Distance-Aware Error for Spline Networks: A Bottom-Up Approach to Uncertainty

Masoud Ataei, Mohammad Javad Khojasteh, *Member, IEEE*, and Vikas Dhiman, *Member, IEEE*

***Abstract*— We develop a new class of error bounds that are distance-aware and tightly characterize the approximation error induced by spline neural networks (SNNs). Our bottom-up approach starts by analyzing the error bounds of each neuron (a spline), and then extends to the neural network (NN). We begin with a review of error bounds for Newton's polynomial, which is then generalized to an arbitrary spline, under higher-order Lipschitz continuity assumptions. We then extend these bounds to function compositions, which are the core of deep networks such as Kolmogorov-Arnold networks (KANs). By the analysis of the propagation of approximation error through composed spline layers, we provide error bounds for the entire spline-based network. These bounds are deterministic, do not rely on sampling or probabilistic assumptions, and hold under mild regularity conditions. We evaluate our method on a variety of experiments, including estimating the object's shape from sparse laser scan points and safe navigation in an unstructured environment to show its effectiveness. We find that our method is faster than Gaussian process (GP) and Monte Carlo (MC) approaches, and that our error bounds reliably enclose the true error. We also develop a metric for the distance-awareness of an uncertainty estimator, and show that distance-aware uncertainty for Kolmogorov network (DAREK) is distance-aware in more regions of the space than the baselines. For ease of adoption, we provide a library that can be integrated into existing KAN models to estimate error bounds, available at: `https://masoud-ataei.github.io/DAREK/`**

***Index Terms*— error bounds, neural networks, splines, navigation, safe control**

I. Introduction

SPLINES are smooth, piecewise polynomials commonly used for approximation and modeling, particularly in control engineering and signal processing [2], [3]. The use of spline functions in neural networks (NNs) dates back to the early 1990s, where techniques such as cubic splines and B-splines (Basis splines) were employed to improve generalization and enable more efficient training [4], [5]. These early efforts aimed to leverage the smoothness and flexibility of splines to build data-efficient models. Recent advancements in computational power and the growing demand for more expressive and interpretable[1] models have sparked a renewed interest in spline neural networks (SNNs) [10]–[16]. In particular, Kolmogorov-Arnold network (KAN) [8] and its extensions [17]–[22] have adopted B-spline as activation functions, while drawing inspiration from the Kolmogorov-Arnold theorem (KAT) [23]. There is also growing interest in applying these models to control systems, physics-based problems, and trajectory planning [24], [25], though challenges related to uncertainty quantification and reliability remain unresolved.

Uncertainty estimation for machine learning (ML) models can be categorized into two classes: probabilistic methods and worst-case methods. Probabilistic methods estimate a probability distribution on the output of the ML model, while worst-case methods provide error bounds on the output. A popular class of probabilistic methods [26]–[29] uses Monte Carlo (MC) approaches to estimate uncertainty. These approaches are architecture invariant and can be applied to any NN, including SNNs, regardless of the number of layers and activation functions. However, being architecture invariant comes at a price. These methods ignore the inductive biases embedded into an architecture, resulting from the choice of layers and activation functions. Additionally, MC uncertainty estimation requires training wider networks [27] or training multiple networks [26], which increases computational cost.

Gaussian processes (GPs) [30] offer a probabilistic alternative approach that, unlike MC sampling methods, provides architecture-specific uncertainty quantification. In GPs, the architecture of the covariance function [30] determines how the predictions and uncertainty change with the input. Over the past decade, Deep GPs [31]–[33] have blurred the boundaries between NNs and GPs by facilitating the learning of covariance functions as well as interpreting NNs as GPs. Moreover, while infinitely wide NNs converge to GPs, this limit is impractical and necessitates approximation [34]–[36]. GPs' limitations are high computational cost and the need to store a large amount of training data for inference. However, scalability can be improved by selecting inducing points [33]. Nevertheless, kernel choice significantly impacts the performance of Deep GPs, and an ensemble of Deep GPs with diverse kernels can enhance flexibility and performance [32].

This work is supported by the National Science Foundation under Grant No. 2218063 and the Gleason Endowment at RIT. Preliminary results of this paper appeared in the Proceedings of the International Conference on Acoustics, Speech, and Signal Processing [1].

Masoud Ataei and Vikas Dhiman are with the Department of Electrical and Computer Engineering of University of Maine, Orono, ME 04469, USA. {masoud.ataei,vikas.dhiman}@maine.edu.

Mohammad Javad Khojasteh is with Electrical and Microelectronic Engineering Department of Rochester Institute of Technology, Rochester, NY 14623, USA. mjkeme@rit.edu.

[1]Interpretability in machine learning is widely valued but inconsistently defined [6], with prescriptions ranging from mechanistic [7] to symbolic explanations [8], [9]. In this paper, we pursue a form of interpretability based on "explanation by examples" [6], which emphasizes a human-understandable mapping between model inputs and outputs—by relating predictions to nearby training instances.

TABLE I
NOTATIONS AND DEFINITIONS OF IMPORTANT QUANTITIES.

Notation	Definition	Notation	Definition
f	True function	$[.]f$	Divided differences of a function f (defined in Eq 1)
$\boldsymbol{\tau}_{1:m}$	Vector or set of m knots, $\{\tau_1, \dots, \tau_m\}$	τ_i	The knot i in the set $\boldsymbol{\tau}_{1:m}$
$\mathcal{T}$	Matrix of knots	$\mathcal{T}_{i,j}$	The jth knot in ith spline
$\boldsymbol{\tau}_{1:k}^{j(x)}(\eta_{1:m})$	k-nearby points to x in set $\eta_{1:m}$ (Def. 1)	$\tau_i^{j(x)}$	ith element in the k-nearby knots $\boldsymbol{\tau}_{1:k}^{j(x)}$
$\mathcal{D}_f(\mathcal{X})$	Operator that creates a dataset of pairs of data $\mathcal{D}_f(\mathcal{X}) \coloneqq \{(\mathbf{x}, f(\mathbf{x}))\}_{\mathbf{x}\in\mathcal{X}}$	$\mathcal{P}_{k,n}[\mathcal{D}_f(\boldsymbol{\tau}_{1:m})](x)$	The nth piece of kth-order Newton's piecewise polynomial approximation of f over set of knots $\boldsymbol{\tau}_{1:m}$ for $x \in [\tau_n, \tau_{n+1})$
$b_{k,i}(x)$	Spline-basis of order k non-zero between $[\tau_i, \tau_{i+1})$ (Def. 2)	$w_{k,i}$	The weight of basis $b_{k,i}$
$\phi_k[\boldsymbol{\tau}_{1:m}]$	A spline of order k on the set of knots $\boldsymbol{\tau}_{1:m}$	$\mathcal{L}_f^k$	kth order Lipschitz constant of f (defined in Eq 10)
$\bar{u}_f(x; \boldsymbol{\tau}_{1:m})$	The error bound of f over set of knots $\boldsymbol{\tau}_{1:m}$ evaluated at x with zero error at knots	$u_f(\mathbf{x}; \boldsymbol{\tau}_{1:m})$	The uncertainty estimator that calculates the error bound of function f over set of knots $\boldsymbol{\tau}_{1:m}$ evaluated at x with non-zero error at knots.
$\mathcal{X}$	Input space	$\mathcal{X}_{\mathcal{D}}$	Set of training inputs
$\hat{f}$	A piecewise polynomial approximation of f on set of knots $\boldsymbol{\tau}_{1:m}$	$\hat{f}_j$	The jth piece of $\hat{f}$ defined on $x \in [\tau_j, \tau_{j+1})$
e_j^f	True error which is difference between the true function f and the approximation $\hat{f}$ at interval (τ_j, τ_{j+1})	$\mathbf{1}_n$	A vector of n ones

Probabilistic uncertainty approaches are often avoided in safety-critical applications due to their computational complexity, large sample size requirements, reliance on hyper-parameters [37], and hazards of poorly specified prior [38]. We instead focus on worst-case analysis [39], which is easier to understand, implement, and debug. Moreover, they are robust to bounded input noise and adversarial attacks, and effective where probabilistic assumptions might not hold due to insufficient data. Furthermore, worst-case analysis is appropriate for safety-critical systems, where the costs of ignoring low-probability events are very high. Combined with suitable robust control algorithms, worst-case analysis is akin to placing a safety cage around dangerous machinery, ensuring a system's robustness, reliability, and failure resilience, even when a single failure can lead to disastrous consequences.

Unlike Bayesian neural networks (BNNs), we develop a *distance-aware* error analysis for SNNs, meaning the error bound at a given input is a monotonically increasing function of its distance from the training data [40]–[43]. Intuitively, the predictions of a well-trained model are more accurate within the neighborhood of the training data, and the model should avoid being overly confident when confronted with out-of-distribution inputs, which is numerically demonstrated in [40]. Distance-awareness is crucial in safety-critical applications, such as autonomous driving and human-robot interaction, where decisions must be made with caution based on how novel the input is compared to the training data. Liu et al. [8] provide error bounds for KANs, but these are potentially loose, not distance-aware, and only defined up to an unknown constant. In this work, we focus on KANs and study distance-aware uncertainty estimation for SNNs, and its implications for sensing and control.

Our approach relies on a trained KAN network, except that instead of randomly sampling the knots[2], we restrict them to be from the training data. We build an uncertainty estimate for a KAN network bottom-up. We first estimate the uncertainty of a single neuron in the network, which is then composed layer-wise to estimate the uncertainty of a complex network. To estimate uncertainty at a neuron level, we use Newton's polynomial error bound [44], which assumes the function to be higher-order Lipschitz continuous and function approximation to be a piecewise continuous polynomial (Section VI-A). We get the required Lipschitz constant for Newton's polynomial by assuming higher-order continuity on the network-level true function, which is then attributed to the continuity at the neuron level (Section VI-D). Similarly, errors made at the knots in the network level function approximation are attributed to the neuron level errors (Section VI-E). Given the neuron-level error bounds, we compose the errors to a two-layer network for ease of exposition and then generalize it to a multi-layer network (Section VI-B). Finally, we extend error bounds to layers that were so far restricted to be piecewise continuous polynomials, which allow for an additive smooth function (e.g., SiLU, Section VI-C).

In summary, we introduce distance-aware uncertainty for Kolmogorov networks (DAREKs), a bottom-up approach that is more *efficient* than traditional GPs, and provides *distance-aware* uncertainty bounds that are more *interpretable* and *tighter* than those provided by BNNs. We specifically make the following **contributions**: 1) establish analytical worst-case *distance-aware* error bounds for a multi-layer spline neural network; 2) propose and evaluate approaches for Lipschitz attribution and error attribution between network layers and neurons; 3) extend error bounds to layers with a sufficiently smooth function added to the spline neurons; 4) develop a metric for distance-awareness and compare commonly used uncertainty estimators for distance-awareness; and 5) validate the scalability and efficacy of our method, as compared to GPs and Ensembles, on various examples, including object shape estimation and safe navigation in an unstructured environment.

[2]"Spline" originates from flexible wooden strips used by shipbuilders, constrained by control points (or knots).

TABLE II
EXPANSIONS OF IMPORTANT ACRONYMS.

Acronym	Expansion
NN	Neural network
SNN	Spline neural network
BNN	Bayesian neural network
GP	Gaussian process
MC	Monte Carlo
RBF	Radial basis function
ReLU	Regularized linear unit
SNGP	Spectral-normalized Gaussian processes
KAT	Kolmogorov-Arnold theorem
KAN	Kolmogorov-Arnold network
PPE	Piecewise polynomial error
DAREK	Distance-aware uncertainty for Kolmogorov network
MPC	Model predictive control
CBF	Control barrier functions

II. RELATED WORK

BNNs quantify uncertainty by treating network parameters as random variables and learning a posterior distribution over them, rather than relying on fixed weights that yield a single-point prediction [45], [46]. Computing the exact posterior distribution is intractable; therefore, several approximate inference methods have been proposed. MC Dropout [27] assumes a Bernoulli distribution over the weights. Bayes By Backprop (BBB) and other variational bayes approaches [47] use variational inference to learn weight distribution [45], and stochastic gradient Langevin dynamics (SGLD) applies stochastic gradients to sample from the posterior [48]. Probabilistic backpropagation (PBP) developed an approximation technique that directly updates weight distribution in the network, without relying on global variational inference [49]. Deep ensembles train multiple models with different initializations, which have been shown to improve generalization [26]. Other approaches include randomized maximum a posteriori (MAP) sampling [50], which injects noise into the optimization process to mimic posterior sampling, and the Laplace approximation for estimating unimodal posteriors [51]. While BNNs can provide confidence levels, they often require careful calibration to reflect the true posterior [52]. Ataei et al. [53] propose an approach to effectively estimate well-calibrated [54], [55] uncertainty in both in-domain and out-of-domain settings [56]. While these approaches estimate a probabilistic *distance-unaware* uncertainty, we adopt a worst-case *distance-aware* analysis instead.

Distance-awareness is a desirable property of uncertainty estimators, which ensures higher uncertainty in regions far from the training data [40]. GPs are non-parametric models that interpolate between training data, providing mean predictions and uncertainty estimates using a given kernel function. These models are inherently distance aware because the GP uncertainty increases with the decrease in kernel value, which acts as a proxy for inverse-distance [30]. The computational complexity of traditional GPs is high, and several approximations have been developed for specific applications; however, each of these methods introduces its trade-offs [57]–[60].

Deep kernel learning (DKL) maps high-dimensional inputs to a feature space using a NN and applies a GP in feature space. Several researchers [40]–[43] explore DKL for distance-aware uncertainty estimation. Spectral-normalized Gaussian processes (SNGP) [40] applies spectral normalization to make the network Lipschitz continuous and replaces the final layer with a scalable GP approximation using random Fourier features, which causes the predictive uncertainty to converge to zero when the number of data points goes to infinity, even for inputs far from the training data [43]. Deterministic uncertainty quantification (DUQ) [42] focuses on stabilizing distance-aware uncertainty using RBF in feature space, with constraints to prevent feature collapse. Feature collapse happens when substantially different model inputs have identical representations in feature space. DUQ requires modeling the centroid of each class; hence, it does not generalize to regression problems. Deterministic uncertainty estimation (DUE) [43] improves on DUQ by enforcing bi-Lipschitz constraints, similar to SNGP [40], and uses inducing point GPs to retain non-parametric behavior, resulting in uncertainty estimates that are more robust, sensitive to prior choice and architectural details. Deep deterministic uncertainty (DDU) [41] fits a class-conditional Gaussian density in the feature space of a spectrally-normalized network to detect out-of-distribution inputs, making it computationally efficient but primarily empirical in nature. One drawback of DKL methods is that they estimate uncertainty using GP only after feature mapping and ignore the uncertainty induced by the learned mapping. We propose a *bottom-up* approach that computes the uncertainty of SNNs and hence can be combined with DKL approaches. Moreover, these methods are probabilistic, whereas we do a worst-case analysis for the Spline-based network. Our method uses Newton's polynomial definition to bring distance awareness in error calculation and uses the Lipschitz constant to determine the bounds.

Error estimation of splines has been extensively studied, with global error bounds derived for cubic splines [16], [44], [61]. Takacs et al. [61] found the error bound of B-splines with maximum smoothness and fixed knot interval, and showed that this error is independent of the polynomial degree, later extended to arbitrary smoothness [62]. Although these works focused on *distance-unaware* error bounds, [44] provides distance-aware error bounds and shows that placing knots at Chebyshev points can significantly reduce error. However, this analysis assumes zero error at knots. Our work introduces a *distance-aware* error bound for spline approximation with non-zero error at knots and tracks it in a hierarchical spline network, which has not been previously explored.

Spline neural networks use splines as activation functions to enhance NN architectures. Early works, such as [10], replaced the univariate functions in the Kolmogorov-Arnold theorem [23] with cubic splines. At the same time, Compolucci et al. [63] and Harris et al. [64] introduced learnable spline and B-spline activations to improve flexibility. Hong

et al. [65] introduce a complex-valued B-spline network for Wiener systems. According to [16], using at least second-degree splines in SNNs ensures differentiability for gradient-based training, while dimensionality reduction and tensor-product B-splines help manage model complexity. Additionally, a *distance-unaware* upper bound for deep regularized linear unit (ReLU) networks is derived [66], which can be viewed as first-order spline networks. Also, [67] demonstrates that any deep ReLU network can be equivalently represented as a multivariate spline network. Moreover, KAN [8] introduces *distance-unaware* error bounds for hierarchical spline networks, with an unknown constant of proportionality. Recently, SNNs have been extended to convolution NNs as well [68]. Our algorithm, DAREK, provides a novel distance-aware error bound for spline networks, a more precise characterization of approximation errors that enhances reliability and interpretability.

III. Overview of DAREK

DAREK is a bottom-up approach to worst-case uncertainty (error bound) estimation in SNNs. We begin by finding error bounds of each neuron and then combine them to estimate the error bounds of the full network. Fig. 1 provides an overview of the DAREK algorithm. Fig. 1 (a) illustrates different components of the error bounds for a single neuron, denoted as PPE block, and (b) shows how predictions and error bounds from each PPE block contribute to produce the model's prediction, as well as the associated error bounds estimate.

DAREK estimates the uncertainty of a trained SNN with input $\mathbf{x} \in \mathbb{R}^d$, output $\hat{\mathbf{y}} \in \mathbb{R}^m$, and corresponding error bound $\mathbf{u} \in \mathbb{R}^m$, under reasonable assumptions of continuity that are enumerated later. Predictions and error bounds are computed layer by layer and combined to obtain the NN's error bound. The core component of DAREK is PPE block, which maps each input $\hat{y}_{l-1}$ to a prediction $\hat{y}_l$ and associated error u_l. This error estimation is driven by aggregating two sources of error: interpolation error $\bar{u}$ (Theorem 1) and error at knot $\mathcal{P}_{k,j}[e_j]$ (Lemma 1). Each layer, $\mathbf{h}_l : \mathbb{R}^{m_{l-1}} \to \mathbb{R}^{m_l}$ for $l \in [1, \dots, L]$, forms output as linear combination of PPEs predictions and errors as linear combination of PPEs errors. The propagated error is computed by scaling the sum of errors from the previous layer by the Lipschitz constant of the current layer and adding the current layer's error, $\mathcal{L}^1_{l-1} \sum \mathbf{u}_{l-1} + \mathbf{u}_l$. Further details are provided in Section VI.

By selecting spline knots from training data, our worst-case error analysis for SNN leverages a semi-parametric model that combines the accuracy and efficiency of parametric methods (e.g., NNs) with the interpretability and analyzability of non-parametric models (e.g., GPs). As discussed below, this design also improves computational efficiency compared to state-of-the-art methods.

Computational Cost: Consider an Monte Carlo (MC) method that estimates uncertainty by sampling n different sets of NN weights. Suppose each NN evaluation has a computational cost of $\mathcal{O}(LH^2)$, where L is the number of layers and H is the average width of the hidden layer. In that case, MC sampling results in a computational complexity of $\mathcal{O}(nLH^2)$. This method is more efficient than exact GPs, which requires matrix inversion with a cost of $\mathcal{O}(N^3)$, where N is the number of data samples and is typically greater than L, H, and n. In contrast, DAREK identifies the $k+1$ closest knots to a given test point at each layer among a set of m knots (see Algorithm 1 for more details). If the $k+1$-consecutive-closest-knots are identified using a binary tree, then the test-time complexity is $\mathcal{O}(\log_2(m)LH) + \mathcal{O}(LH^2)$. The binary tree is constructed once as a post-training step. Typically, MC methods use $n \in \{5, \dots, 20\}$ sampled networks, while we use $H \in \{2, \dots, 20\}$ and $m \in \{10, \dots, 100\}$ for DAREK. When $m < 2^n H$, which is a common scenario, DAREK is more efficient than MC methods. A numerical comparison of computational costs is also demonstrated in Fig. 4.

IV. Background

Notations: Let $\boldsymbol{\tau}_{i:j} \coloneqq \{\tau_i, \tau_{i+1}, \dots, \tau_j\}$ be a sorted sequence of distinct points, and $f : \mathcal{X} \subseteq \mathbb{R} \to \mathbb{R}$ be a scalar function. Given $\boldsymbol{\tau}_{1:m}$ with $\tau_i \in \mathcal{X}$, the set of input-output pairs is $\mathcal{D}_f(\boldsymbol{\tau}_{1:m}) \coloneqq \{(\tau_i, f(\tau_i))\}_{i=1}^m$. Separately, let $\mathcal{D} \coloneqq \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ denote the training dataset, and define its input set as $\mathcal{X}_\mathcal{D} \coloneqq \{\mathbf{x}_i\}_{i=1}^n$. We use $\mathbb{R}^+$ to denote the set of non-negative real numbers. We use capital letters for matrices, lowercase boldface for vectors, and lowercase regular font for scalars. The notation $[.]f$ represents the divided difference of function f on a non-repeating set $\boldsymbol{\tau}_{i:j}$, which is defined recursively as follows [44]:

$$[\tau_1]f \coloneqq f(\tau_1), \qquad [\tau_1, \tau_2]f \coloneqq \frac{[\tau_2]f - [\tau_1]f}{\tau_2 - \tau_1},$$
$$[\boldsymbol{\tau}_{1:k}]f \coloneqq \frac{[\boldsymbol{\tau}_{2:k}]f - [\boldsymbol{\tau}_{1:k-1}]f}{\tau_k - \tau_1}. \tag{1}$$

Intuitively, divided differences are closely related to numerical derivatives. Important notations and definitions used in the paper are summarized in Table I.

Our objective is to determine efficient worst-case distance-aware uncertainty bounds for hierarchical spline architectures. To achieve this, we introduce our definitions of knot selection, distance awareness, piecewise polynomial, and Lipschitz constant. We then draw on Newton's polynomial, which offers a tight bound for piecewise polynomials [44], and review relevant background, including B-splines, KAT [23], and KAN [8].

Definition 1 (k-nearby knots): Let $\eta_{1:m} = \{\eta_1, \dots \eta_m\}$ be a sorted sequence of m unique real knots. Define k-nearby knots $\boldsymbol{\tau}^{j(x)}_{1:k}(\eta_{1:m})$, with $k \geq 2$, given a number $x \in [\eta_1, \eta_m)$ to be a set of k knots nearby x chosen such that:

1) the nearby knots are *consecutive*,
2) the nearby knots includes both nearest knots $\{\eta_{j(x)}, \eta_{j(x)+1}\}$ with nearest-pair index $j(x)$ defined such that $\eta_{j(x)} \leq x < \eta_{j(x)+1}$, and
3) the nearby knots are a *deterministic* function of the nearest-pair index $j(x)$.

We shorten the notation to $\boldsymbol{\tau}^{j(x)}_{1:k}$ when the set of knots $\eta_{1:m}$ are clear from the context.

There are several options for choosing the k-nearby knots. For example, one can choose the right-window as $\boldsymbol{\tau}^{j(x),\text{right}}_{1:k}(\eta_{1:m}) \coloneqq \{\eta_{j(x)}, \dots, \eta_{j(x)+k-1}\}$, or the left window

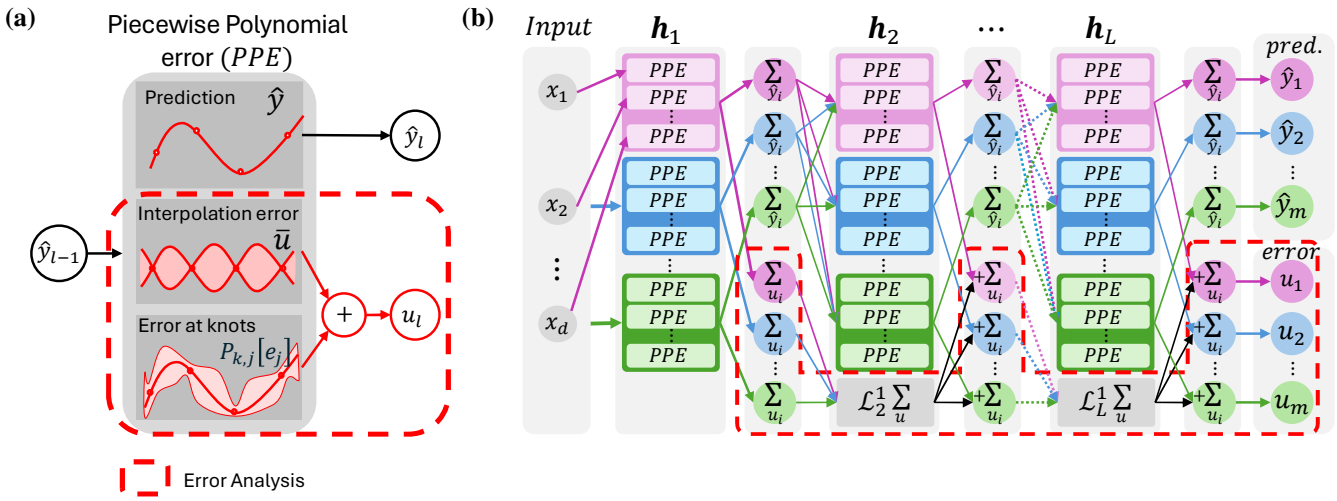

Fig. 1. Block diagram of DAREK. **(a)** PPE takes a scalar $\hat{y}_{l-1}$ as input and computes three values: 1) $\hat{y}_l$ - the spline value at given input, 2) $\bar{u}_l$ - the interpolation error using Theorem 1, and 3) $\mathcal{P}_{k,j}[e_j]$ - the error at knot using Lemma 1. The output of the PPE consists of the mean prediction $\hat{y}_l$ and error bound term $u = \bar{u} + \mathcal{P}_{k,j}[e_j]$. **(b)** DAREK architecture consists of L layers, where each layer h_l has m_l block of PPEs and each has m_{l-1} PPEs in it. The cumulative error bound up to layer l is computed as: $\mathbf{u}_l = \mathbf{u}_{h_l} + \mathcal{L}_l^1 \sum \mathbf{u}_{l-1}$ Theorem 2.

$\boldsymbol{\tau}_{1:k}^{j(x),\text{left}}(\eta_{1:m}) := \{\eta_{j(x)-k+2}, \ldots, \eta_{j(x)+1}\}$, or the left k-nearest neighbors $\boldsymbol{\tau}_{1:k}^{j(x),\text{lNN}}(\eta_{1:m}) = \text{KNN}_{k-1}(\eta_{j(x)}, \eta_{1:m} \setminus \{\eta_{j(x)+1}\}) \cup \{\eta_{j(x)+1}\}$ where $\text{KNN}_{k-1}(x, \mathcal{X})$ finds the $k-1$ nearest neighbors of x in the set $\mathcal{X}$. One counter example that does not agree with our definition is if one takes the nearest neighbors of x directly, $\boldsymbol{\tau}_{1:k}^{j(x),\text{NNx}}(\eta_{1:m}) = \text{KNN}_{k-1}(x, \eta_{1:m}) \cup \{\eta_{j(x)}, \eta_{j(x)+1}\}$, as this may result in a different set of knots depending on x even when $j(x)$ is fixed. Our theory in the rest of the paper is independent of the choice of this method as long as the same method is used consistently. In our experiments, we use the k-nearest neighbor approach, $\boldsymbol{\tau}_{1:k}^{j(x),\text{NN}}(x)$. We use $\tau_i^{j(x)}$ to denote ith element in $\boldsymbol{\tau}_{1:k}^{j(x)}$.

Definition 2 (Newton's polynomial operator): Given data points $\mathcal{D}_f(\boldsymbol{\tau}_{1:m})$, we can fit a kth-order Newton's polynomial using $k+1$ knots, ensuring it passes through all the selected knots. For each interval $x \in [\tau_n, \tau_{n+1})$, let $\boldsymbol{\tau}_{1:k+1}^{(j)}$ to be $k+1$ nearby knots (Definition 1). Then the piecewise polynomial fit in the nth interval $x \in [\tau_n, \tau_{n+1})$ is given by [44, p7]:

$$\mathcal{P}_{k,n(x)}[\mathcal{D}_f(\boldsymbol{\tau}_{1:m})](x) := \left[\tau_1^{n(x)}\right] f + \sum_{i=2}^{k+1} \left[\boldsymbol{\tau}_{1:i}^{n(x)}\right] f \prod_{j=1}^{i-1} \left(x - \tau_j^{n(x)}\right). \quad (2)$$

Any $k+1$ distinct points sampled from a kth-order polynomial determine a unique Newton's polynomial with the same coefficients, regardless of the chosen points. This implies that for a polynomial $f(x)$ of order k, the Newton's polynomial satisfies $\mathcal{P}_{k,n}[\mathcal{D}_f(\boldsymbol{\tau}_{n:n+k})](x) = f(x)$ [44].

Definition 3 (Piecewise polynomial): A piecewise polynomial $\hat{f}(x)$ of order k on a given set of knots $\boldsymbol{\tau}_{1:m}$ consists of $m-1$ polynomials $\hat{f}_{[j]}(x)$ of order k, as follows:

$$\hat{f}(x) = \sum_{i=0}^{k} c_{i,j} x^k =: \hat{f}_{[j]}(x) \qquad \forall x \in [\tau_j, \tau_{j+1}),$$
$$\text{s. t.} \quad \hat{f}_{[j]}(\tau_{j+1}) = \hat{f}_{[j+1]}(\tau_{j+1}). \quad (3)$$

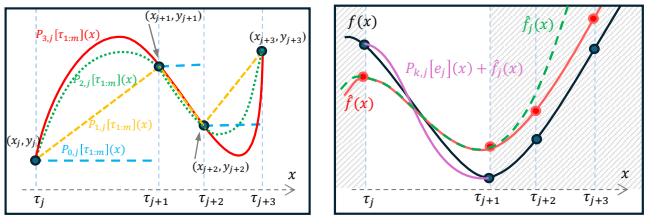

Fig. 2. **left)** Newton's piecewise polynomial of different orders is shown, with **black circles** representing selected knots at real values. The **blue line** shows a zero-order piecewise polynomial, the **yellow line** is the first-order, the **green line** represents a quadratic Newton's polynomial, and the **red line** is the cubic polynomial on the selected knots. **right)** The plot illustrates the notations for error as defined in Lemma 1. Black, red, green, and pink lines are true functions, spline approximation with non-zero error at knots, Newton's polynomial of interval j, and the adjusted polynomial of interval j to compensate for non-zero error, respectively.

The polynomial $\hat{f}_{[j]}(x)$ represents the jth piece of $\hat{f}(x)$. Piecewise polynomials are continuous but not necessarily smooth. When we refer to $\mathcal{P}_{k,j}$, we also mean the jth piece of the piecewise polynomial.

An illustration of this operator for different k values is shown in Fig. 2 (left). Newton's polynomial is rarely used for spline fitting due to non-smooth transitions at knots. Smooth splines are typically constructed using B-splines. We base our spline error analysis on Newton's polynomial, as they might provide tight distance-aware error bounds (cf. Theorem 1).

B-splines: A B-spline of order k is a smooth piecewise polynomial of the same order, defined in terms of basis functions called B-spline basis. B-splines are defined on m knots $\boldsymbol{\tau}_{1:m}$ for ks greater than one as:

$$b_{k,i}(x) := (t_{i+k} - t_i)[t_i, \ldots, t_{i+k}](. - x)_+^{k-1}, \quad (4)$$

where $(x)_+ = \max\{0, x\}$ [69]. Any smooth spline can be written as a linear combination of B-spline basis functions [44, p97], $\phi_k[\boldsymbol{\tau}_{1:m}](x) = \sum_{i=1}^{m-k} \alpha_i b_{k,i}(x) = \boldsymbol{\alpha}^\top \mathbf{b}_{k,\boldsymbol{\tau}_{1:m}}(x)$ where $\alpha_i \in \mathbb{R}$ are the coefficients and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\beta)^\top$,

$\mathbf{b}_{k,\boldsymbol{\tau}_{1:m}}(x) = (b_{k,i}(x))_{i=1}^{\beta}$ are corresponding basis vectors. We denote $\beta \coloneqq m-k$ as the size of the kth-order B-spline basis for m knots and emphasize the dependence of the basis functions on the knots $\boldsymbol{\tau}_{1:m}$ by including it in the subscript. Note that, in total, we have m knots, $m-k$ B-splines, and $m-1$ polynomials defined between two consecutive knots.

Kolmogorov-Arnold Theorem: For any given continuous multivariate function $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}; f \in \mathcal{C}^0$; Kolmogorov-Arnold theorem (KAT) [23] states that the following two-layer representation exists with a set of continuous univariate activation functions, $\{h_{l,i,j} \in \mathcal{C}^0\}_{l,i,j}$, $f(\mathbf{x}) = \sum_{i=1}^{2d+1} h_{2,1,i}(\sum_{j=1}^{d} h_{1,i,j}(x_j))$.

A natural question is whether the continuity assumptions for the multivariate function f and univariate activations functions $h_{l,i,j}$ can be strengthened (higher order continuity, $f, h_{l,i,j} \in \mathcal{C}^k : k \geq 1$) or simplified (discontinuous $f, h_{l,i,j} \notin \mathcal{C}^0$). Notably, the theorem does not hold for continuously differentiable multivariate ($f \in \mathcal{C}^1$) and univariate activation functions ($h_{l,i,j} \in \mathcal{C}^1$) [70]. By increasing the number of layers in the KAT [8] and relaxing equality to approximation [71], one can extend KAT representation results with weaker or alternative assumptions [72]–[74], and one of the extensions is discussed next.

Kolmogorov-Arnold Networks: KAN [8] replaces the univariate activation functions in KAT with splines. A two-layer model $\text{KAN}_2 : \mathbb{R}^{m_1} \to \mathbb{R}$ can be written as,

$$\text{KAN}_2(\mathbf{x}) = \sum_{i=1}^{m_2} \phi_{2,1,i}\big(\sum_{j=1}^{m_1} \phi_{1,i,j}(x_j)\big),$$

where $\phi_{l,i,j}$ is a kth-order spline in layer l that connects input j to output i. Consider the case when $\phi_{l,i,j}(x_j) = \boldsymbol{\alpha}_{l,i,j}^{\top} \mathbf{b}_{k,\boldsymbol{\tau}_{l,j}}(x_j)$. Then KAN can be written as:

$$\text{KAN}_2(\mathbf{x}) = \mathbf{w}_2^{\top} \mathbf{B}_{k,\boldsymbol{\tau}_2}(\mathbf{W}_1 \mathbf{B}_{k,\boldsymbol{\tau}_1}(\mathbf{x})), \tag{5}$$

where $\mathbf{W}_1 = ([(\boldsymbol{\alpha}_{1,i,j})_{j=1}^{m_1}]^{\top})_{i=1}^{m_2} \in \mathbb{R}^{m_2 \times m_1 \beta}$, $\mathbf{w}_2 = (\boldsymbol{\alpha}_{2,1,i})_{i=1}^{m_2} \in \mathbb{R}^{m_2 \beta}$, $\mathbf{B}_{k,\boldsymbol{\tau}_1}(\mathbf{x}) = (\mathbf{b}_{k,\boldsymbol{\tau}_{1,j}}(x_j))_{j=1}^{m_1}$ and $\mathbf{B}_{k,\boldsymbol{\tau}_2}(\mathbf{y}) = (\mathbf{b}_{k,\boldsymbol{\tau}_{2,i}}(y_i))_{i=1}^{m_2}$. Here the notation $(\mathbf{v}_i)_{i=1}^{m_2}$ denotes vertical stacking of the column vectors $\mathbf{v}_i$ into a single column vector. Equation (5) is a multi-layer perceptron (MLP) with $\mathbf{B}_{k,\boldsymbol{\tau}_2}$ as its activation function and $\mathbf{B}_{k,\boldsymbol{\tau}_1}$ as a feature map of the inputs $\mathbf{x}$. Since the basis functions depend on the knots, and the model weights are learned from the training data, KAN is a *semi-parametric* model that brings the computational efficiency and accuracy of parametric models (e.g. NNs) with the interpretability of non-parametric models (e.g. splines, k-nearest neighbors). We use this property of KAN to develop a distance-aware uncertainty estimator.

For ease of notation, let define ith output of layer l, which has m_l components, as $y_{l,i} = \sum_{j=1}^{m_{l-1}} \phi_{l,i,j}(y_{l-1,j})$ and layer l of network predicts the output $\mathbf{y}_l = \mathbf{h}_l(\mathbf{y}_{l-1}) = [y_{l,i}]_{i=1}^{m_l} = [\sum_{j=1}^{m_{l-1}} \phi_{l,i,j}(y_{l-1,j})]_{i=1}^{m_l}$, and $\mathbf{y}_0 = \mathbf{x}$. With this, in general, a spline neural network (SNN) consists of multiple KAN layers each represented by $\mathbf{h}_l : \mathbb{R}^{m_l} \to \mathbb{R}^{m_{l+1}}$, which maps a m_l dimensional input to m_{l+1} dimensional output. Therefore, an L layer multi-input multi-output KAN can be written as:

$$\text{KAN}_L(\mathbf{x}) = \mathbf{h}_L \circ \cdots \circ \mathbf{h}_l \circ \cdots \circ \mathbf{h}_1(\mathbf{x}). \tag{6}$$

Distance awareness: Distance-aware uncertainty is driven by the intuition that a model's prediction uncertainty increases with distance from the training data. We define distance awareness for worst-case analysis inspired by [40].

Definition 4 (Input distance awareness): Consider a predictive model $\hat{y} = \hat{f}(\mathbf{x})$ that estimates a true function $y = f(\mathbf{x})$ using $x \in \mathcal{X}_{\mathcal{D}}$. Let $u_{\hat{f}}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^+$ be an uncertainty estimator such that for all $\mathbf{x} \in \mathcal{X}$, the true output is bounded $y \in [\hat{f}(\mathbf{x}) - u_{\hat{f}}(\mathbf{x}), \hat{f}(\mathbf{x}) + u_{\hat{f}}(\mathbf{x})]$. The uncertainty estimator $u_{\hat{f}}(\mathbf{x})$ is said to be distance-aware if it monotonically increases with the test point's distance from the training data with respect to some distance function $d(\mathbf{x}, \mathcal{X}_{\mathcal{D}})$.

Unlike Liu et al. [40], who define the set distance $d(., \mathcal{X}_{\mathcal{D}})$ as the expected distance of a test point to all the training data, we define it to be the distance of the test point from the nearest training point (or knot) τ^*,

$$d(\mathbf{x}, \mathcal{X}_{\mathcal{D}}) = \min_{\tau \in \mathcal{T}} d_u(\mathbf{x}, \tau) = d_u(\mathbf{x}, \tau^*), \tag{7}$$

where $\mathcal{T} \subset \mathcal{X}_{\mathcal{D}}$ is the set of training points (or knots). We call this distance function $d_u(\mathbf{x}, \tau^*)$ to be the *inducing distance* for the uncertainty estimator $u_{\hat{f}}$. The inducing distance does not need to be Euclidean, or ℓ_p, it can be a geodesic distance over the data manifold. If both the uncertainty estimator and the inducing distance are differentiable, the distance-awareness property can be compactly stated as,

$$[\nabla_{\mathbf{x}} u_{\hat{f}}(\mathbf{x})]^{\top} \nabla_{\mathbf{x}} d_u(\mathbf{x}, \tau^*) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}. \tag{8}$$

The above condition enforces that the uncertainty monotonically increases with distance.

Measuring Distance-Awareness: Evaluating (8) over the entire input space is intractable, as it requires global monotonicity verification over $\mathcal{X}$. Therefore, we adopt a sampling-based metric. For computational efficiency and consistency across uncertainty estimators, we assume the inducing distance to be Euclidean.

Definition 5 (Sampled Distance-Awareness (SDA)): Let $\mathbf{x}_t \sim \mathcal{X}_{\text{test}}$ be sampled uniformly from the test data. Then the *sampled distance-awareness* (SDA) of u is defined as,

$$\text{SDA} = \frac{1}{N} \sum_{\mathbf{x}_t \sim \mathcal{X}_{\text{test}}} \mathbb{1}\left[\nabla_{\mathbf{x}} u_{\hat{f}}(\mathbf{x}_t)^{\top} (\mathbf{x}_t - \boldsymbol{\tau}^*) \geq 0\right], \tag{9}$$

where $\mathbb{1}[.]$ is the indicator function. A higher SDA value indicates that, with high probability, uncertainty increases as one moves farther from the nearest knot, which captures the distance-awareness property of an uncertainty estimator.

We assume Lipschitz continuity to obtain distance-awareness bounds, as it is a common assumption in control literature, and many real-world functions obey this criterion. Since we are working with kth-order splines, we define an extended notion of kth-order Lipschitz continuity as follows:

Assumption 1 (kth-order Lipschitz continuity): For a $k-1$ time differentiable function $f : [a, b] \to \mathbb{R}$, the kth-order Lipschitz constant $\mathcal{L}_f^k$ is the maximum rate of change in the $(k-1)$th derivative of the function f to the change of input,

$$\frac{|f^{(k-1)}(x) - f^{(k-1)}(y)|}{d(x, y)} \leq \mathcal{L}_f^k \quad \forall x \neq y. \tag{10}$$

If such a constant exists, the function is kth-order Lipschitz continuous. The inequality holds for all $x, y \in [a, b]$. As y approaches x, the left-hand side converges to the derivative, $\lim_{y\to x} \frac{|f^{(k-1)}(x)-f^{(k-1)}(y)|}{|x-y|} = |f^{(k)}(x)|$. Thus, the inequality is valid for all x and y, and it is applied to the maximum of the kth derivative as well. When the distance function $d(x, y) = |x-y|$ is the absolute difference and the function f is k times differentiable, then $\max_{x\in[a,b]} |f^{(k)}(x)| \leq \mathcal{L}_f^k$. Note that for a multivariate function f, the Lipschitz constant of its derivatives is the supremum of changes in f along any direction; see Equation (23), similar to Lemma 1 in [75].

If we assume a function f and its approximator $\hat{f}$ to be first-order Lipschitz continuous with constant $\mathcal{L}_f^1$, then a trivial distance-aware uncertainty estimator is

$$\bar{u}_{\hat{f}}^{\ell}(\mathbf{x}) = (2 + \epsilon_{\text{safe}})\mathcal{L}_f^1 d(\mathbf{x}, \tau^*). \tag{11}$$

Here, we considered the true function and approximation to have the same Lipschitz constant and ϵ_{safe} as a safety margin. In the next section, we extend this notion to tighter bounds for higher-order splines and later multi-layer KAN networks.

V. Problem formulation

We formalize the problem of worst-case uncertainty estimation from the perspective of safe control. Suppose the function being learned (e.g., system dynamics or cost-to-go) is $f : \mathbf{x} \mapsto y \in \mathbb{R}$. Also, we know a set of assumptions about the true functions so that we can constrain it to a function class $f \in \mathcal{F}$. In this paper, we consider the class $\mathcal{F}$ to be a class of all $k+1$-order Lipschitz functions with known constants $\mathcal{L}_f^{k+1}$. We also have a dataset of noisy input-output pairs available to learn the desired function f, $\mathcal{D} \coloneqq \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $y_i = f(\mathbf{x}_i) + \epsilon_i \in \mathbb{R}$ are the corresponding observed (possibly noisy) outputs, with bounded noise $\epsilon_i \leq \epsilon_{\text{max}}$. The first step is to learn the function approximation $\hat{f}(\mathbf{x}; \Theta) \in \mathcal{H}$ from the hypothesis space $\mathcal{H}$, where Θ is the combination of spline parameters and knots locations required for the function representation. In this paper, our hypothesis space is the class of multi-layer KAN networks with a fixed architecture. Let a learning algorithm provide us the optimal parameters Θ^* according to (12), with $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ as a given loss function. Our objective is to find the worst-case bound for the learned function approximation. With slight abuse of notation, let $\mathcal{F} \cap_\epsilon \mathcal{D}$ denote the space of all functions in $\mathcal{F}$ that may generate the dataset $\mathcal{D}$, $\mathcal{F} \cap_\epsilon \mathcal{D} \coloneqq \{g \in \mathcal{F} \mid l(g(\mathbf{x}_i), y_i) \leq \epsilon_{\text{max}}, \forall(\mathbf{x}_i, y_i) \in \mathcal{D}\}$[3]. Then the uncertainty bound $u_f(\mathbf{x})$ at a given test point $\mathbf{x}$ is the loss if the true function is chosen to be the worst possible in the given range of possibilities,

$$\Theta^* = \arg\min_{\Theta} \sum_{(\mathbf{x}_i, \mathbf{y}_i)\in\mathcal{D}} l(\hat{f}(\mathbf{x}_i, \Theta), \mathbf{y}_i), \tag{12}$$

$$u_f(\mathbf{x}) \geq \sup_{f\in\mathcal{F}\cap_\epsilon\mathcal{D}} l(f(\mathbf{x}), \hat{f}(\mathbf{x}; \Theta^*)). \tag{13}$$

In this paper, we do not modify the training process (12), but focus on estimating the tight worst-case bound $u_f(\mathbf{x})$. Optimizing over $\mathcal{F}\cap_\epsilon\mathcal{D}$ is in general intractable. In this paper, we focus on a class of kth-order Lipschitz smooth functions.

[3]This bounded feasible set can be extended to probabilitic set as well [76], $\{g \in \mathcal{F} \mid P(l(g(\mathbf{x}_i), y_i) \leq \epsilon_{\text{th}}) \geq 1 - \delta, \forall(\mathbf{x}_i, y_i) \in \mathcal{D}\}$.

VI. Method

In this section, we derive an error bound for the KAN model in three steps. First, we introduce the error bound for Newton's piecewise polynomial approximation. Second, we extend this result to arbitrary splines where the error at knots may be non-zero. Finally, we extend the resulting error bound to KANs through the addition and composition of spline functions. We then present the error bound for multilayer networks and a method for handling residual connections, which is used in structures such as KAN. Additionally, we explored various approaches to dividing Lipschitz constants across layers and effectively managing error propagation.

A. Piecewise Polynomial Error (PPE)

We begin by establishing an error bound for Newton's piecewise polynomial approximation.

Theorem 1 (Newton's polynomial error bound): Let $f \in \mathcal{C}^{k+1}$ which is defined over $[a, b]$ and has $k + 1$ continuous derivatives and is $k+1$th-order Lipschitz continuous with constant $\mathcal{L}_f^{k+1}$. The approximation error for kth-order Newton's piecewise polynomial [44] fit, $\mathcal{P}_{k,j}[\mathcal{D}_f(\boldsymbol{\tau}_{1:m})]$, that passes through the knots $\mathcal{D}_f(\boldsymbol{\tau}_{1:m})$, at the test point $x \in [\tau_j, \tau_{j+1})$ for all $j \in \{1, \ldots, m-1\}$ is bounded as follows:

$$|f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f(\boldsymbol{\tau}_{1:m})](x)| \leq \underbrace{\frac{\mathcal{L}_f^{k+1}}{(k+1)!}\left|\prod_{i=1}^{k+1}(x - \tau_i^{(j)})\right|}_{=:\bar{u}_f^{k+1}(x;\boldsymbol{\tau}_{1:m})}, \tag{14}$$

where $\tau_i^{(j)}$ is a knot in the set $\boldsymbol{\tau}_{1:k+1}^{j(x)}$ of $k + 1$ nearby knots to x in $\boldsymbol{\tau}_{1:m}$ as defined in Definition 1. The above bound is tight when the $k + 1$th derivative is constant at the maximum amount, $f^{(k+1)}(x) = \mathcal{L}_f^{k+1}$.

Proof: A function f can be written as a kth-order polynomial approximation along with the remainder term [77]:

$$\begin{aligned} f(x) = {} & \mathcal{P}_{k,j}[\mathcal{D}_f(\boldsymbol{\tau}_{1:k+1}^{j(x)})](x) \\ & + (x - \tau_1^{(j)}) \ldots (x - \tau_{k+1}^{(j)})[\tau_1^{(j)}, \ldots, \tau_{k+1}^{(j)}, x]f. \end{aligned} \tag{15}$$

Rearranging the above expression and using the mean value theorem for divided differences (Theorem 5), we obtain

$$\begin{aligned} |f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f(\boldsymbol{\tau}_{1:k+1}^{j(x)})](x)| &= \frac{f^{(k+1)}(\zeta)}{(k+1)!}\prod_{i=1}^{k+1}(x - \tau_i^{(j)}) \\ &\leq \frac{\mathcal{L}_f^{k+1}}{(k+1)!}\left|\prod_{i=1}^{k+1}(x - \tau_i^{(j)})\right|, \quad \exists\zeta \in [a, b]. \end{aligned} \tag{16}$$

■

Remark 1: Newton's polynomial bound $\bar{u}_f^{k+1}(x; \boldsymbol{\tau}_{1:m})$, defined in (14), provides the worst-case bound as defined in (13), when the true function class $\mathcal{F}$ consist of all $k + 1$-order Lipschitz continuous function, the hypothesis class $\mathcal{H}$ is the class of splines and $\epsilon_{\text{max}} = 0$. Additionally, the bound is distance-aware with a distance that is proportional to the geometric mean of the distances to the (k+1)-nearby knots, $d(x, \mathcal{X}_\mathcal{D}) \coloneqq \prod_{i=1}^{k+1}\left|x - \tau_i^{(j)}\right|$. In Appendix A-V, we provide the proof that Newton's polynomial is distance-aware.

Remark 2: This higher-order error bound can be loose when the test point is far away from the knots. To address this, we consider the minimum of the trivial bound (11) and Newton's polynomial error bound as interpolation error, $\bar{u}_f(x;\boldsymbol{\tau}_{1:m}) = \min\{\bar{u}_f^{\ell}(x;\boldsymbol{\tau}_{1:m}), \bar{u}_f^{k+1}(x;\boldsymbol{\tau}_{1:m})\}$. Also, if the local Lipschitz constant is known rather than a global Lipschitz constant, it can provide a tighter bound.

As discussed in Section IV, we have selected to use the error bound for Newton's polynomial as the basis of our analysis, because it offers a tight and distance-aware bound for piecewise polynomials, the building block of SNNs. The interpolation error in Theorem 1 assumes zero error at the knots, which is not typically possible in many conditions, such as when data is noisy, when there are fewer knots than data points, or when we require the piecewise polynomial to be smooth. In the next Lemma, we provide an error bound for any spline approximation that has non-zero error at the knots.

Lemma 1 (Piecewise polynomial interpolation at knots): With the same assumptions as Theorem 1, consider a piecewise polynomial approximation $\hat{f}(x)$ that does not necessarily pass through all the knots $\mathcal{D}_f(\boldsymbol{\tau}_{1:m})$. Let $\hat{f}_{[j]}(x)$ be the jth polynomial piece of $\hat{f}(x)$ (Definition 3). Because $\hat{f}_{[j]}(x)$ is a polynomial, we can extend its domain to the domain of $f(x)$, $[a,b]$. Also, define the corresponding error function as $e_j^f(x) := f(x) - \hat{f}_{[j]}(x)$. Note that $e_j^f(x) : [a,b] \to \mathbb{R}$ is defined on the entire domain of $f(x)$, not just $x \in [\tau_j, \tau_{j+1})$ and has known values at the knots, $\mathcal{D}_{e_j^f}(\boldsymbol{\tau}_{1:m})$. Then the error for the interval $x \in [\tau_j, \tau_{j+1})$ for all $j \in \{1,\dots,m-1\}$ is bounded by,

$$\begin{aligned}|f(x) - \hat{f}(x)| \leq& \bar{u}_f(x;\boldsymbol{\tau}_{1:m}) + |\mathcal{P}_{k,j}[\mathcal{D}_{e_j^f}](x)| \\ =:& \, u_f(x;\boldsymbol{\tau}_{1:m}) \quad \text{(EBS)},\end{aligned} \tag{17}$$

where $\bar{u}_f(x)$ is the error bound from Theorem 1. We refer to (17) as the *error bound with spline fit* (EBS). The above bound is tight when the inequality in Theorem 1 is tight and $\text{sign}(f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f](x)) = \text{sign}(\mathcal{P}_{k,j}[\mathcal{D}_{e_j^f}](x))$ for $x \in [\tau_1, \tau_m)$. This Lemma holds for any kth-order piecewise polynomial, regardless of the learning algorithm for $\hat{f}$.

Fig. 2 (right) depicts the actual function f in black, the non-zero spline approximation $\hat{f}$ in red, the jth piece of spline approximation $\hat{f}_{[j]}$ in green, and the adjusted jth piece of kth-order Newton's polynomial in pink. The proof of Lemma 1 is provided in Appendix A-IV and the proof of distance awareness of Theorem 1 and Lemma 1 are provided in Appendices A-VI and A-VII, respectively. Although the bound in Lemma 1 is tight, in practice the knot errors $\mathcal{D}_{e_j^f}$ may contain measurement noise and modeling error.

When knots are densely spaced, fitting a high-order polynomial $\mathcal{P}_{k,j}[\mathcal{D}_{e_j^f}](x)$ can introduce oscillations between knots, similar to Runge's phenomenon [78]. These oscillations may lead to numerical instability, are distance-unaware, and are undesirable because the error at the knots is expected to primarily capture measurement noise and modeling error, which should not contribute more to the interpolation error than the error at nearby knots itself. To avoid these oscillations, we fit a piecewise linear function $|\mathcal{P}_{1,j}[\mathcal{D}_{e_j^f}](x)| = [\tau_j, \tau_{j+1}]|e_j^f|(x-\tau_j) + |e_j^f|$, where the slope is the divided difference $[\tau_j, \tau_{j+1}]|e_j^f| = (|e_{j+1}^f| - |e_j^f|)/(\tau_{j+1} - \tau_j)$ and refer to this approach as *error bound with linear fit* (EBL).

$$u_f(x;\boldsymbol{\tau}_{1:m}) := \bar{u}_f(x;\boldsymbol{\tau}_{1:m}) + |\mathcal{P}_{1,j}[\mathcal{D}_{e_j^f}](x)| \quad \text{(EBL)}. \tag{18}$$

This choice smooths the contribution of the error at knots, reduces sensitivity to noise, and lowers computational complexity. If the worst-case measurement error is known, that can be added to the error at knots. For simplicity, we use the same notation u_f for both EBS (17) and EBL (18), considering EBS as the default error bound and explicitly mentioning EBL when it is used.

B. Multilayer Spline Error Analysis

Next, we discuss how the error propagates when composing spline functions. To extend layer-wise error bound to a two-layer case, consider $f(x) : [a,b] \to \mathbb{R}$ approximated by a two-layer architecture $\hat{f}(x) = \hat{h}(\sum_{i=1}^n \hat{g}_i(x)) = \hat{h}(\mathbf{1}_n^\top \hat{\mathbf{g}}(x))$, where $\mathbf{1}_n$ is a vector of n ones. Any true 1-D function f can always be decomposed into the same network structure as the approximation, $f(x) = h(\mathbf{1}_n^\top \mathbf{g}(x))$, with infinitely many valid decompositions. For example, when $n = 2$, one such decomposition is $g_1(x) = f(x)$, $g_2(x) = 0$, and $h(x) = x$. Moreover, given any decomposition $f(x) = h(g_1(x) + g_2(x))$ and an invertible function $v(x)$, we can get a new decomposition $f(x) = h'(g_1'(x) + g_2(x))$ where $h'(y) := h(v^{-1}(y))$ and $g_1'(x) := v(g_1(x) + g_2(x)) - g_2(x)$. The tightest worst-case error can be achieved by picking a true decomposition that minimizes layer-wise error among all possible decompositions. We revisit the discussion of picking a true decomposition in subsection VI-E. The next theorem provides error bounds given knowledge of the true decomposition values at the knots.

Theorem 2 (Two-layer KAN error bound): Consider a true two-layer function $f(x) : [a,b] \to \mathbb{R}$, approximated by a composition of kth-order piecewise polynomials $\hat{f}(x) = \hat{h}(\mathbf{1}_n^\top \hat{\mathbf{g}}(x))$. Let the true decomposition at the knots be given as $f(\tau_i) = h(\mathbf{1}_n^\top \mathbf{g}(\tau_i))$ for all $\tau_i \in \boldsymbol{\tau}_{1:m}$. Define the errors at knots as $e^h(.) := h(.) - \hat{h}(.)$ and $e^{\mathbf{g}}(.) := \mathbf{g}(.) - \hat{\mathbf{g}}(.)$, and assume h is first-order Lipschitz continuous with constant $\mathcal{L}_h^1$. Let $f, h, \mathbf{g} \in \mathcal{C}^{k+1}$, with known Lipschitz constants, and let $\hat{h}$, $\hat{\mathbf{g}}$ be piecewise polynomials. Then the error bound for the two-layer approximation is given by:

$$|f(x) - \hat{f}(x)| \leq u_h(\mathbf{1}_n^\top \hat{\mathbf{g}}(x); \boldsymbol{\xi}_{\mathbf{g}_m}) + \mathcal{L}_h^1 \mathbf{1}_n^\top \boldsymbol{u}_{\mathbf{g}}(x;\boldsymbol{\tau}_{1:m}), \tag{19}$$

where $\boldsymbol{\xi}_{\mathbf{g}_m} := \{\mathbf{1}_n^\top \hat{\mathbf{g}}(\tau_i)\}_{i=1}^m$ is input for $\hat{h}$ layer at knots, and $\boldsymbol{u}_{\mathbf{g}}(x) := (u_{g_i})_{i=1}^n$ is the vector of error bound in $\hat{g}_i$ spline approximations as estimated in Lemma 1. The above bound is tight when $h^{(1)}(t) = \text{sign}(h(\mathbf{1}_n^\top \hat{\mathbf{g}}(x)) - \hat{h}(\mathbf{1}_n^\top \hat{\mathbf{g}}(x)))\mathcal{L}_h^1$, for all $t \in [\mathbf{1}_n^\top \hat{\mathbf{g}}(x), \mathbf{1}_n^\top \mathbf{g}(x)]$.

Proof: For ease of exposition, we use a hidden layer of size two $\hat{\mathbf{g}}(x) = [g_1(x), g_2(x)]^\top$. Consider the left-hand side,

$$\begin{aligned}|f(x) - \hat{f}(x)| &= |h(g_1(x) + g_2(x)) - \hat{h}(\hat{g}_1(x) + \hat{g}_2(x))| \\ &= |h(y) - \hat{h}(\hat{y})|,\end{aligned} \tag{20}$$

where $\hat{y} \triangleq \hat{g}_1(x) + \hat{g}_2(x)$ and $y \triangleq g_1(x) + g_2(x)$. To use the bound on the approximation of h, we need to consider the

change in h due to y. We use the Taylor series with integral remainder on function $f(y)$ [79]:

$$\begin{aligned}|h(y) - \hat{h}(\hat{y})| &= |h(\hat{y}) + \int_{\hat{y}}^{y} h^{(1)}(t)dt - \hat{h}(\hat{y})| \\ &\leq |h(\hat{y}) - \hat{h}(\hat{y})| + |\int_{\hat{y}}^{y} h^{(1)}(t)dt| \\ &\leq |h(\hat{y}) - \hat{h}(\hat{y})| + \mathcal{L}_h^1 |y - \hat{y}|. \end{aligned} \tag{21}$$

Substituting Theorem 1 into Equation (21) results in Equation (19). The first inequality is tight when the sign of integral term is the same as $h(\hat{y}) - \hat{h}(\hat{y})$ and second inequality is tight when $h^{(1)}(t) = \mathcal{L}_h^1$ for the integral range, $t \in [\hat{y}, y]$. ∎

Error of Multi-Input Function: Consider a function $f(\mathbf{x})$ approximated by $\hat{f}(\mathbf{x}) = \hat{h}(\sum_{i=1}^{m_i} \hat{g}_i(x_i))$. Let the set of input knots be represented by a matrix $\mathcal{T} \in \mathbb{R}^{m_i \times m}$, where m_i is the number of inner spline functions g_i and m is the number of knots per spline. Define $y_1 = \sum_{i=1}^{m_i} g_i(x_i)$ and $\boldsymbol{\xi}_{1:m} = \{\sum_{i=1}^{m_i} g_i(\mathcal{T}_{i,j})\}_{j=1}^{m}$, where $\mathcal{T}_{i,j}$ is jth knot of ith spline. Similar to Theorem 2, the error is bounded by:

$$u_f(\mathbf{x}) = u_h(y_1; \boldsymbol{\xi}_{1:m}) + \sum_{i=1}^{m_i} \mathcal{L}_{\partial h,i}^1 u_{g_i}(\mathbf{x}_i; \mathcal{T}_i), \tag{22}$$

where $\mathcal{L}_{\partial h,i}^1$ is first-order component-wise Lipschitz constant of $h(.)$ with respect to its ith input. Formally for a multi-input output function $\mathbf{h} : \mathbb{R}^{m_i} \to \mathbb{R}^{m_o}$, (similar to [75], [80]),

$$\begin{aligned}\mathcal{L}_{\partial \mathbf{h}_o,i}^{(k)} \coloneqq \sup_{\mathbf{x} \in \mathbb{R}^{m_i}, \lambda \in \mathbb{R} \setminus \{0\}} \left| \frac{\mathbf{h}_o^{(k-1)}(\mathbf{x} + \lambda \mathbf{e}_i) - \mathbf{h}_o^{(k-1)}(\mathbf{x})}{\lambda} \right|, \\ \forall i \in \{1, \ldots, m_i\}, \quad o \in \{1, \ldots, m_o\}, \end{aligned} \tag{23}$$

where $\mathbf{e}_i = (0, 0, \ldots, 1, \ldots, 0)$ is the standard basis vector that is all zeros except at the ith coordinate. This definition is different from that in Assumption 1, as the change in input is allowed only in one dimension. However, the two definitions are equivalent with different constants [75], [80]. Henceforth, we use $\mathcal{L}_{\mathbf{h}}^{(k)}$ for both when clear from context.

Theorem 3 (Multi-layer KAN error): For an L layers network (Equation (6)), let the output of layer l at knots be $\Xi_l = \mathbf{h}_l(\Xi_{l-1})$ and the output evaluated at the input be $\hat{\mathbf{y}}_l = \mathbf{h}_l(\hat{\mathbf{y}}_{l-1})$, where $\Xi_0 = \mathcal{T}$ and $\hat{\mathbf{y}}_0 = \mathbf{x}$ (the input). Then the error of layer l is $\mathbf{u}_l(\hat{\mathbf{y}}_{l-1}; \Xi_{l-1}) + \mathcal{L}_{h_l}^1 \mathbf{u}_{l-1}(\hat{\mathbf{y}}_{l-2}; \Xi_{l-2})$. Note that $\mathbf{u}_0 = \mathbf{0}_{m_0}$ is a zero vector of size m_0. Then the overall error, with all the same assumptions at Theorem 2 for all the L layers, will be:

$$\begin{aligned}|\mathbf{f}(\mathbf{x}) - \text{KAN}_L(\mathbf{x})| &\leq \mathbf{u}_L(\hat{\mathbf{y}}_{L-1}; \Xi_{L-1}) \\ &+ \sum_{l=1}^{L-1} (\mathbf{u}_l(\hat{\mathbf{y}}_{l-1}; \Xi_{l-1}) \prod_{j=l+1}^{L} \mathcal{L}_{h_j}^1). \end{aligned} \tag{24}$$

The above theorem shows that the error has two components: the error introduced by each layer and the error propagated from previous layers. Depending on the Lipschitz constant of the outer layers, any error in the inner layers can quickly scale up in the outer layers.

DAREK Algorithm: We summarize DAREK in Algorithm 1, which computes error bounds for an L-layer spline network with m_l hidden units in layer l. The intermediate layer values are denoted as $\hat{\mathbf{y}}_0(\mathbf{x}) \coloneqq \mathbf{x}$, $\hat{\mathbf{y}}_l(\mathbf{x}) \coloneqq \hat{\mathbf{h}}_l(\ldots \hat{\mathbf{h}}_2(\hat{\mathbf{h}}_1(\mathbf{x})))$, and $\hat{\mathbf{y}}_L(\mathbf{x}) \coloneqq \hat{\mathbf{f}}(\mathbf{x})$. Here, subscript n denotes an element of a vector, and j denotes the jth piece of a piecewise polynomial. To understand the algorithm, consider a two-layer model $\hat{f}(x) = \hat{h}_{2,1}(\hat{h}_{1,1}(x) + \hat{h}_{1,2}(x))$. After dividing the prediction error e^f and Lipschitz constant budgets, $\mathcal{L}_f^{(1)}$ and $\mathcal{L}_f^{(k+1)}$, among neurons ($\hat{h}_{2,1}$, $\hat{h}_{1,1}$, and $\hat{h}_{1,2}$), the algorithm calculates each neuron's error by substituting its knots into Equations (14) and (17). It then computes the error of each layer ($u_{h_2} = u_{h_{2,1}}$ and $u_{h_1} = u_{h_{1,1}} + u_{h_{1,2}}$), and the total error using Equation (19), $u_f = u_{h_2} + \mathcal{L}_{h_2}^{(1)} u_{h_1}$.

Algorithm 1: DAREK

Data: Input-output pairs $\mathbf{f}(\mathcal{T})$, trained model $\hat{\mathbf{f}}(\mathbf{x}) = \hat{\mathbf{h}}_L(\ldots \hat{\mathbf{h}}_2(\hat{\mathbf{h}}_1(\mathbf{x})))$, test point $\mathbf{x}^*$, order k, Lipschitz constants $\mathcal{L}_{\mathbf{f}}^{(k+1)}$, $\mathcal{L}_{\mathbf{f}}^{(1)}$.

1 Precompute errors $e^{\mathbf{f}} = |\mathbf{f}(\mathcal{T}) - \hat{\mathbf{f}}(\mathcal{T})|$.
2 Divide $e^{\mathbf{f}}$ among layers, $e^{\mathbf{h}_1}, \ldots, e^{\mathbf{h}_L}$ at knots.
3 Divide $\mathcal{L}_{\mathbf{f}}^1, \mathcal{L}_{\mathbf{f}}^{k+1}$ among layers.
4 **for** $l, n, p \in (\{0, \ldots, L-1\} \times \{1, \ldots, m_l\} \times \{1, \ldots, m_{l+1}\})$ **do**
  `/* O(log2(m)) binary search. */`
5 Find j such that $\hat{\mathbf{y}}_{l,n}(\mathcal{T}_{1:m_0,j}) \leq \hat{\mathbf{y}}_{l,n}(\mathbf{x}^*) < \hat{\mathbf{y}}_{l,n}(\mathcal{T}_{1:m_0,j+1})$.
6 Fit Newton's Polynomials $\mathcal{P}_{k,j}[\mathcal{D}(e^{h_{l,n}})]$ on the knots $\{(\hat{\mathbf{y}}_{1:k+1,l,n}^{(j)}(\mathcal{T}_{1:m_0}), e^{h_{l+1,p}}(\hat{\mathbf{y}}_{1:k+1,l,n}^{(j)}(\mathcal{T}_{1:m_0})))\}_{i=1}^{k+1}$
7 Find $\mathbf{u}_{l+1,n}(\hat{\mathbf{y}}_{l,n}(\mathbf{x}^*); \hat{\mathbf{y}}_{l,n}(\mathcal{T}))$ [Lemma 1].
8 Compute error bound over f [Thm. 2]

The algorithm requires Lipschitz division and error at knots division in the second and third lines of Algorithm 1, which is discussed in detail in Subsections VI-D and VI-E.

C. Residual Connections in KAN Layer

In KAN [8], the authors use the residual connection in the form of $\hat{f}(x) =: r(x) + \phi_k(x)$ to improve the optimization stability and convergence of the model. The $r(x)$ in KAN implementation is a sigmoid linear unit (SiLU) function defined as $\text{SiLU}(x) = x/(1 + \exp(-x))$. The following theorem shows how to compute the neuron's error when it is added to an arbitrary function, assuming the residuals are errorless.

Theorem 4 (Error bound of spline + any function): The error bound of the general structure for function defined on the interval $[\tau_j, \tau_{j+1})$ as $\hat{f}_{[j]}(x) = \hat{r}(x) + \phi_k(x)$ where ϕ_k is a polynomial of order k, and $\hat{r} \in \mathcal{F}$ has the same assumptions as $f(x) \in \mathcal{F}$, then is given by:

$$|f - \hat{f}_{[j]}| \leq \bar{u}_{f-\hat{r}}(x; \boldsymbol{\tau}_{1:m}) + |\mathcal{P}_{k,j}[\mathcal{D}_{e_j^{f-\hat{r}}}]| = u_{f-\hat{r}}. \tag{25}$$

Intuitively, if we assume the true function is $f - \hat{r}$, then all error arises from the ϕ_k. Proof is provided in Appendix A-V.

D. Lipschitz Division among Neurons

While the Lipschitz constant of the overall true function f might be known from the problem definition $\mathcal{F}$, determining the Lipschitz constants for the intermediate layers remains an open problem. In this section, we explore empirical methods for dividing the Lipschitz constant of the true function

among the neurons in each layer $\mathbf{h}_l$, possibly informed by the Lipschitz constant of the approximated neurons in each layer $\mathbf{h}_l$. As discussed in Section VI-B, our goal is to find the closest true function decomposition that respects the known constraints of the true function f.

The Lipschitz constant of each layer of the approximate function can be estimated directly from the data, representing the sharpest change in the layer's output relative to its input. Alternatively, more efficient methods, such as those proposed by Shi et al., can be used [81]. Other works aim to bound the function approximation during training [12], [40]; however, we aim to assign a Lipschitz constant to each neuron of the trained network. Before introducing our proposed approaches, we first review two properties of Lipschitz constants.

Lemma 2 (Lipschitz constant of a sum): Let $g(x) = \sum_{i=1}^{d} g_i(x)$, where each g_i has Lipschitz constant $\mathcal{L}_{g_i}$. Recalling from Equation (10), $|g(x) - g(y)| \leq \mathcal{L}_g|x-y|$, the Lipschitz constant of g would be:

$$|g(x)-g(y)| = \left|\sum_{i=1}^{d}(g_i(x)-g_i(y))\right| \leq \sum_{i=1}^{d}|g_i(x)-g_i(y)| \leq \sum_{i=1}^{d}\mathcal{L}_{g_i}|x-y| = (\sum_{i=1}^{d}\mathcal{L}_{g_i})|x-y|. \quad (26)$$

Lemma 3 (Lipschitz constant of composite function): Let the composite function $f(x) = h(g(x))$, where g and h have Lipschitz constants $\mathcal{L}_g$ and $\mathcal{L}_h$, respectively. Then:

$$|f(\mathbf{x})-f(\mathbf{y})| = |h(g(\mathbf{x}))-h(g(\mathbf{y}))| \leq \mathcal{L}_h|g(\mathbf{x})-g(\mathbf{y})| \leq \mathcal{L}_h\mathcal{L}_g|\mathbf{x}-\mathbf{y}|. \quad (27)$$

Based on Definition 2 and Equation (3), for a KAN_L model (Equation (6)) with L layers and m_l nodes per layer contributing to each output dimension, the total Lipschitz constant is bounded by: $\mathcal{L}_f = \prod_{l=1}^{L}(m_l\mathcal{L}_l)$, where $\mathcal{L}_l$ is the Lipschitz constant of each node in layer l. We define $\hat{\mathcal{L}}_l$ as the empirical Lipschitz constant of each neuron in layer l, computed numerically using Equation (10). Our goal is to determine, layer-wise Lipschitz constants $\mathcal{L}_L$ for true function f so that they are proportional to the layer-wise Lipschitz constants $\hat{\mathcal{L}}_l$ for the approximated function $\hat{f}$, while satisfying the total Lipschitz budget constraint of $\mathcal{L}_f = \prod_{l=1}^{L}(m_l\mathcal{L}_l)$. Furthermore, we aim to balance the computational complexity of the approaches against their impact on uncertainty estimation. We proposed five different methods varying in computational cost and level of detail in modeling the problem: optimization-based division, logarithmic division, linear division, equal division, and worst-case division. In the following, we explain each one and then compare them in toy examples on a trained model.

1) Optimization Based Division: We start with the most detailed formulation, consequently the most computationally costly, which seeks the tightest possible division by explicitly accounting for higher-order Lipschitz constants. Consider a two-layer model $f = \text{KAN}_2(\mathbf{x}) = \sum_{i=1}^{m_2}\phi_{2,1,i}\big(\sum_{j=1}^{m_1}\phi_{1,i,j}(x_j)\big)$. Our goal is to determine the Lipschitz constants of each layer given the Lipschitz constant of f. Assuming $\mathcal{L}^{(k)}_{\phi_{2,1,i}} = \mathcal{L}^{(k)}_{\phi_2}$ and $\mathcal{L}^{(k)}_{\phi_{1,i,j}} = \mathcal{L}^{(k)}_{\phi_1}$ for all i and j, the first-order Lipschitz constant can be written as

$$\mathcal{L}_f^{(1)} = m_2 m_1 \mathcal{L}^{(1)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}, \quad (28)$$

and, for cubic splines, the 4th-order Lipschitz constant as

$$\mathcal{L}_f^{(4)} = m_2 m_1^4 \mathcal{L}^{(4)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}(\mathcal{L}^{(2)}_{\phi_1})^3 + 3m_2m_1^3\mathcal{L}^{(3)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}\mathcal{L}^{(2)}_{\phi_1}\mathcal{L}^{(3)}_{\phi_1} + m_2m_1^2\mathcal{L}^{(2)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}\mathcal{L}^{(4)}_{\phi_1}, \quad (29)$$

which reflects how the Lipschitz bounds are divided across layers. We know $\mathcal{L}_f^{(1)}$ and $\mathcal{L}_f^{(4)}$, then we need to satisfy constraints (28) and (29) while maximizing the approximation error to account for the worst-case Lipschitz distribution among the layers. We ignore the contribution from error at knots because Lipschitz constants only affect interpolation error. The error is formulated as $u_f(\mathbf{x}^*) = \sum_{i=1}^{m_2}(\bar{u}_{\phi_{2,1,i}}(\mathbf{y}_1^*, \Xi_1) + \mathcal{L}^{(1)}_{\phi_2}\sum_{j=1}^{m_1}\bar{u}_{\phi_{1,i,j}}(\mathbf{x}^*, \mathcal{T}))$, where $\mathbf{x}^*$ is the test point that function approximation makes the most error. For ease of notation, consider Equation (14) as $\bar{u}_f(\mathbf{x}, \boldsymbol{\tau}) = \mathcal{L}_f^{(k+1)}\psi(\mathbf{x}, \boldsymbol{\tau})$, where $\psi(\mathbf{x}, \boldsymbol{\tau}) = \prod_{i=1}^{k+1}|x-\tau_i^{(j)}|/(k+1)!$. Furthermore, since worst-case test point is unknown, we consider the expectation of error as, $\bar{\psi}(\boldsymbol{\tau}) = \mathbb{E}_{\mathbf{x}\in\mathcal{X}}[\psi(\mathbf{x}, \boldsymbol{\tau})] \approx \sum_{\mathbf{x}\in\mathcal{X}_\mathcal{D}}\psi(\mathbf{x}, \boldsymbol{\tau})$. We then formulate the following optimization problem:

$$\begin{aligned}
\max_{\mathcal{L}} \quad & \mathcal{L}^{(k+1)}_{\phi_2}\sum_{i=1}^{m_2}(\bar{\psi}(\Xi_1)) + \mathcal{L}^{(1)}_{\phi_2}\mathcal{L}^{(k+1)}_{\phi_1}\sum_{j=1}^{m_1}\bar{\psi}(\boldsymbol{\tau})) \\
s.t. \quad & \mathcal{L}_f^{(4)} = m_2m_1^4\mathcal{L}^{(4)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}(\mathcal{L}^{(2)}_{\phi_1})^3 \\
& \quad + 3m_2m_1^3\mathcal{L}^{(3)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}\mathcal{L}^{(2)}_{\phi_1}\mathcal{L}^{(3)}_{\phi_1} + m_2m_1^2\mathcal{L}^{(2)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1}\mathcal{L}^{(4)}_{\phi_1} \\
& \mathcal{L}_f^{(1)} = m_2m_1\mathcal{L}^{(1)}_{\phi_2}\mathcal{L}^{(1)}_{\phi_1} \\
& 0 \leq \mathcal{L}^{(k)}_{\phi_2}, \mathcal{L}^{(k)}_{\phi_1} \leq \mathcal{L}_f^{(k)} \quad \forall k \in \{1, 4\}.
\end{aligned} \quad (30)$$

This is a non-convex optimization problem that can be solved using the Lagrangian method [82], but it has a high computational cost.

2) Logarithmic Division: To avoid solving the computationally costly non-convex optimization, we propose a simpler approach. Since the Lipschitz constant accumulates by product over the layers, we can linearize it in the log domain, $\log(\mathcal{L}_f) = \sum_{l=1}^{L}\log(m_l\mathcal{L}_l)$. With this, we can distribute the total Lipschitz proportional to the sampled Lipschitz constant of the approximated function as:

$$\mathcal{L}_l = \frac{1}{m_l}\exp\left(\frac{\log(m_l\hat{\mathcal{L}}_l)}{\sum_{i=1}^{L}\log(m_i\hat{\mathcal{L}}_i)}\log(\mathcal{L}_f)\right). \quad (31)$$

This approach factorizes the network-level Lipschitz constant into layers in proportion to the learned function's layer-wise Lipschitz constant.

3) Linear Division: An even simpler alternative is to distribute the Lipschitz budget linearly based on empirical estimates: $w_l = \hat{\mathcal{L}}_l/(\sum_{i=1}^{L}m_i\hat{\mathcal{L}}_i)$, where the portion assigned to each neuron in layer l is computed as $\mathcal{L}_l = w_l\mathcal{L}_f$.

4) Equal Division: The simplest approach assumes uniform distribution across all nodes, inspired by [40]. Let assume the Lipschitz constant of each node is $\mathcal{L}_h$, then we have, $\mathcal{L}_f = \prod_{l=1}^{L}m_l\mathcal{L}_h = (\mathcal{L}_h)^L\prod_{l=1}^{L}m_l$. Solving for the node's Lipschitz constant, we obtain, $\mathcal{L}_h = \sqrt[L]{\mathcal{L}_f/\prod_{l=1}^{L}m_l}$.

5) Worst-Case Division: Another simplest, yet conservative, approach is to assign the full Lipschitz budget to each component $\mathcal{L}_l = \mathcal{L}_f$. This guarantees a valid bound, but it is overly conservative and produces loose uncertainty estimates.

Case Studies: To evaluate the effectiveness of the Lipschitz division methods, we tested them on a set of benchmark functions, including $f_1(x) = \cos(x)$, $f_2(x, y) = \exp(\sin(\pi x) + y^2)$, a discontinuous function,

$$h(t) = \begin{cases} \frac{5}{4}\sum_{k=1}^{4}\sin(kt), & \text{for } t < 0 \\ 5\cos(5t), & \text{for } t \geq 0 \end{cases}$$

$$f_3(x, y) = h(x)h(y) \tag{32}$$

and $f_4(x, y)$: the Two Moon dataset [83], defined as:

$$\begin{aligned} x_1 &= r\cos(\theta) + \epsilon_x, & y_1 &= r\sin(\theta) + \epsilon_y, \\ x_2 &= r - r\cos(\theta) + \epsilon_x, & y_2 &= -r\sin(\theta) + \epsilon_y, \\ f_4(x, y) &= \begin{cases} 1, & \text{if } (x, y) = (x_1, y_1) \\ -1, & \text{if } (x, y) = (x_2, y_2) \end{cases}, \end{aligned} \tag{33}$$

where θ is sampled from $[0, \pi]$, r is the radius, and noise terms $\epsilon_x, \epsilon_y \sim \mathcal{N}(0, \sigma^2)$.

We trained two-layer KANs on collected data on these functions, assumed zero error at knots to isolate the effect of Lipschitz division, used EBL (18) for spline error, and computed the number of violations of the interpolation error bound to assess each method. The results in Table III indicate that the **non-optimized worst-case** method achieves the fewest violations, the lowest computational complexity, and the most relaxed error bound among the compared approaches. The **worst-case** method yields the second-lowest violation rate but requires costly optimization. The **equal division** method ranks third in terms of violation rate, yet it remains a reasonable choice due to its simplicity and minimal computation overhead, making it our **preferred choice**. Fig. 3 (a) visualizes that all the approaches successfully recover the true error bounds on the cosine function, with both the non-optimized worst-case and worst-case error bounds appearing overly conservative. In contrast, the linear and logarithmic heuristic methods provide a tighter error bound. Please check Appendix C-III for more information on network architecture, calculated Lipschitz, and additional details.

TABLE III
COMPARISON OF LIPSCHITZ DIVISION METHODS ON ESTIMATING DIFFERENT FUNCTIONS.

	Violation rate			
Methods	f_1	f_2	f_3	f_4
Equal	0.171	0.132	0.472	0.150
Linear	0.487	0.057	0.733	0.777
Logarithmic	0.171	0.176	0.558	0.206
Worst-case	0.051	0.004	0.074	0.014
Optimization	0.058	0.005	0.082	0.014

E. Division of Error at Knots

Since only the total approximation error $e^f(\tau_i) := f(\tau_i) - \hat{f}(\tau_i)$ is observable and available at knots, rather than the layer-wise errors required for Theorem 2, we divide the total error

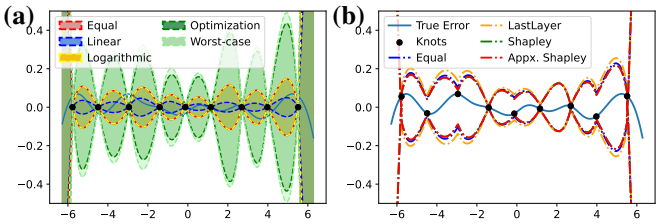

Fig. 3. **a)** The comparison of error bounds of different Lipschitz division methods. All the algorithms can recover the true error. **b)** Error bound using different error division approaches on $\cos$ function.

across layers proportionally to their contributions. For an L-layer network similar to Equation (19), this obtains:

$$e^f(\boldsymbol{\tau}) \leq e^{\mathbf{h}_L}(\hat{\mathbf{y}}_{L-1}) + \sum_{l=1}^{L-1}\left(\hat{e}^{\mathbf{h}_l}(\hat{\mathbf{y}}_{l-1})\prod_{j=l+1}^{L}\mathcal{L}_{h_j}^1\right), \tag{34}$$

where $\hat{e}^{\mathbf{h}_l}(\hat{\mathbf{y}}_{l-1}) = \sum_{j=1}^{m_l} e^{\mathbf{h}_{l,j}}(\hat{\mathbf{y}}_{l-1,j})$ represents the error contribution from layer l, and $\hat{\mathbf{y}}_{l-1,j}$ is the input to the jth node of layer l. To evaluate interpolation error $\bar{u}_h$ (Theorem 1), only the spline's knot locations are needed; however, the error at knots $\mathcal{P}[e_j^h]$ requires the true function values at inner layers, which are unavailable. To address this, we propose three alternative approaches for estimating the inner-layer error: 1- assigning all error to the last layer, 2- equally dividing the error across knots, and 3- distributing the error based on Shapley values. Once the error at the knots is estimated, Theorem 2 can be applied to obtain the error bound for the full approximation. Finally, we compare these approaches using 2D examples.

1) Last Layer Error: Our first approach depends on the following assumption:

Assumption 2 (No early information loss (NEIL)): In approximating a function $f(x)$ using function compositions, for instance, $\hat{f}(x) = \hat{h}(\hat{\mathbf{g}}(x))$, the early layers $\hat{\mathbf{g}}$ do not throw away vital information of f. This means that for any two distinguishable inputs $x_1 \neq x_2$, which map into $f(x_1) \neq f(x_2)$, all the early layer approximations should be distinguishable $\hat{\mathbf{g}}(x_1) \neq \hat{\mathbf{g}}(x_2)$:

$$\text{if } f(x_1) \neq f(x_2) \implies \hat{\mathbf{g}}(x_1) \neq \hat{\mathbf{g}}(x_2) \quad \forall x_1 \neq x_2. \tag{35}$$

This assumption is equivalent to the invertibility of $\hat{\mathbf{g}}$, for all $x_1, x_2 \in \mathcal{X}$ for which $f(x_1) \neq f(x_2)$.

If Assumption 2 is satisfied, that is there is no loss of information in the earlier layers, then all the error can be compensated in the last layer. Moreover this assumption can be enforced by using bi-Lipschitz continuity techniques as used in [40], [43]. So our first method for dividing error at knots assumes zero error in all the layers except the last layer. Then the last layer error at knots is same as the total error at knots.

2) Equally Dividing Error at Knot: We divide this error across neurons in proportion to their Lipschitz constants. The error assigned to neuron $\phi_{l,j,i}$ in layer l with m_l nodes is

$$e^{\phi_{l,j,i}} = \frac{e^f}{C_e}, \quad C_e = \sum_{l=1}^{L-1} m_l \prod_{l'=l+1}^{L}\mathcal{L}_{h_{l'}}. \tag{36}$$

Note that, although e^f is defined on the entire domain, we only divide it at the knots, $x \in \boldsymbol{\tau}_{1:m}$.

3) Shapley Value: While the aforementioned methods are effective, they ignore the interactions between neurons. To address this, we adapt *Shapley value* from cooperative game theory, which fairly distributes the total payoff among players based on each player's contribution to all possible coalitions [84]. The theory incorporates different parts of ML to assess feature, sample, or element importance. This solution is an adaptation of [85] to find the error division of each neuron. Let Φ be the set of all neurons in the network, and let $\Phi_s \subseteq \Phi$. Define the operator $\mathcal{M}(\Phi_s)$ as the network in which neurons in $\Phi \setminus \Phi_s$ have been "zeroed out". Zeroing out a neuron means removing the neuron while keeping the residual connection r. The Shapley value $\varphi_{l,j,i}$ for neuron $\phi_{l,j,i}$ is then computed as:

$$\varphi_{l,j,i} = \sum_{\Phi_s \subseteq \Phi \setminus \{\phi_{l,j,i}\}} \frac{|\Phi_s|!\,(|\Phi| - |\Phi_s| - 1|)!}{|\Phi|!}\,\Delta_{l,j,i}(\Phi_s) \quad (37)$$

$$\Delta_{l,j,i}(\Phi_s) = \mathcal{V}(\mathcal{M}(\Phi_s \cup \{\phi_{l,j,i}\})) - \mathcal{V}(\mathcal{M}(\Phi_s)),$$

which $|.|$ is cardinality operator, $\Delta_i(\Phi_s)$ is marginal contribution. The value function $\mathcal{V}(\mathcal{M})$ measures the error made by the network $\mathcal{M}$, defined as: $\mathcal{V}(\mathcal{M}) = \frac{1}{|\mathcal{X}|}\sum_{x,y\in\mathcal{X}} |y - \mathcal{M}(x)|$. Please check Appendix B for a more detailed example of how the Shapley value works.

Although Shapley approach leads to a fair allocation, computing it exactly is intractable for the large number of permutations when the number of players (neurons) exceeds 30 [85]. To address this, we use an MC sampling-based algorithm that estimates Shapley values by averaging marginal error changes across randomly sampled permutations of neurons.

Case Studies to evaluate error division: We evaluate the proposed error division methods on the same functions defined in Section (VI-D.5). The network structure is identical to that used in the Lipschitz division experiment, with *equal Lipschitz division* applied. Table IV summarizes the number of violations for each method across experiments.

TABLE IV
COMPARISON OF ERROR BOUNDS OF DIFFERENT ERROR DIVISION METHODS.

	Violations' rate			
Methods	f_1	f_2	f_3	f_4
Equal division	0.007	0.013	0.002	0.037
Last layer error	0.005	0.001	0.0	0.002
Shapley value	0.007	0.000	N/A	0.005
Appx. Shapley	0.007	0.001	0.0	0.005

As reported in Table IV, the violation rate of all methods decreased significantly compared to Table III, where error division was assumed to be zero. For the cosine function, all the error division approaches eliminate violations, as illustrated in Fig. 3 (b). The Shapley method for experiment f_3 was infeasible since $|\Phi| = 30$ which results in 2^{30} subsets. Even with an optimized model evaluation that takes 1ms, the full computation would require weeks on a single machine.

VII. CASE STUDIES

In this section, we evaluate the proposed method, DAREK, through several experiments. We compared DAREK's error bounds with probabilistic uncertainty bounds, such as GPs and Ensemble, and evaluated the computational complexity of DAREK with GPs and Ensemble. We also evaluate DAREK for estimating an object's shape from a single laser scan, its distance-awareness in a high-dimensional face-recognition experiment, and its performance in a multi-agent navigation experiment. All experiments use cubic splines ($k = 3$) with first and kth derivative Lipschitz constants set to one, unless otherwise reported.

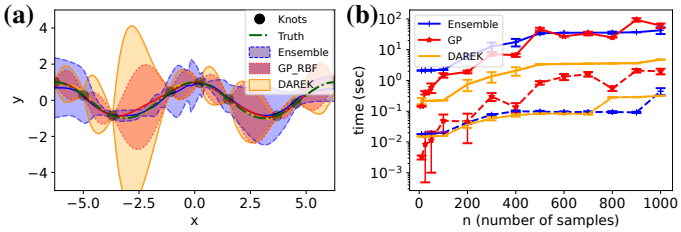

Fig. 4. **(a)** The comparison of error bounds of DAREK on cosine function with Ensemble and GP. Ensemble and GP's uncertainty bounds are shown within the $\pm 3\sigma$ range. **(b)** Computation complexity comparison of DAREK (orange), GP (red), and Ensemble (blue) models. Training and inference times are plotted with solid and dashed lines, respectively.

Comparison with Baselines: To the best of our knowledge, this is the first work to introduce distance-aware worst-case error bounds under higher-order Lipschitz smoothness assumptions. In any case, Fig. 4 presents a comparison between the proposed two-layer DAREK, Ensemble, and GP in terms of both uncertainty estimation and computational complexity. The Ensemble method employs an MC approach with ten KAN models initialized independently. For the GP, we use a radial basis function (RBF) kernel with fixed hyperparameters: unit length scale and unit variance. As shown in image (a), the uncertainty estimated by the Ensemble does not follow a consistent or interpretable pattern, whereas the GP inherently captures the distance awareness through its interpolation mechanism. The comparison with GP is *unfair* without explicitly relating Lipschitz constants to RBF kernel parameters, which is left for future work. Fig. 4 (b) presents the computational performance of the three models for different numbers of training set sizes, demonstrating trends consistent with the theoretical complexity discussed earlier in Section III.

Object Shape Estimation: To evaluate the generalization of DAREK to real-world mapping problem, we consider the object shape estimation from a laser scan in 2D. We train a two-layer KAN with units [2,20,1] and 20 knots to learn the Signed Distance Function (SDF) profile of an object, using laser scan data from [86]. The goal is to estimate the object's shape from scan measurements. The top-left plot of Fig. 5 shows the training and test samples with selected knot locations. The top-middle plot presents the model's predictions at test points, while the top-right plot depicts the corresponding prediction errors. Red lines indicate the positions of knots in the input layer. The bottom row of plots shows the prediction error near the object's boundary as a function of the laser scan angle θ, and the Cartesian coordinates x and y, respectively.

Face detection: To demonstrate the effectiveness of DAREK on high dimension inputs, we evaluate DAREK on the Celebrity Attribute face detection dataset [87]. The task is

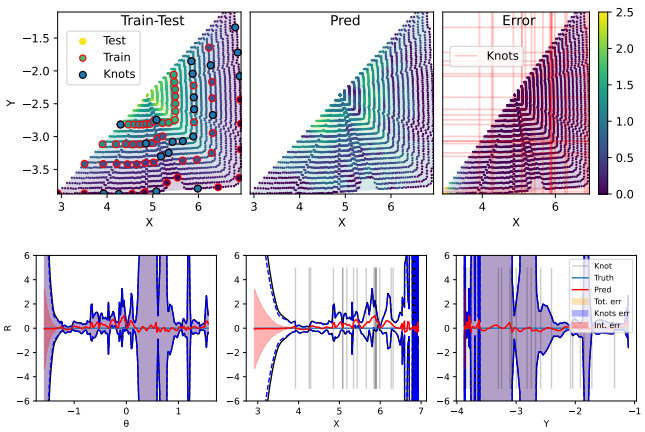

Fig. 5. In the first row, the plots from left to right show the train-test dataset, the trained model's predictions, and the interpolation error of each test point. The second row shows the predicted distance from the object boundary ($\boldsymbol{R}$) versus the laser scanner angle ($\boldsymbol{\theta}$), the $\boldsymbol{x}$ location, and the $\boldsymbol{y}$ location of the boundary point.

to predict the bounding box of the face in the given image. To reduce the dataset size, we randomly selected 8k images for training and 1k images for testing. We reduced feature dimensionality to 2, 20, and 100 using two feature extraction strategies: PCA-based projection and deep Res-Net18 feature projection. These reduced dimensionality inputs were used to train uncertainty-aware methods to predict the location of the bounding box along with corresponding uncertainty. We compared the following methods: one-layer DAREK (DK1), two-layer DAREK with 10 hidden units (DK2), an ensemble of one-layer KANs (ENS1), an ensemble of two-layer KANs with 10 hidden units (ENS2), an exact GP (ExGP) that uses the same knots as the DAREK as its training data, and a variational sparse GP (ApxGP) [60] that uses DAREK knots as inducing points and learns a variational posterior. Networks were trained for 1000 iterations with a learning rate of 0.1 and GPs for 2000 iterations with a learning rate of 0.5. All models used a decay factor of 0.9 every 200 iterations. Performance was evaluated using normalized root mean squared error (RMSE), intersection over union (IOU), and SDA (5). The results are reported in Table V. Additional results on dimensional sizes (5, 10, 50, 200) are shown in Appendix C-IV. RMSE and IOU remain consistent across models and features, with ResNet-18 features yielding lower RMSE and higher IOU. DAREK's SDA (5) increases from approximately 60% at two dimensions to around 95% at 100 dimensions, outperforming ENS1 and ENS2, which do not follow a consistent pattern. GP achieves near-perfect SDA due to its kernel-based formulation. Example predictions are illustrated in Fig. 6.

Safe Multi-Agent Experiment: We set up a multi-agent environment based on Cheng et al. [88] to evaluate DAREK in a safe control application. Each agent follows double-integrator dynamics in 2D space, with state $\mathbf{x} = (p_t, v_t)$ containing positions and velocities, of all agents in x and y directions, and control input $\mathbf{a}$ contains the acceleration in x and y directions of all agents. The system dynamics are $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{a}_t) + \mathbf{d}(\mathbf{x}_t)$, where $\mathbf{d}$ is disturbance. Each agent

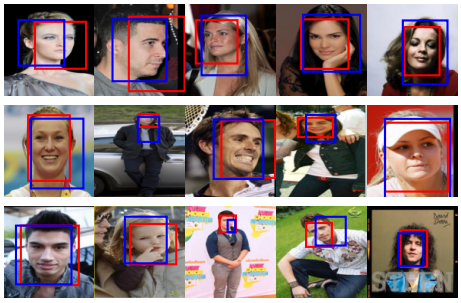

Fig. 6. Face detection on sample test images from CelebA dataset [87] using DAREK. Red boxes indicate ground-truth bounding boxes, and blue boxes indicate DAREK predictions. Note that it uses only 100 projected dimensions out of a 25k-dimensional feature space of ResNet18.

TABLE V
RESULTS FOR HIGH-DIMENSIONAL EXPERIMENTS ACROSS TWO FEATURE EXTRACTION METHODS (PCA AND RESNET-18).

	RMSE ↓			IOU ↑			SDA ↑		
Model	2	20	100	2	20	100	2	20	100
(a) PCA									
DK1	0.136	0.133	**0.129**	**0.451**	0.452	**0.466**	0.676	0.967	0.952
DK2	0.137	0.190	0.184	0.448	0.375	0.339	0.677	0.972	0.950
ENS1	0.136	0.133	0.130	0.450	0.451	0.464	0.571	0.563	0.431
ENS2	**0.135**	**0.128**	0.265	0.449	**0.469**	0.145	0.527	0.530	0.564
ExGP	0.138	0.161	0.137	0.435	0.359	0.448	1.000	1.000	1.000
ApxGP	0.138	0.137	0.137	0.427	0.447	0.447	0.977	1.000	1.000
(b) ResNet-18									
DK1	0.100	0.088	0.074	0.535	0.557	0.606	0.704	0.892	0.837
DK2	0.100	0.103	0.073	0.533	0.470	0.608	0.691	0.918	0.882
ENS1	0.100	0.089	0.075	0.535	0.557	0.603	0.566	0.577	0.468
ENS2	0.100	0.084	0.069	0.530	0.575	0.628	0.515	0.562	0.550
ExGP	0.139	0.140	0.137	0.431	0.427	0.447	0.542	1.000	1.000
ApxGP	0.133	0.137	0.137	0.445	0.447	0.447	0.729	1.000	1.000

has an initial state, a goal state, and a decentralized controller. The blue agent uses a robust controller (Equation (39)), while the red agents use a primal controller without the polytope constraint (38). The control framework first solves a model predictive control (MPC) to generate the optimal trajectory, $\mathbf{x}_{\text{des}}$, and then solves control barrier functions (CBF) to compute the safe control inputs that keep the robot close to the optimal trajectory. Assuming bounded disturbances, a robust CBF can be formulated to jointly optimize safety and optimality. The nonlinear uncertainty bound at a specified confidence level can be approximated by the polytope:

$$\{\mathbf{d} \in \mathbf{R}^n | G\mathbf{d} \leq \mathbf{g}\}, \tag{38}$$

where $\mathbf{d}$ is disturbance within the polytope, G defines its orientation, and $\mathbf{g}$ sets the bounds along those directions.

$$\begin{aligned} \min_{\mathbf{a}\in\mathcal{A},\mathbf{d}\in D} \ & \|\mathbf{f}(\mathbf{x}_t, \mathbf{a}) - \mathbf{x}_{\text{des}}\| \\ \text{s.t.} \ \ & \forall \mathbf{d} \in \{\mathbf{d} \in \mathbb{R}^n | G\mathbf{d} \leq \mathbf{g}\} \\ & H_1(\mathbf{x}_t)\mathbf{d} + \mathbf{a}^\top H_2(\mathbf{x}_t)\mathbf{d} + H_3(\mathbf{x}_t)\mathbf{a} \leq \mathbf{k}_c(\mathbf{x}_t) \\ & \|\mathbf{a}\|_2 \leq \mathbf{a}_{\max} \qquad \text{(actuation limits)}, \end{aligned} \tag{39}$$

where H_1, H_2, and H_3 are projecting the effect of disturbance, control-disturbance coupling, and control input in safety, and k_c defines the safety bounds. These functions are defined in [88]. We used a pretrained DAREK model with architecture

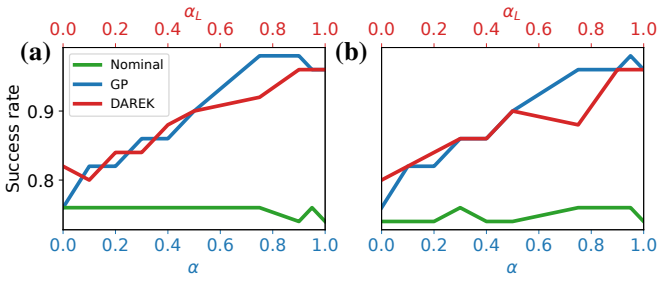

Fig. 7. Success rate in multi-agent simulations under different safety levels. The GP controller uses α to represent the percentage of the uncertainty coverage, while DAREK uses α_L, denoting the percentage of the true Lipschitz constant applied during simulation. **(a)** Uniform noise, **(b)** Gaussian noise.

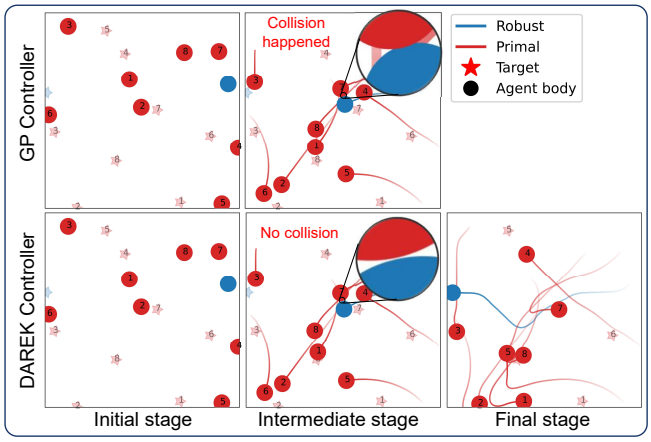

Fig. 8. Illustration of a multi-agent simulation. The plot compares a test scenario where the GP controller failed to complete the trajectory, whereas the DAREK controller successfully reached the goal. It shows the initial positions, the intermediate stage where the GP controller collided and the DAREK controller avoided it, and the final stage where the DAREK controller reached the goal. Agent trajectories are shown as lines, their bodies at the current state as circles, and goals as stars.

$[2, 5, 4]$ and 100 knots for cubic spline, trained offline in a low-noise environment (one percent noise on acceleration), then fine-tuned online. During execution, the model buffers $(\Delta\mathbf{x}, \mathbf{d})$ pairs and retrains every 10 steps for 10 epochs. To ensure distance-awareness, we use the error bound with linear fit (EBL) (18) for error at knots in this experiment. DAREK produces uncorrelated error bounds, which are linearly independent, and the polytope is directly obtained: $-\mathbf{u}_f \leq \mathbf{d} \leq \mathbf{u}_f$.

The system dynamics is defined as follows:

$$\mathbf{x}_{t+1} = \begin{bmatrix} p_t + v_t dt \\ v_t - k_v |v_t|^2 \end{bmatrix} + \begin{bmatrix} 0 \\ k_d(|v_t| + 1)dt \end{bmatrix} a + \underbrace{\begin{bmatrix} 0 \\ k_d(|v_t| + 1)dt \end{bmatrix} \mathbf{d}_a + \begin{bmatrix} \mathbf{d}_p \\ \mathbf{d}_v \end{bmatrix}}_{\mathbf{d}(\mathbf{x})}, \tag{40}$$

where dt is the step time, k_v is the drag coefficient, and k_d is the acceleration scaling factor. The terms $\mathbf{d}_p$, $\mathbf{d}_v$, and $\mathbf{d}_a$ represent disturbances in position, velocity, and acceleration, respectively. Uncertainty model learns the disturbance function $\mathbf{d}(\mathbf{x})$ and Lipschitz constants $L_{\partial p_{+1}/\partial v} = 0$ and $L_{\partial v_{+1}/\partial v} = k_d \mathbf{d}_{a,\max} dt$. These Lipschitz constants cannot capture $\mathbf{d}_p$ and $\mathbf{d}_v$, which have to be captured with error at knots.

For higher order Lipschitz, assuming $\mathcal{L}_f^{(k)}$ is the kth order Lipschitz and bounds kth derivative of f, Assumption 1. Now, the $(k+1)$th Lipschitz using the trivial error bound is:

$$\mathcal{L}_f^{(k+1)} \leq 2\mathcal{L}_f^{(k)} |\Delta\mathbf{x}|. \tag{41}$$

All simulations are conducted with a fixed noise level of 10% and are repeated over 50 randomized environments. A trial is successful if the robust agent reaches its goal without collisions. Fig. 7 depicts the robust controllers' plots, GP and DAREK, which outperform the primal controller and reach up to 95 percent success rate for different confidence levels. Fig. 8 illustrates a failure case for GP due to a collision, while DAREK successfully navigated to the goal. The trivial bound in Equation (41) is used on average 82 percent of the time.

VIII. Conclusion

We introduced DAREK, distance-aware error analysis for SNN that provides theoretically tight and interpretable worst-case error bounds. We evaluate the correctness and quality of these bounds on various function approximation tasks, consistently showing that the proposed bound safely overestimates the true function. We further validate DAREK on an object shape estimation problem. Additionally, we integrate DAREK into a multi-agent safe control framework, demonstrating its utility in a simulated multi-agent environment. Across experiments, DAREK was competitive with GP while being more computationally efficient.

DAREK offers several **advantages**. It combines the strengths of both parametric and non-parametric models, offering flexibility and analytical tractability. The ability to localize errors to the nearest knots makes the model interpretable and facilitates explanation and debugging. Distance awareness is inherently embedded in the error formulation, making the model more conservative as test points move farther from training samples. Moreover, DAREK is computationally efficient, making it suitable for real-time systems. Additionally, DAREK provides worst-case error guarantees, making it highly suitable for safety-critical **applications** such as robotics, autonomous control systems, and medical decision-making, where conservative behavior is essential. It also has potential utility in signal processing, risk management, and other domains that require robust, distance-aware regression.
Despite its strengths, DAREK has several **limitations**. The proposed Lipschitz and error division methods are not guaranteed to match the Lipschitz and errors of the true system. Fixing the spline knots using a subset of the training data can limit generalization if the sample is unrepresentative, and it also requires access to training data. Furthermore, aggregating worst-case bounds, especially across dimensions, can grow quickly and lead to excessive conservatism. Additionally, poor knot placement, either too far or too close, can cause high

interpolation or unwanted oscillations, known as Runge's phenomenon [89]. These limitations are not fundamental to the approach and can be addressed in future research.

REFERENCES

[1] M. Ataei, M. J. Khojasteh, and V. Dhiman, "Darek-distance aware error for kolmogorov networks," in *ICASSP*, 2025, pp. 1–5.

[2] M. Egerstedt and C. Martin, *Control theoretic splines: optimal control, statistics, and path planning*. Princeton University Press, 2009.

[3] M. Unser, "Splines: A perfect fit for signal and image processing," *IEEE Signal processing magazine*, vol. 16, no. 6, pp. 22–38, 2002.

[4] J. Friedman, "Adaptive spline networks," *NeurIPS*, vol. 3, 1990.

[5] A. Uncini and F. Piazza, "Adaptive spline neural networks for signal processing applications," in *INTERNATIONAL SYMPOSIUM on INTELLIGENT SYSTEMS AMSEISIS*, vol. 97, 1997, pp. 11–13.

[6] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, pp. 31–57, 2018.

[7] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt, "Progress measures for grokking via mechanistic interpretability," *arXiv:2301.05217*, 2023.

[8] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov–arnold networks," in *ICLR*, 2025.

[9] T. Tohme, M. J. Khojasteh, M. Sadr, F. Meyer, and K. Youcef-Toumi, "ISR: Invertible symbolic regression," *arXiv:2405.06848*, 2024.

[10] B. Igelnik and N. Parikh, "Kolmogorov's spline network," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 725–733, 2003.

[11] P. Bohra, J. Campos, H. Gupta, S. Aziznejad, and M. Unser, "Learning activation functions in deep (spline) neural networks," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 295–309, 2020.

[12] S. Aziznejad, H. Gupta, J. Campos, and M. Unser, "Deep neural networks with trainable activations and controlled lipschitz constant," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4688–4699, 2020.

[13] N. Diamant, E. Hajiramezanali, T. Biancalani, and G. Scalia, "Conformalized deep splines for optimal and efficient prediction sets," in *International Conference on Artificial Intelligence and Statistics*, 2024, pp. 1657–1665.

[14] C. Keskin and S. Izadi, "Splinenets: Continuous neural decision graphs," *NeurIPS*, vol. 31, 2018.

[15] S. Günther, W. Pazner, and D. Qi, "Spline parameterization of neural network controls for deep learning," *arXiv:2103.00301*, 2021.

[16] D. Fakhoury, E. Fakhoury, and H. Speleers, "Exsplinet: An interpretable and expressive spline-based neural network," *Neural Networks*, vol. 152, pp. 332–346, 2022.

[17] Z. Bozorgasl and H. Chen, "Wav-kan: Wavelet kolmogorov-arnold networks," *arXiv:2405.12832*, 2024.

[18] R. Bresson, G. Nikolentzos, G. Panagopoulos, M. Chatzianastasis, J. Pang, and M. Vazirgiannis, "Kagnns: Kolmogorov-arnold networks meet graph learning," *arXiv:2406.18380*, 2024.

[19] D. W. Abueidda, P. Pantidis, and M. E. Mobasher, "Deepokan: Deep operator network based on kolmogorov arnold networks for mechanics problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 436, p. 117699, 2025.

[20] C. J. Vaca-Rubio, L. Blanco, R. Pereira, and M. Caus, "Kolmogorov-arnold networks (kans) for time series analysis," in *2024 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2024, pp. 1–6.

[21] R. Genet and H. Inzirillo, "Tkan: Temporal kolmogorov-arnold networks," *arXiv:2405.07344*, 2024.

[22] B. C. Koenig, S. Kim, and S. Deng, "Kan-odes: Kolmogorov–arnold network ordinary differential equations for learning dynamical systems and hidden physics," *Computer Methods in Applied Mechanics and Engineering*, vol. 432, p. 117397, 2024.

[23] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," in *Doklady Akademii Nauk*, vol. 114, no. 5, 1957, pp. 953–956.

[24] F. Beltran-Carbajal, H. Yañez-Badillo, D. Galvan-Perez, I. Rivas-Cambero, D. Sotelo, and C. Sotelo, "B-spline artificial neural networks in robust induction motor control," *IEEE Access*, 2024.

[25] K. Qian and M. Kheir, "Investigating kan-based physics-informed neural networks for emi/emc simulations," in *International Conference on Intelligent Systems, Blockchain, and Communication Technologies*, 2024, pp. 40–48.

[26] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *NeurIPS*, vol. 30, 2017.

[27] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *ICML*, vol. 48. PMLR, 20–22 Jun 2016, pp. 1050–1059.

[28] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson, "A simple baseline for bayesian uncertainty in deep learning," *NeurIPS*, vol. 32, 2019.

[29] S. Jantre, N. M. Urban, X. Qian, and B.-J. Yoon, "Learning active subspaces for effective and scalable uncertainty quantification in deep neural networks," in *ICASSP*, 2024, pp. 5330–5334.

[30] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. The MIT Press, 2006, vol. 1, no. 1.

[31] A. Damianou and N. D. Lawrence, "Deep gaussian processes," in *Artificial intelligence and statistics*, 2013, pp. 207–215.

[32] Y. Song, Y. Liu, and P. M. Djurić, "Novel architecture of deep feature-based gaussian processes with an ensemble of kernels," in *ICASSP*, 2024, pp. 6750–6754.

[33] A. Chakraborty and A. Chakraborty, "Scalable model-based gaussian process clustering," in *ICASSP*, 2024, pp. 5730–5734.

[34] G. Pleiss and J. P. Cunningham, "The limitations of large width in neural networks: A deep gaussian process perspective," *NeurIPS*, vol. 34, pp. 3349–3363, 2021.

[35] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," in *ICLR*, 2018.

[36] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *NeurIPS*, vol. 31, 2018.

[37] R. Cheng, R. M. Murray, and J. W. Burdick, "Limits of probabilistic safety guarantees when considering human uncertainty," in *ICRA*, 2021, pp. 3182–3189.

[38] J. W. Seaman III, J. W. Seaman Jr, and J. D. Stamey, "Hidden dangers of specifying noninformative priors," *The American Statistician*, vol. 66, no. 2, pp. 77–84, 2012.

[39] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Interval analysis*. Springer, 2001.

[40] J. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax Weiss, and B. Lakshminarayanan, "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness," *NeurIPS*, vol. 33, pp. 7498–7512, 2020.

[41] J. Mukhoti, A. Kirsch, J. van Amersfoort, P. H. Torr, and Y. Gal, "Deep deterministic uncertainty: A new simple baseline," in *CVPR*, 2023, pp. 24 384–24 394.

[42] J. Van Amersfoort, L. Smith, Y. W. Teh, and Y. Gal, "Uncertainty estimation using a single deep deterministic neural network," in *ICML*, 2020, pp. 9690–9700.

[43] J. Van Amersfoort, L. Smith, A. Jesson, O. Key, and Y. Gal, "On feature collapse and deep kernel learning for single forward pass uncertainty," *arXiv:2102.11409*, 2021.

[44] C. De Boor, *A practical guide to splines*. springer New York, 1978, vol. 27.

[45] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *ICML*, 2015, pp. 1613–1622.

[46] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher *et al.*, "A survey of uncertainty in deep neural networks," *Artificial Intelligence Review*, vol. 56, no. Suppl 1, pp. 1513–1589, 2023.

[47] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernandez-Lobato, and A. L. Gaunt, "Deterministic variational inference for robust bayesian neural networks," *arXiv:1810.03958*, 2018.

[48] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *ICML*, 2011, pp. 681–688.

[49] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *ICML*, 2015, pp. 1861–1869.

[50] X. Lu and B. Van Roy, "Ensemble sampling," *NeurIPS*, vol. 30, 2017.

[51] H. Ritter, A. Botev, and D. Barber, "A scalable laplace approximation for neural networks," in *ICLR*, vol. 6, 2018.

[52] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *ICML*, 2017, pp. 1321–1330.

[53] M. Ataei and V. Dhiman, "DADEE: Well-calibrated uncertainty quantification in neural networks for barriers-based robot safety," *arXiv:2407.00616*, 2024.

[54] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *NeurIPS*, vol. 30, 2017.

[55] F. Berkenkamp, A. P. Schoellig, and A. Krause, "No-regret bayesian optimization with unknown hyperparameters," *Journal of Machine Learning Research*, vol. 20, no. 50, pp. 1–24, 2019.

[56] A. Liu, G. Shi, S.-J. Chung, A. Anandkumar, and Y. Yue, "Robust regression for safe exploration in control," in *Learning for Dynamics and Control*, 2020, pp. 608–619.

[57] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

[58] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[59] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," *NeurIPS*, vol. 18, 2005.

[60] J. Hensman, A. Matthews, and Z. Ghahramani, "Scalable variational gaussian process classification," in *Artificial intelligence and statistics*, 2015, pp. 351–360.

[61] S. Takacs and T. Takacs, "Approximation error estimates and inverse inequalities for b-splines of maximum smoothness," *Mathematical Models and Methods in Applied Sciences*, vol. 26, no. 07, pp. 1411–1445, 2016.

[62] E. Sande, C. Manni, and H. Speleers, "Explicit error estimates for spline approximation of arbitrary smoothness in isogeometric analysis," *Numerische Mathematik*, vol. 144, no. 4, pp. 889–929, 2020.

[63] P. Campolucci, F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini, "Neural networks with adaptive spline activation function," in *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*, vol. 3, 1996, pp. 1442–1445.

[64] C. J. Harris, C. G. Moore, and M. Brown, *Intelligent control: aspects of fuzzy logic and neural nets*. World Scientific, 1993, vol. 6.

[65] X. Hong and S. Chen, "Modeling of complex-valued wiener systems using b-spline neural network," *IEEE Transactions on Neural Networks*, vol. 22, no. 5, pp. 818–825, 2011.

[66] H. Montanelli and H. Yang, "Error bounds for deep relu networks using the kolmogorov–arnold superposition theorem," *Neural Networks*, vol. 129, pp. 1–6, 2020.

[67] R. Balestriero *et al.*, "A spline theory of deep learning," in *ICML*, 2018, pp. 374–383.

[68] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *CVPR*, 2018, pp. 869–877.

[69] G. Wahba, *Spline Functions*. John Wiley & Sons, Ltd, 2006.

[70] S. Morris, "Hilbert 13: Are there any genuine continuous multivariate real-valued functions?" *Bulletin of the American Mathematical Society*, vol. 58, no. 1, pp. 107–118, 2021.

[71] V. Kůrkovà, "Kolmogorov's theorem and multilayer neural networks," *Neural networks*, vol. 5, no. 3, pp. 501–506, 1992.

[72] J. Schmidt-Hieber, "The kolmogorov–arnold representation theorem revisited," *Neural networks*, vol. 137, pp. 119–126, 2021.

[73] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proceedings of the international conference on Neural Networks*, vol. 3, 1987, pp. 11–14.

[74] V. E. Ismailov, "A three layer neural network can represent any multivariate function," *Journal of Mathematical Analysis and Applications*, vol. 523, no. 1, p. 127096, 2023.

[75] V. Dhiman, M. J. Khojasteh, M. Franceschetti, and N. Atanasov, "Control barriers in bayesian learning of system dynamics," *IEEE transactions on automatic control*, vol. 68, no. 1, pp. 214–229, 2021.

[76] D. Bertsimas, D. Den Hertog, and J. Pauphilet, "Probabilistic guarantees in robust optimization," *SIAM Journal on Optimization*, vol. 31, no. 4, pp. 2893–2920, 2021.

[77] G. M. Phillips, *Interpolation and approximation by polynomials*. Springer Science & Business Media, 2003, vol. 14.

[78] J. F. Epperson, "On the runge example," *The American Mathematical Monthly*, vol. 94, no. 4, pp. 329–341, 1987.

[79] T. M. Apostol, *Calculus, Volume 1*. John Wiley & Sons, 1967.

[80] A. Lederer, J. Umlauft, and S. Hirche, "Uniform error bounds for gaussian process regression with application to safe control," *NeurIPS*, vol. 32, 2019.

[81] Z. Shi, Y. Wang, H. Zhang, J. Z. Kolter, and C.-J. Hsieh, "Efficiently computing local lipschitz constants of neural networks via bound propagation," *NeurIPS*, vol. 35, pp. 2350–2364, 2022.

[82] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[84] L. S. Shapley, "A value for n-person games," *Contribution to the Theory of Games*, vol. 2, 1953.

[85] A. Ghorbani and J. Y. Zou, "Neuron shapley: Discovering the responsible neurons," *NeurIPS*, vol. 33, pp. 5922–5932, 2020.

[86] A. Howard, "The robotics data set repository (radish)," *http://radish.sourceforge. net/*, 2003.

[87] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3730–3738.

[88] R. Cheng, M. J. Khojasteh, A. D. Ames, and J. W. Burdick, "Safe multi-agent interaction through robust control barrier functions with learned uncertainties," in *IEEE CDC*, 2020, pp. 777–783.

[89] D. Chen, T. Qiao, H. Tan, M. Li, and Y. Zhang, "Solving the problem of runge phenomenon by pseudoinverse cubic spline," in *2014 IEEE 17th International Conference on Computational Science and Engineering*, 2014, pp. 1226–1231.

Masoud Ataei received his M.Sc. degree in Electrical Engineering from Amirkabir University of Technology and is currently pursuing his Ph.D. in Electrical Engineering at the University of Maine. He serves as a Research Assistant in the Computer Vision and Autonomous Robotics lab at the University of Maine and is passionate about advancing safe control systems, particularly in autonomous robotics. His current research focuses on developing a risk-aware safe control framework for autonomous vehicles, combining model-based techniques with machine learning approaches to enable reliable navigation in uncertain environments.

Mohammad Javad Khojasteh (Member, IEEE) received the Ph.D. and M.Sc. degrees in electrical and computer engineering from the University of California, San Diego in 2019 and 2017, respectively. He is a Gleason Endowed Assistant Professor with the Rochester Institute of Technology (RIT). Before joining RIT, he held postdoctoral positions at Marine Physical Laboratory (MPL) at Scripps Institution of Oceanography (SIO), Department of Mechanical Engineering and Laboratory for Information and Decision Systems (LIDS) at Massachusetts Institute of Technology (MIT), and Center for Autonomous Systems and Technologies (CAST) at California Institute of Technology (Caltech), where he worked with Team CoSTAR as visitor at NASA Jet Propulsion Laboratory. He received the Gleason Chair in 2024 and the 2025 Provost's Learning Innovation Grant at RIT. Dr. Khojasteh's publications, co-authored with colleagues and students, have received awards, including Tammy L. Blair Student Paper Award (second place) from the International Society of Information Fusion.

Vikas Dhiman (Member, IEEE) received the Electrical Engineering degree from the Indian Institute of Technology, Roorkee, in 2008, the M.S. degree from the University at Buffalo, in 2014, and the Ph.D. degree from the University of Michigan, Ann Arbor, in 2019. His works lie in the localization, mapping, and control algorithms for applications in robotics. He was a Postdoctoral Researcher with the University of California, San Diego, from 2019 to 2021. He is currently an Assistant Professor with the ECE Department, the University of Maine.

DAREK - DISTANCE AWARE ERROR FOR KOLMOGOROV NETWORKS (Supplementary Material)

Masoud Ataei* Mohammad Javad Khojasteh† Vikas Dhiman*
*Electrical and Computer Engg. Dept., University of Maine, Orono, ME, USA
†Electrical and Microelectronic Engg. Dept., Rochester Institute of Technology, Rochester, NY, USA

A. Background and Proofs

Here, we review the background and definitions used in the paper. We also provide proof for the theories used in this paper.

I. Mean Value Theorem:

The result of following Theorem establishes the relation between divided differences and derivatives.

Theorem 5 (Mean value theorem [44]): For a $k-1$ times differentiable function $f:\mathbb{R}\to\mathbb{R}$ and a set of k points $\mathcal{T}=\{\tau_1,\dots,\tau_k\}$, there exists $\zeta\in\mathcal{R}(\mathcal{T})$ such that:

$$[\tau_1,\dots,\tau_k]f=\frac{1}{(k-1)!}f^{(k-1)}(\zeta)\quad\exists\zeta\in\mathcal{R}(\mathcal{T}). \tag{42}$$

Here, $f^{(k-1)}(.)$ denotes the $k-1$th derivative of the function and $\mathcal{R}(\mathcal{T})=[\min(\mathcal{T}),\max(\mathcal{T})]$ the interval spanned by the knots $\mathcal{T}$.

Proof: Suppose we approximate a function f with a $k-1$th order newton's polynomial $\hat{f}=\mathcal{P}_{k-1}$, since $\hat{f}$ passes through k points at $f(\tau_{1:k})$ then we know $f(\tau_i)=\hat{f}(\tau_i)\ \forall i\in 1,\dots,k$. It means $e(\tau_i)=f(\tau_i)-\hat{f}(\tau_i)=0$. Since, $f-\hat{f}$ is a continuous function then between every two continuous knots there exists at least one point that derivative of $f-\hat{f}$ will be zero, having k point will make $k-1$ point of e' to be zero. Continuing this process results that there are one zero on $k-1$th derivative of e function which means $(f-\hat{f})^{(k-1)}(\zeta)=0$ at one point, lets call it ζ. Then we have:

$$\begin{aligned}f^{(k-1)}&=(\mathcal{P}_k)^{(k-1)}\\&=([\tau_1]f+\sum_{i=1}^{k-1}[\tau_{1:i+1}]f\prod_{j=1}^{i}(x-\tau_j))^{(k-1)}\\&=(k-1)!\quad[\tau_1,\dots,\tau_k]f\\\Rightarrow[\tau_1,\dots,\tau_k]f&=\frac{1}{(k-1)!}f^{(k-1)}(\zeta).\end{aligned} \tag{43}$$

■

II. Linearity of Divided Difference:

Divided differences are linear with respect to the function values, i.e., $[\boldsymbol{\tau}](\alpha f+\beta g)=\alpha[\boldsymbol{\tau}]f+\beta[\boldsymbol{\tau}]g$ for a set of k distinct knots $\boldsymbol{\tau}=\{\tau_1,\dots,\tau_k\}$. We use this property to establish the linearity of Newton's polynomial; the proof is as follows.

Proof: Let f and g be two functions, and let α and β be two scalars. We will show that the divided difference (1) operator satisfies linearity. For the base case, consider the zeroth-order divided differences:

$$\begin{aligned}[\tau_1](\alpha f+\beta g)&=(\alpha f+\beta g)(\tau_1)\\&=\alpha f(\tau_1)+\beta g(\tau_1)\\&=\alpha[\tau_1]f+\beta[\tau_1]g.\end{aligned} \tag{44}$$

We proceed to the first-order divided differences to confirm linearity:

$$\begin{aligned}[\tau_1,\tau_2](\alpha f+\beta g)&=\frac{[\tau_2](\alpha f+\beta g)-[\tau_1](\alpha f+\beta g)}{\tau_2-\tau_1}\\&=\frac{(\alpha f(\tau_2)+\beta g(\tau_2))-(\alpha f(\tau_1)+\beta g(\tau_1))}{\tau_2-\tau_1}\\&=\alpha\frac{f(\tau_2)-f(\tau_1)}{\tau_2-\tau_1}+\beta\frac{g(\tau_2)-g(\tau_1)}{\tau_2-\tau_1}\\&=\alpha[\tau_1,\tau_2]f+\beta[\tau_1,\tau_2]g.\end{aligned} \tag{45}$$

For higher-order divided differences, we use induction. Assume that for $k-1$th order divided differences, the linearity holds:

$$\begin{aligned}&[\tau_1,\dots,\tau_{k-1}](\alpha f+\beta g)=\\&\qquad\alpha[\tau_1,\dots,\tau_{k-1}]f+\beta[\tau_1,\dots,\tau_{k-1}]g.\end{aligned} \tag{46}$$

Now, for the kth order divided differences, we have:

$$\begin{aligned}&[\tau_1,\dots,\tau_k](\alpha f+\beta g)\\&\quad=\frac{[\tau_2,\dots,\tau_k](\alpha f+\beta g)-[\tau_1,\dots,\tau_{k-1}](\alpha f+\beta g)}{\tau_k-\tau_1}\\&\quad=\alpha[\tau_1,\dots,\tau_k]f+\beta[\tau_1,\dots,\tau_k]g.\end{aligned} \tag{47}$$

Thus, the divided difference operator is linear for all orders. ■

III. Linearity of Newton's polynomial operator:

Lemma 1 implicitly depends on this property, and it has also been used in Equation (50), and its proof is as follows.

Proof: From definition of Newton's polynomial (2) for a function $\alpha f+\beta g$, we have:

$$\begin{aligned}\mathcal{P}_{k,n}[\alpha f+\beta g](x)&=[\tau_n](\alpha f+\beta g)\\&+\sum_{i=1}^{k}[\tau_n,\dots,\tau_{n+i}](\alpha f+\beta g)\prod_{j=n}^{n+i-1}(x-\tau_j).\end{aligned} \tag{48}$$

Using the linearity properties of divided differences, we can rewrite this as:

$$\begin{aligned}&\mathcal{P}_{k,n}[\alpha f+\beta g](x)=(\alpha[\tau_n]f+\beta[\tau_n]g)\\&+\sum_{i=1}^{k}(\alpha[\tau_n,\dots,\tau_{n+i}]f+\beta[\tau_n,\dots,\tau_{n+i}]g)\prod_{j=n}^{n+i-1}(x-\tau_j)\\&=\alpha\mathcal{P}_{k,n}[f](x)+\beta\mathcal{P}_{k,n}[g](x).\end{aligned} \tag{49}$$

Therefore, $\mathcal{P}_{k,n}$ is a linear operator. ■

IV. Proof of Lemma 1:

Note that $\mathcal{P}_{k,j}[\mathcal{D}_{\hat{f}_{[j]}}(\boldsymbol{\tau}_{1:m})](x) = \hat{f}_{[j]}(x)$ because $\hat{f}_{[j]}(x)$ is a kth-order polynomial that generated through the $k+1$ nearby knots $\{(\tau_i, \hat{f}_{[j]}(\tau_i))\}_{i=j}^{j+k}$ and Newton's polynomial is a unique kth-order polynomial that is generated through the same data $k+1$ knots [44]. From the linearity of Newton's polynomial operator at the same knots we have,

$$\mathcal{P}_{k,j}[\mathcal{D}_{\hat{f}_{[j]}}](x) = \mathcal{P}_{k,j}[\mathcal{D}_f](x) - \mathcal{P}_{k,j}[e^f_j](x). \tag{50}$$

To simplify notation, we drop the knots $\boldsymbol{\tau}_{1:m}$ when clear from the context. Combining these observations, for $x \in [\tau_j, \tau_{j+1})$ and $j \in \{1, \dots, m-1\}$, we get,

$$\begin{aligned} f(x) - \hat{f}_j(x) &= f(x) - \mathcal{P}_{k,j}[\mathcal{D}_{\hat{f}_{[j]}}](x) \\ &\overset{(50)}{=} f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f](x) + \mathcal{P}_{k,j}[\mathcal{D}_{e^f_j}](x) \\ \text{or } |f(x) - \hat{f}_{[j]}(x)| &\leq |f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f](x)| + |\mathcal{P}_{k,j}[\mathcal{D}_{e^f_j}](x)| \\ &\overset{Thm.\ 1}{\leq} \bar{u}_f(x) + |\mathcal{P}_{k,j}[\mathcal{D}_{e^f_j}](x)|. \end{aligned} \tag{51}$$

The resulting inequality is tight when inequality from Theorem 1 is tight and $\text{sign}(f(x) - \mathcal{P}_{k,j}[\mathcal{D}_f](x)) = \text{sign}(\mathcal{P}_{k,j}[\mathcal{D}_{e^f_j}](x))$. ■

V. Proof of Theorem 4:

Consider a function f which is approximated with the residual structure of $\hat{f}_{[j]} = \hat{r} + \phi_k$ on interval $[\tau_j, \tau_{j+1})$. Since Newton's polynomial is uniquely determined by $k+1$ points, we can accordingly construct Newton's polynomial of ϕ_k. Also, $\mathcal{P}_{k,j}[\mathcal{D}_{\phi_k}] = \phi_k$ on interval $[\tau_j, \tau_{j+1})$. Note that the non-zero error of neuron can be defined as $e^{f-\hat{r}}_{[j]} = f - \hat{f}_{[j]} = f - \hat{r} - \phi_k$, and since ϕ_k in jth interval is a polynomial, and polynomial interpolation is a linear operation, we can define $\phi_k = \mathcal{P}_{k,j}[\mathcal{D}_{\phi_k}] = \mathcal{P}_{k,j}[\mathcal{D}_{f-\hat{r}}] - \mathcal{P}_{k,j}[\mathcal{D}_{e^{f-\hat{r}}_{[j]}}]$. Substituting this expression into following equation obtains:

$$f - \hat{f}_{[j]} = (f - \hat{r}) - \mathcal{P}_{k,j}[\mathcal{D}_{f-\hat{r}}] + \mathcal{P}_{k,j}[\mathcal{D}_{e^{f-\hat{r}}_{[j]}}]. \tag{52}$$

Then the error bound of the $f - \hat{r}$ would be:

$$\begin{aligned} |f - \hat{f}_{[j]}| &\leq |(f - \hat{r}) - \mathcal{P}_{k,j}[\mathcal{D}_{f-\hat{r}}]| + |\mathcal{P}_{k,j}[\mathcal{D}_{e^{f-\hat{r}}_{[j]}}]| \\ &= u_{f-\hat{r}}(x; \boldsymbol{\tau}_{1:m}). \end{aligned} \tag{53}$$

■

VI. Interpolation Error from Theorem 1 is Distance-aware.

Interpolation error (14) is distance aware because it satisfies distance awareness condition (8). Here, we first show that the interpolation error has a unique maximum between consecutive knots and is monotonically increasing until it reaches the maximum, then decreasing, and show how this satisfies distance-awareness for differentiable uncertainty (8).

For a set of distinct and consecutive knots $\boldsymbol{\tau}_{1:k+1}$, the interpolation error (14), within the interval $\tau_j \leq x < \tau_{j+1}$, is given by

$$\bar{u}_{[j]}(x, \boldsymbol{\tau}^{(j)}) \coloneqq \alpha \left| \prod_{i=1}^{k+1} (x - \tau_i^{(j)}) \right| \tag{54}$$

$$= \alpha \prod_{i=1}^{j} (x - \tau_i^{(j)}) \prod_{i=j+1}^{k+1} (\tau_i^{(j)} - x), \tag{55}$$

where $\alpha \coloneqq \frac{\mathcal{L}_f^{k+1}}{(k+1)!}$. We shorten $\bar{u}_{[j]}(x, \boldsymbol{\tau}^{(j)})$ to $\bar{u}_{[j]}(x)$ when knots are clear from the context. Take the derivative of $\bar{u}_j(x)$ with respect to x, and write it as $\bar{u}'_{[j]}(x) = \bar{u}_{[j]}(x) g_{[j]}(x)$, where $g_{[j]}(x) = \sum_{i=1}^{j} \frac{1}{x - \tau_i} - \sum_{i=j+1}^{k+1} \frac{1}{\tau_i - x}$. The function $g_{[j]}(x)$ is strictly decreasing since its derivative is negative $g'_{[j]}(x) = -\sum_{i=1}^{k+1} (x - \tau_i)^{-2} < 0$. In addition, $g_{[j]}(x) \to +\infty$, $x \to \tau_j^+$, $g_{[j]}(x) \to -\infty$ as $x \to \tau_{j+1}^-$ and $g_{[j]}(x^*) = 0$ when $\sum_{i=1}^{j} \frac{1}{x - \tau_i} = \sum_{i=j+1}^{k+1} \frac{1}{\tau_i - x}$. Since, $g_{[j]}(x)$ is a continuous and strictly decreasing function in interval (τ_j, τ_{j+1}), therefore by the intermediate value theorem then there exists a unique $x^* \in (\tau_j, \tau_{j+1})$ such that $g_{[j]}(x^*) = 0$. Since $\bar{u}'_{[j]}(x) = \bar{u}_{[j]}(x) g_{[j]}(x)$ and $\bar{u}_{[j]}(x) > 0$, the sign of $\bar{u}'_{[j]}(x)$ is completely determined by the sign of $g_{[j]}(x)$. Consequently, $\bar{u}'_{[j]}(x) > 0$ for $x \in (\tau_j, x^*)$ and $\bar{u}'_{[j]}(x) < 0$ for $x \in (x^*, \tau_{j+1})$. Choose an inducing distance function $d_u(x, \tau^*)$ for distance-awareness such that, the nearest knot is $\tau^* = \tau_j$ for $x \in [\tau_j, x^*)$ and $\tau^* = \tau_{j+1}$ for $x \in [x^*, \tau_{j+1})$,

$$d_u(x, \tau^*) \coloneqq \begin{cases} x - \tau_j & \text{if } x \in [\tau_j, x^*) \\ \tau_{j+1} - x & \text{if } x \in [x^*, \tau_{j+1}) \end{cases}. \tag{56}$$

Substituting $\bar{u}_{[j]}(x)$ and $d_u(x, \tau^*)$ into the requirement of distance-awareness (8):

$$\bar{u}'_{[j]}(x) d'_u(x, \tau^*) = \begin{cases} \bar{u}'_{[j]}(x) \frac{d}{dx}(x - \tau_j) \geq 0 & \text{if } x \in [\tau_j, x^*) \\ \bar{u}'_{[j]}(x) \frac{d}{dx}(\tau_{j+1} - x) \geq 0 & \text{otherwise.} \end{cases} \tag{57}$$

■

VII. Distance-Awareness of Spline Error (Lemma 1).

The error at knot $\mathcal{P}_{k,j}(x)$ does not change the number of unique maxima of the interpolation error, and therefore preserves distance awareness.

From (18), the neuron uncertainty consists of two terms: an interpolation error term $\bar{u}_j(x)$ and a correction term $|\mathcal{P}_{1,j}(x)|$ capturing the error at the knots. If we use a linear function for error at knots, $|\mathcal{P}_{1,j}(x)| = |e^f_j| + c_j(x - \tau_j)$, where $c_j = (|e^f_{j+1}| - |e^f_j|)/(\tau_{j+1} - \tau_j)$ is a constant and finite because the knots are distinct. The derivative of (18) becomes $u'_j(x) = \bar{u}'_j(x) + c_j$. From Appendix A-VI, $\bar{u}'_j$ is continuous (τ_j, τ_{j+1}) and changes from $+\infty$ to $-\infty$ with a unique zero. Adding a finite, constant term to it will preserve continuity, monotonicity, and the boundary limits $\pm\infty$ of u_j. Consequently, $u_j(x)$ remains strictly increasing on $x \in (\tau_j, \tilde{x}^*)$ and strictly decreasing on $x \in (\tilde{x}^*, \tau_{j+1})$ and retains the distance awareness property by the substitution into (8) as shown in Appendix A-VI, where $\tilde{x}^*$ is the unique maxima of $u_{[j]}(x)$ within the knots $[tau_j, \tau_{j+1})$.

B. Shapley Example

Here we explain how we Shapley value for error division in a simple example. Consider the network $\hat{f}(x) = h(g_1(x)+g_2(x))$. This has the set of neurons $\Phi = \{h, g_1, g_2\}$ and which can produce eight different architectures including $\{x, g_1(x), g_2(x), h(x), h(g_1(x)), h(g_2(x)), h(g_1(x)+g_2(x))\}$. The operator $\mathcal{M}$ can construct each instance by passing the set of neurons to it, for instance $\mathcal{M}(\{\emptyset\}) = x$ or $\mathcal{M}(\{h, g_2\}) = h(g_2(x))$. Also, note that the permutation in this problem does not affect the output, $\mathcal{M}(\{h, g_1\}) = \mathcal{M}(\{g_1, h\})$.

Assume we want to calculate the contribution of g_1. All possible sets without g_1 would be $\Phi_s \subseteq \Phi \setminus \{g_1\} = \{\emptyset, \{g_2\}, \{h\}, \{g_2, h\}\}$ and we can calculate the error each of eight networks over the entire dataset is made then substitute values in (37). And, we assume we only have access to knots data $f(\tau_{1:m})$ (however it can be any data in the dataset), then the evaluation for one of eight sub-networks would be:

$$\mathcal{V}(\mathcal{M}(\{h, g2\})) = \mathcal{V}(h(g_2)) = \frac{1}{m}\sum_{i=1}^{m}(y_i - h(g_2(\tau_i))).$$

Then the complete equation would be:

$$\begin{aligned}
\varphi_{g_1} = &\frac{0!2!}{3!}\frac{1}{m}\sum_{i=1}^{m}\big(g_1(\tau_i) - \tau_i\big) && \Phi_s=\emptyset \\
&+ \frac{1!1!}{3!}\frac{1}{m}\sum_{i=1}^{m}\big((g_2(\tau_i) + g_1(\tau_i)) - g_2(\tau_i)\big) && \Phi_s=\{g_2\} \\
&+ \frac{1!1!}{3!}\frac{1}{m}\sum_{i=1}^{m}\big(h(g_1(\tau_i)) - h(\tau_i)\big) && \Phi_s=\{h\} \\
&+ \frac{2!0!}{3!}\frac{1}{m}\sum_{i=1}^{m}\big(h(g_2(\tau_i) + g_1(\tau_i)) - h(g_2(\tau_i))\big) && \Phi_s=\{g_2,h\}
\end{aligned} \tag{58}$$

C. Additional Experiments

In this section, we present additional experiments for the paper: studying the error of a single neuron and a two-layer DAREK, providing details on the Lipschitz division experiment, and a high-dimensional face recognition experiment.

I. Spline Error Bound

We trained a one-layer DAREK on a cos function. We used 20 equally spaced points from $[-2\pi, 2\pi]$ as training data, and selected 9 knots from these samples. The model was trained for 200 epochs with $lr = 1.0$. As Fig. 9 (a) depicts, while the fitted model does not perfectly follow the actual function, the overall error bound completely covers the true function. The error bounds between knots grow from one knot and shrink when meeting another knot. In this study, we use the error bound with spline fit (EBS) (17) for the error at knots.

II. Two-layer DAREK Error Bound

In this experiment, we trained a two-layer spline network with two hidden units to approximate the *cos* function. The network was trained on 30 equally spaced points from $[-2\pi, 2\pi]$, using 9 knots selected from these points. Training was conducted over 200 epochs with a learning rate of 0.05.

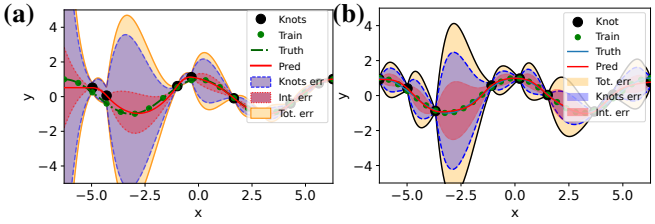

Fig. 9. The error bounds of DAREK model on cos function. **(a)** one-layer model, **(b)** two-layer DAREK model.

Fig. 9 (b) shows the overall output error. To analyze layer-wise behavior, we provide visualizations of the intermediate neuron error in Fig. 10, showing each input-output pair separately. Since the output of the first layer serves as the input domain for the second layer, any extrema in the first layer can reverse the direction of the input to the second layer, making the composite input non-functional unless sorted. Although the interpolation error of the second layer remains continuous with respect to its input, the propagated error from earlier layers may appear discontinuous due to this sorting process. To better illustrate this phenomenon, we color-coded different monotonic segments of the first layer's output. In this study, we use error bound for spline (EBS) (17) for the error of individual neurons.

III. Lipschitz Division Results

In this section, we provide details of the computed Lipschitz constants from section VI-D and Table III. The parameters are provided in Table VI. For function f_1, we used 50 training data equally spaced from -2π to 2π and we used a two-layer DAREK with two hidden units, cubic splines, and 9 knots. We only trained the model for 50 epochs with lr=0.1 to leave some error at knots for error study as well.

For function f_2, we used 2500 training pairs, a two-layer DAREK with two inputs and five hidden units, cubic splines, and 20 knots for each neuron. We trained the model until its MSE loss reached 0.256.

In the experiment for function f_3, we used 2500 training pairs, a two-layer DAREK with two inputs and ten hidden units, cubic splines, and 30 knots. We trained the model until its MSE loss reached 0.592.

For the Twomoon dataset, f_4, we used two moon datasets from the sklearn package, each with 500 training samples, and a model similar to the one we used for f_2, and trained the model until its MSE loss reached 0.735. In this study, we use EBL (18) for the error of individual neurons.

IV. Face detection

We provide the full results for face recognition experiments in Table VII, extending the *High-dimensional experiment* in Section VII to additional feature dimensions (2, 5, 10, 20, 50, 100, and 200). These results are consistent with the main findings: DAREK (DK1 and DK2) maintains stable RMSE and IoU, and SDA improves as the feature dimension increases. ENS1 and ENS2 do not provide consistent SDA performance, and GP-based methods (ExGP and ApxGP) achieve near-perfect SDA.

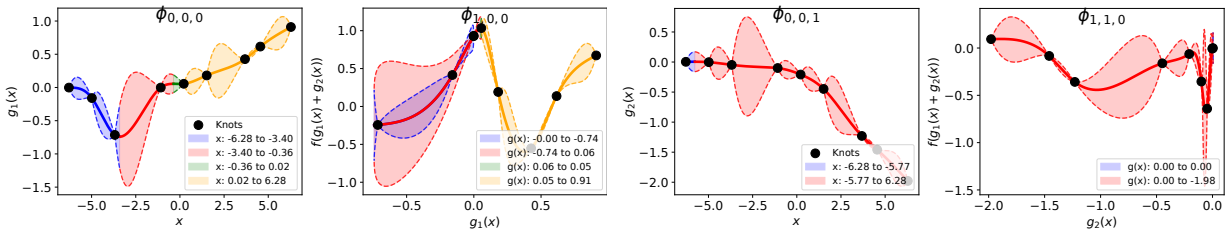

Fig. 10. Layer-wise interpolation errors of two-layer DAREK model on cos function; color highlights monotonic segments of the first layer affecting the second layer's input.

TABLE VI

LIPSCHITZ CONSTANTS PARAMETERS ESTIMATED USING DIFFERENT METHODS ON DIFFERENT DATASETS IN SECTION VI-D.

	f_1			f_2			f_3			f_4		
Methods	$\mathcal{L}_g^k$	$\mathcal{L}_h^1$	$\mathcal{L}_h^k$	$\mathcal{L}_g^k$	$\mathcal{L}_h^1$	$\mathcal{L}_h^k$	$\mathcal{L}_g^k$	$\mathcal{L}_h^1$	$\mathcal{L}_h^k$	$\mathcal{L}_g^k$	$\mathcal{L}_h^1$	$\mathcal{L}_h^k$
Equal	0.500	0.500	0.500	1.414	0.548	1.414	0.577	0.258	0.577	0.816	0.365	0.816
Linear	0.000	0.345	0.500	0.001	0.030	3.999	0.500	0.001	0.000	1.000	0.001	0.000
Logarithmic	0.500	0.500	0.500	0.397	0.297	1.008	0.175	0.120	0.285	0.381	0.218	0.524
Worst-case	1.000	1.000	1.000	20.00	3.000	20.00	10.0	2.0	10.00	10.00	2.000	10.00
Optimization	1.000	1.000	0.368	6.439	1.645	20.00	10.0	1.406	10.00	10.00	2.000	10.00

TABLE VII

RESULTS FOR HIGH-DIMENSIONAL EXPERIMENTS ACROSS FOUR FEATURE EXTRACTION METHODS (PCA AND RESNET-18).

(a) PCA: resize to $224 \times 224 \times 3$ then select top n PCA dimensions

	RMSE ↓							IoU ↑							SDA ↑						
Model	2	5	10	20	50	100	200	2	5	10	20	50	100	200	2	5	10	20	50	100	200
DK1	0.136	0.133	0.133	0.133	0.126	0.129	0.149	0.451	0.456	0.454	0.452	0.469	0.466	0.410	0.676	0.856	0.947	0.967	0.974	0.952	0.969
DK2	0.137	0.139	0.134	0.190	0.123	0.184	0.135	0.448	0.443	0.454	0.375	0.486	0.339	0.451	0.677	0.857	0.947	0.972	0.974	0.950	0.971
ENS1	0.136	0.133	0.133	0.133	0.126	0.130	0.151	0.450	0.457	0.455	0.451	0.467	0.464	0.407	0.571	0.554	0.568	0.563	0.501	0.431	0.378
ENS2	0.135	0.131	0.131	0.128	0.183	0.265	0.129	0.449	0.456	0.446	0.469	0.311	0.145	0.431	0.527	0.542	0.545	0.530	0.588	0.564	0.525
ExGP	0.138	0.149	0.137	0.161	0.138	0.137	0.143	0.435	0.400	0.450	0.359	0.437	0.448	0.420	1.000	1.000	1.000	1.000	1.000	1.000	1.000
ApxGP	0.138	0.137	0.137	0.137	0.137	0.137	0.137	0.427	0.447	0.447	0.447	0.447	0.447	0.447	0.977	1.000	1.000	1.000	1.000	1.000	1.000

(b) ResNet-18: resize to $224 \times 224 \times 3$, extract ResNet-18 features, select top n via PCA

	RMSE ↓							IoU ↑							SDA ↑						
Model	2	5	10	20	50	100	200	2	5	10	20	50	100	200	2	5	10	20	50	100	200
DK1	0.100	0.097	0.090	0.088	0.079	0.074	0.079	0.535	0.538	0.562	0.557	0.586	0.606	0.589	0.704	0.686	0.875	0.892	0.788	0.837	0.935
DK2	0.100	0.100	0.093	0.103	0.083	0.073	0.072	0.533	0.535	0.553	0.470	0.573	0.608	0.626	0.691	0.707	0.899	0.918	0.858	0.882	0.902
ENS1	0.100	0.097	0.089	0.089	0.080	0.075	0.081	0.535	0.540	0.565	0.557	0.585	0.603	0.583	0.578	0.551	0.566	0.562	0.364	0.222	0.408
ENS2	0.100	0.094	0.087	0.084	0.174	0.069	0.065	0.530	0.551	0.569	0.575	0.161	0.628	0.646	0.515	0.533	0.573	0.562	0.461	0.550	0.542
ExGP	0.139	0.140	0.139	0.140	0.140	0.137	0.138	0.431	0.447	0.434	0.427	0.426	0.447	0.444	0.542	1.000	1.000	1.000	1.000	1.000	1.000
ApxGP	0.133	0.137	0.137	0.137	0.137	0.137	0.137	0.445	0.447	0.447	0.447	0.447	0.447	0.447	0.729	1.000	1.000	1.000	1.000	1.000	1.000

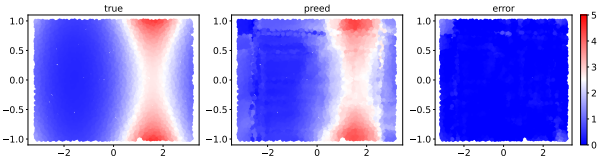

Fig. 11. The true dataset, mean prediction, and error estimation of function $f_2(x, y) = \exp(\sin(x) + y^2/2)$.

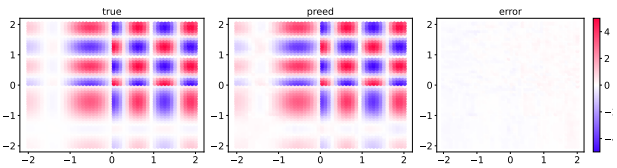

Fig. 12. The true dataset, mean prediction, and error estimation of function f_3 as defined in (32).

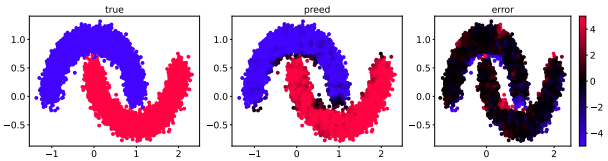

Fig. 13. The true dataset, mean prediction, and error estimation of the twomoon dataset.