomly and set the bias to zero. In either case, the link matrix $\mathbf{L}$ is updated to attach the two children to the parent's path (Figure 1); the gate-ancestry matrix used to compute $m_k(x)$ is updated analogously (Appendix B.1).

3.4 The Expansion Cycle

In the previous section, we defined each step of the expansion cycle (the loop in Algorithm 1). In each cycle, the model is trained until the training loss plateaus, and then one expert is split. Every

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

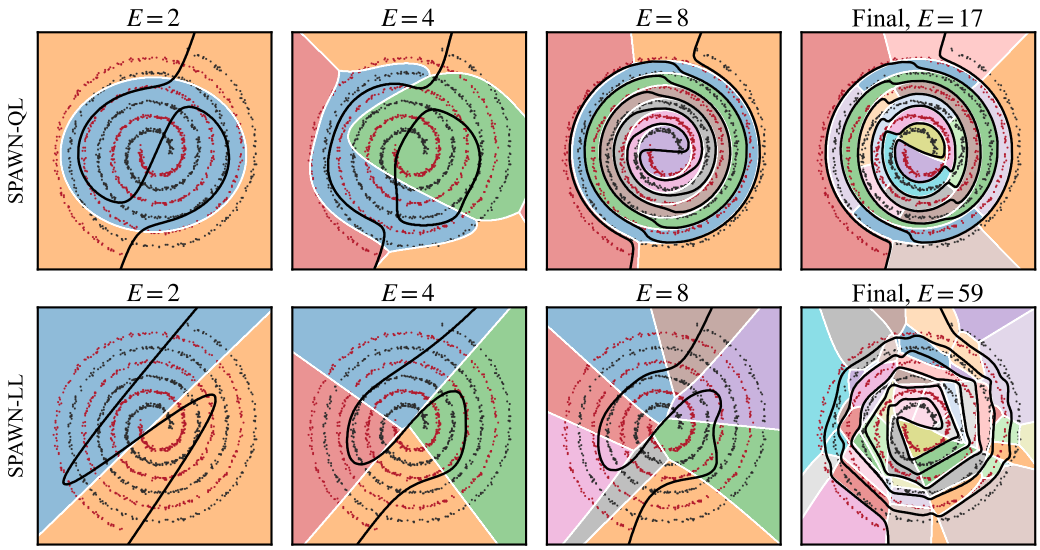


Figure 3: SPAWN expert partition on Spirals (same settings as Table 1). The columns show, from left to right, the progression of growing from two experts ($E = 2$) to the final architecture shape, for both variants (QL-top and LL-bottom). Each region is colored by its dominant expert. Dots are training points colored by class, and black curves are decision boundaries.

split adds new parameters, so we reset the optimizer state and the learning-rate schedule at the start of each cycle. After every epoch, we check the validation loss and keep the checkpoint with the lowest value seen so far. Growth stops when the validation loss has not improved for P consecutive cycles (the growth patience), or after $C_{\max}$ cycles. The validation loss only decides which checkpoint we keep. The expert to split is selected based on the training-loss scores s_h in Equation 12.

When growth stops, we restore the best checkpoint, which fixes the final architecture. We then fine-tune all its parameters without further splits. Fine-tuning uses a separate learning rate, usually smaller than the one used during growth, and early stopping on the validation loss. We call the resulting model the *retained model*.

4 Experiments

Notation. We denote an MLP by wW_d, where d is the number of hidden layers and W is the number of units per layer; all hidden layers share the same width. We denote an XGBoost ensemble by tT_d, where T is the maximum number of boosting rounds and d the maximum tree depth.

4.1 Experimental Setup

We evaluate whether SPAWN, starting from a single shallow predictor, discovers compact architectures that match or surpass the predictive performance of larger fixed-size and growing models. Section 4.2 compares SPAWN with the best configuration of each baseline family on classification and regression benchmarks. For more detail on the choice of the baseline architectures, refer to Appendix E. The benchmark comprises synthetic tasks in two dimensions (Spirals, Checkerboard and Rastrigin (Törn & Žilinskas, 1989)), small tabular datasets (Breast Cancer (Wolberg et al., 1993), Diabetes (Efron et al., 2004), kin8nm (Neal, 1996) and CCPP (Tüfekci, 2014)), and larger datasets with hundreds of features or hundreds of thousands of examples (UCI HAR (Anguita et al., 2013), MNIST (LeCun et al., 1998), Covertype (Blackard & Dean, 1999) and CT Slice (Graf et al., 2011)). Each experiment is averaged over ten seeds.

We compare SPAWN with five baseline families. MLPs (Rumelhart et al., 1986) are the reference, fixed-size neural model. XGBoost (Chen & Guestrin, 2016) is a strong gradient boosting method and is closely related to SPAWN, as it also partitions the input space into regions, each with its own prediction. Budding Trees (Irsoy et al., 2014) are soft decision trees (Irsoy et al., 2012) that

		Classic baselines		Growing baselines			SPAWN
	Dataset	MLP	XGBoost	Budding	GradMax	Firefly	Best variant
Classification	Spirals	100.00 ±0.00 8.6k ($w64_3$)	97.67 ±0.67 33.0k ($t1000_8$)	52.83 ±2.03 143	99.98 ±0.05 17.2k	99.75 ±0.28 1.3k	**99.88 ±0.16** **248** (QL)
	Checkerboard	91.09 ±0.52 2.9k ($w576_1$)	92.83 ±0.47 39.1k ($t1000_8$)	51.80 ±0.16 110	85.65 ±3.72 17.2k	89.70 ±0.47 1.3k	**91.97 ±0.54** **1.0k** (LL)
	Breast cancer	97.54 ±0.93 197 ($w5_2$)	96.75 ±0.84 1.8k ($t300_4$)	98.16 ±0.69 891	98.25 ±0.98 1.0k	98.33 ±0.81 2.0k	**97.54 ±1.21** **192** (LL)
	UCI HAR	**93.52 ±0.56** **5.2k** ($w9_2$)	93.50 ±0.53 24.0k ($t300_4$)	92.81 ±0.40 35.8k	93.24 ±0.73 22.2k	93.53 ±0.70 18.2k	93.71 ±0.74 13.3k (QL)
	MNIST	97.40 ±0.17 225.0k ($w283_1$)	**97.85 ±0.04** **511.3k** ($t1200_8$)	88.69 ±1.87 24.7k	96.96 ±0.28 118.3k	97.27 ±0.08 101.1k	96.66 ±0.13 262.6k (LL)
	Covertype	**94.40 ±0.11** **129.7k** ($w330_2$)	94.42 ±0.07 1.35M ($t1000_8$)	— —	90.35 ±0.24 8.1k	85.36 ±0.09 4.0k	92.67 ±0.13 42.4k (LL)
Regression	Rastrigin	0.907 ±0.023 17.0k ($w128_2$)	0.993 ±0.001 89.7k ($t1200_8$)	0.217 ±0.104 241	0.675 ±0.046 4.4k	0.875 ±0.031 4.1k	**0.988 ±0.005** **3.6k** (QL)
	Diabetes	0.449 ±0.055 277 ($w23_1$)	0.389 ±0.056 1.6k ($t300_4$)	0.421 ±0.053 195	0.430 ±0.051 1.4k	0.456 ±0.044 169	**0.459 ±0.054** **43** (QL)
	Kin8nm	0.931 ±0.002 134.1k ($w256_3$)	0.817 ±0.005 263.0k ($t1000_8$)	0.894 ±0.004 693	0.927 ±0.004 4.7k	0.924 ±0.003 3.9k	**0.929 ±0.004** **736** (LL)
	CT slice	0.960 ±0.009 56.5k ($w113_2$)	0.931 ±0.013 241.4k ($t1200_8$)	0.935 ±0.017 23.1k	0.954 ±0.012 65.9k	0.927 ±0.020 45.1k	**0.937 ±0.017** **19.2k** (LL)
	CCPP	0.956 ±0.003 133.1k ($w256_3$)	**0.970 ±0.003** **167.9k** ($t1000_8$)	0.943 ±0.004 267	0.952 ±0.003 4.5k	0.953 ±0.003 6.1k	0.950 ±0.004 575 (QL)

Table 1: Test comparison on six classification and five regression datasets. Top: mean test accuracy (%) or R^2 over ten seeds, with the half-width of the 95% CI. Bottom: mean retained size (red) and model ID. Gray entries have CIs that do not overlap the CI of the best mean, and the shaded entry is the smallest model among the others. Dashes mark families that were not evaluated.[4]

grow during training and are optimized via gradient descent, making them the closest compact relative of SPAWN. Finally, GradMax (Evci et al., 2022) and Firefly (Wu et al., 2020) are the neural growing methods closest to SPAWN, as they also grow a network in width by widening the hidden layers of a small initial network during training, with neurons chosen based on gradient information. For each family, we train several candidate configurations per dataset, varying the architecture and hyperparameters, and keep the configuration with the lowest validation loss averaged over the ten seeds. For SPAWN, the gate family in Section 2.2 is treated as an additional hyperparameter in this selection, so each dataset uses either linear or quadratic gates. We report the test accuracy or R^2 of the selected configuration and its retained size.

All experiments were run on a single machine with an Intel Core i5-14600K CPU and an NVIDIA RTX 4060 Ti GPU. Datasets and baselines are detailed in Appendix C, and the hyperparameter search, with the configuration kept for each dataset, is reported in Appendix E. Additional results are presented in Appendix D.

4.2 Experimental Results

In Table 1 we report the means across the ten seeds with 95% confidence intervals (CIs). Entries whose CI does not overlap that of the best mean are shown in gray, and among the remaining ones, which we call competitive, we shade the smallest model,[3] since equal predictive quality with fewer parameters means lower memory and inference cost. Table 3 reports both gate families of SPAWN.

On most datasets, SPAWN is competitive with the best model at a smaller size. SPAWN is competitive on eight of the eleven datasets and is the smallest competitive model on seven of them. It

[3]XGBoost sizes are node counts, which are not comparable with parameter counts.

[4]We did not run Budding Trees on Covertype, because their training time on this dataset exceeded our compute budget. We therefore mark the family as not applicable there.

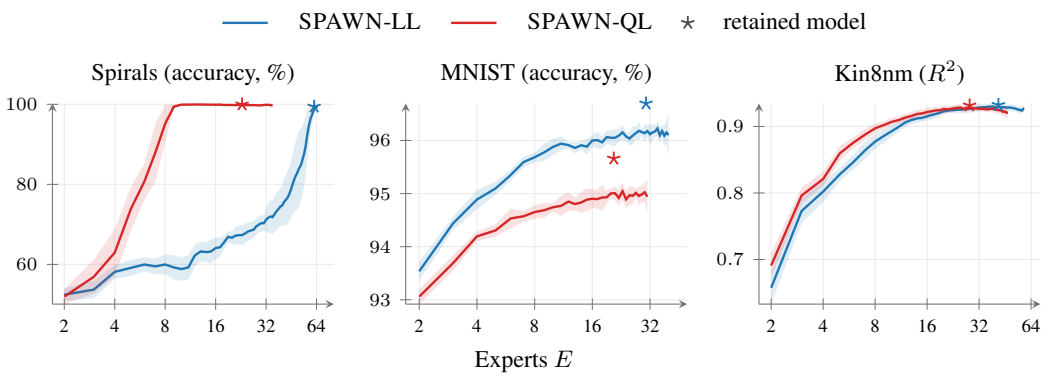


Figure 4: Mean validation score of the kept SPAWN-LL and SPAWN-QL configurations on Spirals, MNIST and kin8nm against the number of experts E (log scale). Curves show the model at the end of each growth cycle, averaged over the seeds that reached that size, and bands show 95% CIs of the mean. Stars mark the retained models of Table 1, i.e., the validation-best checkpoints after fine-tuning (Section 3.4), at their mean number of experts.

has the highest mean on UCI HAR and Diabetes. On Rastrigin, it is the only neural model competitive with XGBoost, with 3.6k parameters against 89.7k nodes. GradMax and Firefly, which also expand the network in width, are never the smallest competitive models. On the remaining three datasets, SPAWN stays close to the best mean: 1.2 points behind on MNIST, 1.8 points behind on Covertype, and 0.02 points behind in R^2 on CCPP. On Covertype, it uses a third of the parameters of the competitive MLP and is the most accurate growing model by more than two points, and on CCPP it is at least seven times smaller than any model with a higher mean. More specifically, this performance difference is also justified by the structure of the two models. The hidden units of an MLP compute features shared across the whole input space, which pays off when many input features interact, as in these datasets. SPAWN allocates a separate expert to each region, which is most effective when the structure varies locally in few input dimensions, as in interleaved spirals, checkerboard boundaries, and oscillating surfaces. Since the expansion procedure is agnostic to the function class of the experts, richer experts can also bring shared features into SPAWN (Section 2.2).

To visually illustrate the effects of the network expansion, we present the growth of SPAWN on Spirals in Figure 3. With quadratic gates, splits target the region where the arms are still mixed, forming rings that follow them inward, while linear gates form polygonal cells, and quadratic gates reach the same accuracy with less than a third of the experts. Moreover, in Figure 4 we show how the validation score evolves as experts are added on Spirals, MNIST, and kin8nm. The curves track the model at the end of each growth cycle, and the stars mark the retained models. The curves continue past the stars because growth stops only once the growth patience has run out, and the stars can lie above the curves because the retained model is the best checkpoint of any epoch, fine-tuned after growth (Section 3.4). We report a detailed graphical analysis for each dataset in Appendix D.

5 Conclusion

We introduced the Self-Partitioning Adaptive Widening Network (SPAWN), a novel neural architecture that unifies parameter optimization and architectural design by dynamically increasing its capacity during training. Through extensive evaluation, we demonstrated that SPAWN can devise efficient model structures that match or exceed the performance of substantially larger, fixed-capacity baselines across standard regression and classification tasks. Despite its parameter efficiency, SPAWN incurs computational overhead during its expansion phase, as a newly introduced predictor must be optimized before the next expansion. However, hierarchical mixing of experts enables concurrent evaluation of underfit regions, thereby further reducing the time burden. 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.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology, support in writing the code for the generation of synthetic datasets. We have not used generative AI tools for developing theoretical models or conceptual frameworks, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for grammar checking and support in writing the manuscript, support in writing the codebase for the proposed approach, and searching for information. We have reviewed all AI-assisted work. We checked LLM-generated edited and suggested text and carefully checked the code for correctness by all the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ethem Alpaydin and Cenk Kaynak. Cascading classifiers. *Kybernetika*, 34(4):369–374, 1998.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 437–442, 2013.

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.

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.

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.

Lucas Caccia, Jing Xu, Myle Ott, Marcaurelio Ranzato, and Ludovic Denoyer. On anytime learning at macroscale. In *Conference on Lifelong Learning Agents*, pp. 165–182. PMLR, 2022.

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*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016.

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.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. GradMax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022.

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

Mark Fanty and Ronald Cole. Spoken letter recognition. In *Advances in Neural Information Processing Systems*, volume 3, pp. 220–226, 1990.

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.

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.

Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Sebastian Pölsterl, and Alexander Cavallaro. 2D image registration in CT images using radial image descriptors. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 6892 of *Lecture Notes in Computer Science*, pp. 607–614, 2011. doi: 10.1007/978-3-642-23629-7_74.

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

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

Ozan Irsoy, Olcay Taner Yildiz, and Ethem Alpaydin. Budding trees. In *22nd International Conference on Pattern Recognition*, pp. 3582–3587, 2014. doi: 10.1109/ICPR.2014.616.

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.

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.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1467–1475. IEEE, 2015. doi: 10.1109/ICCV.2015.172.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a.

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, 2019b.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019c.

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.

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.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. ICLR 2017 Workshop, 2017.

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.

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.

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

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

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*, pp. 6166–6175. PMLR, 2019.

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.

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

Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014. doi: 10.1016/j.ijepes.2014.02.027.

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, pp. 10015–10023. AAAI Press, 2021. doi: 10.1609/aaai.v35i11.17202.

William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast cancer wisconsin (diagnostic). UCI Machine Learning Repository, 1993.

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.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

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

A Related Work

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

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

Tree-Based and Partitioning Models. A parallel line of work embeds tree structure into differentiable models, including soft decision trees (Irsoy et al., 2012) and Deep Neural Decision Forests (Kontschieder et al., 2015). While models like those of Frosst & Hinton (2017) use similarly to SPAWN soft routing gates to determine the policy associated to the leaves, their tree depth is fixed in advance, and their leaves emit static class distributions rather than input-dependent predictions. Budding trees (Irsoy et al., 2014) instead learn the tree structure during training, with each node interpolating between a leaf and an internal node through a soft leafness parameter whose penalty controls growth. Their leaves, however, still emit constant responses, and the tree grows as a consequence of this penalty rather than through an explicit decision on when and where to split. Most closely related to our growth mechanism are Adaptive Neural Trees (ANTs, Tanno et al., 2019), which also expand during training. We differ from ANTs on three major points: (i) ANT routers and leaves can be deep neural networks, whereas our gates and experts also admit linear solutions, locating the non-linear capacity entirely in the partition. (ii) ANT growth is driven by validation-based search, whereas ours relies purely on training-time signals. (iii) ANT trains new modules locally, while we train the full model jointly at every cycle, achieving smooth continuation via output-preserving parameter inheritance. Closer in inductive bias to our architecture is the family of Gated Linear Networks (GLNs, Veness et al., 2021; Budden et al., 2020; Li & Sompolinsky, 2022). A GLN is a feed-forward stack of neurons in which each neuron stores a bank of linear weight vectors and selects among them via a halfspace context function, with predictions combined across layers by gated geometric mixing and trained locally via online convex optimization at each neuron. Yet, unlike GLNs, which rely on a (deep) fixed stack of randomly sampled gates trained locally, our architecture discovers a shallow, learned binary tree optimized globally end-to-end.

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

number of experts grows. This scale-invariance is crucial for continuous expansion and represents a requirement previously unaddressed by fixed-size MoE architectures.

B Regularization Losses: Extended Derivation

This section provides additional details on the entropy-based regularizers introduced in Section 2.3. We first formalize the closed-form computation of the per-gate path weights $m_k(x)$ via a gate-ancestry matrix, and then expand on the comparison with related formulations from the soft-tree and sparsely-gated MoE literature.

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

Recall from Section 2.3 that $m_k(x) \in [0, 1]$ denotes the soft path weight of sample x reaching gate k, defined as the product of $p_j(x)$ and $1 - p_j(x)$ factors along the root-to-k path. We encode this path structure in a *gate-ancestry matrix* $\mathbf{A} \in \{-1, 0, +1\}^{G \times G}$, analogous to the link matrix $\mathbf{L}$ of Section 2 and read off the tree in the same way (Figure 1):

$$\mathbf{A}_{kj} = \begin{cases} +1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its positive side,} \\ -1 & \text{if gate } j \text{ is an ancestor of } k \text{ and } k \text{ lies on its negative side,} \\ 0 & \text{otherwise (including } j = k\text{).} \end{cases} \tag{14}$$

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

$$\log m_k(x) = \sum_{j \in \mathcal{G}} \mathbf{A}^+_{kj} \log p_j(x) + \mathbf{A}^-_{kj} \log\big(1 - p_j(x)\big), \tag{15}$$

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

B.2 Comparison with Related Formulations

Soft decision trees. The per-gate balance penalty of Equation 10 is closely related to the regularizer of Frosst & Hinton (2017), which penalizes the cross-entropy of the path-weighted split ratio α_i against the uniform target $(0.5, 0.5)$. Their α_i corresponds exactly to our $\bar{p}_k$, and both penalties are minimized by the balanced split $\bar{p}_k = \frac{1}{2}$. The two penalties are however not equivalent. Their cross-entropy equals $\log 2 + \mathrm{KL}(u \,\|\, \alpha_i)$, where u is the uniform distribution, and diverges as a gate collapses. Our penalty $\log 2 - H_b(\bar{p}_k) = \mathrm{KL}(\bar{p}_k \,\|\, u)$ is instead bounded by $\log 2$, which allows the normalization described in Appendix B.3. The two formulations differ in how they handle tree depth. In Frosst & Hinton (2017) each gate's contribution is multiplied by an explicit 2^{-d} factor that down-weights deeper gates a priori, while our formulation encodes tree geometry directly through the path weights $m_k(x)$ in the definition of $\bar{p}_k$ (Equation 9). Each gate is thus weighted by the actual fraction of the data it routes, rather than by a fixed function of its depth, which adapts naturally to the unbalanced expansion trajectories produced by the data-driven splitting criterion of Section 3, where deeper sub-trees may still receive substantial mass.

Sparsely gated MoE. A related load-balancing objective appears in the auxiliary losses of sparsely gated MoE architectures (Shazeer et al., 2017; Fedus et al., 2022), where the goal is to prevent a small number of experts from dominating routing under a flat softmax. There, balance is enforced globally over the full E-way expert distribution. Our per-gate formulation differs in two ways. (i) It is *local*, i.e., it applies soft pressure to each binary split independently. (ii) It is *hierarchical*, i.e., it exploits the tree structure to evaluate balance at the level of individual partitions rather than over the leaf set. As argued in Section 2.3, evaluating balance per gate avoids conflating shallow gates, which partition large regions, with deep gates, which partition small sub-regions, while still allowing the task loss to override the balance bias when an asymmetric split materially improves prediction.

B.3 Normalization and Invariance to Expansion

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

B.4 Task-Loss Gradients at Expansion

The output-preserving inheritance of Section 3.3, formalized by Equation 13, also helps explain the role of the routing regularizers: at insertion, the task-loss gradient on the parameters of the newly inserted gate is identically zero. This property does not depend on the gate or expert family. We make it precise below.

Adopting the notation of Section 3.3, let k' be the new gate with parameters $\psi_{k'}$ and h^+, h^- its two children inheriting $\phi_{h^\star}$. Let $\theta' = \{\psi_{k'}, \phi_{h^+}, \phi_{h^-}\}$ denote their parameters, and write $\cdot|_0$ for evaluation at this initialization. The gate and the two experts have separate parameters.

Proposition B.1. *For differentiable gates and experts, every input x, and every task loss $\mathcal{L}_t$ differentiable in $f_\theta(x)$,*

$$\nabla_{\psi_{k'}} \mathcal{L}_t\big|_0 = 0,$$

whereas $\nabla_{\phi_{h^+}} \mathcal{L}_t\big|_0$ and $\nabla_{\phi_{h^-}} \mathcal{L}_t\big|_0$ are in general non-zero.

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

$$g_{h^+}(x)\, e_{h^+}(x) + g_{h^-}(x)\, e_{h^-}(x). \tag{16}$$

Since $g_{h^+}(x) + g_{h^-}(x) = g_{h^\star}(x)$ and $g_{h^\star}(x)$ is independent of $\psi_{k'}$, differentiating yields

$$\nabla_{\psi_{k'}} g_{h^+}(x) = -\nabla_{\psi_{k'}} g_{h^-}(x).$$

For any coordinate ψ of $\psi_{k'}$, differentiating Equation 16 and using $e_{h^+}(x) = e_{h^-}(x) = e_{h^\star}(x)$ at initialization gives

$$\frac{\partial}{\partial \psi}[g_{h^+} e_{h^+} + g_{h^-} e_{h^-}]\bigg|_0 = \frac{\partial g_{h^+}}{\partial \psi}\bigg|_0 \big(e_{h^+}(x) - e_{h^-}(x)\big)\big|_0 = 0.$$

For any coordinate ϕ of a child's parameters, $\partial f_\theta(x)/\partial \phi = g_{h^\pm}(x)\, \partial e_{h^\pm}(x)/\partial \phi$, which is in general non-zero. □

The regularizers $\mathcal{L}_s$ (Equation 8) and $\mathcal{L}_m$ (Equation 10) depend on $\psi_{k'}$ through the split weights assigned to h^+ and h^-, so they can provide a gradient signal to the new gate at the first post-split optimizer step. Meanwhile, the task loss trains the child experts through gradients weighted by their respective gating weights. As the children develop different predictions, the task loss can also guide the new gate. The regularizers therefore allow routing to adapt from the moment of expansion, while the inherited experts begin to specialize.

C Experimental Details

This section describes the datasets, the baselines, and the adaptive hyperparameters used in Section 4.

C.1 Datasets

The comparison covers 14 datasets, i.e., nine classification and five regression tasks, grouped into synthetic benchmarks, small tabular datasets, and large-scale datasets. Each seed $s \in \{0, \ldots, 9\}$ draws its own train/validation/test split, and the same split is shared by all methods. Datasets without an official test partition are split into 64/16/20 % train/validation/test, stratified by class for

classification, except kin8nm and CCPP, which use an $80/10/10\,\%$ split. Datasets with an official test partition keep it as the test set, and a stratified $20\,\%$ of the official training set is held out for validation. The dimensions and split sizes are reported in Table 2. Inputs are standardized using training-set statistics as described in Section 2, and classification targets are integer class labels.

Synthetic benchmarks. The synthetic block contains *Checkerboard*, *Spirals*, and *Rastrigin*, which test piecewise decision logic, nested manifolds, and oscillating regression surfaces in two dimensions. Checkerboard is a two-class rule on an 80×80 grid over $[-1.5, 1.5]^2$, where the label is the sign of a sum of alternating Gaussian bumps centered on a 7×7 lattice, with a small amount of noise. Spirals contains two interleaved noisy spirals with 1200 points per class. Rastrigin is the two-dimensional Rastrigin function (Törn & Žilinskas, 1989) on $[-5.12, 5.12]^2$ with additive Gaussian noise, which combines a quadratic envelope with a dense periodic lattice.

Small tabular datasets. The small tabular block contains *Breast Cancer*, *Digits*, *Diabetes*, *kin8nm*, and *CCPP*. Breast Cancer is the 30-feature Wisconsin diagnostic binary classification dataset (Wolberg et al., 1993), and Digits is the scikit-learn (Pedregosa et al., 2011) version of the optical handwritten digits dataset (Alpaydin & Kaynak, 1998), with 8×8 images and ten classes. Diabetes is the 10-feature regression benchmark popularized with least-angle regression (Efron et al., 2004), kin8nm is the 8-feature robot-arm regression task distributed with the DELVE benchmark collection (Neal, 1996), and CCPP (Tüfekci, 2014) predicts the net hourly electrical output of a combined cycle power plant from 4 ambient variables.

Large-scale datasets. The large-scale block contains *MNIST*, *Fashion-MNIST*, *ISOLET*, *UCI HAR*, *Covertype*, and *CT Slice*. MNIST (LeCun et al., 1998) and Fashion-MNIST (Xiao et al., 2017) are ten-class image datasets, whose 28×28 grayscale images are flattened into 784 features. ISOLET (Fanty & Cole, 1990) contains 617 acoustic features of spoken letters from 26 classes, and UCI HAR (Anguita et al., 2013) contains 561 features computed from smartphone inertial signals, labeled with six activities performed by 30 subjects. Covertype (Blackard & Dean, 1999) predicts one of seven forest cover types from 54 cartographic features, 44 of which are binary indicators. CT Slice (Graf et al., 2011) is a regression task that predicts the relative axial position of a CT slice from 384 image descriptors, collected from 97 patients. For UCI HAR and CT Slice the splits are grouped by subject and by patient, respectively, so that no subject or patient appears in more than one split. Their split sizes therefore vary across seeds.

Dataset	Task	d_{in}	Classes	Train / val / test	Total
Checkerboard	classification	2	2	4,096 / 1,024 / 1,280	6,400
Spirals	classification	2	2	1,536 / 384 / 480	2,400
Rastrigin	regression	2	–	1,408 / 352 / 440	2,200
Breast Cancer	classification	30	2	364 / 91 / 114	569
Digits	classification	64	10	1,149 / 288 / 360	1,797
Diabetes	regression	10	–	282 / 71 / 89	442
kin8nm	regression	8	–	6,552 / 820 / 820	8,192
CCPP	regression	4	–	7,654 / 957 / 957	9,568
MNIST	classification	784	10	48,000 / 12,000 / 10,000	70,000
Fashion-MNIST	classification	784	10	48,000 / 12,000 / 10,000	70,000
ISOLET	classification	617	26	4,990 / 1,248 / 1,559	7,797
UCI HAR	classification	561	6	5,449–5,676 / 1,676–1,903 / 2,947	10,299
Covertype	classification	54	7	371,847 / 92,962 / 116,203	581,012
CT Slice	regression	384	–	29,613–36,223 / 8,215–11,289 / 8,504–14,197	53,500

Table 2: Dataset dimensions and split sizes. Split sizes are identical across seeds, except for the subject- and patient-grouped splits of UCI HAR and CT Slice, for which the range over the ten seeds is reported.

C.2 Baselines

We compare SPAWN against two fixed architecture families, MLPs and XGBoost, and three growing families, Budding trees, GradMax and Firefly. For every family, candidate settings vary by dataset,

and each family keeps, for all seeds, the configuration with the lowest validation loss averaged over the ten seeds (Section 4).

Multi-layer perceptron (MLP). Fully connected feed-forward network with ReLU activations, trained by back-propagation (Rumelhart et al., 1986), with width and depth varied across candidates. Its retained size is its parameter count.

Gradient-boosted trees (XGBoost). Stage-wise additive ensemble of regression trees with second-order gradient updates and shrinkage (Chen & Guestrin, 2016), with the number of boosting rounds and the maximum depth varied across candidates. Its retained size is its number of tree nodes, which is not comparable to a neural parameter count.

Budding trees. Soft decision tree (Irsoy et al., 2012) whose structure is learned during training (Irsoy et al., 2014). Each node interpolates between its own response and the gated combination of its children's responses through a leafness parameter $\gamma \in [0, 1]$. When γ drops below one, the node acquires two children, and a penalty $\lambda \sum(1-\gamma)$ controls how much the tree grows. As in the original formulation, the leaves emit constant responses rather than input-dependent predictions. Budding trees were not evaluated on Covertype due to their computational cost.

GradMax. Grows the hidden layers of an MLP by inserting neurons with zero incoming weights, which preserves the network function at insertion. The outgoing weights are set to the leading singular vectors of a gradient statistic, so as to maximize the gradient norm of the new weights (Evci et al., 2022). Our dense implementation follows the authors' released code.

Firefly. Grows the width of a hidden layer by splitting existing neurons and adding new ones, and keeps the candidates that most decrease the loss in a first-order neighborhood of the current network (Wu et al., 2020). Since the original method targets convolutional networks, we use a dense, width-only variant.

C.3 Adaptive Hyperparameters

The adaptive procedure is controlled by three groups of hyperparameters. The inner stopper closes a training cycle when the rolling difference of the training loss over a window of k epochs stays below a threshold τ after w warm-up epochs. The cycle and growth stoppers close cycles and terminate growth based on the validation loss, and the hard cap $C_{\max}$ bounds the number of expansion cycles. After growth, the fine-tuning phase of Section 3.4 trains the validation-best checkpoint with its own learning rate, epoch cap and validation patience. The remaining settings are the optimizer and its learning rates, the weight decay, the entropy weights $\lambda_s = \lambda_m$, and the batch size, while the gradient norm is always clipped at 1 and splits always use the gate-weighted deviation score of Equation 12. The growth patience, i.e., the number of consecutive expansion cycles without improvement of the validation loss after which growth stops, determines how readily the outer loop commits to a final architecture. A long patience keeps expanding when the validation loss has temporarily stalled and recovers models that grow further before stopping, while a short patience terminates earlier and returns smaller models at an earlier point of the trade-off between size and accuracy. These settings are tuned per dataset, and Table 7 lists the configuration kept on each dataset.

D Extended Results

Table 3 reports the test comparison of Table 1 on all datasets, including Digits, ISOLET and Fashion-MNIST, with the two SPAWN variants shown separately. On the three datasets added there, the results follow the pattern of Section 4.2. On Digits, SPAWN-LL is the smallest competitive model, with half the parameters of the MLP. On ISOLET and Fashion-MNIST, whose inputs have hundreds of features, Firefly and XGBoost score highest, and SPAWN stays within about two points of them. A dagger marks entries where at least one of the ten selected runs stopped at its budget cap, i.e., the cap on expansion cycles for SPAWN, the caps on epochs, growth steps or nodes for the growing baselines and MLPs, and the cap on boosting rounds for XGBoost. Figure 5 extends Figure 4 to all datasets, showing how the validation score of the kept SPAWN configurations evolves as experts are added. Figure 6 shows the growth on Checkerboard and Rastrigin as Figure 3 does on Spirals. We produce the same visualization for four simple classification datasets with two-dimensional inputs

		Classic baselines		Growing baselines			SPAWN	
	Dataset	MLP	XGBoost	Budding	GradMax	Firefly	LL	QL
Classification	Spirals	100.00 ±0.00 8.6k ($w64_3$)	97.67 ±0.67 33.0k ($t1000_8$)	52.83 ±2.03 143†	99.98 ±0.05 17.2k	99.75 ±0.28 1.3k†	99.35 ±0.51 554†	**99.88 ±0.16** **248**
	Checkerboard	91.09 ±0.52 2.9k† ($w576_1$)	92.83 ±0.47 39.1k ($t1000_8$)	51.80 ±0.16 110	85.65 ±3.72 17.2k†	89.70 ±0.47 1.3k†	**91.97 ±0.54** **1.0k**	91.55 ±0.69 735
	Breast cancer	97.54 ±0.93 197† ($w5_2$)	96.75 ±0.84 1.8k† ($t300_4$)	98.16 ±0.69 891	98.25 ±0.98 1.0k	98.33 ±0.81 2.0k	**97.54 ±1.21** **192**	97.54 ±0.83 210
	Digits	96.97 ±0.53 3.5k† ($w32_2$)	96.17 ±0.49 20.0k† ($t300_4$)	93.69 ±1.37 4.8k†	97.11 ±0.62 7.1k	96.03 ±0.56 2.9k	**96.89 ±0.73** **1.7k**	96.39 ±0.73 3.4k
	UCI HAR	**93.52 ±0.56** **5.2k** ($w9_2$)	93.50 ±0.53 24.0k† ($t300_4$)	92.81 ±0.40 35.8k†	93.24 ±0.73 22.2k	93.53 ±0.70 18.2k	93.21 ±0.50 9.3k	93.71 ±0.74 13.3k
	ISOLET	94.96 ±0.13 99.0k ($w128_2$)	94.72 ±0.22 203.2k† ($t1200_6$)	45.68 ±3.84 20.0k†	94.74 ±0.32 99.0k	**95.76 ±0.16** **82.5k**	94.37 ±0.40 42.8k	94.32 ±0.33 40.3k
	MNIST	97.40 ±0.17 225.0k ($w283_1$)	**97.85 ±0.04** **511.3k**† ($t1200_8$)	88.69 ±1.87 24.7k†	96.96 ±0.28 118.3k	97.27 ±0.08 101.1k†	96.66 ±0.13 262.6k†	95.61 ±0.11 193.4k
	Fashion-MNIST	88.08 ±0.23 118.3k ($w128_2$)	**89.91 ±0.08** **547.5k** ($t1200_8$)	79.24 ±2.52 24.7k†	88.15 ±0.21 118.3k	87.41 ±0.14 62.3k	87.75 ±0.14 131.3k†	85.49 ±0.11 17.3k
	Covertype	**94.40 ±0.11** **129.7k**† ($w330_2$)	94.42 ±0.07 1.35M† ($t1000_8$)	— —	90.35 ±0.24 8.1k†	85.36 ±0.09 4.0k†	92.67 ±0.13 42.4k†	91.07 ±0.12 47.5k†
Regression	Rastrigin	0.907 ±0.023 17.0k† ($w128_2$)	0.993 ±0.001 89.7k† ($t1200_8$)	0.217 ±0.104 241†	0.675 ±0.046 4.4k†	0.875 ±0.031 4.1k†	0.777 ±0.163 1.9k†	**0.988 ±0.005** **3.6k**†
	Diabetes	0.449 ±0.055 277 ($w23_1$)	0.389 ±0.056 1.6k ($t300_4$)	0.421 ±0.053 195†	0.430 ±0.051 1.4k	0.456 ±0.044 169†	**0.465 ±0.050** **40**	0.459 ±0.054 43
	Kin8nm	0.931 ±0.002 134.1k ($w256_3$)	0.817 ±0.005 263.0k† ($t1000_8$)	0.894 ±0.004 693†	0.927 ±0.004 4.7k	0.924 ±0.003 3.9k	0.929 ±0.004 736†	**0.928 ±0.004** **716**
	CT slice	0.960 ±0.009 56.5k ($w113_2$)	0.931 ±0.013 241.4k† ($t1200_8$)	0.935 ±0.017 23.1k	0.954 ±0.012 65.9k	0.927 ±0.020 45.1k	**0.937 ±0.017** **19.2k**	0.895 ±0.041 21.5k
	CCPP	0.956 ±0.003 133.1k ($w256_3$)	**0.970 ±0.003** **167.9k** ($t1000_8$)	0.943 ±0.004 267†	0.952 ±0.003 4.5k†	0.953 ±0.003 6.1k	0.950 ±0.003 499†	0.950 ±0.004 575†

Table 3: Test comparison on all datasets, with the two SPAWN variants shown separately. Entries follow Table 1, with gray text and shading computed within each row of this table. A dagger marks entries where at least one of the ten selected runs stopped at its budget cap.

(Figure 7): Circles and Moons from scikit-learn (Pedregosa et al., 2011), three concentric rings, and an XOR of four Gaussian clusters. These datasets are not part of Table 1, and SPAWN uses the configuration kept on Spirals without further tuning. Their class boundaries are smooth curves, which few experts represent with few parameters. With at most four experts, i.e., at most 51 parameters, both variants reach at least 98% validation accuracy on all four datasets. With quadratic gates, two experts, i.e., 17 or 23 parameters, already reach more than 99% on Circles, Three rings and Gaussian XOR, since a single quadratic gate already separates the rings, or the diagonal clusters, with a conic boundary.

The MLP search of Table 1 includes a dense size sweep, trained after SPAWN and sized from its retained model. On each dataset, we train between 9 and 30 uniform-width MLPs with one to three hidden layers, whose widths give evenly spaced parameter counts from about a third to up to ten times the size of SPAWN. Only the architecture varies: the other training settings are those of the MLP selected before the sweep, on the same ten splits. The selection of Table 1 keeps a sweep MLP on seven datasets. Since this selection favors low validation loss rather than small size, Table 4 compares the retained SPAWN models with all sweep MLPs on validation data, with the display rules of Table 1. Each sweep thus uses the size found by SPAWN, and the best of many MLPs is compared with a single SPAWN model. On nine of eleven datasets, SPAWN lies among the best models, with a CI overlapping that of the best mean. On Spirals, Checkerboard and Rastrigin, it is the smallest of them, while on the other six, smaller MLPs overlap with it. SPAWN falls behind only on MNIST, where a smaller MLP scores significantly higher, and on Covertype, where larger MLPs do.

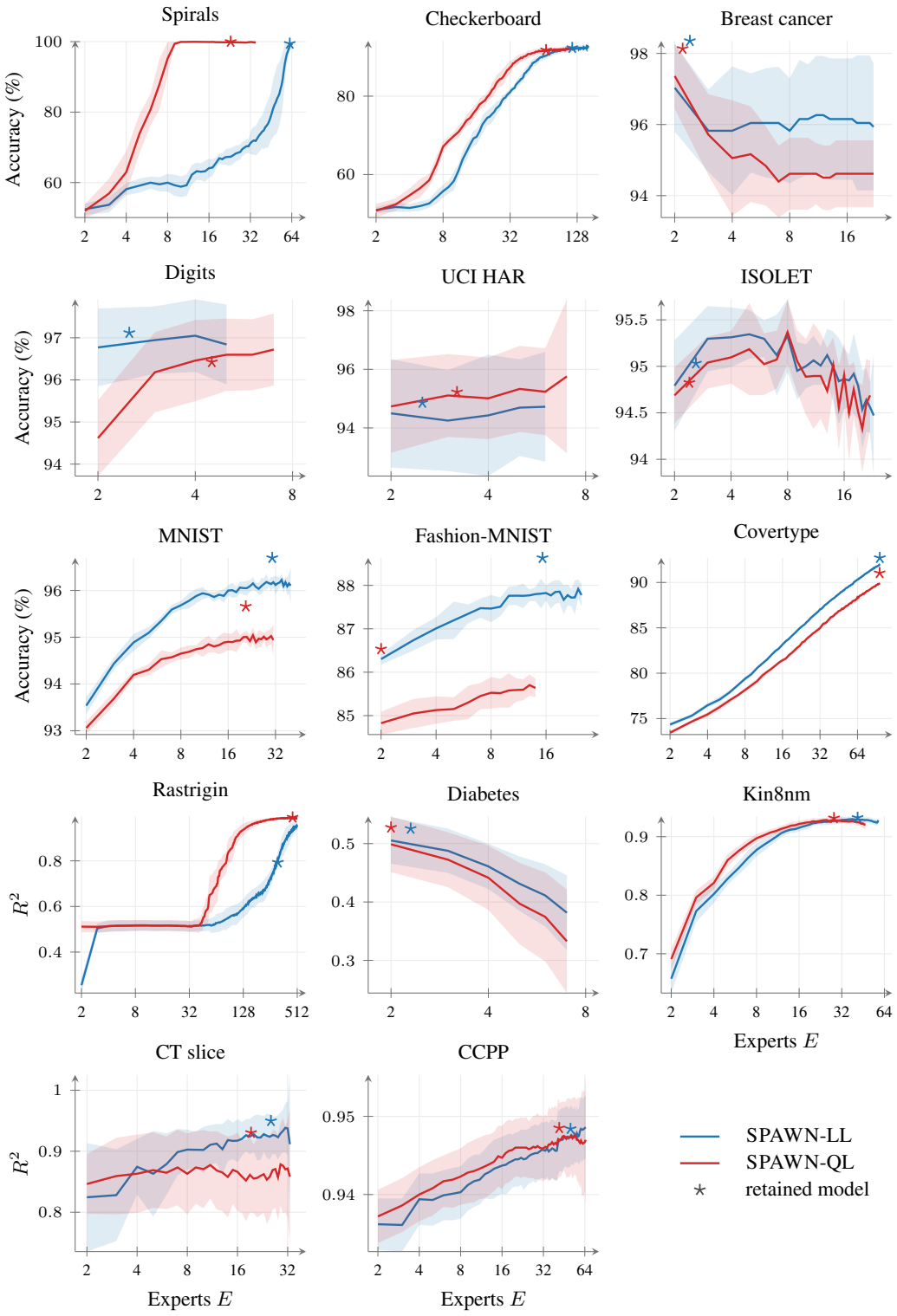


Figure 5: Mean validation score of the kept SPAWN-LL and SPAWN-QL configurations on every dataset against the number of experts E (log scale). Curves show the model at the end of each growth cycle, averaged over the seeds that reached that size, and bands show 95% CIs of the mean. Stars mark the retained models of Table 1, i.e., the validation-best checkpoints after fine-tuning (Section 3.4), at their mean number of experts.

	Dataset	SPAWN	MLPs of the size sweep		
Classification	Spirals	**99.97** ±0.06 **248** (QL)	83.12 ±5.16 149† ($w7_3$)	100.00 ±0.00 626† ($w16_3$)	
	Checkerboard	**92.31** ±0.73 **1.0k** (LL)	87.18 ±1.58 618† ($w22_2$)	87.83 ±0.83 642† ($w128_1$)	90.64 ±0.88 942 ($w20_3$)
	Breast cancer	98.35 ±0.85 192 (LL)	**98.35** ±0.76 **68**† ($w2_1$)	98.68 ±0.89 197† ($w5_2$)	98.90 ±0.64 227 ($w5_3$)
	UCI HAR	95.22 ±2.24 13.3k (QL)	**94.11** ±2.67 **5.1k** ($w9_1$)	94.56 ±1.96 5.2k ($w9_2$)	
	MNIST	96.70 ±0.10 262.6k (LL)	**96.87** ±0.13 **74.7k** ($w94_1$)	97.19 ±0.20 225.0k ($w283_1$)	97.25 ±0.33 375.2k ($w472_1$)
	Covertype	92.70 ±0.08 42.4k (LL)	93.62 ±0.20 43.3k† ($w179_2$)	**94.34** ±0.19 **108.2k** ($w299_2$)	94.40 ±0.15 129.7k† ($w330_2$)
Regression	Rastrigin	**0.990** ±0.004 **3.6k** (QL)	0.723 ±0.062 2.1k ($w43_2$)	0.787 ±0.026 2.1k† ($w525_1$)	0.836 ±0.021 3.1k ($w38_3$)
	Diabetes	0.528 ±0.041 43 (QL)	**0.421** ±0.118 **31**† ($w2_2$)	0.506 ±0.060 37† ($w3_1$)	
	Kin8nm	0.932 ±0.004 736 (LL)	0.898 ±0.004 243† ($w11_2$)	0.920 ±0.006 477† ($w17_2$)	**0.926** ±0.006 **727**† ($w22_2$)
	CT slice	0.950 ±0.020 19.2k (LL)	**0.946** ±0.017 **9.4k** ($w23_2$)	0.954 ±0.016 28.4k ($w63_2$)	0.961 ±0.011 56.5k ($w113_2$)
	CCPP	0.949 ±0.004 575 (QL)	**0.944** ±0.004 **193** ($w8_3$)	0.950 ±0.004 1.4k ($w34_2$)	0.951 ±0.004 1.6k ($w26_3$)

Table 4: Validation comparison of the retained SPAWN models of Table 1 with the MLPs of the size sweep. Entries follow Table 1: mean score over ten seeds with the half width of the 95% CI, parameter count in red, and gate family or architecture ($\text{width}_{\text{depth}}$) in parentheses. The rules are applied over SPAWN and all sweep MLPs of each dataset: entries whose CI does not overlap that of the best mean are grayed out, and the smallest remaining model is bold and shaded. Each row shows the best entry, the shaded one, and, in the remaining slots, the MLPs scoring higher than every smaller model, starting from the next one larger than SPAWN and then those closest to it in size. A dagger marks architectures with at least one run stopped at its epoch cap.

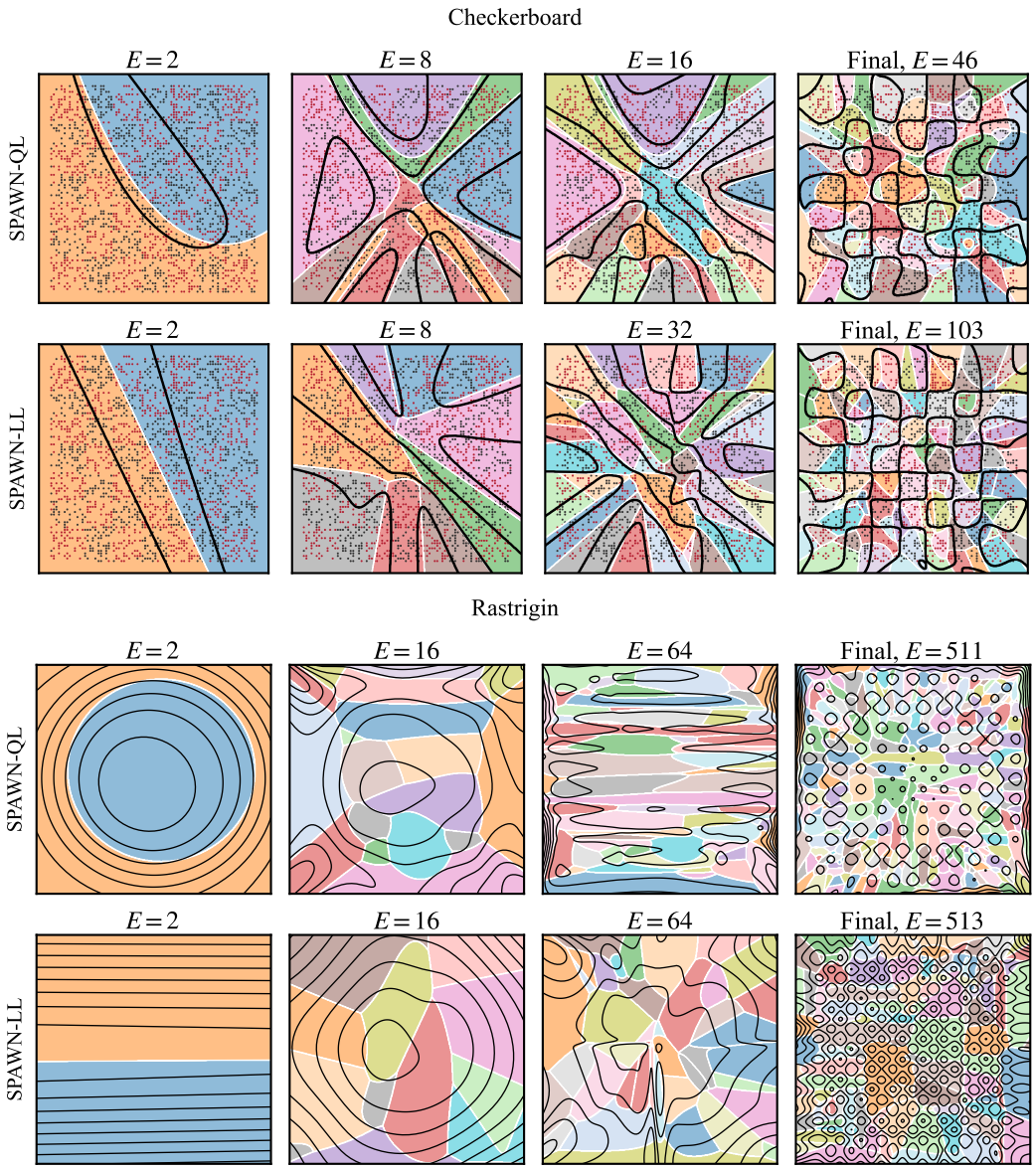


Figure 6: Growth of SPAWN-QL and SPAWN-LL on Checkerboard (top) and Rastrigin (bottom), from the runs of Table 1. The first three panels of each row show the model at the end of the training cycle with E experts, and the last panel shows the retained model after fine-tuning. Regions are colored by the dominant expert. On Checkerboard, dots are training points colored by class and black curves are decision boundaries. On Rastrigin, black curves are contour lines of the prediction.

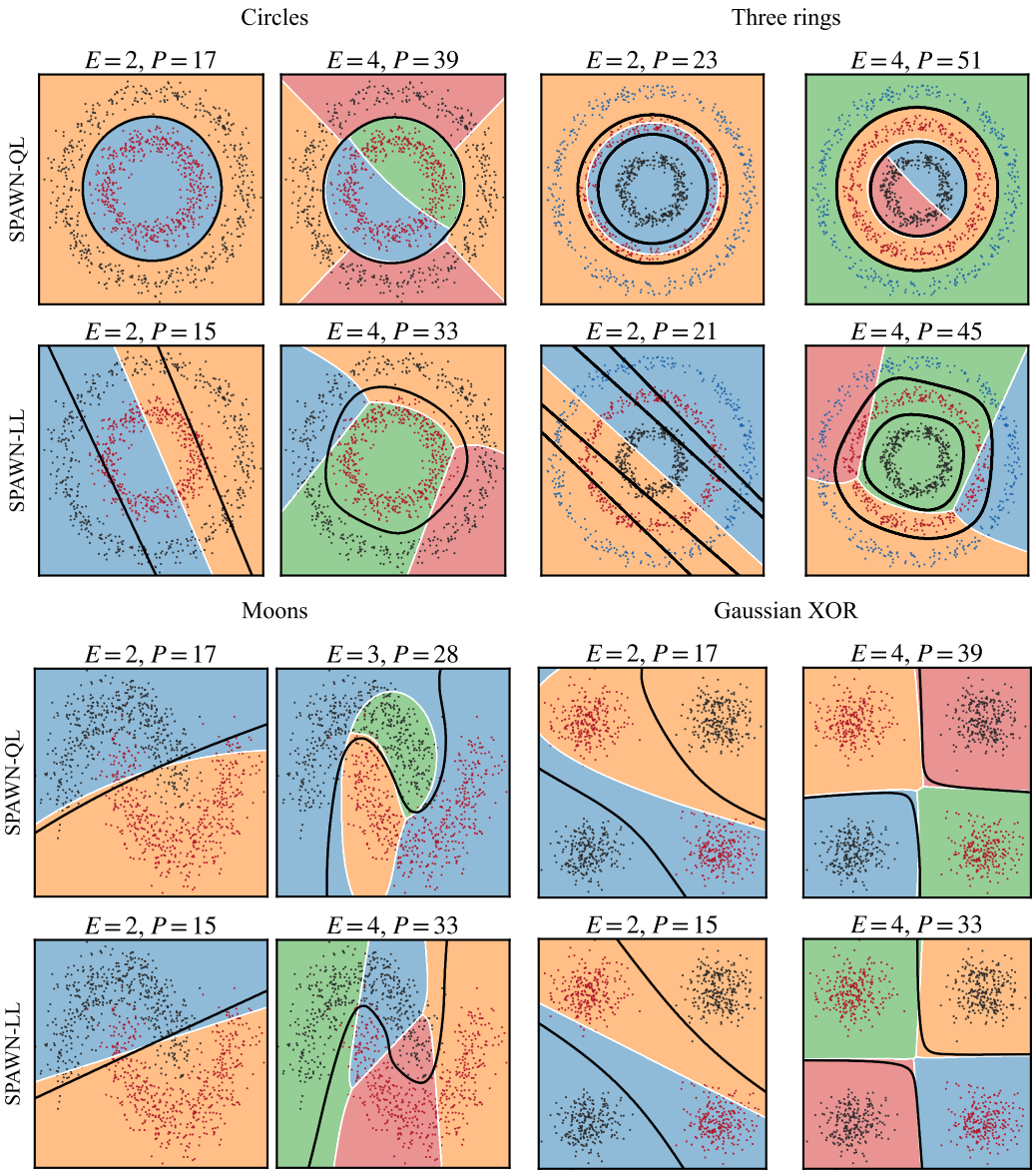


Figure 7: First growth steps of SPAWN-QL (top row of each block) and SPAWN-LL (bottom row) on Circles, Three rings, Moons and Gaussian XOR, trained with the SPAWN configuration kept on Spirals in Table 1. Each panel shows the model at the end of the training cycle with E experts and P parameters. Regions are colored by the dominant expert, dots are training points colored by class, and black curves are decision boundaries.

E HYPERPARAMETER SEARCH AND SELECTED CONFIGURATIONS

For every family, we train a set of candidate configurations on each dataset over the same ten paired splits, and keep for all ten seeds the configuration with the lowest mean validation loss. The candidates vary the settings that control capacity and growth, together with the learning rate and batch size, so that each baseline is tuned to the dataset rather than run with a single default. MLPs range from width 2 to 1024 with one to three hidden layers, including the size sweep of Table 4, and XGBoost from 300 to 1200 boosting rounds with trees of depth 4 to 8. Budding trees are capped between 15 and 127 nodes. GradMax grows two hidden layers towards a target width between 16 and 128, and Firefly grows a single hidden layer up to a width between 16 and 1024. For SPAWN, the expansion-cycle cap ranges from 8 to 800 and the entropy weight from 10^{-4} to 10^{-2}, with both gate types evaluated on every dataset.

Tables 5, 6 and 7 report the configuration kept on each dataset. Tables 8–11 list every XGBoost candidate and every MLP candidate outside the size sweep with its mean validation score and retained size, and Tables 12 and 13 report how the validation score of the kept SPAWN configurations evolves as experts are added, which Figure 5 plots. The validation scores of every candidate and seed are released with the code.

		MLP					**XGBoost**			
	Dataset	Layers	Batch	LR	WD	Ep./Pat.	Rounds	Depth	η	Reg.
Classification	Spirals	3×64	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	1000	8	$5 \cdot 10^{-2}$	
	Checkerboard	1×576	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	1000	8	$5 \cdot 10^{-2}$	
	Breast cancer	2×5	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	300	4	$5 \cdot 10^{-2}$	
	Digits	2×32	64	10^{-3}	10^{-4}	100/15	300	4	0.1	
	UCI HAR	2×9	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	300	4	$5 \cdot 10^{-2}$	
	ISOLET	2×128	256	10^{-3}	10^{-2}	400/40	1200	6	$3 \cdot 10^{-2}$	
	MNIST	1×283	256	10^{-3}	10^{-2}	400/40	1200	8	$5 \cdot 10^{-2}$	✓
	Fashion-MNIST	2×128	256	10^{-3}	10^{-4}	400/40	1200	8	$5 \cdot 10^{-2}$	✓
	Covertype	2×330	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	1000	8	$5 \cdot 10^{-2}$	
Regression	Rastrigin	2×128	64	$3 \cdot 10^{-3}$	10^{-4}	8000/500	1200	8	$5 \cdot 10^{-2}$	✓
	Diabetes	1×23	64	10^{-2}	10^{-4}	100/15	300	4	$5 \cdot 10^{-2}$	
	Kin8nm	3×256	512	$3 \cdot 10^{-3}$	10^{-4}	2000/200	1000	8	$5 \cdot 10^{-2}$	
	CT slice	2×113	256	10^{-3}	10^{-2}	400/40	1200	8	$5 \cdot 10^{-2}$	✓
	CCPP	3×256	512	$3 \cdot 10^{-3}$	10^{-4}	2000/200	1000	8	$5 \cdot 10^{-2}$	

Table 5: Configuration kept for the fixed-architecture baselines on each dataset, used for all ten seeds. MLPs use ReLU activations and AdamW, with early stopping on the validation loss (Ep./Pat.: epoch cap and patience). XGBoost uses the histogram tree method with early stopping after 25 rounds without validation improvement. Reg. marks the regularized setting, with row and column subsampling 0.8, minimum child weight 5 and L_2 leaf penalty 10, instead of 0.9, 1, 1 and 1. Retained sizes are in Table 3.

		Budding			GradMax			Firefly			
	Dataset	LR	λ	Nodes	Target	LR	Batch	Width	Step	LR	Batch
Classification	Spirals	0.1	10^{-4}	63	128	$3 \cdot 10^{-3}$	64	16→256	+16	$3 \cdot 10^{-3}$	64
	Checkerboard	0.1	10^{-4}	63	128	$3 \cdot 10^{-3}$	64	16→256	+16	$3 \cdot 10^{-3}$	64
	Breast cancer	0.1	10^{-4}	63	64	$3 \cdot 10^{-3}$	512	4→64	+8	$3 \cdot 10^{-3}$	512
	Digits	0.1	10^{-3}	63	64	$3 \cdot 10^{-3}$	64	4→64	+8	$3 \cdot 10^{-3}$	64
	UCI HAR	0.1	10^{-4}	63	64	$3 \cdot 10^{-3}$	512	4→64	+8	$3 \cdot 10^{-3}$	512
	ISOLET	0.1	10^{-4}	31	128	10^{-3}	256	16→128	+8	10^{-3}	256
	MNIST	0.1	10^{-4}	31	128	10^{-3}	256	16→128	+8	10^{-3}	256
	Fashion-MNIST	0.1	10^{-4}	31	128	10^{-3}	256	16→128	+8	10^{-3}	256
	Covertype	—	—	—	64	$3 \cdot 10^{-3}$	1024	4→64	+8	$3 \cdot 10^{-3}$	1024
Regression	Rastrigin	0.1	10^{-4}	63	64	$3 \cdot 10^{-3}$	64	16→1024	+32	$3 \cdot 10^{-3}$	64
	Diabetes	0.1	10^{-3}	15	64	$3 \cdot 10^{-3}$	64	4→16	+4	$3 \cdot 10^{-3}$	64
	Kin8nm	0.1	10^{-4}	63	64	$3 \cdot 10^{-3}$	512	16→1024	+32	$3 \cdot 10^{-3}$	64
	CT slice	0.1	10^{-3}	127	128	10^{-3}	256	16→128	+8	10^{-3}	256
	CCPP	0.1	10^{-4}	63	64	$3 \cdot 10^{-3}$	512	16→1024	+32	$3 \cdot 10^{-3}$	64

Table 6: Configuration kept for the growing baselines on each dataset. Budding: learning rate, leafness penalty λ and node cap. GradMax: two hidden layers grown from width 4 towards the target width. Firefly: one hidden layer grown from the initial to the maximum width, adding Step neurons per growth step. Budding was not evaluated on Covertype.

		Budget		Optimization					Stopping	
	Dataset	$C_{\max}$	Ep.	Opt.	LR	WD	λ	Batch	Inner	Patience
Classification	Spirals	64	300/300	AdamW	$10^{-2}/10^{-3}$	10^{-4}	10^{-2}	512	$30/5 \cdot 10^{-3}/50$	20/100/60
	Checkerboard	800	2000/1500	Adam	$10^{-2}/10^{-3}$	0	10^{-2}	512	$50/10^{-2}/100$	50/2500/200
	Breast cancer	64	300/300	AdamW	$10^{-2}/10^{-3}$	10^{-4}	10^{-2}	512	$30/5 \cdot 10^{-3}/50$	20/100/60
	Digits*	8	30/60	AdamW	$10^{-2}/3 \cdot 10^{-3}$	10^{-4}	10^{-3}	64	$10/5 \cdot 10^{-3}/10$	3/10/15
	UCI HAR	12	60/100	AdamW	$10^{-2}/3 \cdot 10^{-3}$	10^{-4}	10^{-3}	512	$10/5 \cdot 10^{-3}/20$	4/15/20
	ISOLET	64	600/600	AdamW	$3 \cdot 10^{-3}/10^{-3}$	0.1	10^{-3}	512	$30/5 \cdot 10^{-3}/50$	20/10/60
	MNIST	48	2000/1500	Adam	$3 \cdot 10^{-3}/10^{-3}$	$3 \cdot 10^{-4}$	10^{-2}	512	$50/10^{-2}/100$	12/2500/200
	Fashion-MNIST*	24	2000/1500	Adam	$3 \cdot 10^{-3}/10^{-3}$	10^{-3}	10^{-2}	512	$50/10^{-2}/100$	12/10/200
	Covertype	96	60/150	AdamW	$10^{-2}/3 \cdot 10^{-3}$	10^{-3}	10^{-2}	1024	$10/5 \cdot 10^{-3}/20$	6/20/30
Regression	Rastrigin	512	2000/1500	Adam	$10^{-2}/10^{-3}$	0	10^{-3}	512	$50/10^{-2}/100$	50/2500/200
	Diabetes	20	30/60	AdamW	$10^{-2}/3 \cdot 10^{-3}$	10^{-4}	10^{-3}	64	$10/5 \cdot 10^{-3}/10$	5/10/15
	Kin8nm	64	300/300	AdamW	$10^{-2}/10^{-3}$	10^{-4}	10^{-2}	512	$30/5 \cdot 10^{-3}/50$	20/100/60
	CT slice*	64	200/300	AdamW	$3 \cdot 10^{-3}/10^{-3}$	0.1	10^{-3}	512	$50/5 \cdot 10^{-3}/100$	10/80/60
	CCPP	64	300/300	AdamW	$10^{-2}/10^{-3}$	10^{-4}	10^{-2}	512	$30/5 \cdot 10^{-3}/50$	20/100/60

Table 7: Configuration kept for SPAWN on each dataset, shared by the LL and QL variants unless marked with an asterisk. $C_{\max}$: expansion-cycle cap. Ep.: epoch cap per growth cycle / of the final fine-tuning. LR: growth / fine-tuning learning rate. λ: entropy weight $\lambda_s = \lambda_m$. Inner: rolling-difference stopper $k/\tau/w$ (window, threshold, warm-up). Patience: validation patience per cycle / for growth / for fine-tuning. All runs clip the gradient norm at 1 and split with the gate-weighted deviation score. On the marked datasets, SPAWN-QL keeps a different configuration: Digits (Adam, LR $10^{-3}/10^{-3}$, WD 0), Fashion-MNIST (WD 10^{-4}, patience 12/2500/200) and CT slice ($C_{\max}$ 128, Ep. 400/600, patience 20/20/60).

		MLP candidate					Validation	
	Dataset	Layers	Batch	LR	WD	Ep./Pat.	Score	Size
Classification	Spirals	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	69.92 ±3.08	1.2k†
		2×32	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	99.97 ±0.06	1.2k
		3×64	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	100.00 ±0.00	8.6k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	99.74 ±0.29	17.2k†
		2×128	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	100.00 ±0.00	17.2k
	Checkerboard	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	82.44 ±1.46	1.2k†
		2×32	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	89.81 ±1.32	1.2k
		1×576	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	91.56 ±0.44	2.9k†
		3×64	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	91.83 ±0.76	8.6k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	90.75 ±1.04	17.2k†
		2×128	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	89.54 ±2.50	17.2k
	Breast cancer	2×4	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	98.57 ±0.65	154
		2×5	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	98.68 ±0.89	197†
		2×8	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	98.46 ±1.18	338†
		2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	98.13 ±1.11	2.1k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	98.46 ±1.12	20.7k
	Digits	2×32	64	10^{-2}	10^{-4}	100/15	97.15 ±1.01	3.5k†
		2×32	64	10^{-3}	10^{-4}	100/15	97.01 ±0.92	3.5k†
		2×64	64	10^{-2}	10^{-4}	100/15	97.05 ±1.00	9.0k
	UCI HAR	2×9	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	94.56 ±1.96	5.2k
		2×32	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	94.09 ±3.30	19.2k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	94.24 ±2.46	89.2k
	ISOLET	2×32	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	94.71 ±0.48	21.7k
		2×32	256	10^{-3}	10^{-2}	400/40	95.02 ±0.44	21.7k
		2×32	256	10^{-3}	10^{-4}	400/40	95.00 ±0.44	21.7k
		2×128	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	95.49 ±0.43	99.0k
		2×128	256	10^{-3}	10^{-2}	400/40	95.87 ±0.43	99.0k
		2×128	256	10^{-3}	10^{-4}	400/40	95.86 ±0.46	99.0k
	MNIST	2×32	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	95.81 ±0.15	26.5k
		2×32	256	10^{-3}	10^{-2}	400/40	95.83 ±0.13	26.5k
		2×32	256	10^{-3}	10^{-4}	400/40	95.83 ±0.12	26.5k
		2×128	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	96.76 ±0.20	118.3k
		2×128	256	10^{-3}	10^{-2}	400/40	96.80 ±0.16	118.3k
		2×128	256	10^{-3}	10^{-4}	400/40	96.75 ±0.13	118.3k
		1×283	256	10^{-3}	10^{-2}	400/40	97.19 ±0.20	225.0k
	Fashion-MNIST	2×32	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	88.01 ±0.11	26.5k
		2×32	256	10^{-3}	10^{-2}	400/40	88.17 ±0.11	26.5k
		2×32	256	10^{-3}	10^{-4}	400/40	88.10 ±0.10	26.5k
		2×128	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	88.75 ±0.17	118.3k
		2×128	256	10^{-3}	10^{-2}	400/40	88.90 ±0.30	118.3k
		2×128	256	10^{-3}	10^{-4}	400/40	88.94 ±0.25	118.3k
	Covertype	2×32	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	85.53 ±0.14	3.0k†
		2×32	1024	$3 \cdot 10^{-3}$	10^{-4}	450/50	86.32 ±0.18	3.0k†
		2×128	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	92.67 ±0.19	24.5k†
		2×330	1024	$3 \cdot 10^{-3}$	10^{-4}	150/20	94.40 ±0.15	129.7k†

Table 8: MLP candidates on the classification datasets, with the settings of Table 5. Score: mean validation accuracy (%) or R^2 over the ten seeds, with the half width of its 95% CI. Size: mean retained size (red), in parameters for MLPs and tree nodes for XGBoost. The shaded line is the configuration kept for all ten seeds, the one with the lowest mean validation loss, which need not have the highest mean score. A dagger marks candidates for which at least one of the ten runs stopped at its budget cap. Of the MLP size sweep (Table 4), only the kept architecture is listed.

		XGBoost candidate				Validation	
	Dataset	Rounds	Depth	η	Reg.	Score	Size
Classification	Spirals	300	4	$5 \cdot 10^{-2}$		91.85 $\pm$0.90	6.0k†
		600	6	$5 \cdot 10^{-2}$		97.32 $\pm$0.80	23.2k†
		1000	8	$5 \cdot 10^{-2}$		97.84 $\pm$0.59	33.0k
	Checkerboard	300	4	$5 \cdot 10^{-2}$		88.99 $\pm$1.42	8.3k†
		600	6	$5 \cdot 10^{-2}$		92.97 $\pm$0.61	37.4k†
		1000	8	$5 \cdot 10^{-2}$		92.91 $\pm$0.64	39.1k
	Breast cancer	300	4	$5 \cdot 10^{-2}$		96.70 $\pm$1.11	1.8k†
		600	6	$5 \cdot 10^{-2}$		97.03 $\pm$0.91	2.0k†
	Digits	300	4	0.1		96.91 $\pm$1.20	20.0k†
		600	6	0.1		96.63 $\pm$1.21	22.2k†
		300	4	$5 \cdot 10^{-2}$		96.84 $\pm$1.05	32.6k†
	UCI HAR	300	4	$5 \cdot 10^{-2}$		95.89 $\pm$1.32	24.0k†
		600	6	$5 \cdot 10^{-2}$		95.28 $\pm$1.55	33.7k
	ISOLET	300	4	$5 \cdot 10^{-2}$		95.12 $\pm$0.51	92.8k†
		600	6	$5 \cdot 10^{-2}$		95.11 $\pm$0.49	119.4k†
		1200	8	$5 \cdot 10^{-2}$	✓	95.10 $\pm$0.48	125.8k†
		1000	8	$5 \cdot 10^{-2}$		95.13 $\pm$0.48	129.8k
		1200	6	$3 \cdot 10^{-2}$		95.22 $\pm$0.56	203.2k†
	MNIST	300	4	$5 \cdot 10^{-2}$		96.17 $\pm$0.15	86.7k†
		600	6	$5 \cdot 10^{-2}$		97.72 $\pm$0.09	394.4k†
		1200	8	$5 \cdot 10^{-2}$	✓	97.73 $\pm$0.10	511.3k†
		1000	8	$5 \cdot 10^{-2}$		97.67 $\pm$0.10	602.1k
		1200	6	$3 \cdot 10^{-2}$		97.77 $\pm$0.07	711.1k†
	Fashion-MNIST	300	4	$5 \cdot 10^{-2}$		88.94 $\pm$0.15	87.4k†
		600	6	$5 \cdot 10^{-2}$		90.66 $\pm$0.26	436.7k†
		1200	8	$5 \cdot 10^{-2}$	✓	90.77 $\pm$0.20	547.5k
		1000	8	$5 \cdot 10^{-2}$		90.52 $\pm$0.24	562.5k
		1200	6	$3 \cdot 10^{-2}$		90.71 $\pm$0.26	735.7k
	Covertype	300	4	$5 \cdot 10^{-2}$		78.00 $\pm$0.08	52.9k†
		600	6	$5 \cdot 10^{-2}$		87.05 $\pm$0.11	314.4k†
		1200	6	$3 \cdot 10^{-2}$		88.05 $\pm$0.12	638.5k†
		1200	8	$5 \cdot 10^{-2}$	✓	94.19 $\pm$0.08	1.32M†
		1000	8	$5 \cdot 10^{-2}$		94.42 $\pm$0.08	1.35M†

Table 9: XGBoost candidates on the classification datasets, with the settings of Table 5. Score: mean validation accuracy (%) or R^2 over the ten seeds, with the half width of its 95% CI. Size: mean retained size (red), in parameters for MLPs and tree nodes for XGBoost. The shaded line is the configuration kept for all ten seeds, the one with the lowest mean validation loss, which need not have the highest mean score. A dagger marks candidates for which at least one of the ten runs stopped at its budget cap.

		MLP candidate					Validation	
	Dataset	Layers	Batch	LR	WD	Ep./Pat.	Score	Size
Regression	Rastrigin	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.522 ±0.024	1.2k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.525 ±0.023	17.0k†
		2×128	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	0.900 ±0.017	17.0k†
		2×128	64	$3 \cdot 10^{-3}$	10^{-4}	8000/500	0.917 ±0.021	17.0k†
		3×256	64	$3 \cdot 10^{-3}$	10^{-4}	4000/500	0.901 ±0.015	132.6k†
		3×256	64	$3 \cdot 10^{-3}$	10^{-4}	8000/500	0.902 ±0.015	132.6k
	Diabetes	3×2	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	0.260 ±0.175	37
		2×8	64	$3 \cdot 10^{-3}$	10^{-4}	2000/200	0.497 ±0.041	169
		1×23	64	10^{-2}	10^{-4}	100/15	0.523 ±0.046	277
		2×32	64	10^{-2}	10^{-4}	100/15	0.517 ±0.043	1.4k
		2×32	64	10^{-3}	10^{-4}	100/15	0.508 ±0.049	1.4k
		2×64	64	10^{-2}	10^{-4}	100/15	0.507 ±0.040	4.9k
	Kin8nm	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.926 ±0.005	1.4k†
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.929 ±0.004	17.8k
		3×256	512	$3 \cdot 10^{-3}$	10^{-4}	2000/200	0.934 ±0.004	134.1k
	CT slice	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	0.947 ±0.019	13.4k
		2×32	256	10^{-3}	10^{-2}	400/40	0.947 ±0.020	13.4k
		2×32	256	10^{-3}	10^{-4}	400/40	0.947 ±0.020	13.4k
		2×113	256	10^{-3}	10^{-2}	400/40	0.961 ±0.011	56.5k
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	150/20	0.958 ±0.014	65.9k
		2×128	256	10^{-3}	10^{-2}	400/40	0.960 ±0.015	65.9k
		2×128	256	10^{-3}	10^{-4}	400/40	0.960 ±0.014	65.9k
	CCPP	2×32	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.947 ±0.004	1.2k†
		2×128	512	$3 \cdot 10^{-3}$	10^{-4}	300/40	0.951 ±0.004	17.3k†
		3×256	512	$3 \cdot 10^{-3}$	10^{-4}	2000/200	0.956 ±0.004	133.1k

Table 10: MLP candidates on the regression datasets, with the settings of Table 5. Score: mean validation accuracy (%) or R^2 over the ten seeds, with the half width of its 95% CI. Size: mean retained size (red), in parameters for MLPs and tree nodes for XGBoost. The shaded line is the configuration kept for all ten seeds, the one with the lowest mean validation loss, which need not have the highest mean score. A dagger marks candidates for which at least one of the ten runs stopped at its budget cap. Of the MLP size sweep (Table 4), only the kept architecture is listed.

		XGBoost candidate				Validation	
	Dataset	Rounds	Depth	η	Reg.	Score	Size
Regression	Rastrigin	300	4	$5 \cdot 10^{-2}$		0.953 $\pm$0.002	9.2k†
		600	6	$5 \cdot 10^{-2}$		0.969 $\pm$0.003	58.7k†
		1200	8	$5 \cdot 10^{-2}$	✓	0.993 $\pm$<0.001	89.7k†
		1200	6	$3 \cdot 10^{-2}$		0.971 $\pm$0.003	115.1k†
		1000	8	$5 \cdot 10^{-2}$		0.947 $\pm$0.004	201.3k†
	Diabetes	300	4	0.1		0.453 $\pm$0.060	826
		300	4	$5 \cdot 10^{-2}$		0.456 $\pm$0.060	1.6k
		600	6	0.1		0.432 $\pm$0.056	2.3k
	Kin8nm	300	4	$5 \cdot 10^{-2}$		0.690 $\pm$0.013	9.1k†
		600	6	$5 \cdot 10^{-2}$		0.820 $\pm$0.009	65.5k†
		1000	8	$5 \cdot 10^{-2}$		0.821 $\pm$0.008	263.0k†
	CT slice	300	4	$5 \cdot 10^{-2}$		0.896 $\pm$0.024	9.1k†
		600	6	$5 \cdot 10^{-2}$		0.917 $\pm$0.023	59.3k†
		1200	6	$3 \cdot 10^{-2}$		0.917 $\pm$0.023	94.8k†
		1000	8	$5 \cdot 10^{-2}$		0.918 $\pm$0.025	184.6k
		1200	8	$5 \cdot 10^{-2}$	✓	0.926 $\pm$0.025	241.4k†
	CCPP	300	4	$5 \cdot 10^{-2}$		0.955 $\pm$0.004	9.0k†
		600	6	$5 \cdot 10^{-2}$		0.966 $\pm$0.004	62.9k†
		1000	8	$5 \cdot 10^{-2}$		0.968 $\pm$0.004	167.9k

Table 11: XGBoost candidates on the regression datasets, with the settings of Table 5. Score: mean validation accuracy (%) or R^2 over the ten seeds, with the half width of its 95% CI. Size: mean retained size (red), in parameters for MLPs and tree nodes for XGBoost. The shaded line is the configuration kept for all ten seeds, the one with the lowest mean validation loss, which need not have the highest mean score. A dagger marks candidates for which at least one of the ten runs stopped at its budget cap.

		Growth checkpoint									Retained
	Dataset	E=2	E=4	E=8	E=16	E=32	E=64	E=128	E=256	E=512	Final
Classification	Spirals	52.42 15	58.12 33	59.97 69	64.09 141	71.17 285	99.32 573	—	—	—	99.43 ±0.39 554† (E=61.9)
	Checkerboard	50.93 15	51.46 33	55.74 69	69.63 141	81.04 285	90.23 573	92.18 1.1k	—	—	92.31 ±0.73 1.0k (E=115.7)
	Breast cancer	97.03 155	95.82 341	95.82 713	96.15 1.5k	—	—	—	—	—	98.35 ±0.85 192 (E=2.4)
	Digits	96.77 1.4k	97.05 2.8k	98.26 5.7k (1)	—	—	—	—	—	—	97.12 ±1.10 1.7k (E=2.5)
	UCI HAR	94.50 7.3k	94.43 15.2k	96.28 30.9k (1)	—	—	—	—	—	—	94.86 ±2.01 9.3k (E=2.5)
	ISOLET	94.79 32.8k	95.31 66.1k	95.33 132.9k	94.86 266.4k	—	—	—	—	—	95.03 ±0.52 42.8k (E=2.6)
	MNIST	93.54 16.5k	94.89 33.8k	95.68 68.3k	96.00 137.4k	96.12 275.5k	—	—	—	—	96.70 ±0.10 262.6k† (E=30.5)
	Fashion-MNIST	86.30 16.5k	87.01 33.8k	87.47 68.3k	87.82 137.4k	—	—	—	—	—	88.63 ±0.25 131.3k† (E=15.3)
	Covertype	74.34 825	76.46 1.7k	79.38 3.5k	83.16 7.0k	87.03 14.0k	90.31 28.1k	—	—	—	92.70 ±0.08 42.4k† (E=96.6)
Regression	Rastrigin	0.255 9	0.514 21	0.516 45	0.517 93	0.515 189	0.527 381 (7)	0.591 765 (6)	0.719 1.5k (6)	0.955 3.1k (6)	0.793 ±0.152 1.9k† (E=312.0)
	Diabetes	0.505 33	0.461 77	0.292 165 (1)	—	—	—	—	—	—	0.526 ±0.042 40 (E=2.3)
	Kin8nm	0.657 27	0.802 63	0.877 135	0.915 279	0.929 567	0.926 1.1k (4)	—	—	—	0.932 ±0.004 736† (E=41.4)
	CT slice	0.824 1.2k	0.874 2.7k	0.903 5.8k	0.917 11.9k	0.937 24.3k (5)	—	—	—	—	0.950 ±0.020 19.2k (E=25.4)
	CCPP	0.936 15	0.939 35	0.940 75	0.944 155	0.946 315	0.948 635 (8)	—	—	—	0.948 ±0.004 499† (E=50.4)

Table 12: Validation score of the kept SPAWN-LL runs as experts are added. Each growth checkpoint reports the mean validation accuracy (%) or R^2 at the end of the growth cycle with E experts, and the mean number of parameters (red) over the runs that reached it, with their number in parentheses when fewer than ten. Growth continues past the retained checkpoint until the growth patience runs out. Final reports the retained model, i.e., the validation-best checkpoint after fine-tuning (Section 3.4), with the half width of its 95% CI, parameters and mean number of experts. A dagger marks datasets where at least one run stopped at its expansion-cycle cap.

		Growth checkpoint									Retained
	Dataset	E=2	E=4	E=8	E=16	E=32	E=64	E=128	E=256	E=512	Final
Classification	Spirals	51.95 17	62.94 39	95.18 83	99.90 171	99.89 347 (7)	—	—	—	—	99.97 ±0.06 248 (E=23.0)
	Checkerboard	50.76 17	54.69 39	67.10 83	76.68 171	87.33 347	91.55 699	92.24 1.4k (2)	—	—	91.74 ±0.84 735 (E=67.3)
	Breast cancer	97.36 185	95.05 431	94.62 923	94.62 1.9k	—	—	—	—	—	98.13 ±0.75 210 (E=2.2)
	Digits	94.62 1.4k	96.46 3.0k	97.11 6.1k (3)	—	—	—	—	—	—	96.42 ±0.97 3.4k (E=4.5)
	UCI HAR	94.73 7.9k	95.01 16.9k	95.47 34.8k (4)	—	—	—	—	—	—	95.22 ±2.24 13.3k (E=3.2)
	ISOLET	94.69 33.4k	95.10 68.0k	95.37 137.2k	94.90 275.6k	—	—	—	—	—	94.82 ±0.30 40.3k (E=2.4)
	MNIST	93.06 17.3k	94.19 36.1k	94.65 73.8k	94.90 149.1k	95.07 299.8k (4)	—	—	—	—	95.66 ±0.14 193.4k (E=20.7)
	Fashion-MNIST	84.83 17.3k	85.13 36.1k	85.53 73.8k	—	—	—	—	—	—	86.53 ±0.24 17.3k (E=2.0)
	Covertype	73.44 879	75.46 1.9k	78.15 3.8k	81.43 7.8k	84.96 15.7k	88.39 31.5k	—	—	—	91.04 ±0.10 47.5k† (E=96.3)
Regression	Rastrigin	0.512 11	0.513 27	0.517 59	0.516 123	0.512 251	0.689 507	0.936 1.0k	0.981 2.0k	0.988 4.1k (6)	0.990 ±0.004 3.6k† (E=454.7)
	Diabetes	0.499 43	0.442 107	—	—	—	—	—	—	—	0.528 ±0.041 43 (E=2.0)
	Kin8nm	0.691 35	0.821 87	0.897 191	0.922 399	0.927 815	—	—	—	—	0.932 ±0.004 716 (E=28.2)
	CT slice	0.846 1.5k	0.863 3.8k	0.863 8.5k	0.866 17.7k	0.872 36.2k (6)	0.817 73.1k (1)	—	—	—	0.930 ±0.030 21.5k (E=19.3)
	CCPP	0.937 19	0.940 47	0.942 103	0.945 215	0.946 439	0.947 887 (6)	—	—	—	0.949 ±0.004 575† (E=41.7)

Table 13: Validation score of the kept SPAWN-QL runs as experts are added. Each growth checkpoint reports the mean validation accuracy (%) or R^2 at the end of the growth cycle with E experts, and the mean number of parameters (red) over the runs that reached it, with their number in parentheses when fewer than ten. Growth continues past the retained checkpoint until the growth patience runs out. Final reports the retained model, i.e., the validation-best checkpoint after fine-tuning (Section 3.4), with the half width of its 95% CI, parameters and mean number of experts. A dagger marks datasets where at least one run stopped at its expansion-cycle cap.