Institute of Electrical Engineering and Information Technology
Paderborn University
Department of Power Electronics and Electrical Drives
Prof. Dr.-Ing. Jakub Kučka

Master Thesis

Uncertainty quantification for data-driven motor drive temperature estimation models

by

Shubham Gupta

Student ID: Redacted

First examiner: Prof. Dr.-Ing. Jakub Kucka

Second examiner: Prof. Dr.-Ing. Erdal Kayacan

Supervisor: Prof. Dr.-Ing. Jakub Kucka

Submission Date: May 17, 2026

UNIVERSITÄT PADERBORN

Selbstständigkeitserklärung für die Abgabe von Bachelor-, Master-, Studien- und Hausarbeiten (§ 63 Absatz 5 HG)

Gupta Shubham _____ _____

(Name, Vorname) (Matrikelnummer)

Hiermit versichere ich, dass ich meine ~~Bachelorarbeit~~/Masterarbeit/~~Studienarbeit/Hausarbeit~~ (Unzutreffendes bitte streichen) mit dem Titel

Uncertainty quantification for data-driven motor drive temperature estimation models

_____

– bei einer Gruppenarbeit den entsprechend gekennzeichneten Teil der Arbeit – selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, habe ich in jedem Fall unter Angabe der Quellen entsprechend der wissenschaftlichen Standards sowohl an den Stellen im Text als auch im Literaturverzeichnis kenntlich gemacht. Das Gleiche gilt auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw.

Sofern ich bei der Erstellung der Arbeit KI-generierte Inhalte (Text, Darstellungen, Programmcode etc.) verwendet habe, habe ich den Einsatz der KI-Tools sachgerecht dokumentiert und transparent gekennzeichnet. Hierbei habe ich mich an die Vorgaben der Prüfenden gehalten. Falls keine Vorgaben zur Kennzeichnung und Dokumentation des KI-Einsatzes gemacht wurden, habe ich ein den Gepflogenheiten meiner jeweiligen Fachdisziplin angemessenes Dokumentationssystem[1] verwendet.

Mir ist bekannt, dass die ungekennzeichnete oder nicht angemessen gekennzeichnete Übernahme von fremdem geistigem Eigentum unabhängig von dessen Herkunft (auch aus dem Internet) oder die ungekennzeichnete oder nicht angemessen gekennzeichnete Übernahme von KI-generierten Texten eine Täuschung darstellen kann, die zur Folge hat, dass die Arbeit als mit der Note „mangelhaft" (5,0) bewertet gilt. Dies umfasst auch die ungekennzeichnete oder nicht angemessen gekennzeichnete Übernahme von über das Allgemeinwissen hinausgehenden Fakten, Ideen, Argumenten oder spezifischen Formulierungen sowie deren Paraphrasierung oder Übersetzung.

Ich habe die Arbeit nicht, auch nicht auszugsweise, für eine andere abgeschlossene Prüfung angefertigt.

Paderborn 17.05.2026 _____

Ort und Datum Unterschrift

[1] Eine Übersicht der gängigen Dokumentationssysteme finden Sie in der „Handreichung zur Dokumentation und Kennzeichnung von KI-Tools" (http://go.upb.de/ki-wiss-arbeiten).

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor and first examiner, Prof. Dr.-Ing. Jakub Kucka, and my second examiner, Prof. Dr.-Ing. Erdal Kayacan, for their invaluable guidance, support, and evaluation of this Master's thesis at Universität Paderborn.

I am deeply thankful to Prof. Dr.-Ing. Oliver Wallscheid for proposing the original idea for this research. A special thank you goes to Mr. Darius Jakobeit for his outstanding day-to-day supervision, technical feedback, and continuous support throughout the implementation and evaluation phases of this project.

Finally, I would like to acknowledge the Paderborn Center for Parallel Computing (PC2) for providing the computational resources on the Noctua 2 supercomputer. The extensive hyperparameter optimizations and massive dataset evaluations required for this research would not have been possible without their infrastructure.

Abstract

Permanent magnet synchronous motors dominate many industries due to their superior power and torque density. However, temperature sensitivity remains a key vulnerability, as high temperatures can damage components like permanent magnets, stator windings, etc. Monitoring the temperature inside these motors with physical sensors comes with its own set of challenges, such as difficulty in measuring temperatures in a spinning rotor, sensor drift, etc. In addition to that, multiple sensors are needed to obtain a complete thermal profile, which poses another challenge in cost-prohibitive industries, such as automotive. To solve these problems, alternative methods of temperature estimation have been a focus of research. Data-driven temperature estimation models that integrate physical knowledge, such as thermal neural networks (TNNs), have shown great potential here. But their black-box nature and deterministic point predictions make them challenging to deploy in safety-critical applications.

This work intends to address this challenge by quantifying the uncertainty in data-driven temperature estimation models. It evaluates Gaussian processes, dropouts, deep ensembles, evidential deep learning (EDL) optimized with negative log-likelihood (NLL), and EDL optimized with the Continuous Ranked Probability Score (CRPS) on TNNs. To maintain the baseline TNN's gray-box physical interpretability, novel architectural solutions are proposed. Furthermore, this work demonstrates that CRPS is a viable loss function for EDL, an approach that has not previously been explored in the literature.

The performance of these uncertainty quantification (UQ) methods has been rigorously compared using a wide range of metrics, including CRPS, NLL, reliability diagrams, miscalibration area, inference time, and model storage requirements. Based on the analysis, Gaussian processes emerged as the winner, with the best overall uncertainty calibration performance. Dropout achieved the best accuracy on the MSE metric. EDL-based methods showed good performance, with negligible computational overhead. This work also demonstrated the unsuitability of deep ensembles for UQ in TNNs.

Contents

1 Introduction

Electric drives have seen an increase in demand, largely driven by the electric vehicle industry. Permanent magnet synchronous motors (PMSMs) are the most commonly used motors here due to their high power and torque density [1], [2]. However, PMSMs have many heat-sensitive components, such as the permanent magnet, stator yoke, stator tooth, and stator windings. Elevated temperatures in PMSMs do not merely result in efficiency losses, but they can also cause bearing failure, irreversible magnet demagnetization, and winding insulation breakdown. Conversely, to maximize performance, manufacturers aim to operate these motors at their physical limits. This vulnerability is why there has been a significant amount of research into the thermal modeling of PMSMs.

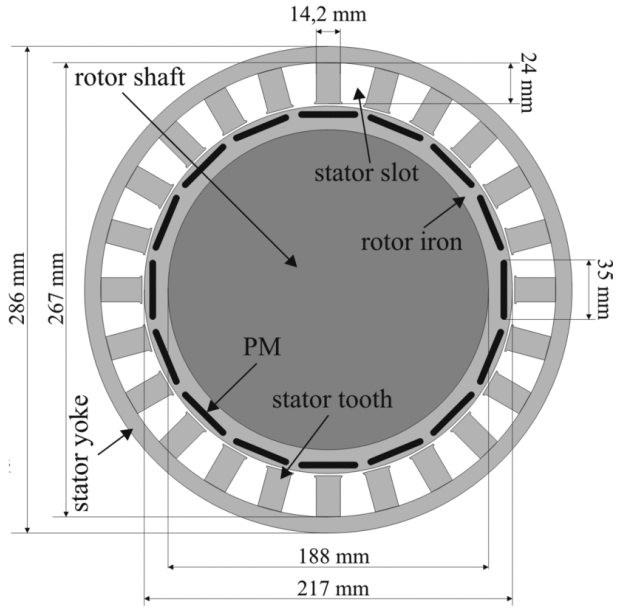

Fig. 1.1: Cross section of a PMSM (taken from [3])

Accurate internal temperature measurement through physical sensors alone presents several limitations. The work presented in [4] outlines several drawbacks of relying solely on sensor-based measurements. Sensors are prone to long-term failure and are costly to repair. They also struggle with spatial resolution, specifically in identifying localized

hotspots. Measuring temperatures within the rotating rotor with a physical sensor is also rather challenging.

Reliance on a single sensor also often results in an incomplete thermal profile. The author highlights the economic burden of integrating multiple sensors in cost-sensitive industries like automotive. Finally, the necessity of independent validation tests under standards such as ISO 26262 further complicates the use of physical sensors alone, motivating the need for estimation models.

Data-driven models that incorporate physical knowledge have shown great potential in this domain. A thermal neural network (TNN) is one such method. In [5], the authors showed that this method outperformed all existing models suitable for real-time temperature estimation.

However, these data-driven models have a critical problem, namely, their unexplainability. This makes their use in safety-critical applications quite difficult. This study aims to address this problem by quantifying the uncertainty in data-driven motor drive temperature estimation models. It focuses on TNN in particular because of its accuracy and gray-box nature that allows for a limited degree of explainability. This work investigates five uncertainty quantification methods, integrates them into the TNN, and rigorously evaluates their estimates.

This work is divided into three main sections. In the fundamentals section, a brief overview of temperature estimation models is provided, followed by an introduction to uncertainty quantification (UQ) and the methods used for UQ and calibration.

In the implementation section, a detailed overview of how the UQ methods were integrated into the TNN is provided, along with an explanation of the novel branched architecture. It also summarizes the hyperparameter optimization that was performed for the UQ methods discussed in this work.

Finally, the results are presented, including plots that visualize model performance.

2 Fundamentals

2.1 Thermal modeling of PMSMs

Historically, computational fluid dynamics (CFD) and finite element analysis (FEA) have been used to estimate temperatures, but these methods are too slow for online monitoring and control(cf Fig. 2.1).

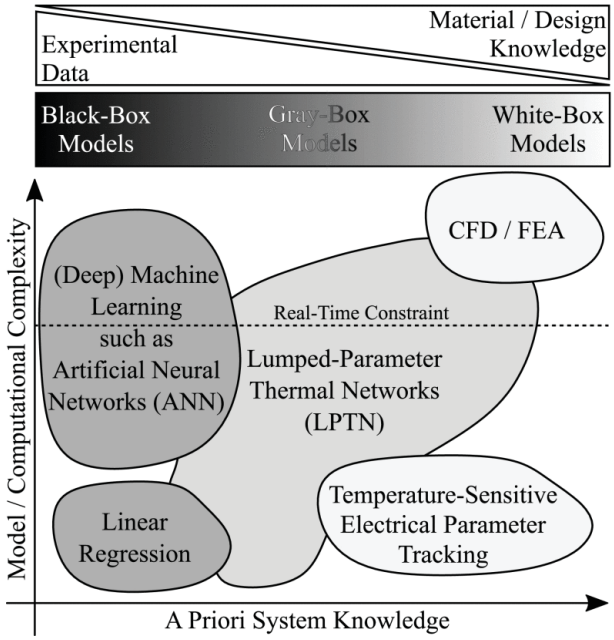

Fig. 2.1: Overview of temperature estimation models (taken from [4])

As per [1], online temperature estimation models for PMSMs can be classified into 3 classes.

- Thermal-based models: These classes of models use thermal properties of materials and fundamental heat theory to extrapolate the temperatures from temperature measurements at easily accessible locations. Lumped-parameter thermal networks (LPTNs) can be included in this section.

- Electric-based models: These models try to estimate winding resistance and permanent magnet flux to estimate the temperature using electrical equations. Several challenges arise when estimating temperatures using these models, including discrepancies between actual and reference voltages due to inverter nonlinearities [1].
- Data-driven models: Modern systems can collect large amounts of data, including operating speed, torque, reference voltage, and current. Data-driven models try to estimate temperature from these measurements. These methods could be completely black-box, such as neural networks, or they could integrate the structure from electrical or thermal-based models, yielding gray-box models. A current overview of data-driven methods is available in [1]. These models can model nonlinearities in the system and address approximation errors associated with other modeling methods. Unexplainability remains a key weakness for this class of models.

The following sections will first introduce LPTNs and TNNs. Subsequent sections will then focus on UQ methods in data-driven methods.

2.1.1 Lumped Parameter Thermal Network

LPTN is a thermal-based modeling technique for temperature estimation in complex systems. It exploits the electrical analog of thermal components, i.e., certain thermal elements behave like electrical components. For e.g., elements that store heat can be thought of behaving like a capacitor. Temperature can be modeled as voltage, since a temperature gradient drives heat flow. Heat flow can be modeled as a current. Thermal resistance becomes an electrical resistance. The thermal behavior of the system is then described through energy balance equations between these components. Fig 2.2 shows

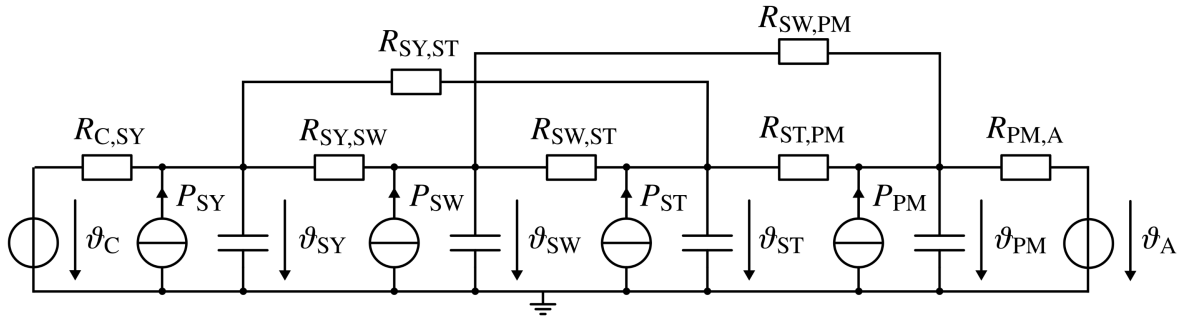

Fig. 2.2: Thermal equivalent circuit of a PMSM including nodes for coolant and ambient temperature (taken from [3]).

a thermal equivalent circuit of a PMSM that is taken from [3]. Values of the thermal resistances, power losses, and capacitance are generally unknown and must be estimated. These estimates require material and geometric data and sometimes use approximations for geometric shapes. After accounting for boundary conditions, material and geometrical data, and other approximations, the dynamics of the thermal equivalent circuit can be

written as a system of ordinary differential equations, as shown below

$$\boldsymbol{C}_i(\boldsymbol{\zeta}(t))\frac{\mathrm{d}\boldsymbol{\vartheta}_i}{\mathrm{d}t} = \boldsymbol{\pi}_i(\boldsymbol{\zeta}(t)) + \sum_{j\in\mathcal{M}\setminus i}\frac{\boldsymbol{\vartheta}_j - \boldsymbol{\vartheta}_i}{\boldsymbol{R}_{i,j}(\boldsymbol{\zeta}(t))} + \sum_{j=1}^{n}\frac{\tilde{\boldsymbol{\vartheta}}_j - \boldsymbol{\vartheta}_i}{\boldsymbol{R}_{i,j}(\boldsymbol{\zeta}(t))}. \tag{2.1}$$

Here $\boldsymbol{C}$,$\boldsymbol{\pi}$,$\boldsymbol{R}$, and $\boldsymbol{\vartheta}$ represent thermal capacitance, power losses, resistance, and temperature, respectively. Temperatures that are measured during operation are referred to as ancillary temperatures, and are denoted by $\tilde{\boldsymbol{\vartheta}}$. Due to the approximations involved in deriving Eq. 2.1, parameters vary not just with time but also vary with operating conditions. $\boldsymbol{\zeta}(t)$, known as the scheduling vector, is used to denote this property of these parameters.

LPTNs can be classified into dark gray-box, light gray-box, and white-box depending on the level of abstraction of the components. White-box LPTNs are not suitable for online temperature monitoring in PMSMs due to computational demands. Gray-box and black LPTNs remain a contender for online temperature estimation. A detailed overview of LPTNs is available in [3].

2.1.2 Thermal Neural Network

TNN is a data-driven method inspired by gray-box LPTN. It uses neural networks to estimate the LPTN parameters. This enables temperature estimation without material information or geometric approximations. Unlike LPTN, TNN can learn to approximate parameters across a wide range of operating conditions with a single model, provided these conditions are sufficiently captured in the training data. TNNs were introduced in [5]. Eq. 2.1 can be rewritten using first-order Euler discretization as

$$\hat{\vartheta}_i[k+1] = \hat{\vartheta}_i[k] + T_s\kappa_i[k]\left(\pi_i[k] + \sum_{j\in\mathcal{M}\setminus i}(\hat{\vartheta}_j[k] - \hat{\vartheta}_i[k])\gamma_{i,j}[k] + \sum_{j=1}^{n}(\tilde{\vartheta}_j[k] - \hat{\vartheta}_i[k])\gamma_{i,j}[k]\right). \tag{2.2}$$

T_s denotes sampling time. Thermal resistance R has been replaced by thermal conductance γ for numerical stability, $\frac{1}{R} = \gamma$. Similarly, thermal capacitance C has been replaced by inverse capacitance, $\frac{1}{C} = \kappa$. Eq. 2.2, can be rewritten in matrix form as follows,

$$\hat{\boldsymbol{\vartheta}}[k+1] = \hat{\boldsymbol{\vartheta}}[k] + T_\mathrm{s}\boldsymbol{\kappa}[k] \odot \left(\boldsymbol{\pi}[k] + \left(\left(\mathbf{1}_{[\mathcal{M}]}\cdot\hat{\boldsymbol{\vartheta}}_\mathrm{ext}[k]^\top - \hat{\boldsymbol{\vartheta}}[k]\cdot\mathbf{1}^\top_{[\mathcal{M}+n]}\right)\odot\boldsymbol{G}[k]_{\{1\ldots\mathcal{M}\}}\right)\cdot\mathbf{1}_{[\mathcal{M}+n]}\right). \tag{2.3}$$

Here, $\mathcal{M}$ denotes the number of nodes being modeled, n denotes the number of ancillary temperatures being available, $\mathbf{1}_a$ denotes a row matrix of ones with dimensions (1, a), $\hat{\boldsymbol{\vartheta}}_\mathrm{ext} = \begin{bmatrix}\hat{\boldsymbol{\vartheta}}^T & \tilde{\boldsymbol{\vartheta}}^T\end{bmatrix}^T$ and $\odot$ represents element-wise multiplication. $\boldsymbol{G}$ denotes thermal conductance adjacency matrix such that

$$\boldsymbol{G} = \begin{bmatrix} 0 & \gamma_{1,2} & \gamma_{1,3} & \cdots & \gamma_{1,|\mathcal{M}|+n} \\ \gamma_{1,2} & 0 & \gamma_{2,3} & \cdots & \gamma_{2,|\mathcal{M}|+n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{1,|\mathcal{M}|+n-1} & \gamma_{2,|\mathcal{M}|+n-1} & \gamma_{3,|\mathcal{M}|+n-1} & \cdots & \gamma_{|\mathcal{M}|+n-1,|\mathcal{M}|+n} \\ \gamma_{1,|\mathcal{M}|+n} & \gamma_{2,|\mathcal{M}|+n} & \gamma_{3,|\mathcal{M}|+n} & \cdots & 0 \end{bmatrix}. \tag{2.4}$$

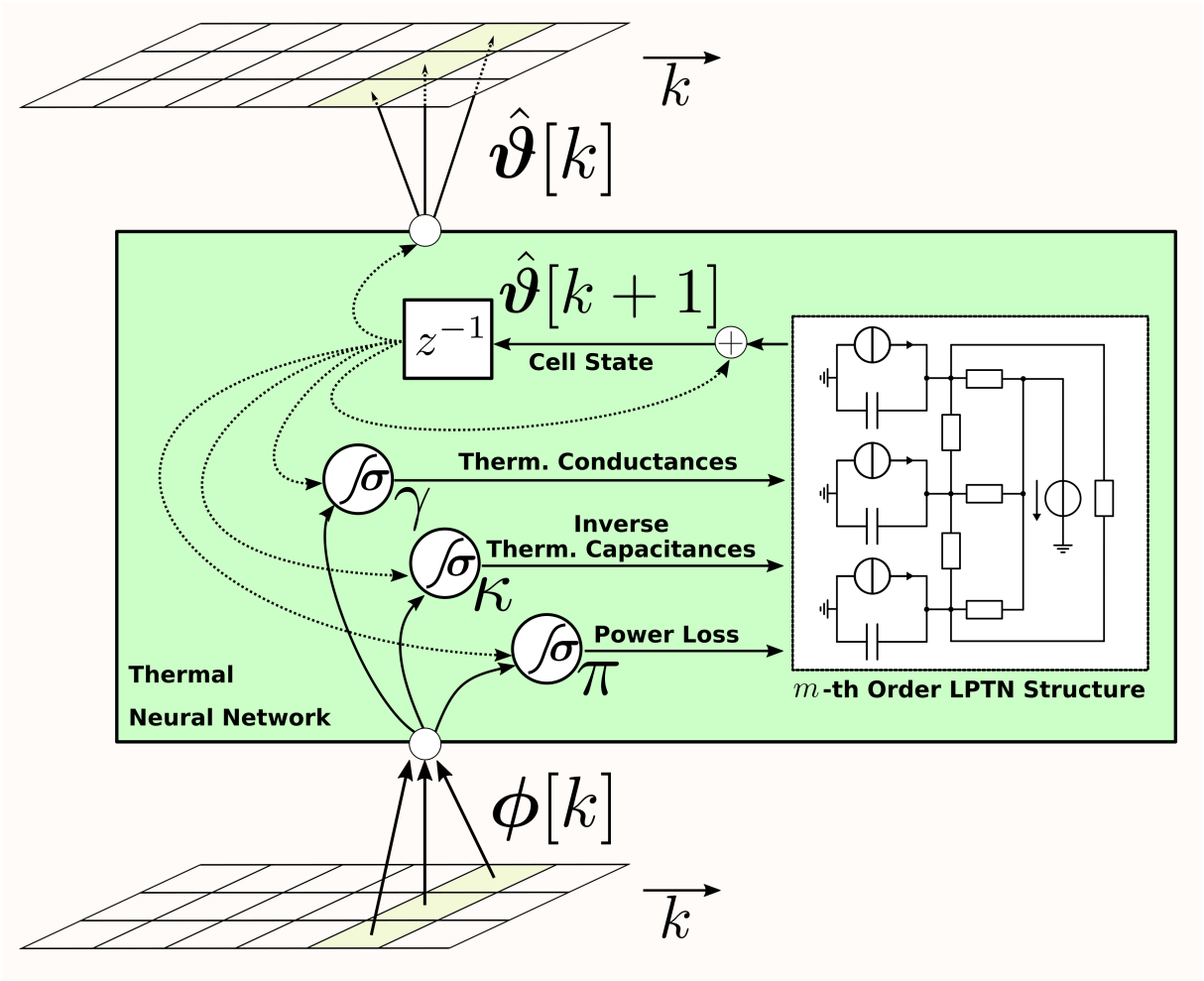

Fig. 2.3: TNNs as defined by [5]. Here $\boldsymbol{\gamma}$ indicates thermal capacitances $\boldsymbol{\pi}$ indicates power losses, $\boldsymbol{\kappa}$ indicates inverse capacitances and $\boldsymbol{\vartheta}$ indicates the temperatures.

G is time-varying; Eq. 2.4 is one such realization. In TNNs, neural networks are used to estimate equivalent power losses and thermal conductance. The inverse thermal capacitance is treated as a learned parameter for simplification. Data-driven TNNs offer high accuracy and low computational overhead. But they are black-box models, incorporating UQ is essential in these models for safety-critical applications to provide a measure of confidence in the model's predictions, especially when operating in transient or edge-case conditions where thermal runaway is a risk. By comparing different UQ methods, this study seeks to enhance the reliability and interpretability of TNNs.

2.2 Uncertainty Quantification

In real-world modeling and data acquisition, unknowns are inevitable. These range from microscopic sensor defects to the numerous approximations inherent in the modeling pro-

cess itself. Such unknowns result in prediction errors, while these may be tolerable in certain fields, safety-critical applications necessitate that they are rigorously quantified. This discipline, known as UQ, provides a mathematical framework to transform these unknowns into measurable, actionable confidence intervals. Historically, UQ has focused on uncertainty propagation, analyzing how input variabilities, such as sensor tolerances, manifest in the output of a system (e.g., a Finite Element Analysis model). This method range includes Monte-Carlo simulations, unscented transforms, polynomial chaos expansion, etc. An overview of some of these methods can be found in [6]. Even though [6] focuses on uncertainty in orbital mechanics, these methods are generalizable. Uncertainties are classified into two categories: aleatoric and epistemic. Aleatoric uncertainty is

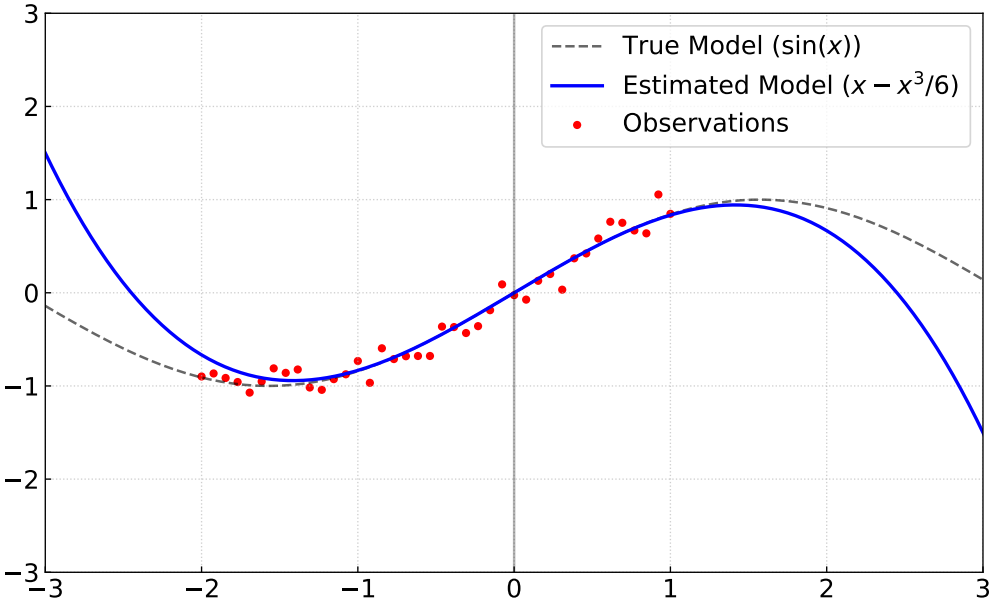

Fig. 2.4: The true underlying data generation model is $sin(x)$, but the observations shown here by red dots are noisy. The predictive model is a third-order Taylor expansion of $sin(x)$. Here, epistemic uncertainty is 0 near the origin but explodes away from it. Aleatoric uncertainty remains the same everywhere.

defined as inaccuracies in predictions arising from uncertainty in the data acquisition process. These could be due to sensor noise or environmental factors such as moisture or vibrations. It is informally referred to as *data uncertainty*. In thermal modeling, this typically stems from stochastic sources such as sensor noise, measurement tolerances, or environmental fluctuations that arise during data collection. This type of uncertainty is irreducible without using lower-tolerance sensors or controlling the environment. Collecting more data using the same sensors under the same conditions will not reduce it.

Epistemic uncertainty is defined as inaccuracies in predictions due to assumptions made during the modeling process. It could include a vast array of factors, such as poor model choice (e.g., using a linear model on non-linear data), poor input parameter choices, insufficient data, etc. It is informally referred to as *model uncertainty*. Unlike aleatoric uncertainty, epistemic uncertainty is reducible. It can be reduced by providing the model with more diverse training data or by refining the modeling architecture.

The difference between them can be visually seen in Fig. 2.4.

2.2.1 Overview of uncertainty quantification methods

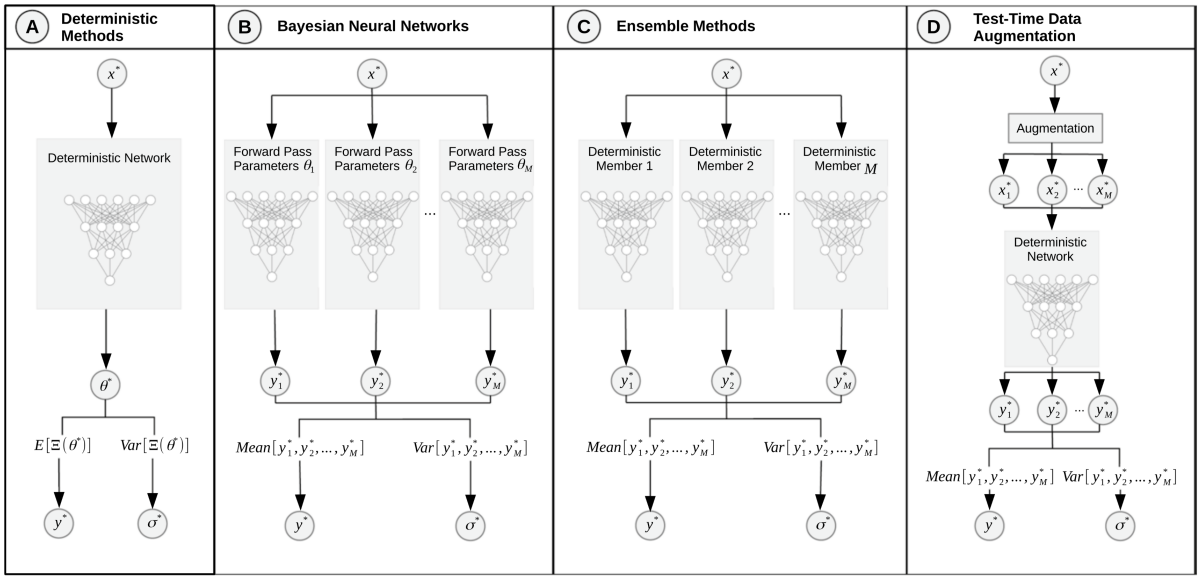

Fig. 2.5: Classification of UQ methods in neural networks (taken from [7])

Fig 2.5 shows the classification of UQ methods for neural networks as presented in [7]. The authors of [7] classified them into the following categories.

- Deterministic methods use fixed weights and attempt to estimate uncertainty with a single forward pass. These methods were historically suitable only for aleatoric uncertainties, but new methods, such as evidential deep learning, demonstrate their suitability for epistemic uncertainty as well.
- In Bayesian neural networks(BNNs), instead of using deterministic weights, weights themselves are probabilistic. By marginalizing over their distributions, the model captures epistemic uncertainty. They are introduced in Sec. 2.5.
- In ensemble methods, multiple deterministic methods are combined, and the variance between them is used for estimating the uncertainty. They are introduced in Sec. 2.6.

- Test-time augmented methods(TTA) methods estimate uncertainty by passing several modified versions of the same input (like rotated or flipped images) through a trained network. If the outputs vary significantly, the model is considered uncertain.

There are also other methods that are not strictly limited to neural networks and can be used for UQ in TNN, such as Gaussian processes (GP). GP are introduced in Sec. 2.4. In this work, GP and one representative method from each of the first three classes are compared. TTA methods are excluded from this comparative analysis due to time constraints and their lower effectiveness for regression-related tasks.

2.3 Scoring Rules

Scoring rules are used to evaluate the fit of predictive distributions to deterministic data. A scoring rule is considered proper if it is minimized when the predictive distribution is equal to the true distribution. When there exists only a unique value that minimizes the score, it is called a strictly proper metric [8]. In this work, CRPS(Continuous ranked probability score) and NLL(Negative log likelihood) are considered. Both are strictly proper scoring rules.

2.3.1 Negative log likelihood

NLL comes from Bayesian decision theory, where the goal is to find the optimal parameters of a probabilistic distribution that maximizes the likelihood of the observed input data $\boldsymbol{x}$. Input data is assumed to be independent and identically distributed (i.i.d).

$$p(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \mid \boldsymbol{\theta}_{LL}) = \prod_{i=1}^{n} p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}_{LL}). \tag{2.5}$$

Here, $\boldsymbol{\theta}_{LL}$ indicates the parameters of the assumed probabilistic distribution and $\boldsymbol{x}^{(i)}$ indicates the i-th sample in the input set. Equation (2.5) represents the likelihood of observing the data. To separate out the multiplicative term, a logarithm of the likelihood is used. For numerical reasons, likelihood is minimized, therefore, a negative sign is added.

$$\text{NLL} = -\sum_{i=1}^{n} \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}_{LL}). \tag{2.6}$$

For a single-dimensional normally distributed data,

$$NLL_{normal} = \sum_{i=1}^{n} \left[\frac{1}{2} \log(2\pi\hat{\sigma}_i^2) + \frac{(y^{(i)} - \hat{\mu}_i)^2}{2\hat{\sigma}_i^2} \right], \tag{2.7}$$

where $\hat{\mu}_i$ indicates predicted mean and $\hat{\sigma}_i$ indicates predicted standard deviation for the i-th sample.

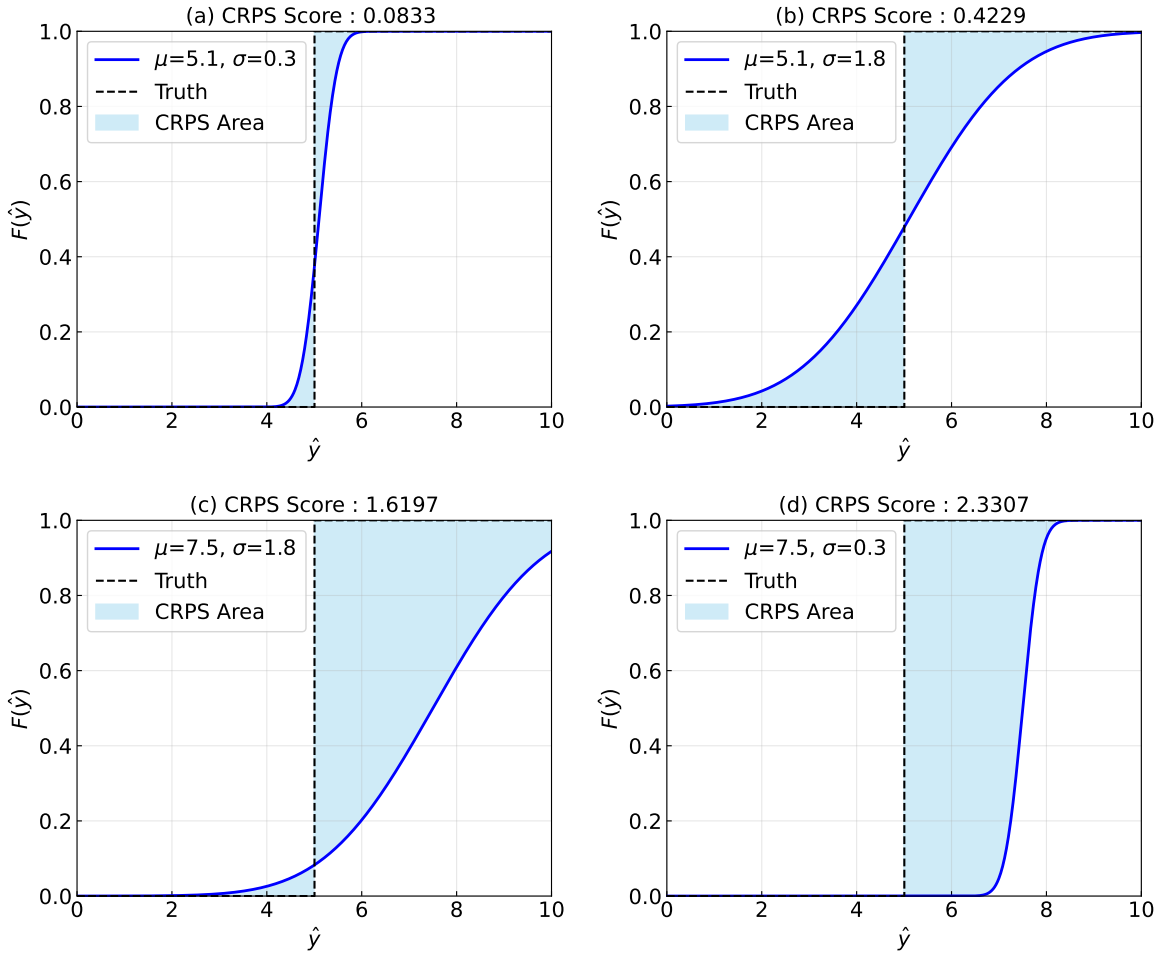

Fig. 2.6: This illustration of CRPS assumes the predictive distribution to be normal. Ground truth is $y^{(i)} = 5$. In figure (a), $\hat{y}^{(i)} = 5.1, \hat{\sigma} = 0.3$, there is low error and low uncertainty. This is the ideal case. In figure (b), $\hat{y}^{(i)} = 5.1, \hat{\sigma} = 1.8$, there is low error but high uncertainty. Predictions are punished for being underconfident. In figure (c), $\hat{y}^{(i)} = 7.5, \hat{\sigma} = 1.8$, there is high error and high uncertainty. Here, the model is being punished for being inaccurate. In figure (d), $\hat{y}^{(i)} = 7.5, \hat{\sigma} = 0.3$, there is high error but low uncertainty. Here, the model is being punished for being overconfident.

2.3.2 Continuous Ranked Probability Score

CRPS was first defined in [9], [10]. Since then, it has been used for weather forecasting. With an increased focus on probabilistic machine learning, CRPS is being adopted more widely. Mathematically,

$$\text{CRPS}(F_i, y^{(i)}) = \int_{-\infty}^{\infty} [F_i(\hat{y}) - H(\hat{y} - y^{(i)})]^2 d\hat{y}. \tag{2.8}$$

here $F_i(.)$ is the cumulative distribution function at i-th prediction and H(.) is the Heaviside function.

$$H(\hat{y} - y^{(i)}) = \begin{cases} 0 & \text{if } \hat{y} - y^{(i)} < 0 \\ 1 & \text{if } \hat{y} - y^{(i)} \geq 0 \end{cases}. \tag{2.9}$$

Fig. 2.6 gives a graphical overview of CRPS. The lowest possible value for CRPS can be 0. For NLL, there is no minimum value. CRPS and NLL can be considered as extensions of maximum absolute error and mean squared error, respectively, in probabilistic modeling [8], [11].

2.4 Gaussian Processes

GP provides a probabilistic framework for regression that naturally incorporates uncertainty in predictions. Before introducing GP regression (GPR), it is helpful to first discuss Bayesian regression, which serves as a conceptual and mathematical foundation for understanding GP.

2.4.1 Bayesian regression with Gaussian prior

Given a dataset

$$\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\}, \tag{2.10}$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ denotes the input vector and $y^{(i)} \in \mathbb{R}$ the corresponding scalar output. Observations are assumed to be corrupted by additive Gaussian noise ε such that

$$y^{(i)} = f(\boldsymbol{x}^{(i)}) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.11}$$

Here, $\mathcal{N}$ denotes a Gaussian distribution and σ^2 denotes measurement noise variance. The discussion first focuses on linear regression models. A linear function can be represented as

$$f(\boldsymbol{x}^{(i)}) = w_1 + \boldsymbol{w}_2^\top \boldsymbol{x}^{(i)}. \tag{2.12}$$

where $w_1 \in \mathbb{R}$ is a bias term and $\boldsymbol{w}_2 \in \mathbb{R}^d$ is a vector of weights. Augmented feature vectors are defined as

$$\boldsymbol{\phi}(\boldsymbol{x}^{(i)}) = \begin{bmatrix} 1 \\ \boldsymbol{x}^{(i)} \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} w_1 \\ \boldsymbol{w}_2 \end{bmatrix}. \tag{2.13}$$

With this notation, the model can be written compactly as

$$f(\boldsymbol{x}^{(i)}) = \boldsymbol{\phi}(\boldsymbol{x}^{(i)})^\top \boldsymbol{w}. \tag{2.14}$$

Let the design matrix $\boldsymbol{\Phi} \in \mathbb{R}^{n \times (d+1)}$ be defined as

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}^{(1)})^\top \\ \boldsymbol{\phi}(\boldsymbol{x}^{(2)})^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}^{(n)})^\top \end{bmatrix}. \tag{2.15}$$

and let output y be

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}. \tag{2.16}$$

The observation model for the full dataset can then be written as

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma}^2 \boldsymbol{I}_n). \tag{2.17}$$

Here, $\boldsymbol{I}_n$ denotes an identity matrix of size n. A Gaussian prior is placed over the weights:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w). \tag{2.18}$$

Here, $\boldsymbol{\mu}_w$ and $\boldsymbol{\Sigma}_w$ denote the mean and the covariance matrix of the weights, respectively. Conditioned on the weights $\boldsymbol{w}$, the likelihood of the observed targets is given by

$$p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\Phi}\boldsymbol{w}, \sigma^2 \boldsymbol{I}_n). \tag{2.19}$$

Using Bayes' theorem, the posterior distribution over the weights is

$$p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{x}) \propto p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{x})\, p(\boldsymbol{w}). \tag{2.20}$$

Substituting the expressions for the likelihood and the prior yields

$$p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{x}) \propto \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\Phi}\boldsymbol{w}, \sigma^2 \boldsymbol{I}_n)\, \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w). \tag{2.21}$$

Since both the prior and the likelihood are Gaussian and the model is linear in $\boldsymbol{w}$, The posterior distribution is also Gaussian:

$$p(\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}). \tag{2.22}$$

The posterior mean and covariance can be obtained in closed form and are given by

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\boldsymbol{\Sigma}_w^{-1} + \frac{1}{\sigma^2} \boldsymbol{\Phi}^\top \boldsymbol{\Phi} \right)^{-1}, \tag{2.23}$$

$$\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}} \left(\boldsymbol{\Sigma}_w^{-1} \boldsymbol{\mu}_w + \frac{1}{\sigma^2} \boldsymbol{\Phi}^\top \boldsymbol{y} \right). \tag{2.24}$$

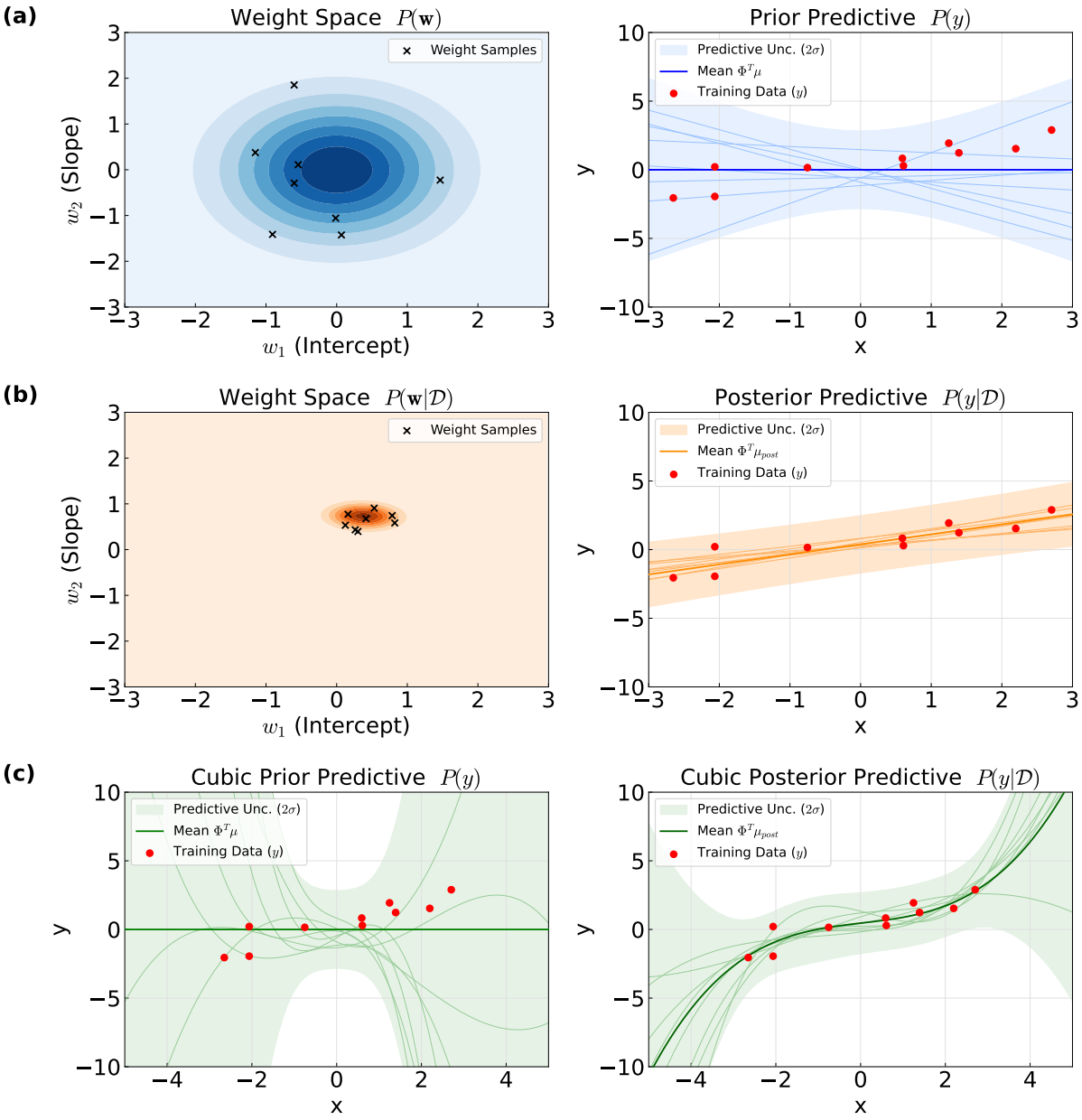

Fig. 2.7: Illustration of Bayesian linear regression on a synthetic dataset. The data is generated according to $y = 0.5\,x + 1.0 + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, 1)$, where the inputs x are sampled uniformly from the interval $[-1, 1]$. Panel (a) shows the prior distribution over the weights, $\mathcal{N}([0,\, 0]^\top, \boldsymbol{I}_2)$, visualized in both weight space and function space. Panel (b) shows the posterior distribution obtained after conditioning on the observed data, using the posterior mean and covariance defined in Equations (2.23) and (2.24) on and linear mapping. In Figure (c), the prior and posterior function spaces for the same dataset, using a cubic mapping, are shown. These illustrations are inspired by the lecture series of Universität Tübingen [12].

The Bayesian linear regression framework is not restricted to linear functions of the input. The model allows for nonlinear dependence on the input through a suitable choice of basis functions.

As an example, a regression model of the following form can be considered.

$$f(\boldsymbol{x}) = w_1 + \boldsymbol{w}_2\boldsymbol{x} + \boldsymbol{w}_3 sin(\boldsymbol{x}) + \boldsymbol{w}_4 cos(\boldsymbol{x}). \tag{2.25}$$

Defining the feature vector as

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} 1 \\ \boldsymbol{x} \\ sin(\boldsymbol{x}) \\ cos(\boldsymbol{x}) \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} w_1 \\ \boldsymbol{w_2} \\ \boldsymbol{w_3} \\ \boldsymbol{w_4} \end{bmatrix}. \tag{2.26}$$

In case of $\boldsymbol{x}$ being multidimensional, operations are done element-wise. The model can be written compactly as

$$f(x) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{w}. \tag{2.27}$$

Despite the nonlinear dependence on the input x, the model remains linear in the weights $\boldsymbol{w}$. Consequently, the posterior distribution over the weights is still Gaussian, and the expressions for the posterior mean and covariance given in equations (2.23) and (2.24) remain valid.

2.4.2 Kernel Trick

As seen in section 2.4.1, $\boldsymbol{\Phi}$ maps an input $\boldsymbol{x}$ into a higher-dimensional feature space. It is also possible to choose a feature map whose output has infinitely many dimensions. For example, one can define $\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} 1 \\ 1/\boldsymbol{x} \\ 1/\boldsymbol{x}^2 \\ \vdots \end{bmatrix}$. The kernel trick allows one to work with such infinite-dimensional objects. Working with an infinite-dimensional feature space allows GP models to represent a very rich class of functions, making them universal function approximators. Equations (2.23) and (2.24) for the posterior mean and covariance matrix can be rewritten using the matrix inverse lemma identity [13]. The equations then become:

$$\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\Sigma}_w - \boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I}_n\right)^{-1} \boldsymbol{\Phi}\boldsymbol{\Sigma}_w. \tag{2.28}$$

$$\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\mu}_w + \boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I}_n\right)^{-1} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}_w). \tag{2.29}$$

The kernel trick is then used to compute the value of $\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top$ directly, instead of keeping the individual matrices in memory. The derivation uses the matrix inverse lemma identity and algebraic manipulation. A full detailed derivation is available in [14]. It defines a kernel (covariance) function $k(\boldsymbol{x}, \boldsymbol{x}')$ such that

$$k(\boldsymbol{x}, \boldsymbol{x}') \;=\; \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle_{\boldsymbol{\Sigma}_w} \;=\; \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\Sigma}_w \boldsymbol{\phi}(\boldsymbol{x}'). \tag{2.30}$$

Here, $k(\boldsymbol{x}, \boldsymbol{x}')$ denotes the i,j-th element of the hence obtained kernel matrix, whereas $\boldsymbol{x}$ denotes the i-th sample and $\boldsymbol{x}'$ denotes the j-th sample. In GP literature, it is also common practice not to include $\boldsymbol{\Sigma_w}$, as it is often assumed to be the identity matrix of appropriate size, with prior information already encoded in the kernel function. GP can be thought of as using Bayesian regression for a Gaussian prior and defining the covariance matrix using a kernel function. A GP is specified by a mean function and the corresponding kernel function.

$$f_{gp}(\boldsymbol{x}) \sim \mathcal{GP}\Big(m(\boldsymbol{x}),\, k(\boldsymbol{x}, \boldsymbol{x}')\Big). \tag{2.31}$$

Here, $m(\boldsymbol{x})$ is the mean function and $\mathcal{GP}$ represents GP. It represents the modeled function's baseline behavior in the absence of the data, while the kernel controls how the function varies around that baseline and how values at different inputs are correlated. In principle, $m(\boldsymbol{x})$ can be any function of $\boldsymbol{x}$, for example, a constant, a linear trend, or a parametric model.

$$m(\boldsymbol{x}) = \boldsymbol{c}, \qquad m(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} + \boldsymbol{b}, \qquad m(x) = g(x; \boldsymbol{\theta}). \tag{2.32}$$

Here, $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ represent arbitrary matrices of appropriate dimensions, and $\boldsymbol{\theta}$ represents the model parameters when the mean function is defined parametrically. However, in many applications, a complex mean function is unnecessary. It is common to use a simple choice such as $m(\boldsymbol{x}) = 0$ (often after centering the targets), and let the kernel capture the required variation and structure in the data. A convenient decomposition is

$$f_{gp}(\boldsymbol{x}) = m(\boldsymbol{x}) + g(\boldsymbol{x}), \qquad g(\boldsymbol{x}) \sim \mathcal{GP}(0, \boldsymbol{K}). \tag{2.33}$$

where $m(\boldsymbol{x})$ models any known global trend and the GP term $g(\boldsymbol{x})$ models the remaining structured deviations. Given inputs $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$, the kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is defined by $K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$. Computing all entries of $\boldsymbol{K}$ typically costs $O(n^2)$ kernel evaluations. Total cost also depends on the cost of evaluating k. Many common kernel choices and their function space are summarized in Fig 2.8. A detailed overview of kernels is available in [14]. A function $k(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel if, for any finite set of inputs $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$, the kernel matrix $\boldsymbol{K}$ with entries K_{ij} is a valid covariance matrix. Therefore, $\boldsymbol{K}$ must be symmetric and positive semi-definite, i.e.,

$$\boldsymbol{K} = \boldsymbol{K}^\top \quad \text{and} \quad \boldsymbol{v}^\top \boldsymbol{K} \boldsymbol{v} \geq 0 \;\text{ for all } \boldsymbol{v} \in \mathbb{R}^n. \tag{2.34}$$

Multiple kernels can be combined to capture different characteristics of the data. For example, if one expects a periodic component and a general trend otherwise, two separate kernels can be used. A common approach is to add kernels:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}'). \tag{2.35}$$

In a GP, this corresponds to modeling the function as a sum of independent latent functions,

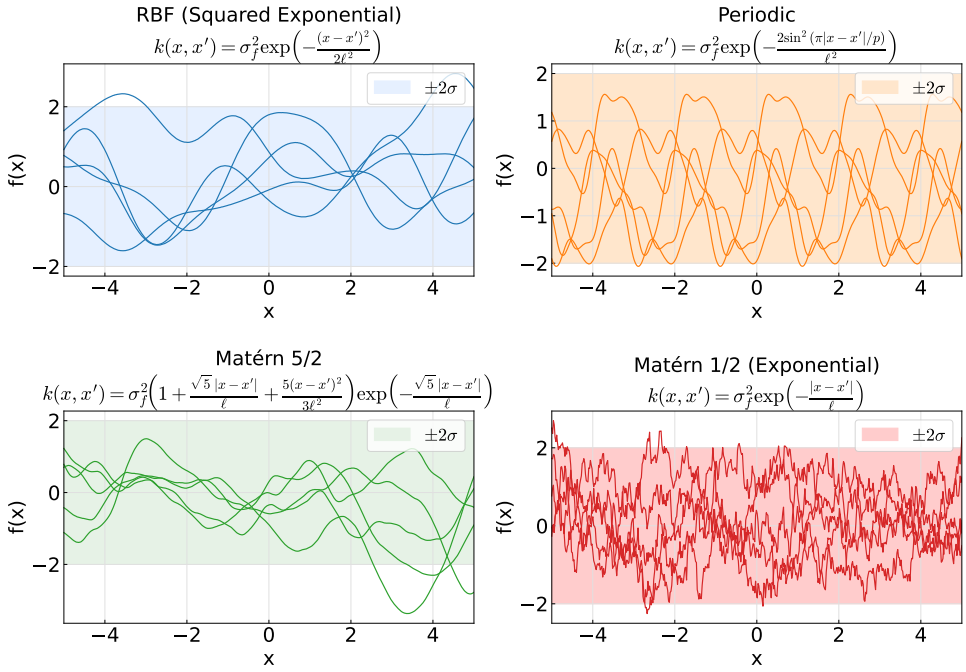

Fig. 2.8: Illustration of common kernels and their associated function space with 95% intervals(2σ for Gaussian), here l indicates length-scale, a hyperparameter for kernels.

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) + g_2(\boldsymbol{x}), \tag{2.36}$$

where $g_1 \sim \mathcal{GP}(0, k_1)$ and $g_2 \sim \mathcal{GP}(0, k_2)$. For this reason, kernel addition is often described informally as an "OR" operation: either component can explain structure in the data. For example, if the data is believed to contain short-term, rough spike-like behavior but is otherwise smoothly varying, a sum of a Matérn $\nu = 1/2$ kernel and a Matérn $\nu = 5/2$ kernel can be used:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_{\text{Matérn-}\nu=1/2}(\boldsymbol{x}, \boldsymbol{x}') + k_{\text{Matérn-}\nu=5/2}(\boldsymbol{x}, \boldsymbol{x}'). \tag{2.37}$$

This lets the GP capture both local irregularities and a smooth underlying trend. Kernels can also be combined by multiplication:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')\, k_2(\boldsymbol{x}, \boldsymbol{x}'). \tag{2.38}$$

This is sometimes described informally as an "AND" operation: two inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$ have high covariance only if both kernels assign them high similarity. In other words, if either $k_1(\boldsymbol{x}, \boldsymbol{x}')$ or $k_2(\boldsymbol{x}, \boldsymbol{x}')$ becomes small, the product kernel and hence the covariance also becomes small. Multiplication is useful for expressing interactions or modulation between structures. For example, multiplying a periodic kernel by a radial-basis function(RBF) kernel produces a locally periodic kernel. This function exhibits periodic behavior, but correlations between points far apart decay, so the periodicity is only maintained over a limited range:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k_{\text{Per}}(\boldsymbol{x}, \boldsymbol{x}')\, k_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{x}'). \tag{2.39}$$

Under the GP prior $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, the vector of function values at any finite set of inputs is jointly Gaussian. In particular, for training inputs $\boldsymbol{x}_{\text{train}} = [\boldsymbol{x}_{\text{train}}^{(1)}, \ldots, \boldsymbol{x}_{\text{train}}^{(n)}]^\top$ and test inputs $\boldsymbol{x}_{\text{test}} = [\boldsymbol{x}_{\text{test}}^{(1)}, \ldots, \boldsymbol{x}_{\text{test}}^{(m)}]^\top$,

$$\boldsymbol{f}_{\text{train}} = f(\boldsymbol{x}_{\text{train}}), \qquad \boldsymbol{f}_{\text{test}} = f(\boldsymbol{x}_{\text{test}}). \tag{2.40}$$

The kernel blocks are defined as

$$\boldsymbol{K}_{\text{train}} = k(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}}), \qquad \boldsymbol{K}_{\text{train,test}} = k(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{test}}), \qquad \boldsymbol{K}_{\text{test}} = k(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_{\text{test}}). \tag{2.41}$$

where, $[\boldsymbol{K}_{\text{train}}]_{ij} = k(\boldsymbol{x}_{\text{train}}^{(i)}, \boldsymbol{x}_{\text{train}}^{(j)})$ and so on for other terms. Then

$$\begin{bmatrix} \boldsymbol{f}_{\text{train}} \\ \boldsymbol{f}_{\text{test}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_{\text{train}}) \\ m(\boldsymbol{x}_{\text{test}}) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{\text{train}} & \boldsymbol{K}_{\text{train,test}} \\ \boldsymbol{K}_{\text{train,test}}^\top & \boldsymbol{K}_{\text{test}} \end{bmatrix} \right). \tag{2.42}$$

If noisy targets $\boldsymbol{y}_{train} = \boldsymbol{f}_{\text{train}} + \boldsymbol{\varepsilon}$ with i.i.d. Gaussian noise are observed,

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_n), \tag{2.43}$$

then the joint distribution of $(\boldsymbol{y}_{test}, \boldsymbol{f}_{\text{test}})$ is

$$\begin{bmatrix} \boldsymbol{y}_{train} \\ \boldsymbol{f}_{\text{test}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x}_{\text{train}}) \\ m(\boldsymbol{x}_{\text{test}}) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{\text{train}} + \sigma^2 \boldsymbol{I}_n & \boldsymbol{K}_{\text{train,test}} \\ \boldsymbol{K}_{\text{train,test}}^\top & \boldsymbol{K}_{\text{test}} \end{bmatrix} \right). \tag{2.44}$$

Conditioning this joint Gaussian yields the GP posterior (predictive) mean and covariance:

$$p(\boldsymbol{f}_{\text{test}} \mid \boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{train}, \boldsymbol{x}_{\text{test}}) = \mathcal{N}(\hat{\boldsymbol{\mu}}_{\text{test}}, \hat{\boldsymbol{\Sigma}}_{\text{test}}). \tag{2.45}$$

$$\hat{\boldsymbol{\mu}}_{\text{test}} = m(\boldsymbol{x}_{\text{test}}) + \boldsymbol{K}_{\text{train,test}}^\top (\boldsymbol{K}_{\text{train}} + \sigma^2 \boldsymbol{I}_n)^{-1} (\boldsymbol{y}_{train} - m(\boldsymbol{x}_{\text{train}})). \tag{2.46}$$

$$\hat{\boldsymbol{\Sigma}}_{\text{test}} = \boldsymbol{K}_{\text{test}} - \boldsymbol{K}_{\text{train,test}}^\top (\boldsymbol{K}_{\text{train}} + \sigma^2 \boldsymbol{I}_n)^{-1} \boldsymbol{K}_{\text{train,test}}. \tag{2.47}$$

Here, σ is typically learned through data using maximum likelihood or similar estimation methods. It can also be used directly if sensor noise is known. For the predictive distribution of noisy observations $\boldsymbol{y}_{\text{test}}$ at $\boldsymbol{x}_{\text{test}}$:

$$p(\hat{\boldsymbol{y}}_{\text{test}} \mid \boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{train}, \boldsymbol{x}_{\text{test}}) = \mathcal{N}(\hat{\boldsymbol{\mu}}_{\text{test}}, \hat{\boldsymbol{\Sigma}}_{\text{test}} + \sigma^2 \boldsymbol{I}_n). \tag{2.48}$$

The posterior equations also reveal the main computational bottleneck of GP regression. The matrix $\boldsymbol{K}_{\text{train}} + \sigma^2 \boldsymbol{I}_n \in \mathbb{R}^{n \times n}$ is typically dense, and computing its inverse is computationally expensive. The time complexity for inverse is $\mathcal{O}(n^3)$ and the memory cost is $\mathcal{O}(n^2)$.

As a result, exact GP inference becomes computationally infeasible for large datasets. In [15], the authors demonstrated an approach to use an exact GP on a larger dataset, but it required significant computational resources (8 V100 GPUs). Some solutions work with low-dimensional data, such as state-space GP[16] or by projecting high-dimensional data

onto lower-dimensional ones, as shown in [17]. There are a lot of other solutions for approximating the posteriors, such as KISS-GP[18], stochastic variational GP (SVGP)[19], vecchia approximations [20], etc. The most famous and most commonly used approximation approach is SVGP.

In this work, SVGP is chosen over other approximations for its numerical stability and support in standard libraries. SVGP is used for residual modeling, to quantify uncertainty and capture systematic discrepancies in a baseline TNN. Concretely,

$$\hat{y}(\boldsymbol{x}) = \hat{y}_{\text{TNN}}(\boldsymbol{x}) + r(\boldsymbol{x}), \qquad r(\boldsymbol{x}) \sim \mathcal{GP}\Big(0, k_{residual}(\boldsymbol{x}, \boldsymbol{x}')\Big). \tag{2.49}$$

where $\hat{y}_{\text{TNN}}(\boldsymbol{x})$ is the prediction of the baseline TNN and $r(\boldsymbol{x})$ models the remaining error. This decomposition allows the TNN to capture the dominant thermal dynamics, while the GP focuses on smaller, structured nonlinearities and provides a principled estimate of predictive uncertainty. A similar idea was used in [21].

2.4.3 Variational Inference

Variational Inference (VI) is used to approximate the posterior in GP without incurring the n^3 time complexity of SVGPs. VI is typically used in latent variable models for calculating the posterior. In latent variable models, a latent variable $\boldsymbol{z}$ and $p(\boldsymbol{x} \mid \boldsymbol{z})$ are assumed to have a simple distribution, typically a Gaussian. The mapping from $\boldsymbol{z}$ to parameters of $\boldsymbol{x}$ given $\boldsymbol{z}$ can be arbitrarily complex. The goal is to find $p(\boldsymbol{z} \mid \boldsymbol{x})$, but calculating this using Bayes theorem requires evaluating the marginal likelihood,

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\, d\boldsymbol{z}, \tag{2.50}$$

which is generally intractable. VI replaces the true posterior with a tractable distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and optimizes this approximation. Rewriting the log marginal

$$\log p(\boldsymbol{x}) = \log \int p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\, d\boldsymbol{z}. \tag{2.51}$$

Multiplying and dividing inside the integral by $q(\boldsymbol{z} \mid \boldsymbol{x})$:

$$\log p(\boldsymbol{x}) = \log \int p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\, \frac{q(\boldsymbol{z} \mid \boldsymbol{x})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\, d\boldsymbol{z}. \tag{2.52}$$

This then becomes

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\right]. \tag{2.53}$$

Since the log is concave, Jensen's inequality can be applied to obtain the evidence lower bound (ELBO):

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z} \mid \boldsymbol{x})}\right]. \tag{2.54}$$

Expanding the log yields:

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x} \mid \boldsymbol{z})] + \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{z})] - \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log q(\boldsymbol{z} \mid \boldsymbol{x})]. \tag{2.55}$$

Grouping these terms produces

$$\text{ELBO} = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x} \mid \boldsymbol{z})] - \text{KL}(q(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z}))\,. \tag{2.56}$$

Here, KL refers to KL divergence. The first term encourages the model to explain the data, while the KL term keeps the variational posterior close to the prior. $\log p(\boldsymbol{x})$ can be rewritten as

$$\log p(\boldsymbol{x}) = \text{ELBO} + \text{KL}(q(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p(\boldsymbol{z} \mid \boldsymbol{x}))\,. \tag{2.57}$$

Because the KL divergence is always non-negative, maximizing the ELBO pushes $q(\boldsymbol{z} \mid \boldsymbol{x})$ toward the true posterior.

2.4.4 SVGP Approximation using VI

In SVGP [19], the latent variable $\boldsymbol{z}$ is replaced by a set of inducing variables $\boldsymbol{u}$. These inducing variables are selected from the input space. There are multiple options for selecting inducing points, such as equidistant spacing, random selection, or clustering algorithms like K-Means. In general, the number of inducing points (M) is much smaller than the number of data points (n), denoted as $M \ll n$. The parameters of the variational distribution $q(\boldsymbol{u})$ are also initialized. In the case of a Gaussian distribution:

$$q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{m}, \boldsymbol{S}) \tag{2.58}$$

where m denotes the mean, and S denotes the covariance function.

$$p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K_{uu}}). \tag{2.59}$$

$$\text{ELBO} = \sum_{i=1}^{n} \left\{ \log \mathcal{N}(y_i \mid \boldsymbol{k_{iu}} \boldsymbol{K_{uu}^{-1}} \boldsymbol{m}, \sigma^2) - \frac{1}{2\sigma^2} \hat{\sigma}_{f_i}^2 \right\} - \text{KL}(q(\boldsymbol{u}) \,\|\, p(\boldsymbol{u})). \tag{2.60}$$

Here, $\boldsymbol{K_{uu}}$ is the kernel matrix obtained from the inducing points. $\hat{\sigma}_{f_i}$ indicates the estimated epistemic uncertainty of the data point and is given by

$$\hat{\sigma}_{f_i}^2 = k_{ii} - \boldsymbol{k_{iu}} \boldsymbol{K_{uu}^{-1}} \left(\boldsymbol{K_{uu}} - \boldsymbol{S} \right) \boldsymbol{K_{uu}^{-1}} \boldsymbol{k_{ui}}. \tag{2.61}$$

In the case of a Gaussian distribution, the KL divergence also has a closed-form solution and is given by:

$$\text{KL}(q(\boldsymbol{u}) \,\|\, p(\boldsymbol{u})) = \frac{1}{2} \left[\log \frac{|\boldsymbol{K_{uu}}|}{|\boldsymbol{S}|} - M + \text{tr}(\boldsymbol{K_{uu}^{-1}} \boldsymbol{S}) + \boldsymbol{m}^\top \boldsymbol{K_{uu}^{-1}} \boldsymbol{m} \right]. \tag{2.62}$$

A detailed derivation of the above equations is available in [19]. Using SVGP drops the time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(nM^2 + M^3)$.

2.5 Bayesian Neural Networks

Bayesian neural networks (BNNs) build upon the concepts of Bayesian linear regression introduced in Sec. 2.4.1. Eq.(2.12) is analogous to the formulation of a simple perceptron in a classical neural network. By treating the network weights as random variables rather than fixed deterministic values, the model becomes a BNN. BNNs can be used to predict both epistemic and aleatoric uncertainty of the predictions. A visual representation of BNNs can be seen in Fig 2.9 During inference, weights are sampled from their posterior

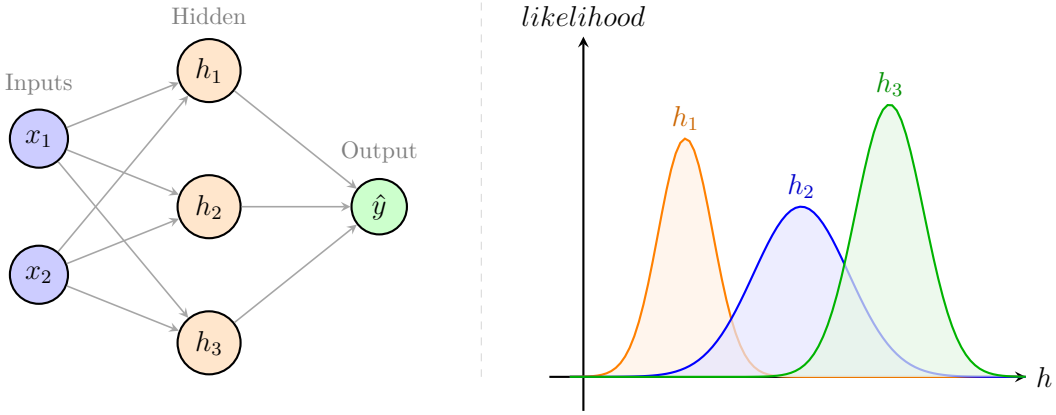

Fig. 2.9: A sample Bayesian Neural Network. The left panel shows the network, and the right panel shows the probabilistic nature of the hidden nodes, depicting their distributions with varying means and variances.

distribution, and a prediction is made. This is repeated multiple times, resulting in multiple predictions. The mean of the predictions is taken as the final prediction, and the variance among them is treated as uncertainty. [22] gives an overview of methods that can be used for training BNNs. But in general, training and inference with BNNs is computationally more expensive than conventional neural networks of similar size. Because of the computational overhead of training BNNs, alternative ways of inferring uncertainty have been explored. One such method is based on dropouts [23].

2.5.1 Dropout as Bayesian approximation

Dropouts were first introduced in 2014 as a simple way to avoid overfitting in neural networks [24]. When used to reduce overfitting, it involves dropping a pre-defined percentage of neurons during training. Dropout is then no longer used during inference. An overview of dropout is shown in Fig 2.10. In [23], the authors demonstrated that keeping the dropouts active even during inference resulted in a predictive distribution that was an approximation of the BNN posterior. In this approach, M inference runs are performed with different dropout masks, yielding M predictions that are then combined using their mean and variance to estimate the posterior distribution.

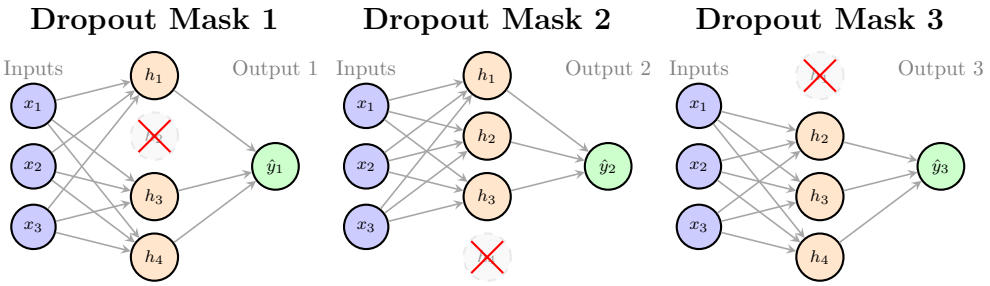

Fig. 2.10: Illustration of dropouts.

$$\hat{y}_{dropout} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i \quad , \qquad \hat{\sigma}^2_{epistemic-dropouts} = \frac{\sum_{i=1}^{M}(\hat{y}_i - \hat{y}_{dropout})^2}{M-1} \tag{2.63}$$

If the network also explicitly predicts an uncertainty, the mean of them becomes the aleatoric uncertainty.

2.6 Deep Ensembles

Another popular technique for UQ is deep ensembles. They were first introduced in [25]. Deep ensembles can be considered as a frequentist alternative to BNNs. The loss landscape for neural networks tends to be quite complex, as shown in Fig 2.11. Due to the complex loss landscape, neural networks often fail to converge to the global minimum and instead settle into a local minimum. This minimum could differ due to stochasticity in training. This stochasticity could arise from randomness in the initial weights or from the sampling performed in this work. Deep ensembles involve training multiple networks with different initial weights, leading to convergence to different local minima, as shown in 2.12. It is also possible to combine models with different architectures here, but in standard deep ensembles, networks with the same architecture are used. If R models are trained then,

$$\hat{y}_{ensembles} = \frac{1}{R}\sum_{i=1}^{R}\hat{y}_i \quad , \qquad \hat{\sigma}^2_{epistemic-ensembles} = \frac{\sum_{i=1}^{R}(\hat{y}_i - \hat{y}_{ensembles})^2}{R-1} \tag{2.64}$$

The idea behind deep ensembles is that if the model encounters out-of-distribution data, each model will output noticeably different values, resulting in wider uncertainty estimates. Similar to dropouts, they can be used for estimating both aleatoric and epistemic uncertainty.

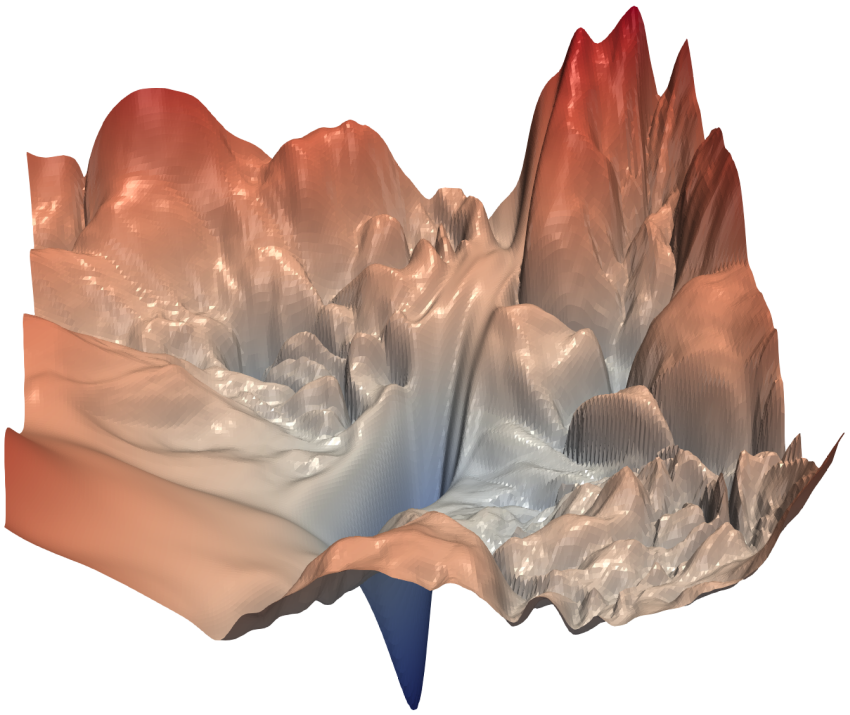

Fig. 2.11: Loss landscape visualization of a neural network (taken from [26]).

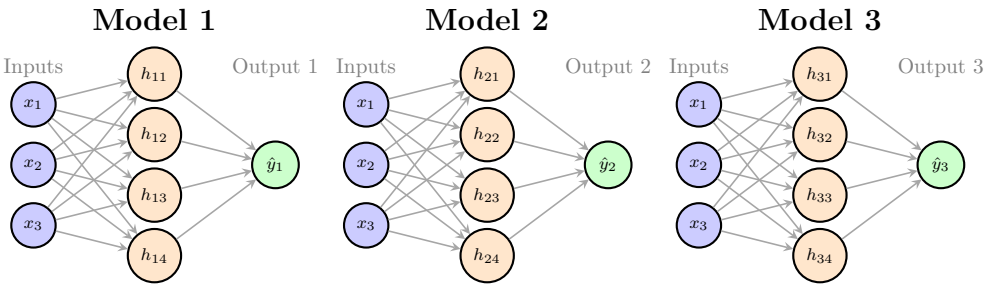

Fig. 2.12: Demonstration of how deep ensembles work.

2.7 Evidential Deep Learning

Evidential deep learning (EDL) offers an alternative way of quantifying uncertainties to BNN-based approaches. Instead of placing priors over the weights of neural networks, here, priors are placed directly on the parameters of the likelihood for the network's output. It is achieved by predicting the parameters of a higher-order distribution referred to in the literature as an evidential distribution. EDL solves the problems associated with other discussed UQ methods as neither it have the scalability bottlenecks as seen in GP nor it requires sampling or multiple forward passes as required in dropouts and deep ensembles. EDL comes with different sets of problems that will be addressed later in this Sec. 2.7.1.

EDL was initially introduced for classification [27]. Later, in [28], it was extended to regression. This work uses the approach presented in [28], but modifies the architecture to accommodate the existing TNN architecture. This approach is suitable only for 1-D targets. It is possible to extend EDL to multi-dimensional targets, but the conjugate distribution then changes, as shown in [29]. In this work, in addition to the NLL as proposed in [28], CRPS loss is used for training and compared with NLL. The integration of the EDL layer into the existing architecture is detailed in the implementation section. In EDL, data is assumed to be Gaussian with unknown mean and variance:

$$p(y_{EDL}^{(i)}) \sim \mathcal{N}(\mu^{(i)}, \sigma_{(i)}^2). \tag{2.65}$$

Conjugate priors are placed over the estimated mean and variance:

$$p(\hat{\mu}_{EDL}^{(i)}|\hat{\sigma}_{(i)EDL}^2) \sim \mathcal{N}\left(\psi^{(i)}, \frac{\hat{\sigma}_{(i)EDL}^2}{v^{(i)}}\right). \tag{2.66}$$

$$p(\hat{\sigma}_{(i)EDL}^2) \sim \Gamma^{-1}(\alpha^{(i)}, \beta^{(i)}). \tag{2.67}$$

Here $\alpha^{(i)}, \beta^{(i)}, \psi^{(i)}$ and $v^{(i)}$ are referred to as evidential parameters. In standard EDL literature, the predicted mean $\psi^{(i)}$ is written as $\gamma^{(i)}$. For numerical reasons, $\alpha^{(i)} > 1$, $\beta^{(i)}, v^{(i)} > 0$. The parameter $\alpha^{(i)}$ can be thought of as virtual observations contributing toward the estimation of the variance. Conversely, $v^{(i)}$ denotes the virtual observations contributing toward the estimation of the predictive mean $\psi^{(i)}$. Together, $2v^{(i)} + \alpha^{(i)}$ denotes the total evidence. A similar physical interpretation of $\beta^{(i)}$ is not explicitly available in the foundational paper [28]. The Γ^{-1} distribution is defined only for positive real values. Effects of changing its parameters and function itself are visualized in Fig. 2.13. Combining the two equations above yields the higher-order normal inverse-gamma (NIG) distribution:

$$p(\hat{\mu}_{EDL}^{(i)}, \hat{\sigma}_{(i)EDL}^2) = p(\hat{\mu}_{EDL}^{(i)}|\hat{\sigma}_{(i)EDL}^2) \cdot p(\hat{\sigma}_{(i)EDL}^2) = NIG(\alpha^{(i)}, \beta^{(i)}, \psi^{(i)}, v^{(i)}). \tag{2.68}$$

Since,

$$p(\hat{y}_{EDL}^{(i)}) = \int_0^\infty \int_{-\infty}^\infty p(\hat{y}_{EDL}^{(i)}|\hat{\mu}^{(i)}, \hat{\sigma}^{(i)2}) \cdot p(\hat{\mu}^{(i)}, \hat{\sigma}^{(i)2}), d\mu, d\sigma^2. \tag{2.69}$$

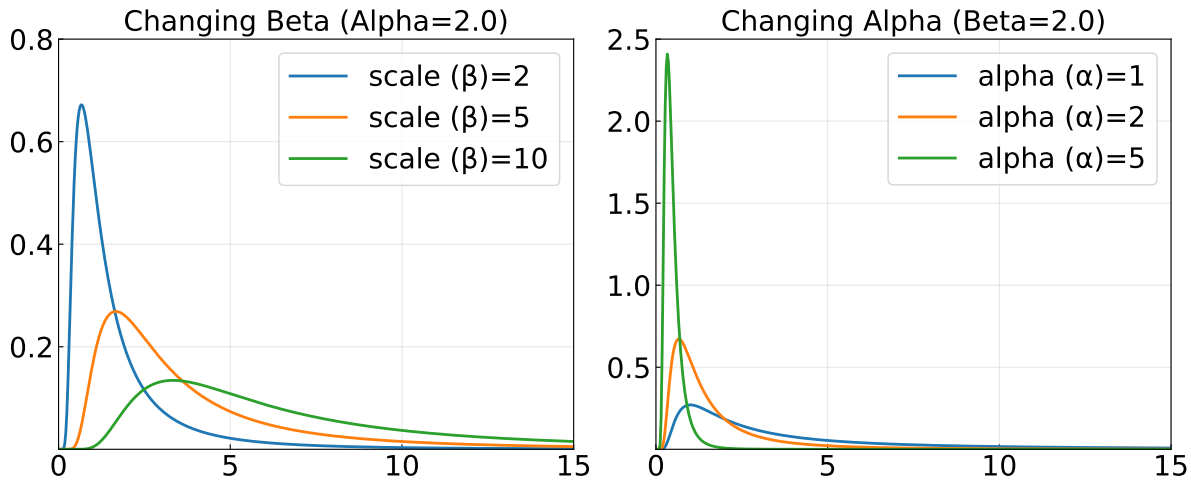

Fig. 2.13: Illustration of Γ^{-1} and the effect of changing its parameters $\alpha^{(i)}$ and $\beta^{(i)}$. α controls the shape of the distribution. Increasing it reduces the variance and creates sharper peaks. $\beta^{(i)}$, on the other hand, controls the scale of the distribution. A higher value of $\beta^{(i)}$ increases the variance and shifts the peak to the right.

Solving this marginalization yields the predictive distribution in the form of a Student-t distribution:

$$p(\hat{y}_{EDL}^{(i)}) = St\left(y^{(i)}; \psi^{(i)}, \frac{\beta^{(i)}(1+v^{(i)})}{v^{(i)}\alpha^{(i)}}, 2\alpha^{(i)}\right). \tag{2.70}$$

Here, St denotes student-distribution, $\frac{\beta^{(i)}(1+v^{(i)})}{v^{(i)}\alpha^{(i)}}$ denotes the total variance, and $2\alpha^{(i)}$ denotes the degrees of freedom (DOF) of the student-t distribution. Effect of changing DOF is shown in Fig. 2.14. The negative log likelihood can be expanded as:

$$\begin{aligned}\mathcal{L}_{i_{EDL}}^{NLL} = &\frac{1}{2}\log\left(\frac{\pi}{v^{(i)}}\right) - \alpha^{(i)}\log(\Omega_{EDL}^{(i)}) + \left(\alpha^{(i)} + \frac{1}{2}\right)\log\left((y^{(i)} - \psi^{(i)})^2 v^{(i)} + \Omega_{EDL}^{(i)}\right) \\ &+ \log\left(\frac{\Gamma(\alpha^{(i)})}{\Gamma(\alpha^{(i)} + \frac{1}{2})}\right).\end{aligned} \tag{2.71}$$

where $\Omega_{EDL}^{(i)} = 2\beta^{(i)}(1+v^{(i)})$. [28], also introduced a regularizer defined as:

$$\mathcal{L}_{i_{EDL}}^{R} = |y^{(i)} - \psi^{(i)}| \cdot (2v^{(i)} + \alpha^{(i)}). \tag{2.72}$$

The total loss for the EDL is defined as

$$\mathcal{L}_{total_{EDL}} = \mathcal{L}_{i_{EDL}}^{NLL} + \lambda_R \mathcal{L}_{i_{EDL}}^{R}. \tag{2.73}$$

Here, λ_R punishes the model for being overconfident. Total uncertainty as given by Equation (2.70) can be further decomposed into aleatoric uncertainty and epistemic uncertainty. Expressions of them are provided below. A detailed derivation is available in the appendix of [28].

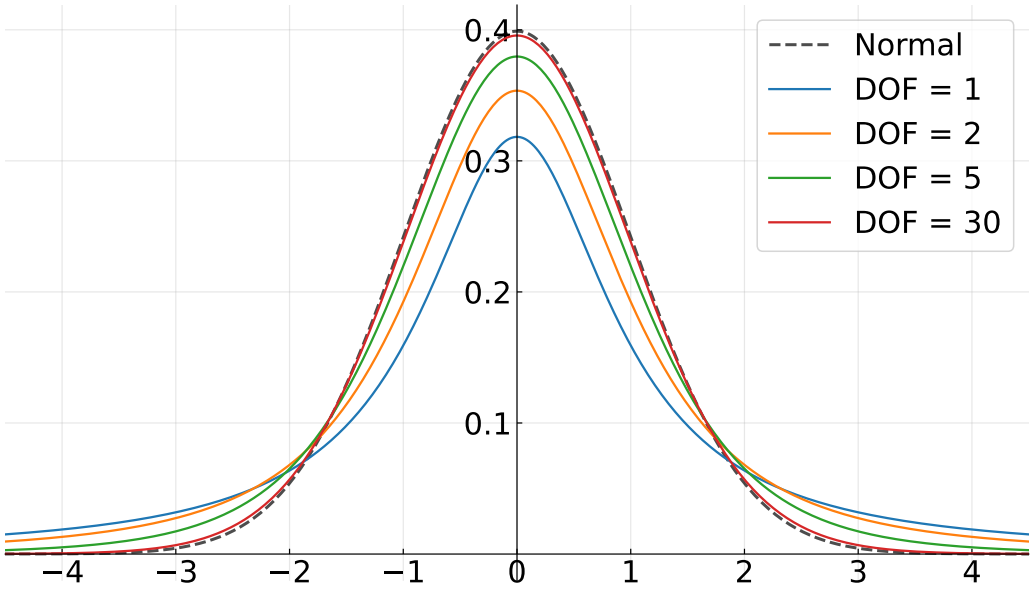

Fig. 2.14: Illustration of student-t distribution and the effect of changing its DOF. As DOF increases, it converges to a normal distribution. For small values of DOF, it has heavier tails.

$$\hat{\sigma}^2_{(i)EDL-aleatoric} = \frac{\beta^{(i)}}{\alpha^{(i)}}, \qquad \hat{\sigma}^2_{(i)EDL-epistemic} = \frac{\beta^{(i)}}{v^{(i)}\alpha^{(i)}}. \tag{2.74}$$

In this work, in addition to NLL, CRPS loss is also considered. Analytical expression as derived in [30] is being used,

$$\begin{aligned}\mathcal{L}^{CRPS}_{i_{EDL}} = \sigma^{(i)}_{st}\Bigg[&\frac{y^{(i)}-\psi^{(i)}}{\sigma^{(i)}_{st}}\left(2F_{2\alpha^{(i)}}\left(\frac{y^{(i)}-\psi^{(i)}}{\sigma^{(i)}_{st}}\right)-1\right)\\ &+2o_{2\alpha^{(i)}}\left(\frac{y^{(i)}-\psi^{(i)}}{\sigma^{(i)}_{st}}\right)\frac{2\alpha^{(i)}+(\frac{y^{(i)}-\psi^{(i)}}{\sigma^{(i)}_{st}})^2}{2\alpha^{(i)}-1}\\ &-\frac{2\sqrt{2\alpha^{(i)}}}{2\alpha^{(i)}-1}\frac{B(\frac{1}{2},2\alpha^{(i)}-\frac{1}{2})}{B(\frac{1}{2},\alpha^{(i)})^2}\Bigg].\end{aligned} \tag{2.75}$$

Here, F and o indicate the CDF and PDF of a Student's t distribution with DOF = 2α, respectively. B indicates a beta function [31]. The regularizer function used is the same as in equation (2.73).

2.7.1 Criticism of EDL

There have been a few issues with predictive distributions under EDL. As seen by equation (2.74), there is a mathematical relation between aleatoric and epistemic uncertainty, i.e.,

$\hat{\sigma}^2_{EDL-aleatoric} = v\hat{\sigma}^2_{EDL-epistemic}$. This kind of relationship between uncertainties is scientifically inaccurate. In [32], the authors introduce a reference distribution to represent the frequentist ground truth for predictive uncertainty. Their analysis demonstrates that the epistemic uncertainty predicted by EDL is unfaithful, as it fails to accurately recover the reference distribution. However, they observed that while absolute values are unreliable, EDL measures remain useful in a relative sense, effectively identifying the ranking of certain versus uncertain regions. There are also issues with positivity constraints on evidential parameters, which can cause zero gradients and lead to evidence contraction. This limitation, though, can be addressed using a different regularization as presented in [33]. This work adopts the stance that while the uncertainty breakdown provided by EDL may be unfaithful in isolation, the total uncertainty remains a valid metric for the predictive distribution. Despite issues raised by [32], [33], EDL is still promising for UQ because of noticeably faster inference, and a lot of these criticisms can be resolved by doing uncertainty calibration.

2.8 Uncertainty Calibration

Despite UQ methods being rooted in statistics, predictive distributions are frequently miscalibrated, manifesting either as overconfident or as under-confident. As per [34], a primary driver of miscalibration is the generalization gap. MSE is typically higher on an unseen test set than on the training set. Unless the predicted uncertainty is inflated proportionally, the model will produce miscalibrated uncertainty. This issue is further exacerbated by the use of approximations, such as in dropout and SVGP. It is also crucial to calibrate uncertainty when working with EDL for reasons discussed in section 2.7.1. For regression, common alternatives include variance scaling, isotonic regression, and conformal prediction. This work focuses exclusively on variance scaling. This choice is motivated by the fact that, for neural network-based modeling, variance scaling often yields superior performance and robustness, as it prevents overfitting to the calibration set[34]. The idea for variance scaling is summarized in the equation below. Here, $\hat{\sigma}^2$ is the predicted uncertainty, $\hat{\sigma}^2_{calibrated}$, and s is a scaling factor.

$$\hat{\sigma}^2_{calibrated} = s\hat{\sigma}^2. \tag{2.76}$$

The goal is to find s on a held out calibration set such that a loss function with $\hat{\sigma}^2_{calibrated}$ is minimized. Choices here include NLL, CRPS, miscalibration area, etc. There are a lot of other loss functions that can be used here, but covering every single one of them is beyond the scope of this work.

2.9 Uncertainty Evaluation Metrics

According to the authors of [35], the quality of predictive distributions can be evaluated using two key properties: calibration and sharpness. Calibration measures whether predicted probabilities are statistically consistent with observed frequencies. For example,

if a model predicts a 95% prediction interval, approximately 95% of the observed data should fall within that interval. Calibration alone, however, is insufficient, since a model with excessively large uncertainty bounds may still achieve good calibration. Sharpness measures the concentration of predictive distributions and therefore penalizes overly broad uncertainty estimates. In practice, most metrics implicitly balance calibration and sharpness, though some place greater emphasis on one than the other. In this work, in addition to CRPS and NLL, the miscalibration area has been used as an uncertainty evaluation metric. Other metrics exist as well, but they have not been discussed in this work, as the metrics covered are sufficient for thorough UQ evaluation.

2.9.1 Reliability diagram

A reliability diagram, also referred to as a calibration plot in some literature, is a graphical method of representing the calibration quality. It plots the predicted probabilities against the observed probabilities. They were initially used for classification tasks [36], [37], [38], but [35] adapted it for regression-related tasks. A reliability diagram is plotted by first calculating prediction intervals for multiple confidence intervals. For each interval, the ratio of observed points in the predicted interval to the total points is calculated. In an ideal case, for each interval, the observed ratio should match the confidence interval, resulting in a straight line as shown by the dotted orange line in Fig. 2.15. Reliability diagrams allow for the determination of underconfidence or overconfidence in predictions. A line passing under this orange line signifies overconfidence, whereas a line above shows underconfidence.

Performance measured by the reliability diagram can be quantified in a single number, referred to as the miscalibration area. Miscalibration area is the area spanned between the reliability curve of the ideal case and the prediction [39]. Miscalibration area ranges between 0 and 0.5, where 0 indicates perfect calibration, and 0.5 indicates the worst possible calibration.

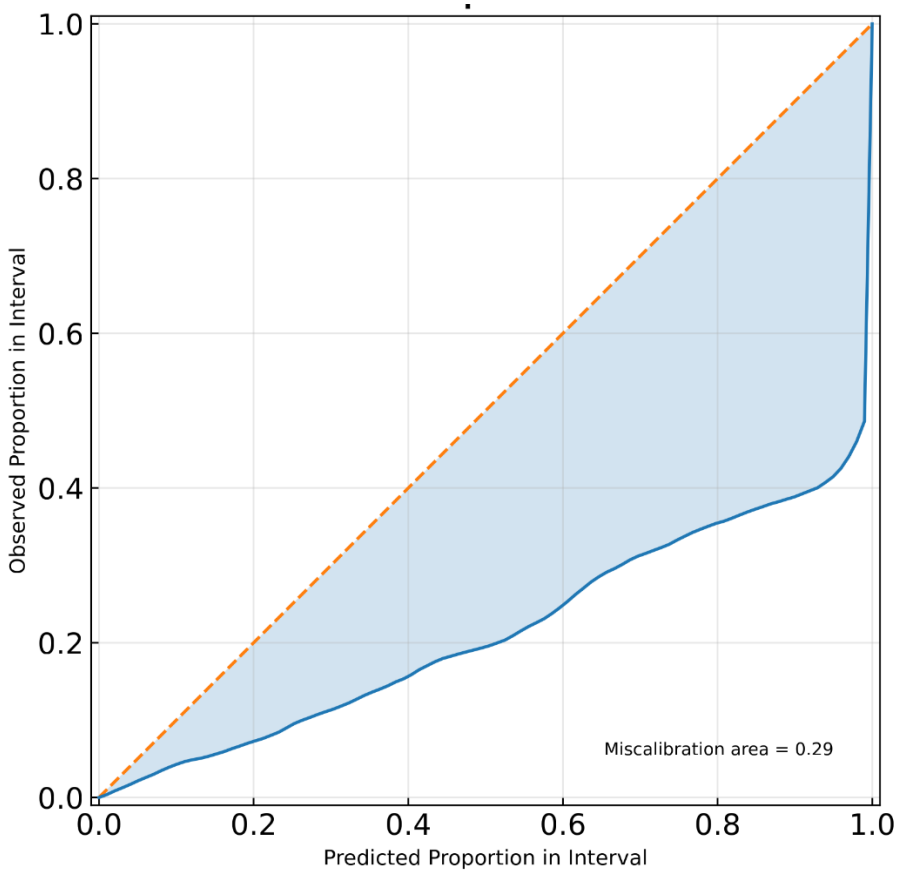

Fig. 2.15: Here, the dotted line represents perfect calibration, the blue line represents the observed probabilities. The area spanned between these lines is the miscalibration area. Here, the miscalibration area is 0.29. This reliability diagram is created using the uncertainty toolbox [39].

3 Implementation

All experiments and models in this work were implemented in Python (version 3.10.14) and performed on the Noctua 2 high-performance computing (HPC) cluster [40]. The deep learning architectures and automatic differentiation are implemented using PyTorch (version 2.4) [41], GP is handled by GPyTorch (version 1.15.1) [42], and hyperparameter optimization is performed using Optuna (version 4.7.0) [43]. Data pre-processing and tabular manipulations were performed using standard scientific libraries. To ensure reproducibility, all environment dependencies are fully documented in the supplementary code repository[1].

3.1 Dataset

Tab. 3.1: Considered input and target parameters.

Parameter name	Symbol
ambient temperature	ϑ_{a}
liquid coolant temperature	ϑ_{c}
actual voltage d-axis component	u_{d}
actual voltage q-axis component	u_{q}
actual current d-axis component	i_{d}
actual current q-axis component	i_{q}
torque applied	τ
motor speed in rpm	v_{rpm}
permanent magnet temperature	ϑ_{PM}
stator teeth temperature	ϑ_{ST}
stator winding temperature	ϑ_{SW}
stator yoke temperature	ϑ_{SY}

This work utilizes the dataset provided by [44], [45]. The dataset comprises 185 hours of recordings from a 52 kW torque-controlled PMSM. Data is sampled at 2 Hz and dis-

[1] https://github.com/745Shubham/UQ_TNN

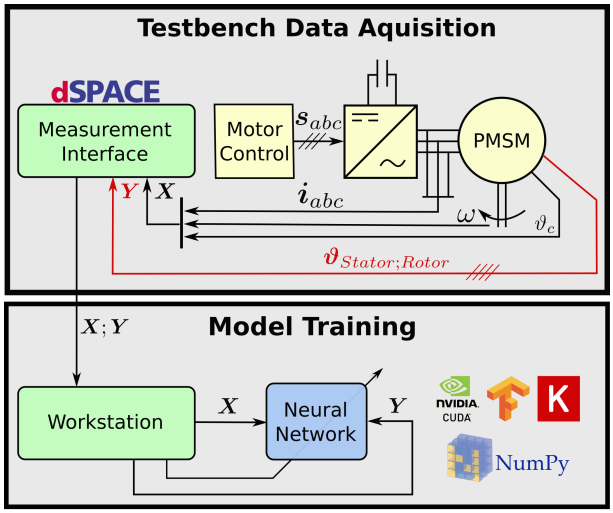

Fig. 3.1: Schematic showing the test bench and an overview of the model training environment (taken from [45]). In this schematic, $\boldsymbol{X}$ represents the input data, $\mathbf{Y}$ represents the measured temperatures of the 4 targets, $\boldsymbol{s}_{abc}$ indicates the motor control signal, and ω denotes the angular velocity of the PMSM.

tributed across 69 distinct driving profiles, yielding 1,330,815 multidimensional samples. Each profile represents a 2-D random walk in the torque-speed plane to effectively simulate real-world driving conditions. The data was collected at a test bench at the University of Paderborn. A schematic of the test bench is shown in Fig. 3.1. A detailed overview of the dataset, test bench architecture, and sensor selection is available in [46]. All measured parameters utilized in this study are summarized in Table 3.1. This dataset is selected because it is the same one used in the original TNN implementation, providing a consistent, direct benchmark for evaluating the proposed additions.

In this work, in addition to the ten measured parameters, feature engineering is performed to compute the magnitudes of the current and voltage, denoted by $\boldsymbol{i}_\mathrm{s}$ and $\boldsymbol{u}_\mathrm{s}$, respectively. As per [46], this combination can be assumed to be correlated with the thermal loss in PMSMs.

$$\boldsymbol{u}_\mathrm{s} = \sqrt{\boldsymbol{u}_\mathrm{d}^2 + \boldsymbol{u}_\mathrm{q}^2} \qquad \boldsymbol{i}_\mathrm{s} = \sqrt{\boldsymbol{i}_\mathrm{d}^2 + \boldsymbol{i}_\mathrm{q}^2}. \tag{3.1}$$

For data normalization, all temperature-related variables were scaled by dividing by 200, while all other variables were normalized by dividing by their respective maximum observed values. Since the maximum temperature value is less than 200 °C, this normalization ensures all inputs and targets are between -1 and 1. This normalization approach is the same as that done in [46].

3.2 Baseline TNN architecture

In the foundational paper for TNN [5], the author proposed an HPO-optimized configuration that outperformed other approaches compared in that work. In the same work,

Tab. 3.2: Baseline TNN Configuration

Hyperparameters	Value
hidden layers	1
nodes per hidden layer	32
π activation	tanh $\rightarrow$ biased ELU
γ activation	sigmoid $\rightarrow$ biased ELU
input activation	linear
optimizer	Adam
initial learning rate	10^{-3}
learning rate decay	$0.5\times$ after 75 epochs
total epochs	150
loss function	MSE
batch size	32
TBPTT window	512
chunk size	2048
inv. capacitance κ	$\mathcal{N}(7, 0.5)$

the author also proposed that a smaller architecture can achieve similar results.

In this work, instead of using the HPO-optimized configuration, a smaller configuration inspired by [5] is chosen. Since the exact configuration of the smaller architecture proposed in [5] is not publicly available, an approximate implementation inspired by the reported setup was used. The architecture is shown in Tab. 3.2. An overview of activation functions, such as linear, ReLU, tanh, sigmoid, biased ELU, and softplus, can be found in [47]. TNN is recursive by definition; this recursion creates exploding/vanishing gradients during training due to the influence of past gradients. To address exploding/vanishing gradients, truncated backpropagation through time (TBPTT) is used. TBPTT places an upper bound on the number of these recursive time steps during backpropagation. Furthermore, to keep all sequences of similar size and avoid long sequences padded with zeros, training profiles were split into chunks. Only the last section of this chunk is then padded. The network is then trained with the adam optimizer [48] for 150 epochs. This configuration achieved an MSE of $\sim 3.3\ °C^2$, similar to that of the small configuration mentioned in [5].

3.3 Dropout

Dropout approximates uncertainty in predictions by running multiple forward passes through the network with different dropout masks. This section provides implementation details, including the proposed architecture and other testing performed during the integration of dropout-based uncertainty quantification in TNNs. In the initial dropout

testing, the number of nodes in both the power loss and conductance networks was increased to 64; everything else remained as in Tab. 3.2. Dropout rates from 0.1 to 0.9 were tested. Results from testing with 20 forward passes for dropout rates 0.1, 0.5, and 0.9 are shown in Fig. 3.2. Here, the mean prediction and the uncertainty were calculated using Eq 2.63. Fig. 3.2 shows only the permanent magnet (pm) temperature results for better readability, but similar results were observed across all targets. Since the model itself does not explicitly predict uncertainty, the uncertainty shown in the figure is mostly epistemic. Predictive uncertainty generally increased with increasing dropout rate. There is no incentive for the model to learn calibrated uncertainty estimates. The solution to this problem is to move beyond a deterministic loss function, such as MSE, and adopt a probabilistic loss function, such as CRPS or NLL. It is also possible to continue using MSE here, pick a random dropout rate, and perform uncertainty calibration. This is not explored in this work because that approach cannot capture aleatoric uncertainty, and accuracy depends on calibration profiles to sufficiently capture all motor operating conditions. To predict aleatoric uncertainty, branching is introduced in both the $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ networks. An explicit uncertainty prediction branch for temperatures was intentionally avoided. This design choice preserves interpretability by constraining uncertainty estimation to physically meaningful quantities such as $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$. An overview of the branched network for $\boldsymbol{\pi}$ is shown in Fig. 3.3. The $\boldsymbol{\gamma}$ network also uses a similar branching structure. In both networks, 8 nodes were used in the uncertainty branch, with Biased Elu during initial testing. Various approaches were explored to propagate this uncertainty in $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ through the non-linear LPTN equation (Eq. 2.2) to estimate the uncertainty in the predicted temperature.

- In the boundary value propagation approach, a combination of mean plus 1 standard deviation is used across all predicted $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ to estimate an upper bound. A similar lower bound is also calculated using the mean minus 1 standard deviation. This approach produced pessimistic estimates (for permanent magnet) because all variables were simultaneously pushed toward their extrema, implicitly assuming highly correlated worst-case deviations.

- In the unscented transform, a fixed number of points distributed around the predicted mean are calculated. These points are referred to as sigma points and were calculated based on [49]. These sigma points were then propagated through the LPTN equation. This resulted in overly optimistic uncertainty predictions. Likely causes here include insufficient tuning of the unscented transform parameters.

- In Monte-Carlo sampling, $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ are assumed to be normally distributed. 50 samples are drawn for $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$, and then 50 temperature estimates are predicted. The mean of these temperatures is taken as the final prediction, and the variance across them is treated as the predicted aleatoric uncertainty. Of the 3 approaches, only this method provided plausible uncertainties and is therefore chosen. Models were trained using the reparameterization trick [50]. The reparameterization trick allows gradient flow through sampling. The proposed branched prediction architecture is shown in Fig. 3.4.

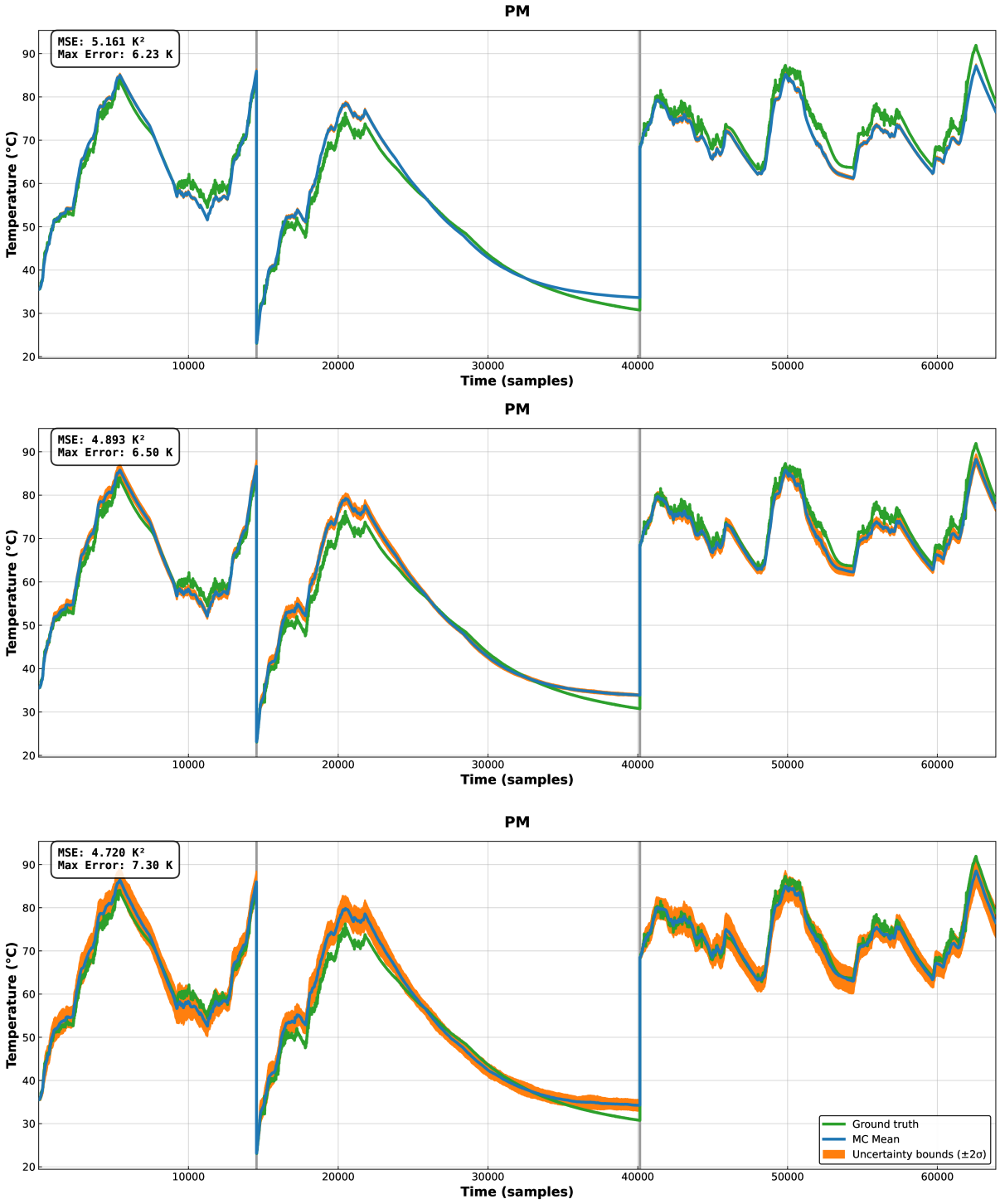

Fig. 3.2: Results from dropouts trained with MSE as a loss function on permanent magnet (pm). The first plot shows performance with a dropout rate of 0.1, the middle plot with a dropout rate of 0.5, and the last plot with a dropout rate of 0.9. Mean standard deviation (uncertainty) is 0.16, 0.45, and 1.20, respectively.

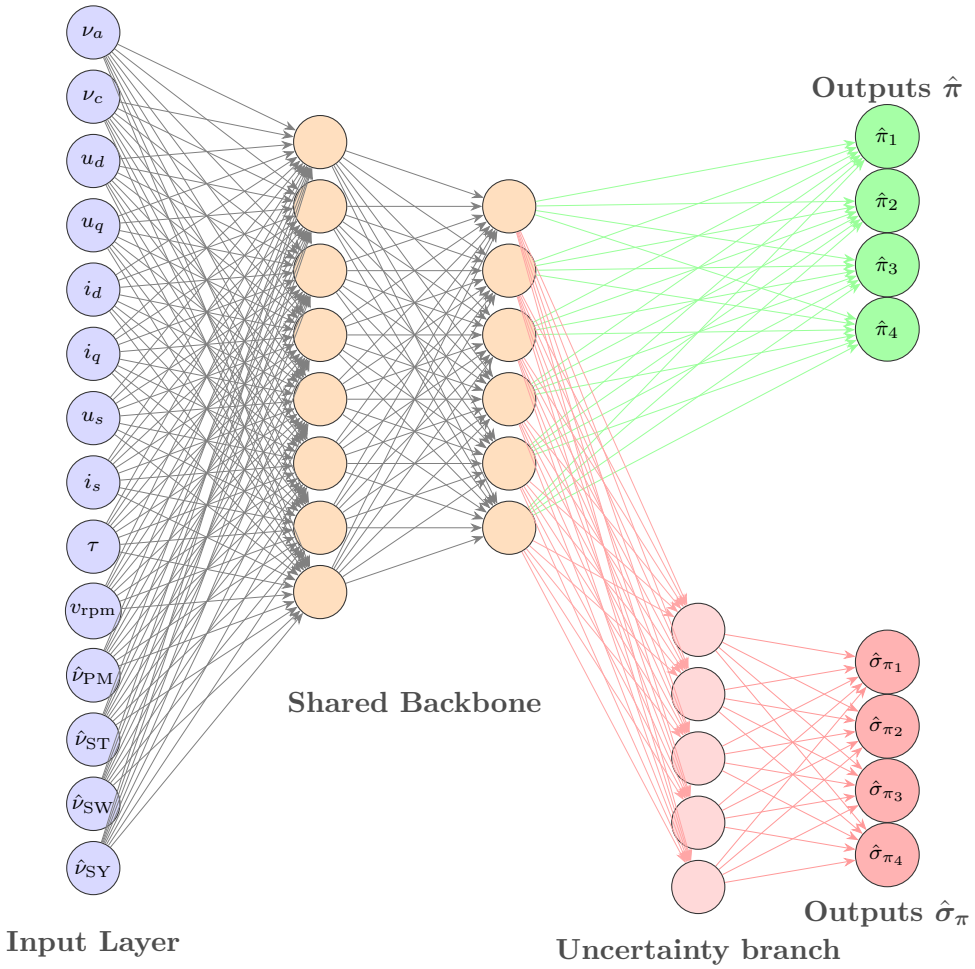

Fig. 3.3: Representative branched neural network for $\boldsymbol{\pi}$. The $\boldsymbol{\gamma}$ network also uses a similar branching architecture.

The extended Kalman filter was not considered as the method of choice because the PyTorch implementation of the Jacobian is computationally slow for real-time predictions.

In the proposed architecture 3.4, both NLL and CRPS were tested. Initial experiments showed that models trained using NLL performed substantially worse than those trained using CRPS on both predictions and uncertainty estimates. It was likely because NLL penalizes over-confident estimates more severely. NLL for normal distribution also suffers from numerical instability, as low uncertainty values can cause the log term to blow up to infinity. Hence, the NLL-trained model converged to a sub-optimal minimum. Because NLL has not been sufficiently explored with a different set of hyperparameters due to the aforementioned reasons and time constraints, this work does not conclusively rule out NLL as a possible loss function for the proposed architecture.

The following equations detail how uncertainty is estimated in the proposed dropout-based UQ architecture.

$$\hat{\boldsymbol{\sigma}}_{\vartheta-dropout} = \sqrt{\hat{\boldsymbol{\sigma}}^2_{epistemic-dropout} + \hat{\boldsymbol{\sigma}}^2_{aleatoric-dropout}} \tag{3.2}$$

Here, $\hat{\boldsymbol{\sigma}}^2_{epistemic-dropout}$ and $\hat{\boldsymbol{\vartheta}}_{dropout}$ are calculated as shown in Eq.2.63, and $\hat{\boldsymbol{\sigma}}_{aleatoric-dropout}$ is the mean of M predictions from the uncertainty head, if M forward passes are performed.

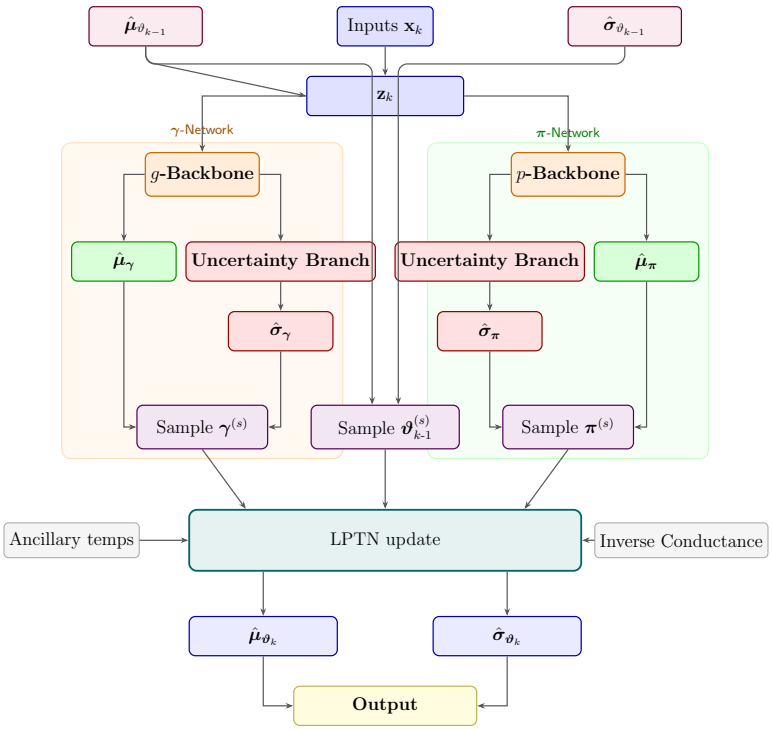

Fig. 3.4: Block diagram showing the detailed structure of the intended uncertainty-enhanced branched TNN.

3.4 Deep Ensembles

This section introduces how uncertainty quantification can be done in TNNs using deep ensembles. A straightforward integration could use a standard TNN network and retrain it multiple times with a different seed and calculate the uncertainty as discussed in Sec. 2.6. Using this approach has the same issues as discussed in Sec. 3.3 for the motivation of probabilistic loss functions.
Learnings from the dropout implementation were adopted here. Deep ensembles use the same architecture as shown in Fig. 3.4 with sampling for uncertainty propagation. Nodes in $\boldsymbol{\pi}$ and $\boldsymbol{\gamma}$ networks were reduced to 32 during initial testing. The uncertainty network still had 8 nodes. CRPS is chosen as the loss function. Multiple versions of this model are trained, and the uncertainty is calculated as follows,

$$\hat{\boldsymbol{\sigma}}_{\vartheta-ensembles} = \sqrt{\hat{\boldsymbol{\sigma}}^2_{epistemic-ensembles} + \hat{\boldsymbol{\sigma}}^2_{aleatoric-ensembles}}. \tag{3.3}$$

Here, $\hat{\boldsymbol{\sigma}}^2_{epistemic-ensembles}$ and $\hat{\boldsymbol{\vartheta}}_{ensembles}$ are calculated as shown in Eq.2.64, and $\hat{\boldsymbol{\sigma}}_{aleatoric-ensembles}$ is the mean of R predictions from the uncertainty head, if R models are trained.

3.5 Evidential Deep Learning

This section introduces implementation details of EDL and integration into TNN. It is not possible to use EDL directly in the $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ networks due to the EDL regularizer. In EDL regularizer (equation 2.72), there is a $|y^{(i)} - \psi^{(i)}|$ term. Using EDL directly in these networks would require access to true $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ values, which are non-trivial to measure or estimate. Furthermore, due to data normalization, TNN estimated $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ will be on a different scale. To solve this challenge, a new architecture is proposed. $\boldsymbol{\gamma}$ networks and $\boldsymbol{\pi}$ networks remain the same and predict the temperatures as in the standard TNN. This prediction is then treated as ψ in EDL. This ψ is then fed back into a new network, referred to as an evidential network. This network predicts the remaining EDL parameters, i.e., α, β, and v. The evidential network takes in input data, ψ, and outputs of the evidential network from the previous timestep as input. A detailed block diagram is shown in Fig. 3.5.

In addition to the architecture shown in Fig. 3.5, two other architectures were also tested. In one of them, ψ is explicitly predicted by the evidential network and used as a temperature estimate. In the other one, feedback for α, β, and v is skipped. Both versions had worse performance and were not considered further. The configuration used for initial testing is documented in Tab. 3.3. As discussed in Sec. 2.7, both CRPS and NLL were tested in this configuration. Because the Student-t posterior has heavier tails, the NLL-optimized model does not suffer from the same issues observed with dropout.

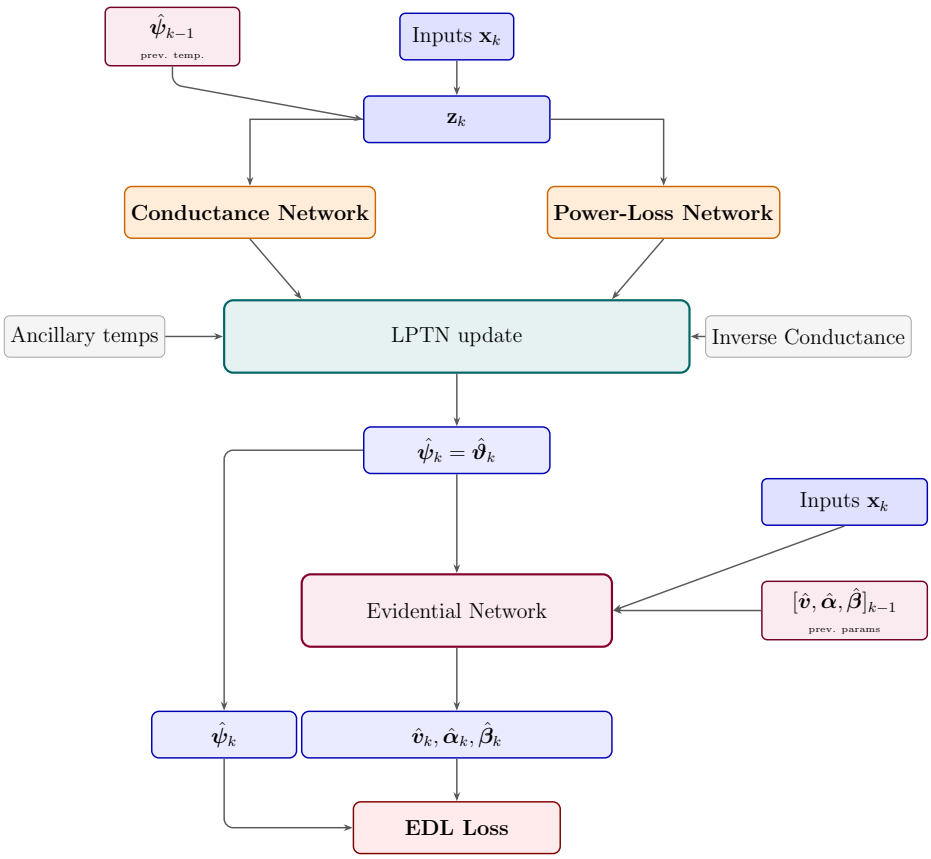

Fig. 3.5: Block diagram showing the detailed structure of the proposed EDL integrated TNN.

Tab. 3.3: EDL Configuration for testing

Hyperparameters	Value
hidden layers	1
nodes per hidden layer	32
π activation	tanh $\rightarrow$ biased ELU
γ activation	sigmoid $\rightarrow$ biased ELU
evidential activation	tanh $\rightarrow$ softplus
input activation	linear
optimizer	Adam
initial learning rate	10^{-2}
lambda_reg	0.01
learning rate decay	0.5$\times$ after 75 epochs
total epochs	150
loss function	MSE
batch size	32
TBPTT window	512
chunk size	2048
inv. capacitance κ	$\mathcal{N}(-7, 0.5)$

3.6 Gaussian Processes

This section discusses how GP is used for UQ in TNN, including the reasoning behind kernel choice. As discussed in Sec. 2.4, for the implementation of GP in TNN, SVGP is selected as the method of choice. The GP framework is implemented as a probabilistic residual model operating on top of a pre-trained TNN. This TNN is trained using the same hyperparameters as detailed in Tab. 3.2. The TNN estimate, along with input data, is fed into the SVGP model. The model is then trained using the Adam optimizer using variational ELBO as a loss function.

Neural networks tend to overfit the training data. Using training data already seen by the TNN risks producing artificially small residuals and overconfident uncertainty estimates. It is also not ideal to remove a substantial amount of data and use an underperforming TNN. To tackle this issue, in addition to the test and calibration set, four additional profiles were removed from the TNN training. These four profiles were then added back into the training samples used for GP. These profiles serve as a regularizer and a punishment for overfitting. Experiments were also conducted in which only these 4 additional profiles were used for GP training, but this caused convergence issues with the temperature uncertainty in the permanent magnet. These 4 profiles were selected arbitrarily without any cross-validation (CV). Data splitting is done as shown in the Fig. 3.6.

For the choice of kernel k, an addition of Matérn 5/2 and Matérn 1/2 is used. Matérn 5/2 is chosen to map the slowly changing global pattern in the data. Matérn- 5/2 is chosen over

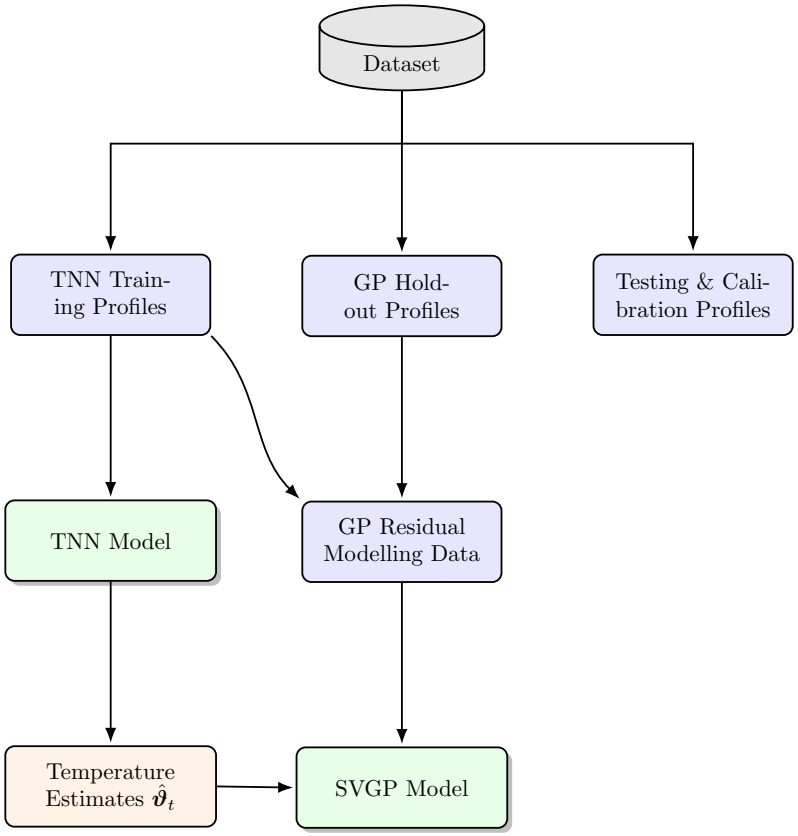

Fig. 3.6: Data split to minimize data leakage in residual modeling. Here ϑ_t shows the estimated temperatures at time step t.

the RBF kernel because they permit less restrictive smoothness assumptions and tends to model physical systems better [14], [51]. Matérn 1/2 is included to model rapidly varying local behavior and non-smooth residual structure. Their length scale (Fig . 2.8) is automatically tuned during training using automatic relevance determination (ARD) [14]. In addition to kernel hyperparameters, the observation noise variance (aleatoric uncertainty) is also learned during training. The number of inducing points is correlated with prediction accuracy, but comes at the cost of longer training and inference times. In this work, 1024 inducing points are adopted for final training. This number is chosen after a trial and error. Before training, these inducing points were placed using k-means clustering.

The learning rate is the only parameter treated as a hyperparameter here. The optimal learning rate is determined by trial and error without the use of optuna. Learning rate is searched between 0.0001 to 0.01 for 200 epochs. The learning rate is also halved after every 20 epochs. In this configuration, the best performing learning rate is 0.001.

3.7 HPO

This section discusses how HPO was implemented for the four models, introduces their search spaces, and presents the optimal configurations found. For HPO, Bayesian optimization with the TPE optimizer [52] is chosen and implemented using Optuna. It is also possible to use other optimizers, like GP. But, TPE is preferred because it can handle both numerical and categorical hyperparameters natively. Bayesian optimization was selected over grid or random search due to the prohibitively large size of the hyperparameter search space.

Prior to hyperparameter optimization (HPO), three profiles were reserved for testing and one profile for calibration. HPO was conducted using five-fold cross-validation for 120 epochs per configuration. The AdamW optimizer was applied to dropout and ensemble models to add L2 regularization. AdamW is preferred over Adam for L2 regularization [53]. Adam optimizer was utilized for EDL HPO. In EDL, a regularizer is already built into the loss function, so no additional regularizer was used. A fixed chunk size of 2048 was used, and the learning rate was reduced every third epoch across all four HPOs. Results from distributed trainings are recorded in a PostgreSQL database [54]. At least 2000 trials were conducted for each of the four HPOs.

For the dropout and deep-ensemble models, only a single parallel trial per configuration was conducted due to computational constraints. To mitigate the influence of stochastic effects in training, a multi-stage optimization procedure was employed. The top 50 and top 10 HPO configurations for dropout and deep ensembles, respectively, were re-evaluated using three individual training runs. The resulting objective values were averaged and used for comparison. Computational cost was not an issue with EDL because there is no sampling, so 3 parallel training runs were performed during the HPO itself.

Search space and optimal values for dropout are summarized in Table 3.4. Optimal values here are those found by comparing the top 50 trials. A full set of HPO plots is available in Appendix A.1. In this and subsequent HPO tables, g and p are used instead of γ and π respectively. For example, g_fc1 indicates the first layer of the γ backbone. Bottleneck refers to the uncertainty branch. Weight decay is a hyperparameter for L2 regularization with the AdamW optimizer.

Search space and optimal values for deep ensembles are summarized in Table 3.5. HPO optimal was also optimal after secondary optimization. A full set of HPO plots is available in Appendix A.2. Search space and optimal values for EDL with NLL as a loss function are summarized in Table 3.6. In this and subsequent HPO tables, e refers to the evidential network. For example, e_hidden_layers refers to hidden layers in the evidential network. A full set of HPO plots is available in Appendix A.3.

Search space and optimal values for EDL with CRPS as a loss function are summarized in Table 3.7. A full set of HPO plots is available in Appendix A.4.

Tab. 3.4: Hyperparameter search space and optimal configuration for dropouts: Dropout rates were the same in the main branch and the uncertainty branch.

Category	Hyperparameter	Search space	Optimal configuration
Conductance network	g_fc1_units	[2, 140]	77
	g_fc1_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
	g_hidden_layers	[0, 2]	1
	g_hidden_*i*_units	[2, 128]	59
	g_hidden_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	Softplus
	g_output_activation	{Softplus, relu, biased_elu, linear}	Softplus
	dropout_rate_g	[0.05, 0.5]	0.4218
Conductance uncertainty branch	bottleneck_hidden_layers_g	[1, 2]	2
	bottleneck_hidden_g_*i*_units	[2, 64]	60, 22
	bottleneck_hidden_g_*i*_activation*	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh, relu
	bottleneck_output_activation_g	{Softplus, relu, biased_elu, linear}	relu
Power loss network	p_fc1_units	[2, 140]	140
	p_fc1_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	sigmoid
	p_hidden_layers	[0, 2]	0
	p_hidden_*i*_units	[2, 128]	-
	p_hidden_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	-
	p_output_activation	{Softplus, relu, biased_elu, linear}	Softplus
	dropout_rate_p	[0.05, 0.5]	0.2894
Power loss uncertainty branch	bottleneck_hidden_layers_p	[1, 2]	2
	bottleneck_hidden_p_*i*_units	[2, 64]	29, 17
	bottleneck_hidden_p_*i*_activation*	{sigmoid, Softplus, tanh, relu, biased_elu}	biased_elu, sigmoid
	bottleneck_output_activation_p	{Softplus, relu, biased_elu, linear}	linear
Training	learning_rate	[1e-4, 1e-2]	5.5e-3
	weight_decay	[1e-6, 1e-2]	8.6e-3
	lr_decay	[0.90, 1.0]	0.91
	tbptt_size	[16, 2048]	398
	batch_size	[16, 128]	16

Tab. 3.5: Hyperparameter search space and optimal configuration for deep ensembles.

Category	Hyperparameter	Search space	Optimal configuration
Conductance network	g_fc1_units	[2, 140]	137
	g_fc1_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
	g_hidden_layers	[0, 2]	1
	g_hidden_*i*_units	[2, 128]	116
	g_hidden_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	relu
	g_output_activation	{Softplus, relu, biased_elu, linear}	Softplus
Conductance uncertainty branch	bottleneck_hidden_layers_g	[1, 2]	1
	bottleneck_hidden_g_*i*_units	[2, 64]	32
	bottleneck_hidden_g_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
	bottleneck_output_activation_g	{Softplus, relu, biased_elu, linear}	linear
Power loss network	p_fc1_units	[2, 140]	97
	p_fc1_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
	p_hidden_layers	[0, 2]	1
	p_hidden_*i*_units	[2, 128]	18
	p_hidden_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	Softplus
	p_output_activation	{Softplus, relu, biased_elu, linear}	Softplus
Power loss uncertainty branch	bottleneck_hidden_layers_p	[1, 2]	2
	bottleneck_hidden_p_*i*_units	[2, 64]	{58, 64}
	bottleneck_hidden_p_*i*_activation	{sigmoid, Softplus, tanh, relu, biased_elu}	{sigmoid, tanh}
	bottleneck_output_activation_p	{Softplus, relu, biased_elu, linear}	relu
Training	learning_rate	[1e-4, 1e-2]	5.53e-3
	weight_decay	[1e-6, 1e-2]	1.30e-4
	lr_decay	[0.90, 1.0]	0.92
	tbptt_size	[16, 2048]	542
	batch_size	[16, 128]	18

Tab. 3.6: Hyperparameter search space and optimal configuration for EDL (NLL Loss)

Category	Hyperparameter	Search space	Optimal configuration
Conductance network	`g_fc1_units`	[2, 128]	15
	`g_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	sigmoid
	`g_hidden_layers`	[0, 2]	1
	`g_hidden_`i`_units`	[2, 128]	105
	`g_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
	`g_output_activation`	{Softplus, relu, biased_elu, linear}	biased_elu
Power loss network	`p_fc1_units`	[2, 128]	108
	`p_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	relu
	`p_hidden_layers`	[0, 2]	1
	`p_hidden_`i`_units`	[2, 128]	21
	`p_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	Softplus
	`p_output_activation`	{Softplus, relu, biased_elu, linear}	biased_elu
Evidential network	`e_fc1_units`	[2, 128]	3
	`e_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	Softplus
	`e_hidden_layers`	[0, 2]	0
	`e_hidden_`i`_units`	[2, 128]	-
	`e_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	-
Training	`learning_rate`	[1e-4, 1e-2]	8.64e-3
	`lambda_reg`	[1e-4, 1e-2]	8.24e-3
	`lr_decay`	[0.90, 1.0]	0.9078
	`tbptt_size`	[16, 2048]	1038
	`batch_size`	[16, 128]	90

Tab. 3.7: Hyperparameter search space and optimal configuration for EDL (CRPS loss)

Category	Hyperparameter	Search space	Optimal configuration
Conductance network	`g_fc1_units`	[2, 128]	51
	`g_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	biased_elu
	`g_hidden_layers`	[0, 2]	0
	`g_hidden_`i`_units`	[2, 128]	-
	`g_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	-
	`g_output_activation`	{Softplus, relu, biased_elu, linear}	relu
Power loss network	`p_fc1_units`	[2, 128]	40
	`p_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	biased_elu
	`p_hidden_layers`	[0, 2]	0
	`p_hidden_`i`_units`	[2, 128]	-
	`p_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	-
	`p_output_activation`	{Softplus, relu, biased_elu, linear}	Softplus
Evidential network	`e_fc1_units`	[2, 128]	53
	`e_fc1_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	sigmoid
	`e_hidden_layers`	[0, 2]	1
	`e_hidden_`i`_units`	[2, 128]	49
	`e_hidden_`i`_activation`	{sigmoid, Softplus, tanh, relu, biased_elu}	tanh
Training	`learning_rate`	[1e-4, 1e-2]	4.18e-3
	`lambda_reg`	[1e-4, 1e-2]	2.20e-3
	`lr_decay`	[0.90, 1.0]	0.9165
	`tbptt_size`	[16, 2048]	648
	`batch_size`	[16, 128]	16

3.8 Uncertainty calibration

As mentioned in Sec.2.8, raw uncertainty estimates should not be used without calibration. In this work, an uncertainty toolbox (version 0.1.1) [39] is used to calibrate uncertainty and plot reliability diagrams for dropout, deep ensembles, and GP. For EDL, a custom calibration function is written because the toolbox [39] does not support the Student-t distribution. In either case, a scaling factor, as discussed in Sec. 2.8, is calculated for each target such that the miscalibration area on the calibration profile is minimized. Predicted uncertainties are then scaled by this scaling factor.

4 Results

This section evaluates the presented UQ methods and the proposed architecture on a held-out test set (profiles 60, 62, and 74). This assessment is performed after applying uncertainty calibration on a separate held-out calibration set (profile 36). Metrics including CRPS, NLL, MSE, and the miscalibration area are used to measure the quality of the predictive distributions and the impact of the UQ enhancements on overall accuracy. Additionally, inference time and model storage size have been compared to evaluate real-time capabilities on edge devices. Finally, performance on test sets is visualized, and reliability diagrams are presented.

Both CRPS and NLL quantify the relationship between the prediction error and the predicted uncertainty at a given point. However, NLL relies on a quadratic penalty, whereas CRPS uses a linear penalty, making NLL highly sensitive to a few outliers. CRPS is more robust by comparison. Nonetheless, analyzing them together is critical, a model exhibiting high NLL alongside a low CRPS can indicate issues about the quality of uncertainty. Such models should not be used in safety-critical applications. The miscalibration area, on the other hand, strictly evaluates the quality of the calibration, ensuring that the predicted uncertainty bounds are neither too narrow nor excessively wide. Unlike CRPS and NLL, the miscalibration area does not take the magnitude of the errors into account. This makes it a particularly robust metric for evaluating targets where point-prediction errors are inherently low, such as the stator yoke. Together, these metrics allow for a comprehensive and robust analysis. The quantitative results across the evaluated metrics are presented in Tab. 4.1.

As demonstrated in the data, GP significantly outperformed the other models in terms of uncertainty calibration, achieving the best overall NLL, CRPS, and miscalibration area. The slightly higher MSE observed for the GP approach can be attributed to the baseline TNN model, which possessed an inherent average MSE of approximately 3.4 $°C^2$ (slightly less than as presented in Sec. 3.2, because of data splitting as described in Sec. 3.6). The fact that the MSE dropped after residual modeling indicates that the GP successfully captured residual patterns that the TNN missed.

Dropout emerged as the strongest alternative, with excellent MSE and a similar CRPS score. The two EDL approaches yielded comparable results. Even though NLL-optimized EDL performed slightly better on average, its larger miscalibration area on the permanent

Component	Metric	Deep ensembles	Dropout	EDL (NLL)	EDL (CRPS)	GP
Permanent magnet	CRPS	1.31	**1.08**	1.4	1.29	1.16
	NLL	11.67	4.67	2.62	2.47	**2.13**
	miscal	0.29	0.18	0.26	0.13	**0.04**
	MSE	3.80	**2.93**	4.76	4.37	4.12
Stator yoke	CRPS	0.62	0.61	**0.586**	0.77	0.59
	NLL	1.87	1.53	1.5	1.84	**1.46**
	miscal	0.13	0.04	0.05	0.15	**0.03**
	MSE	1.08	1.09	**1.05**	1.37	1.09
Stator tooth	CRPS	0.81	0.77	0.78	0.74	**0.69**
	NLL	2.51	1.83	1.77	1.79	**1.64**
	miscal	0.19	0.06	**0.03**	0.15	**0.03**
	MSE	1.83	1.79	1.92	**1.45**	1.53
Stator windings	CRPS	0.75	0.71	0.72	**0.69**	0.7
	NLL	2.17	1.7	1.67	1.69	**1.64**
	miscal	0.15	0.07	**0.02**	0.15	**0.02**
	MSE	1.7	**1.5**	1.7	1.34	1.55
Average	CRPS	0.87	**0.79**	0.87	0.87	**0.79**
	NLL	4.6	2.43	1.89	1.95	**1.72**
	miscal	0.19	0.088	0.09	0.145	**0.03**
	MSE	2.1	**1.83**	2.36	2.13	2.07

Tab. 4.1: Model performance metrics with averages across components.

magnet may be an issue. Finally, deep ensembles, as trained in this study, are not recommended for this application. The exceptionally high NLL on the permanent magnet indicates that the ensemble makes highly confident but erroneous predictions.

Compared with [5], all tested models have lower MSEs than those reported there, indicating a positive effect on MSE from UQ enhancement.

Tab. 4.2 shows the inference time per prediction in milliseconds(ms) and model storage requirements in Megabytes(MB) based on inference on an AMD Ryzen 7 7840 U processor. As seen from the data, EDL provides uncertainty estimates with negligible computational overhead. Tab. 4.3 gives aleatoric uncertainties for all four targets as

Model	Inference time (ms)	Storage requirements (MB)
Baseline TNN	0.019	0.025
EDL (CRPS)	0.034	0.054
EDL (NLL)	0.036	0.058
GP	0.330	17.1
Deep Ensembles	1.360	1.8
Dropout	1.460	0.12

Tab. 4.2: Inference time per prediction and storage size across models. The values for deep ensembles reflect 10 constituent models, whereas the inference time for Dropout reflects 20 forward passes.

Component	Deep Ensembles	Dropout	EDL (NLL)	EDL (CRPS)	GP
Permanent magnet	0.089	0.18	0.75	0.53	1.62
Stator yoke	0.32	0.44	0.82	0.50	0.71
Stator tooth	0.24	0.47	0.88	0.59	0.99
Stator windings	0.32	0.49	0.68	0.56	0.91

Tab. 4.3: Aleatoric uncertainty in °C. For Deep ensembles, dropouts, and GP, the first standard deviation is shown (Normal distribution). For the EDL-based model, the corresponding certainty bound is shown (0.68 certainty for Student-t distribution)

predicted by different models. The value in the table represents the calibrated standard deviation for all models except EDL-based models. For EDL-based models, the Student-t model's 68% certainty value is used. In the absence of the true value of sensor noise for this dataset, the following interpretation is speculative. Typically, among all the components, measuring the temperature of the permanent magnet is most challenging due to the spinning rotor. So, in theory, aleatoric uncertainty should be highest for the permanent magnet compared to other components. This dynamic is only faithfully captured by the GP, though the values obtained are on the higher side compared to what is typically seen in the sensor datasheet. One possible explanation of this mismatch

could be that the designed kernel was not sufficiently able to capture all residual dynamics.

Figures 4.1, 4.2, 4.3, and 4.4 show the ground truth, predicted mean, and 95% certain bounds for permanent magnet, stator yoke, stator tooth, and stator windings, respectively.

Similarly, the reliability diagrams for Deep ensembles, dropouts, EDL (NLL), EDL (CRPS), and GP are shown in Fig. 4.5, 4.6,4.7, 4.8, and 4.9, respectively.

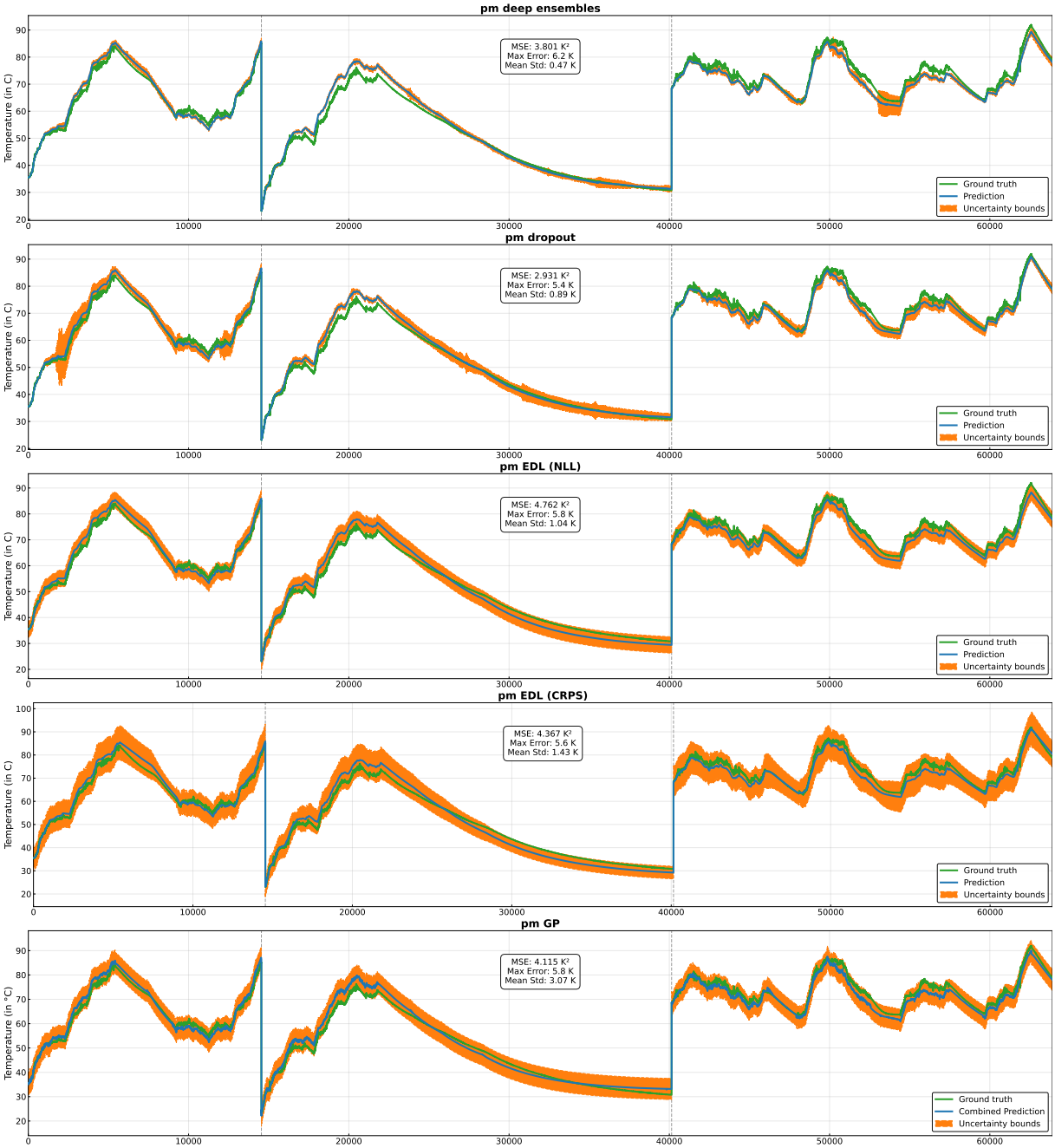

Fig. 4.1: Comparison of ground-truth and predicted permanent magnet (PM) temperatures for all five UQ models with corresponding 95% certainty bounds.

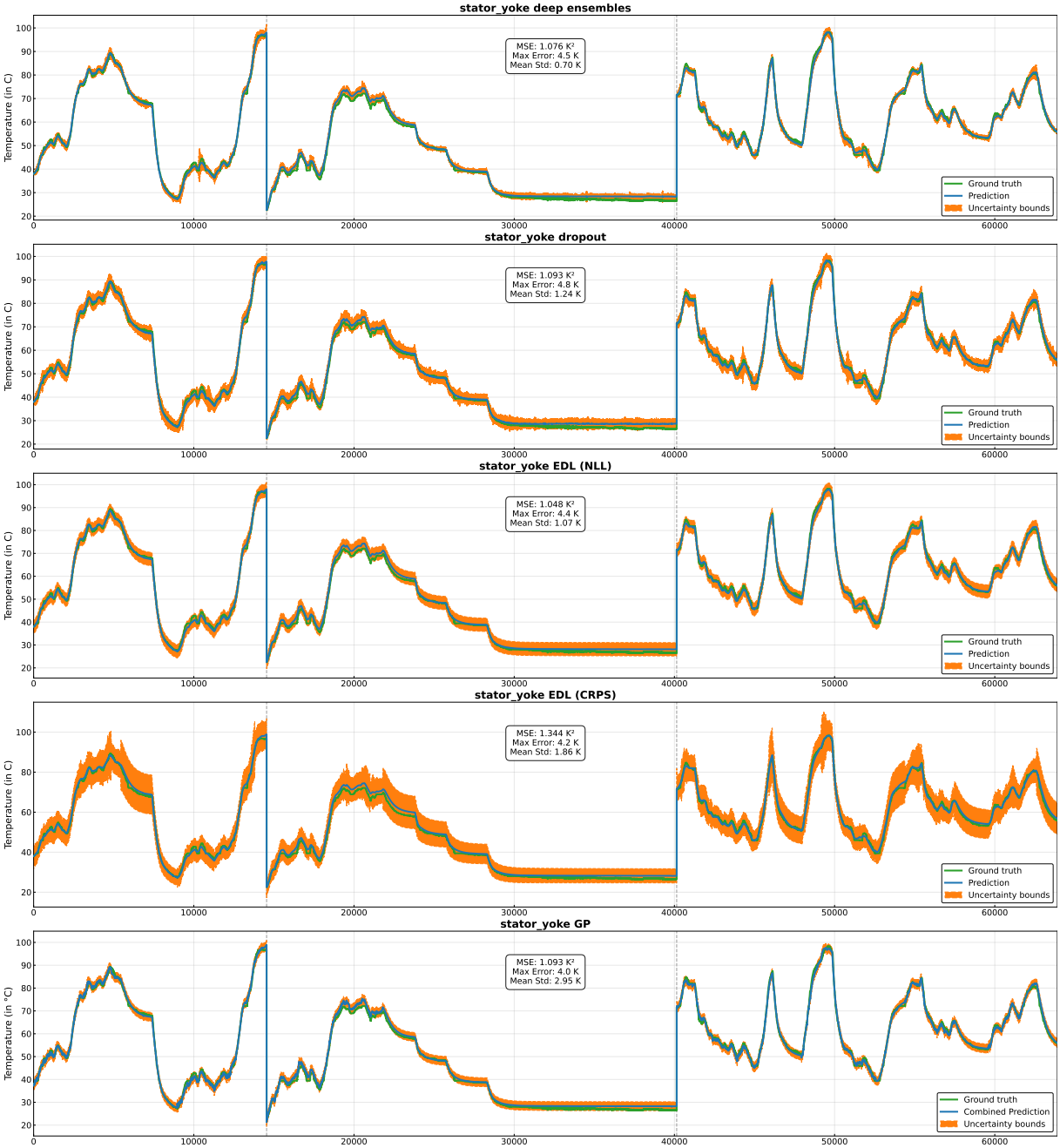

Fig. 4.2: Comparison of ground-truth and predicted stator yoke temperatures for all five UQ models with corresponding 95% certainty bounds.

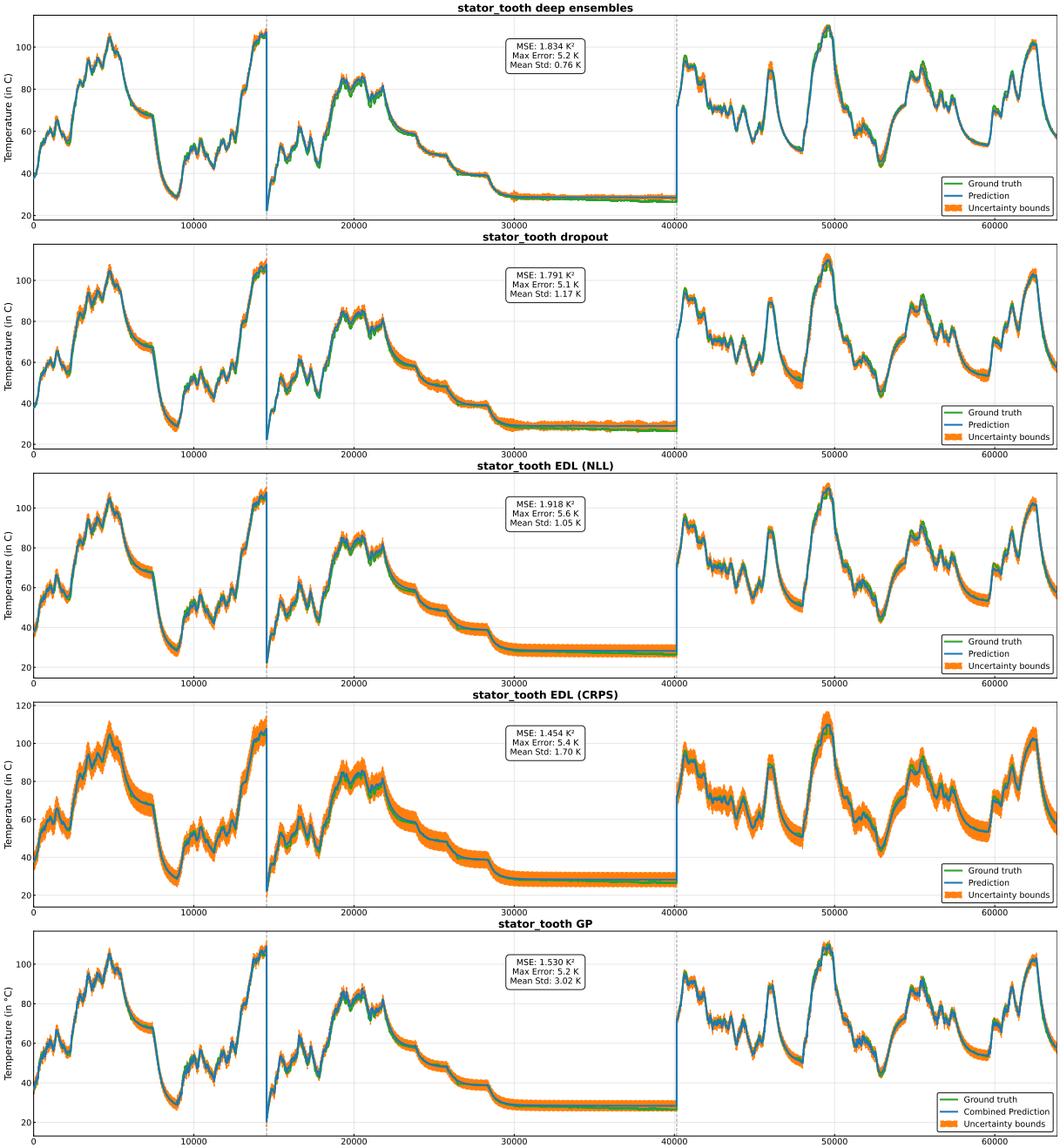

Fig. 4.3: Comparison of ground-truth and predicted stator tooth temperatures for all five UQ models with corresponding 95% certainty bounds.

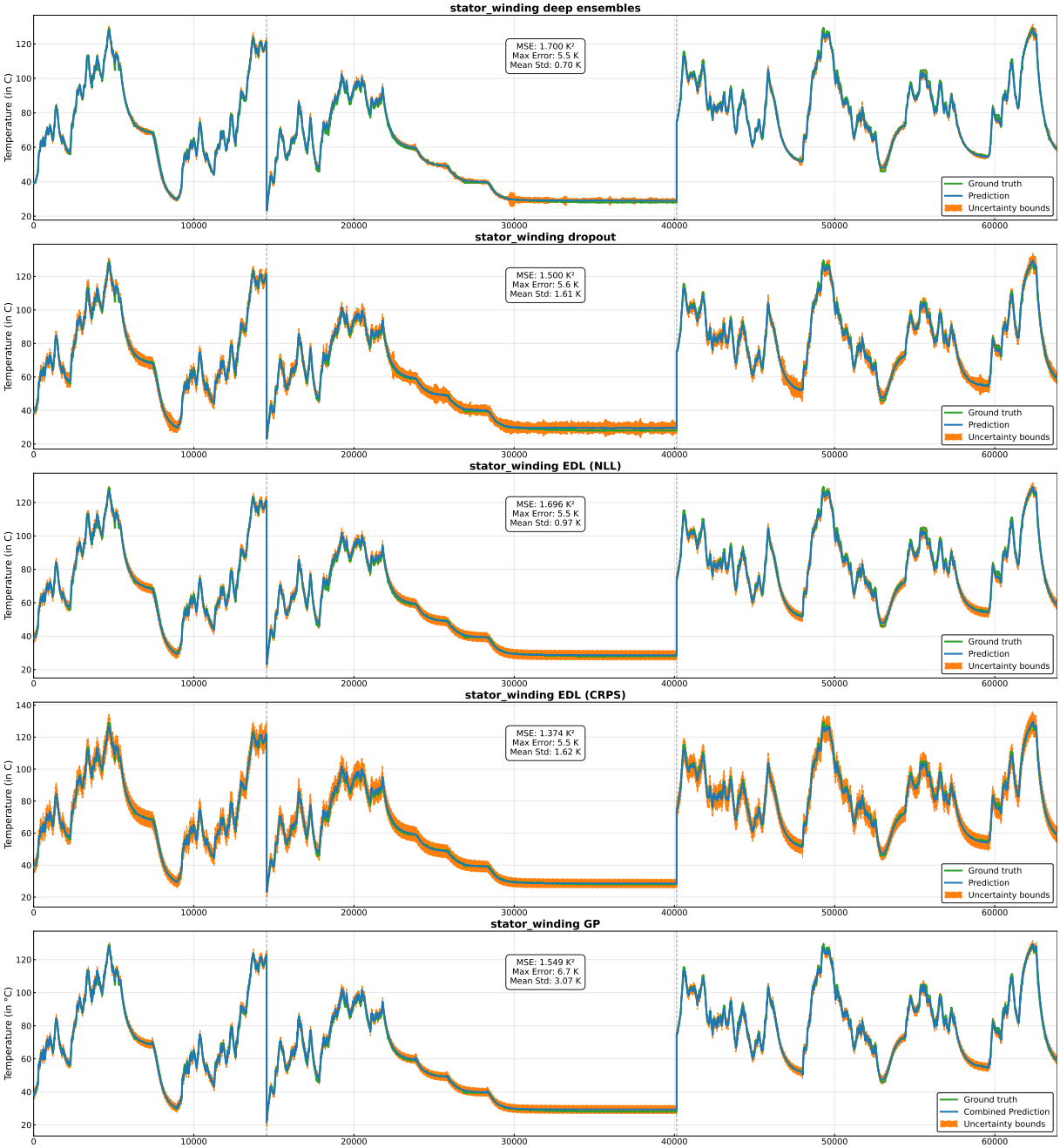

Fig. 4.4: Comparison of ground-truth and stator windings temperatures for all five UQ models with corresponding 95% certainty bounds.

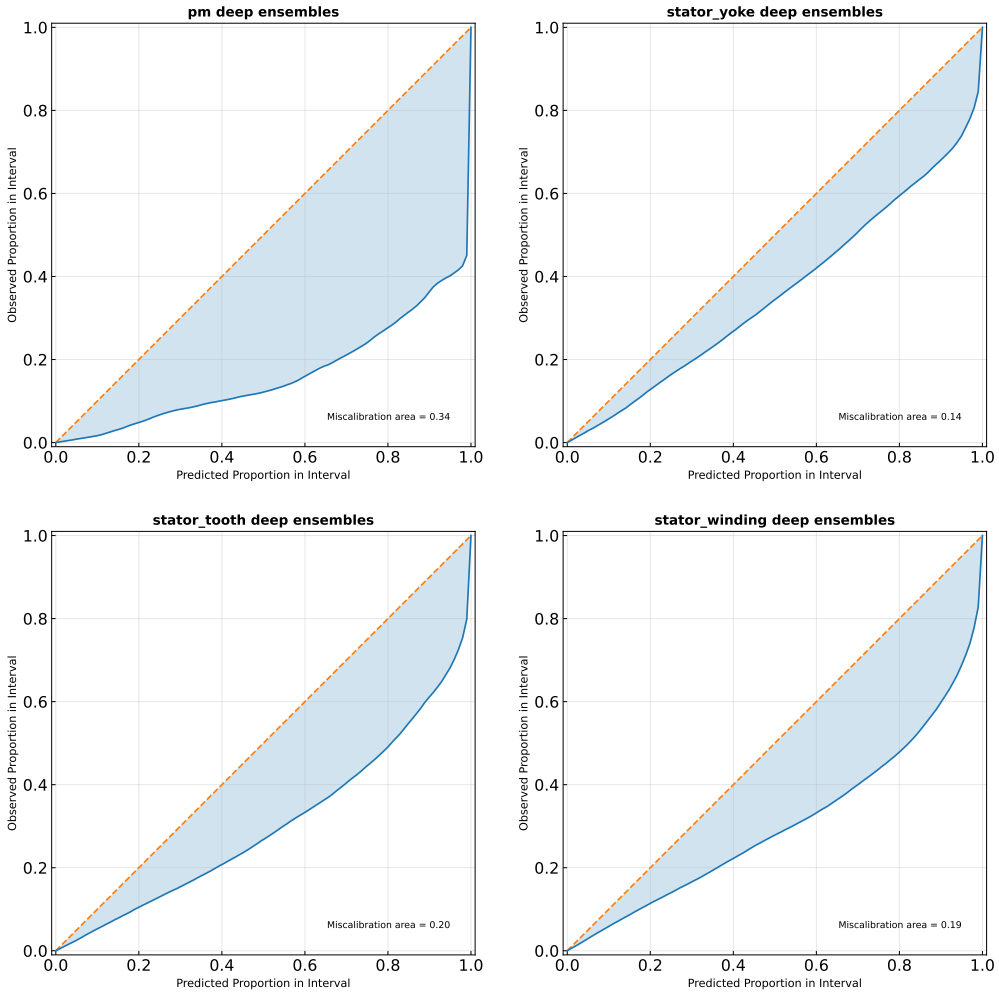

Fig. 4.5: Reliability diagram from deep ensembles. Based on the reliability diagram, it can be seen that deep ensembles have been underconfident.

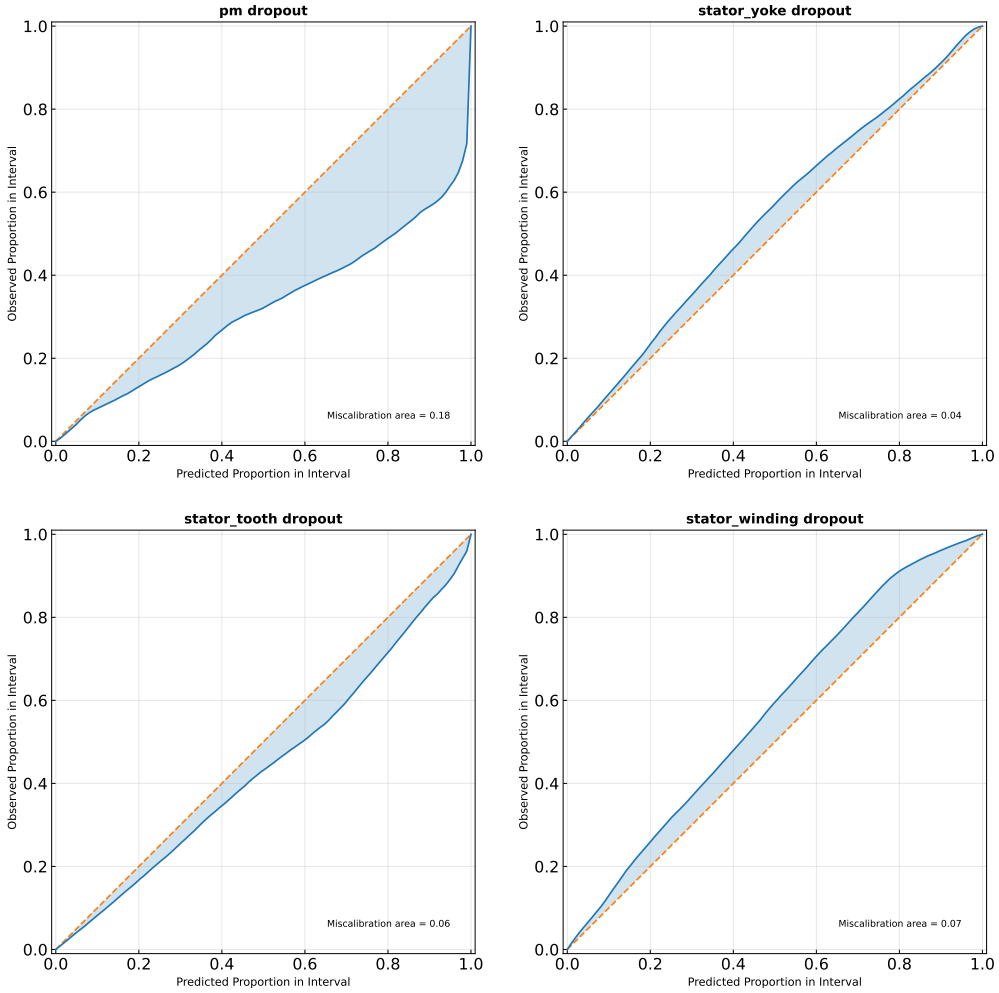

Fig. 4.6: Reliability diagram from Dropout. Based on the reliability diagram, it can be seen that dropout is overconfident for the permanent magnet but well-calibrated for the stator yoke, tooth, and winding.

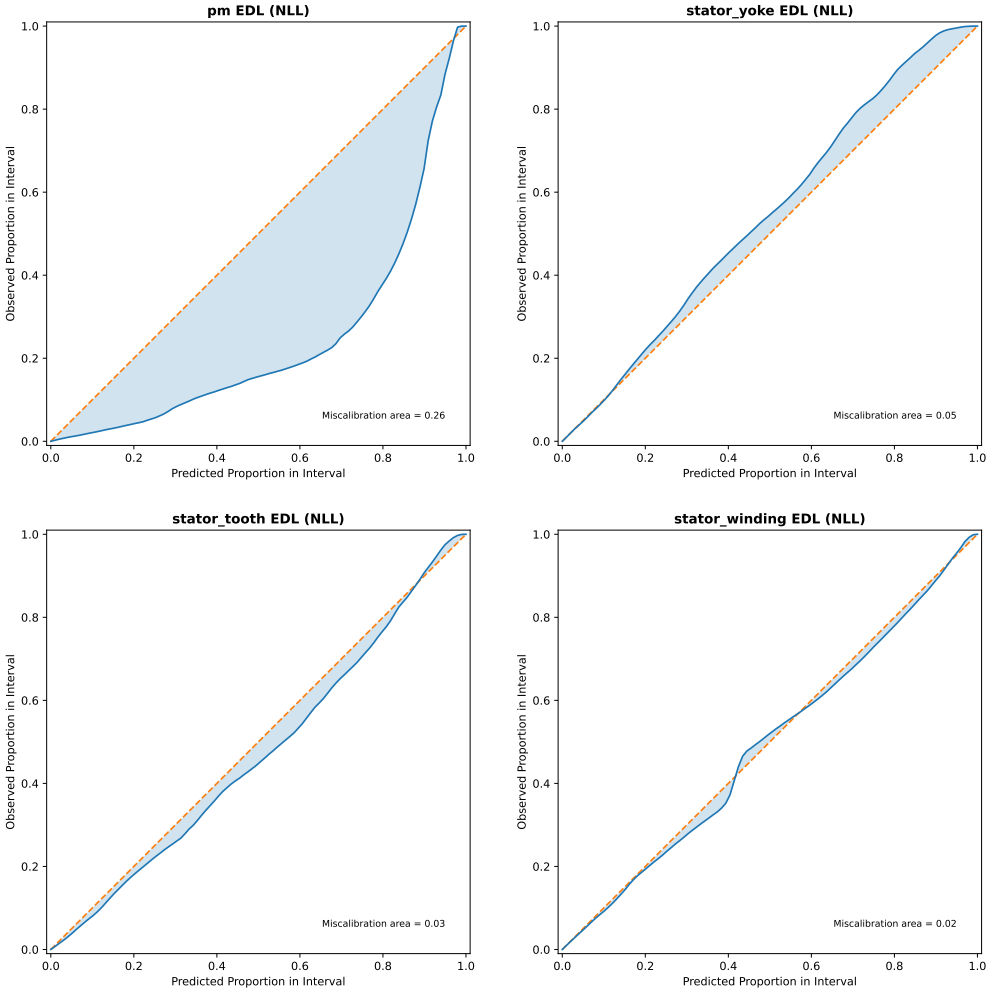

Fig. 4.7: Reliability diagram from EDL (NLL). Based on the reliability diagram, it can be seen that EDL (NLL) is overconfident for the permanent magnet, but highly calibrated for the stator yoke, tooth, and winding.

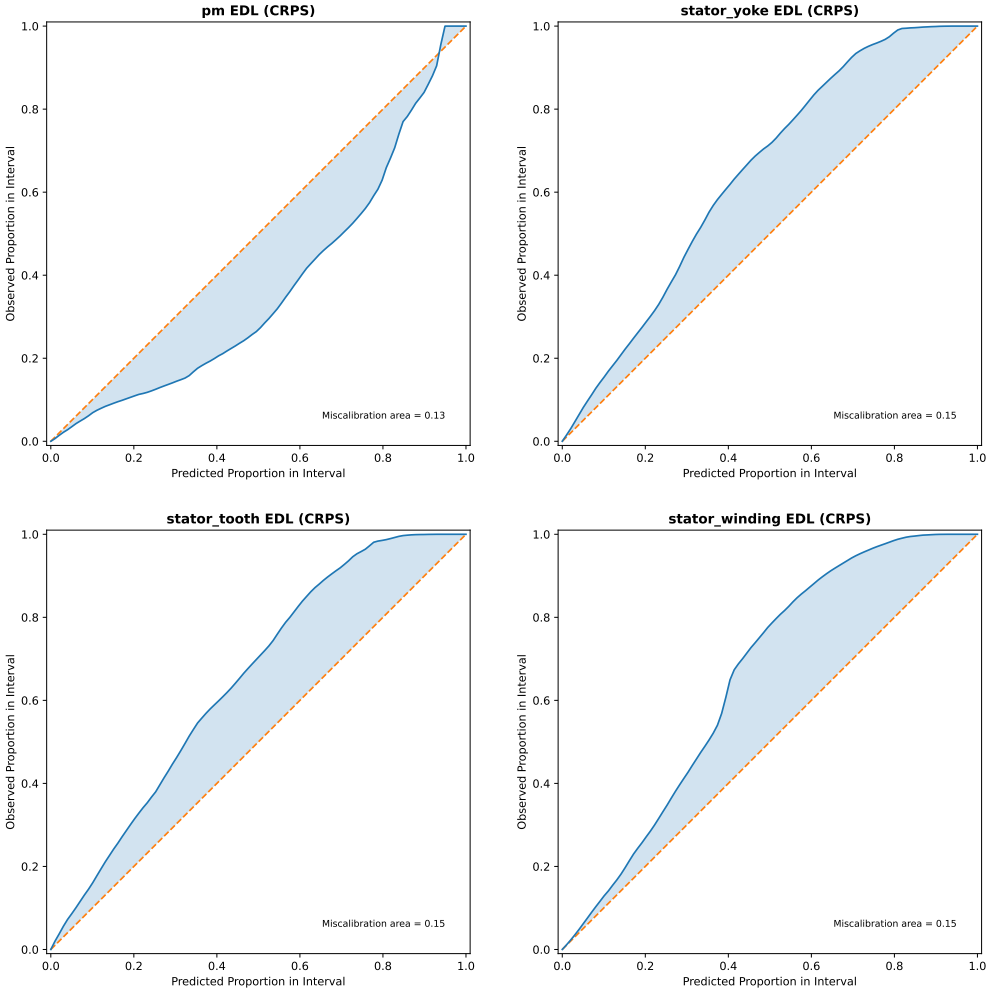

Fig. 4.8: Reliability diagram from EDL (CRPS). Based on the reliability diagram, it can be seen that EDL (CRPS) is overconfident for the permanent magnet, but underconfident for the stator yoke, tooth, and winding.

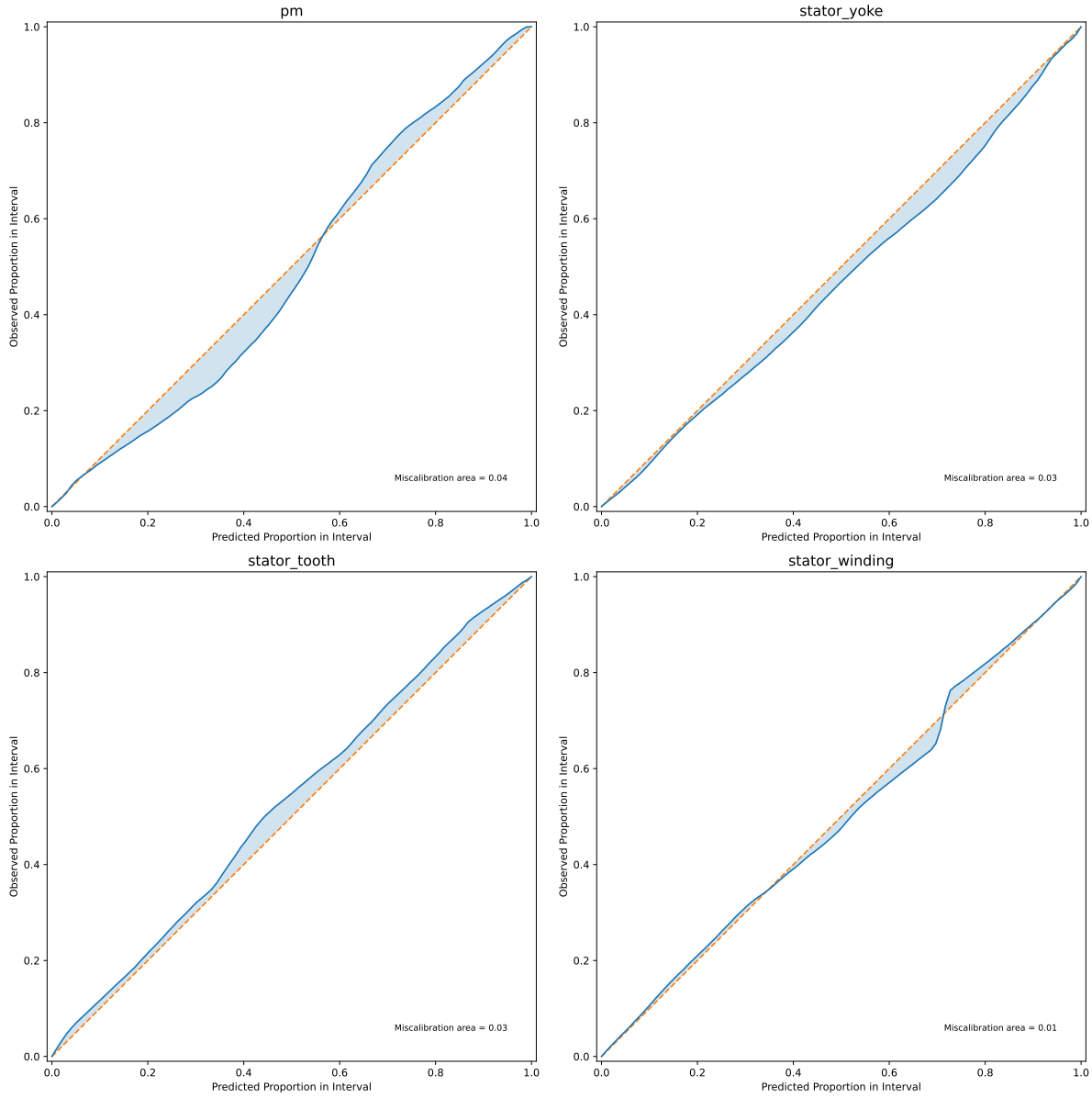

Fig. 4.9: Reliability diagram for GP. Based on the reliability diagram, GP is highly calibrated.

5 Conclusion

Accurate temperature estimation in PMSMs is crucial to avoid catastrophic failures such as magnet demagnetization or winding insulation damage. This work investigated five UQ models: deep ensembles, dropouts, EDL with NLL loss, EDL with CRPS loss, and residual modeling with GP (SVGP approximation) on TNN, a SOTA data-driven motor drive temperature estimation model. These models, besides GP, were integrated into TNN using novel architectural designs. For deep ensembles and dropouts, a branched architecture was used. This choice helped retain the physical interpretability of TNN. For EDL-based models, to bypass the limitations of the standard EDL regularizer when ground truth $\boldsymbol{\gamma}$ and $\boldsymbol{\pi}$ parameters are unavailable, this work introduced a novel evidential feedback architecture. All models presented in this work predicted both aleatoric and epistemic uncertainties.

The models were then compared in terms of MSE, CRPS, NLL, and the miscalibration area. GP came out on top with the lowest overall NLL, CRPS, and miscalibration area. It also had the lowest NLL across all four targets, making it suitable for safety-critical thermal modeling of a PMSM. Dropout was the best-performing model in terms of MSE and had a similar overall CRPS score to GP, but its performance was noticeably worse for the permanent magnet. This work demonstrated that Deep Ensembles, with the proposed architecture, are unsuitable for UQ in TNN because of their poor NLL for the permanent magnet. EDL-based models didn't perform as well as GP, but their inference time is only marginally worse than that of standard TNN, and hence provides uncertainty bounds with relatively low computational overhead.

In addition to quantifying uncertainties in thermal modeling of PMSM, this work also identified a research gap in the current UQ literature. To the best of the author's knowledge, there is no published literature that uses CRPS as a loss function for EDL. This work successfully trained an EDL with CRPS and obtained results similar to those of the standard NLL-based EDL implementation, proving it as a viable option for EDL.

5.1 Limitations and Future Work

Despite the promising results, several limitations and opportunities for future work remain. This study assumed no uncertainty in the inverse capacitance $\boldsymbol{\kappa}$ and treated it as

a learned parameter. A natural extension would be to model uncertainty in $\boldsymbol{\kappa}$ directly by learning a distribution over its values and sampling accordingly during inference.

Because of the large number of models being trained, the computational demands of HPO, and the time constraints, some assumptions were made that could lead to suboptimal convergence. For all HPOs, only a single optimizer (Adam or AdamW) was tested. As already discussed in Sec. 3.7, for deep ensembles and dropouts, only a single parallel trial was used for HPO. Even though a multi-stage optimization was performed using the top 10/50 trials, there is a possibility that the optimal value did not make it into the top 10/50. Furthermore, the kernel choice for GP was manually engineered, and the learning rate was selected by trial and error from a small set of rates. To address both of these limitations, an HPO with a larger search space and parallel trials is recommended. For GP, the search space could include multiple kernel combinations and learning rates.

This work arbitrarily split profiles for GP residual training as detailed in Sec. 3.6. Similarly, a single profile was arbitrarily chosen for calibration. Choosing a single profile risks not capturing all the operating dynamics. Although excellent miscalibration area results across multiple models proved that this was not a limiting factor, it has been disclosed here for transparency.

This work treated predictions across all temperature ranges as similar, even though mostly high temperatures are critical. Future work could evaluate the predicted uncertainties using a weighted score that penalizes low uncertainty and high error more severely in high-temperature regions.

Finally, to extend the novel integration of CRPS as a loss function for EDL, future work could benchmark its performance on the standard regression datasets used in the foundational EDL regression literature [28].

Appendix

The following sections contain plots from the HPO for dropouts, deep ensembles, and both versions of EDL

A.1 HPO plots for dropout

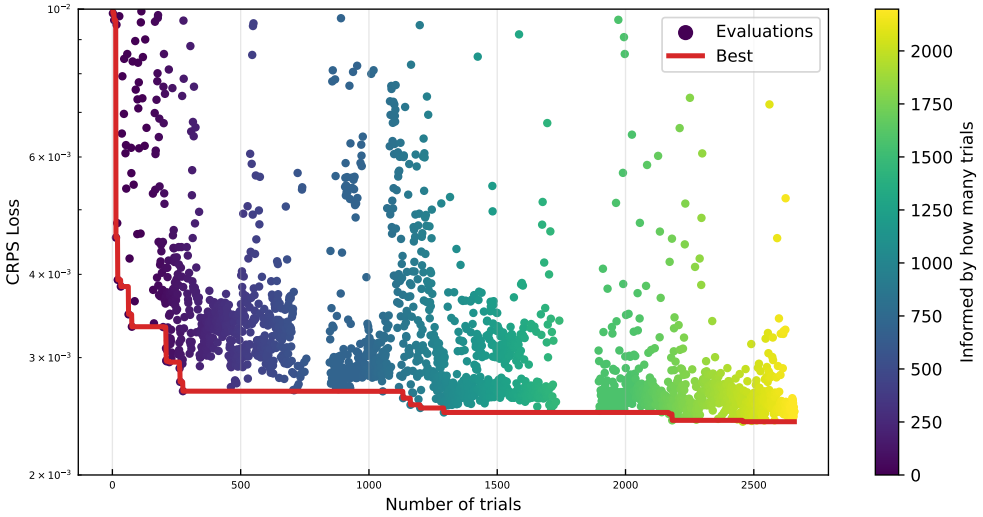

Fig. A.1: HPO progress plot for dropout

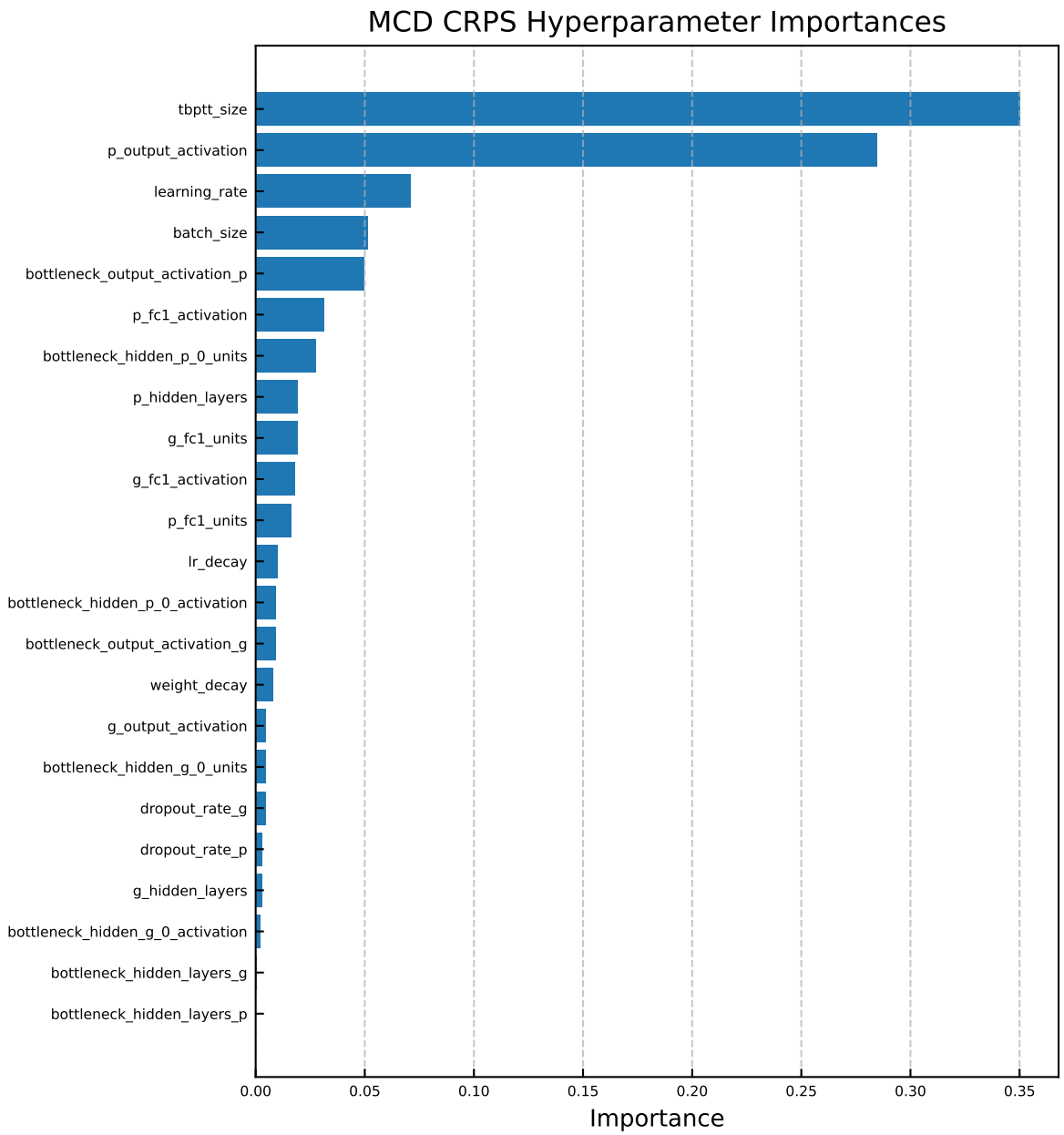

Fig. A.2: Hyperparameter importance plot for dropout

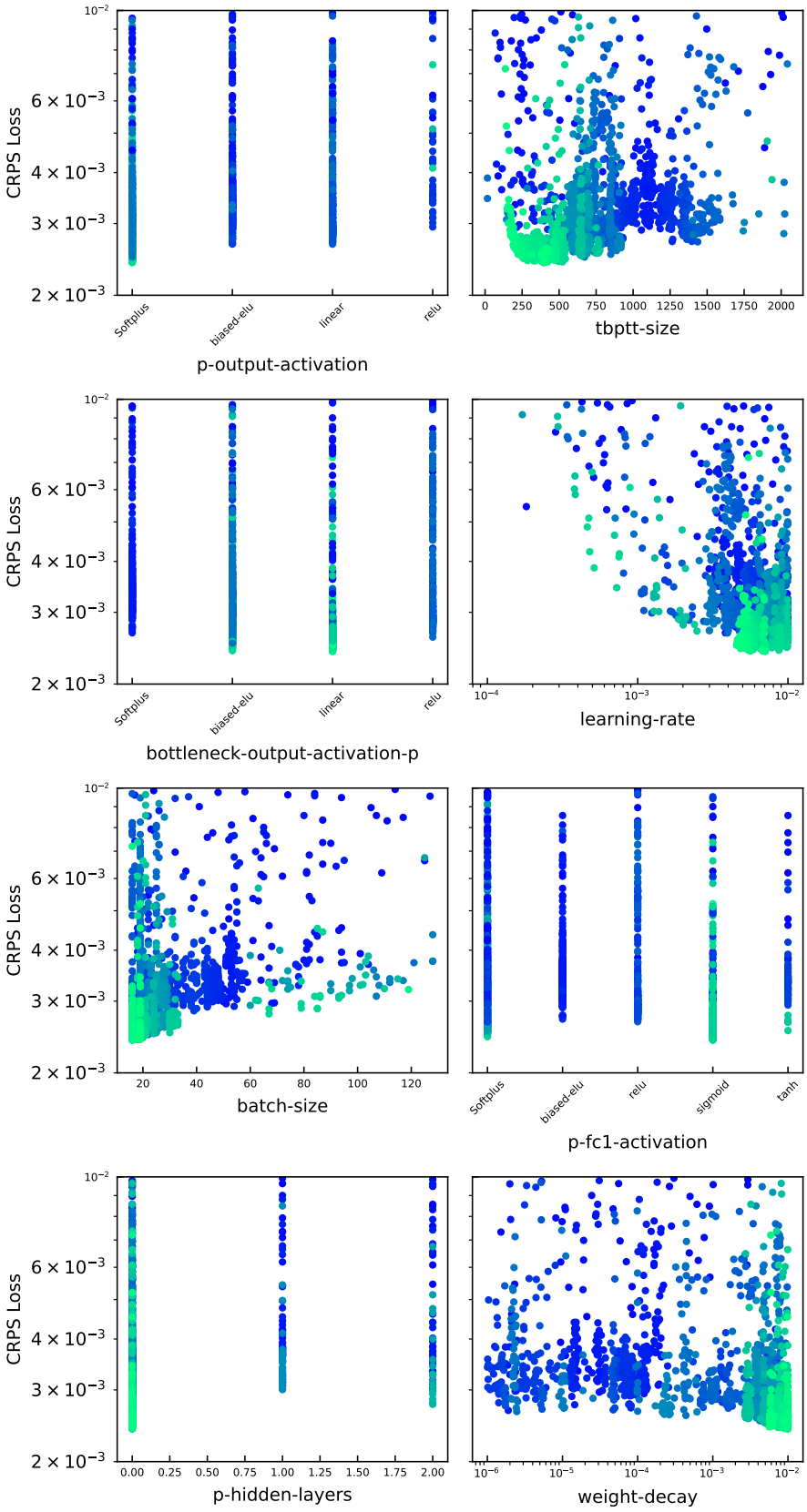

Fig. A.3: Detailed HPO plots for dropout (Part 1): Colour indicates the number of trials, blue being the initial trials and green being later trials.

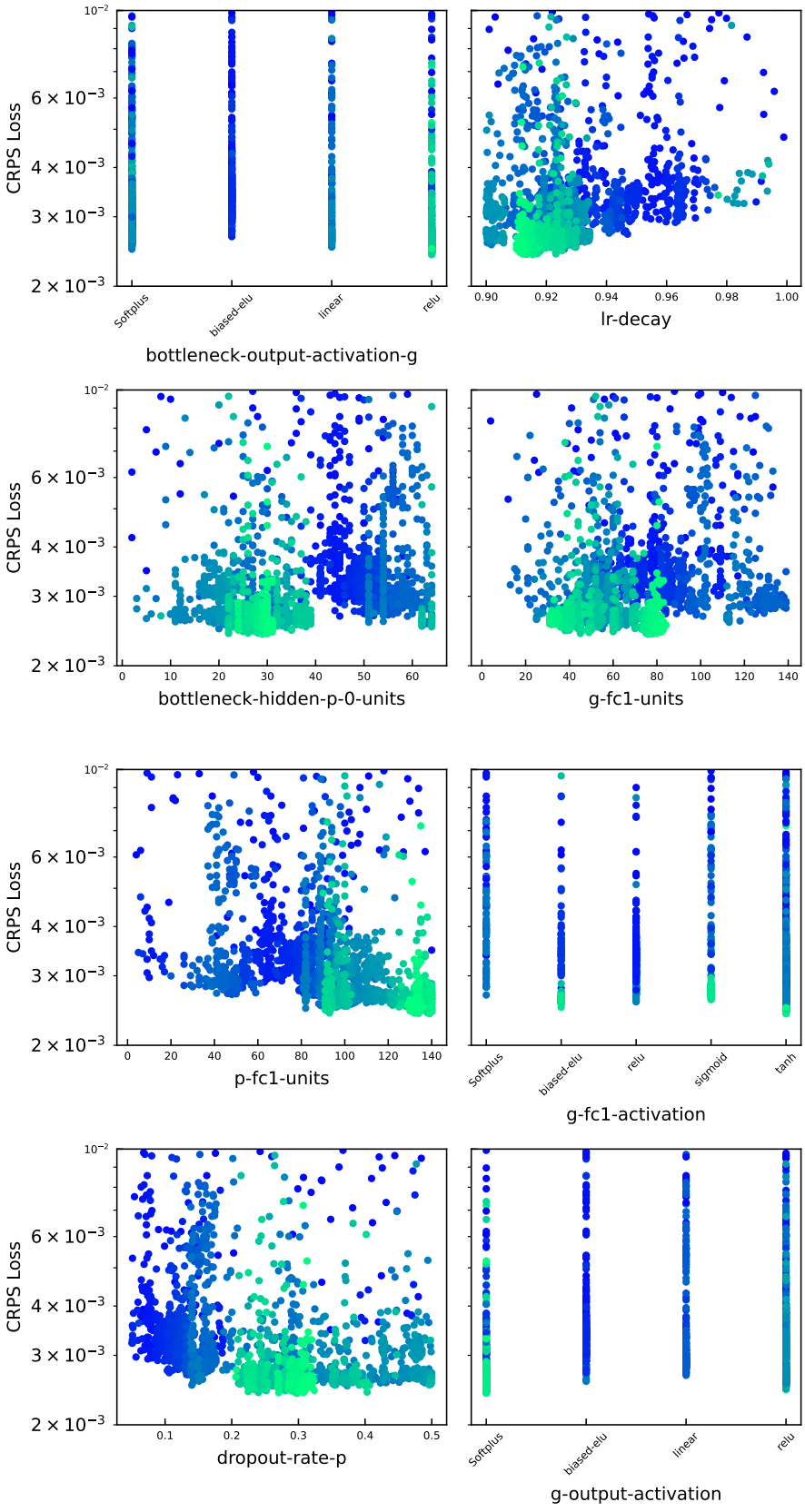

Fig. A.4: Detailed HPO plots for dropout (Part 2)

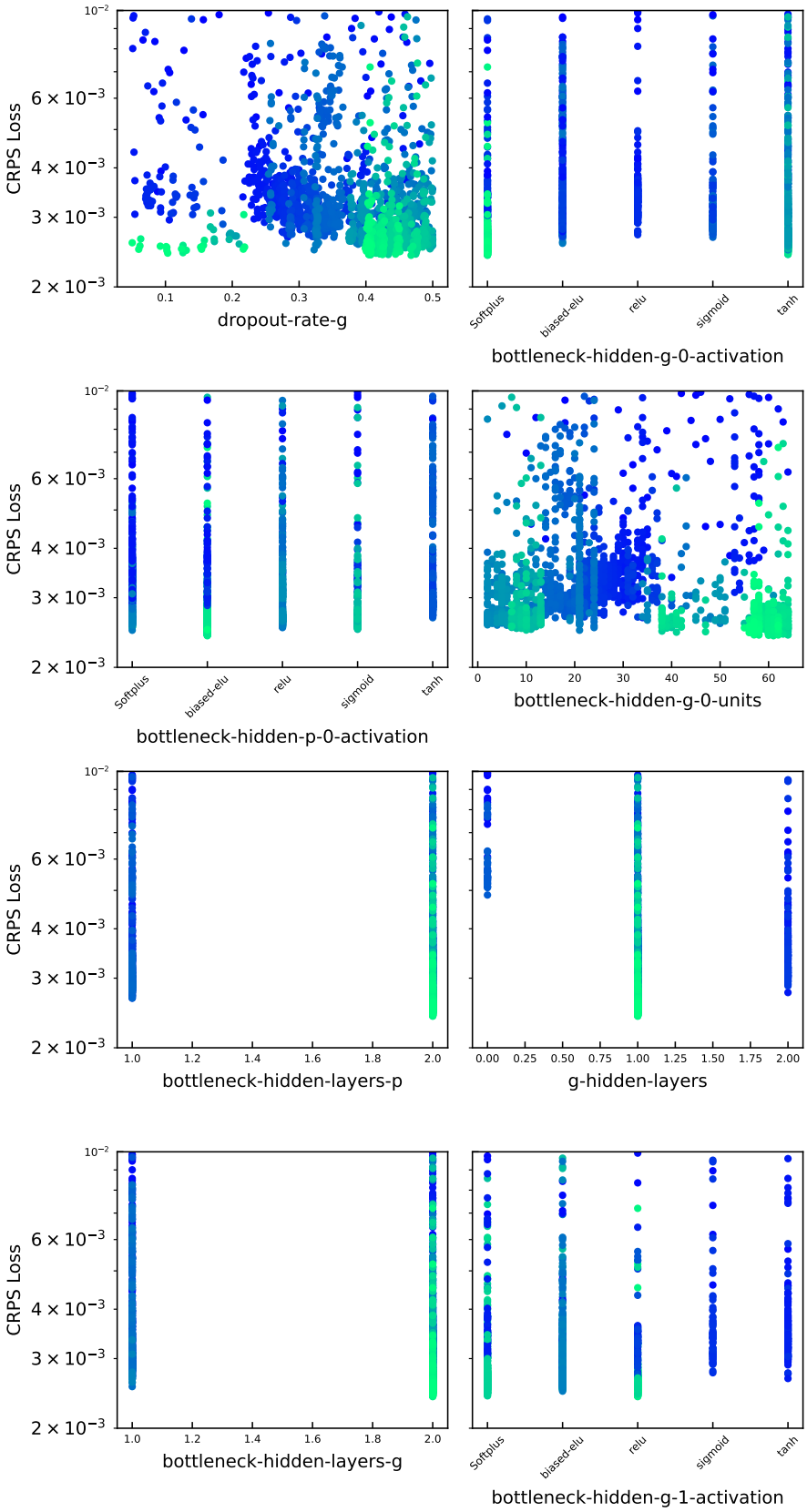

Fig. A.5: Detailed HPO plots for dropout (Part 3)

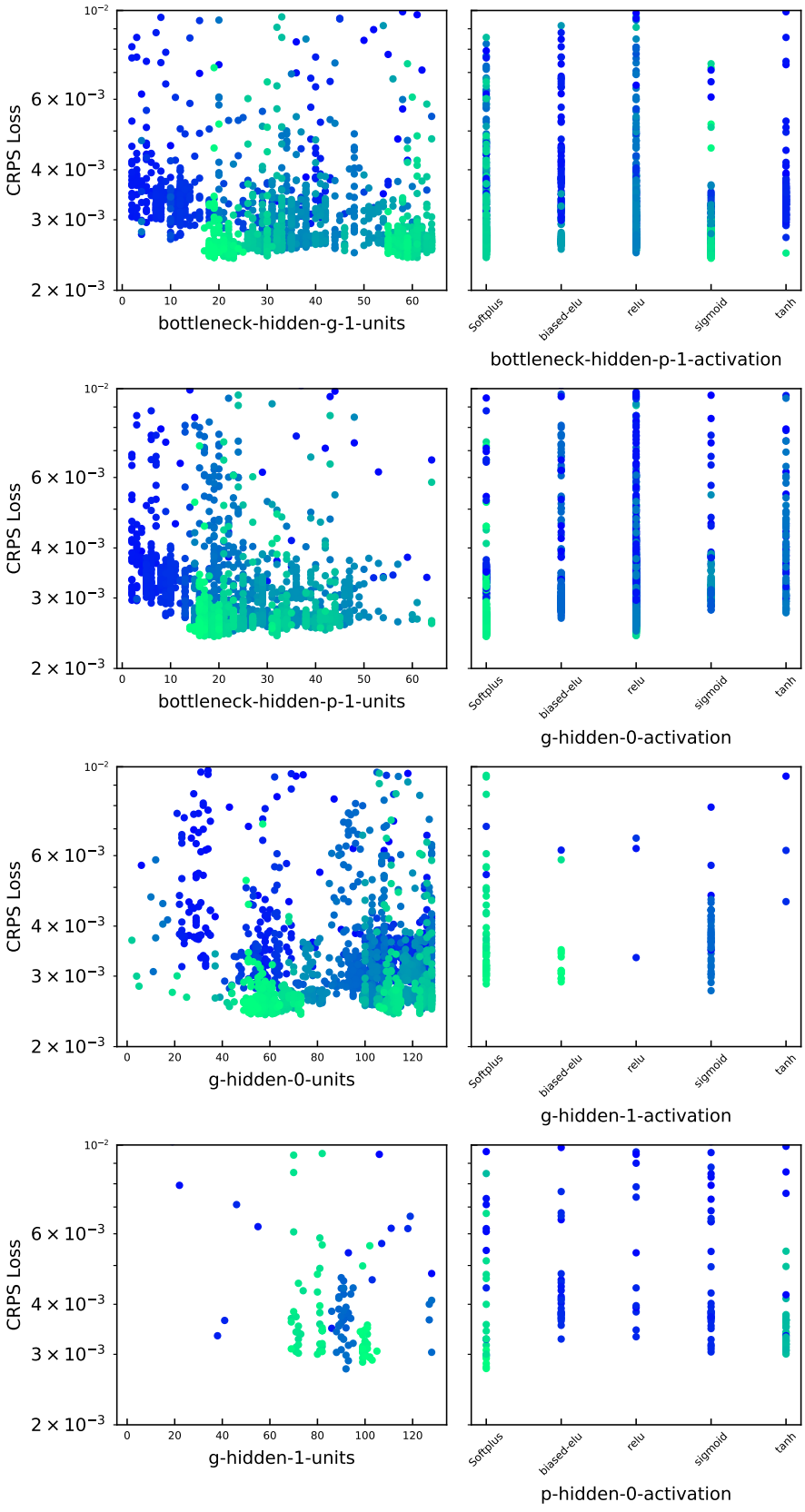

Fig. A.6: Detailed HPO plots for dropout (Part 4)

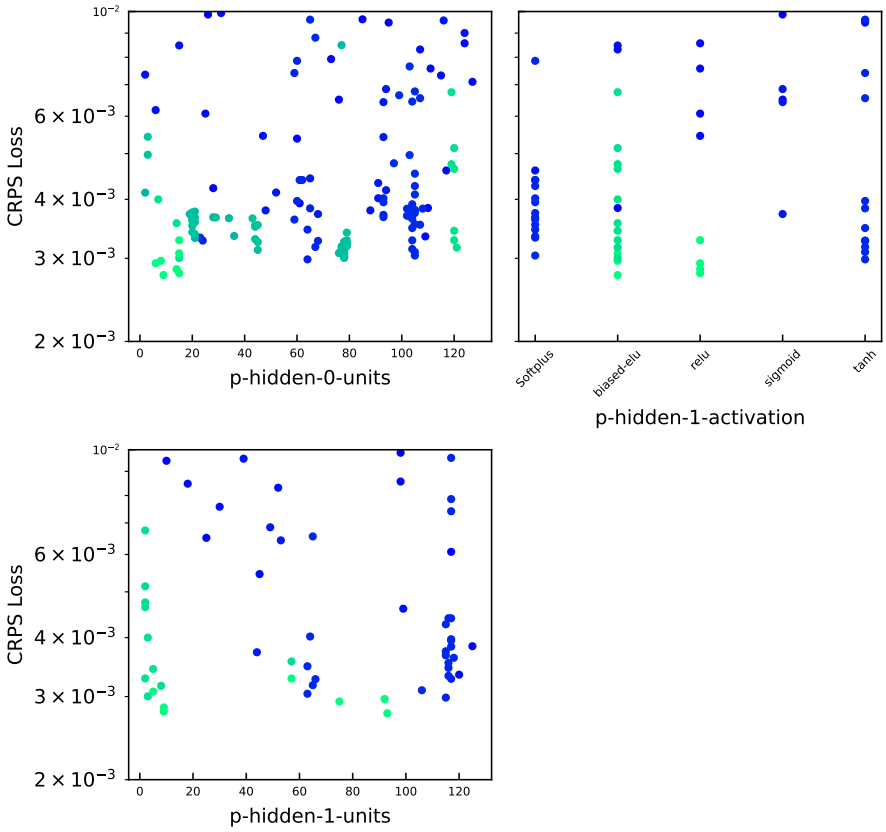

Fig. A.7: Detailed HPO plots for dropout (Part 5)

A.2 HPO plots for deep ensembles

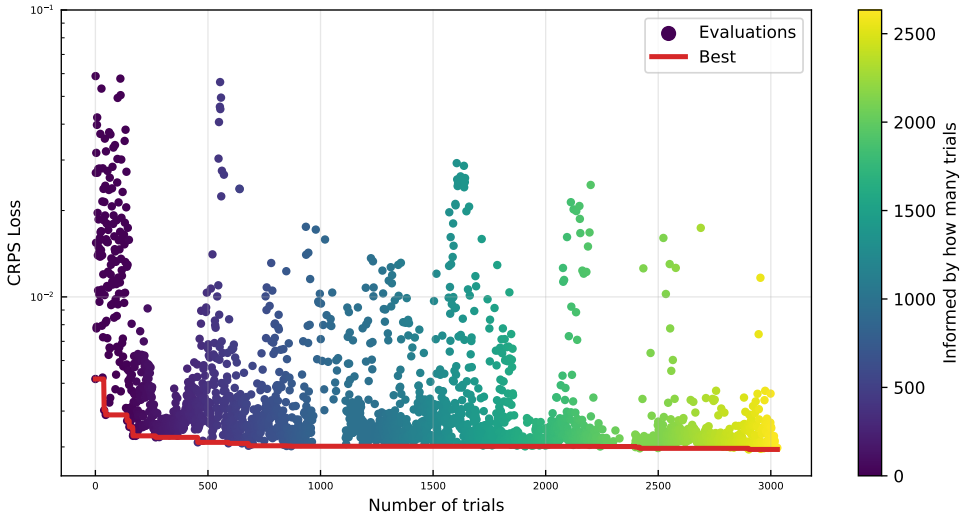

Fig. A.8: HPO progress plot for deep ensembles

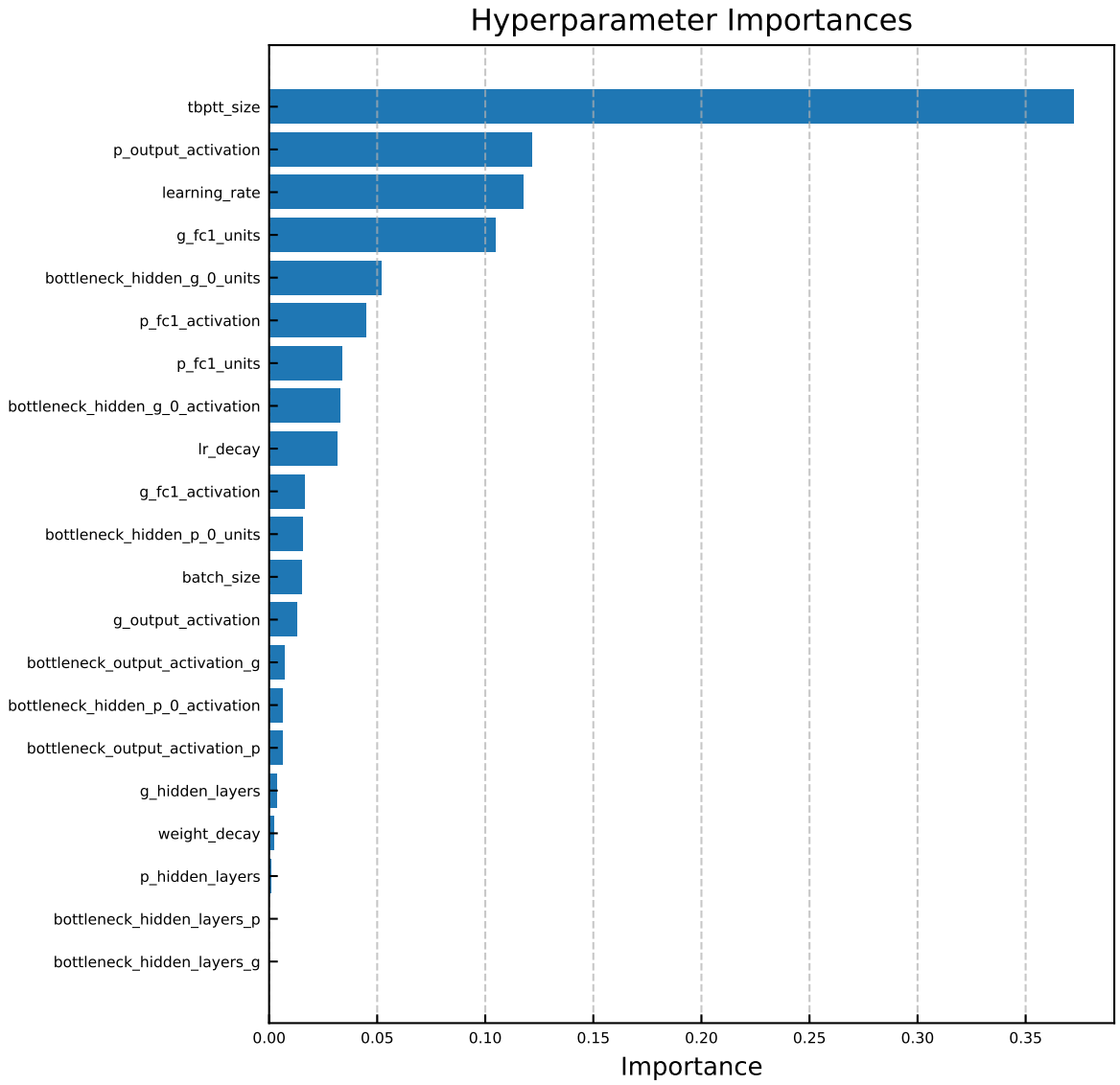

Fig. A.9: Hyperparameter importance plot for deep ensembles

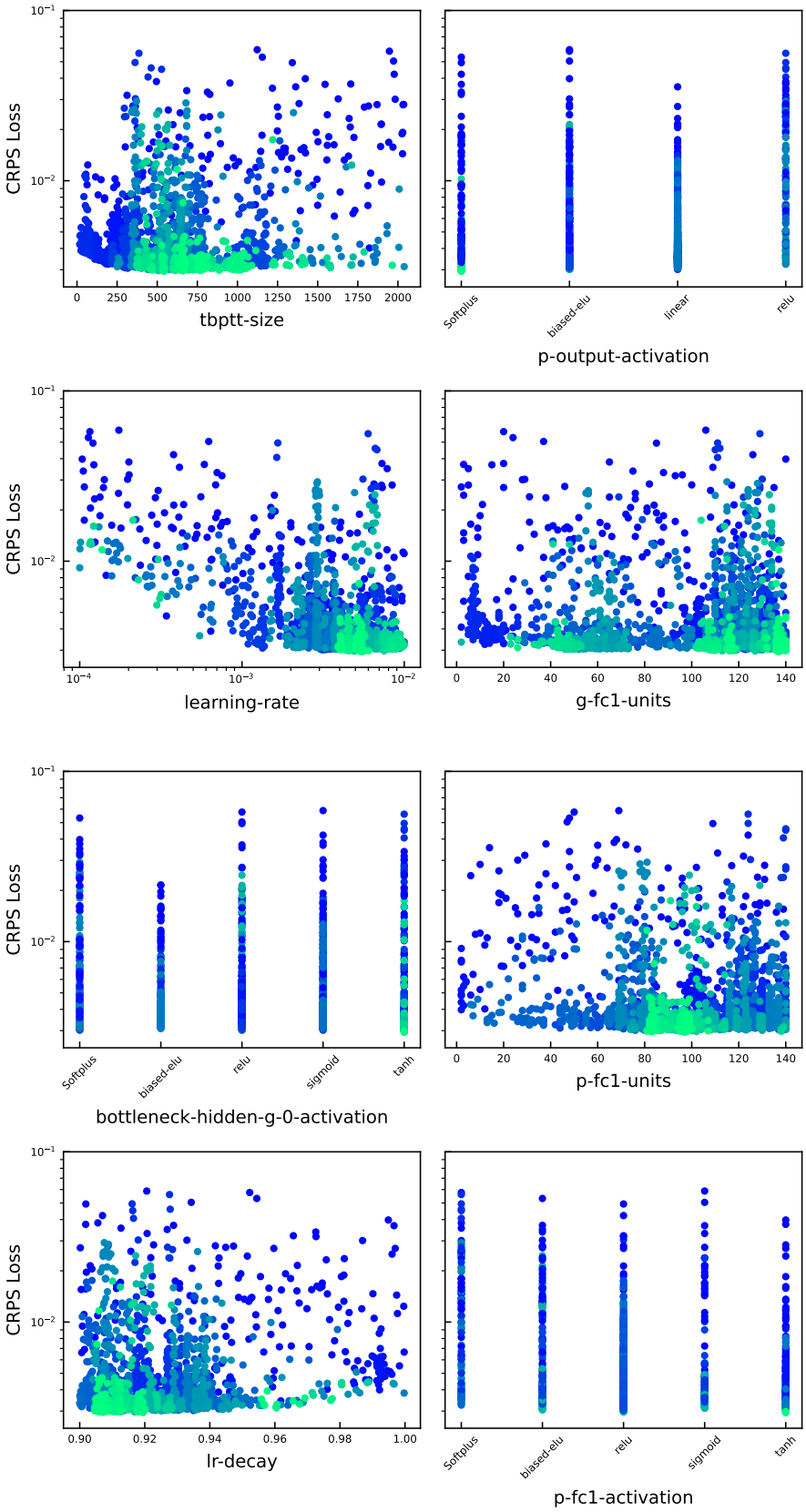

Fig. A.10: Detailed HPO plots for deep ensembles (Part 1): Colour indicates the number of trials, blue being the initial trials and green being later trials.

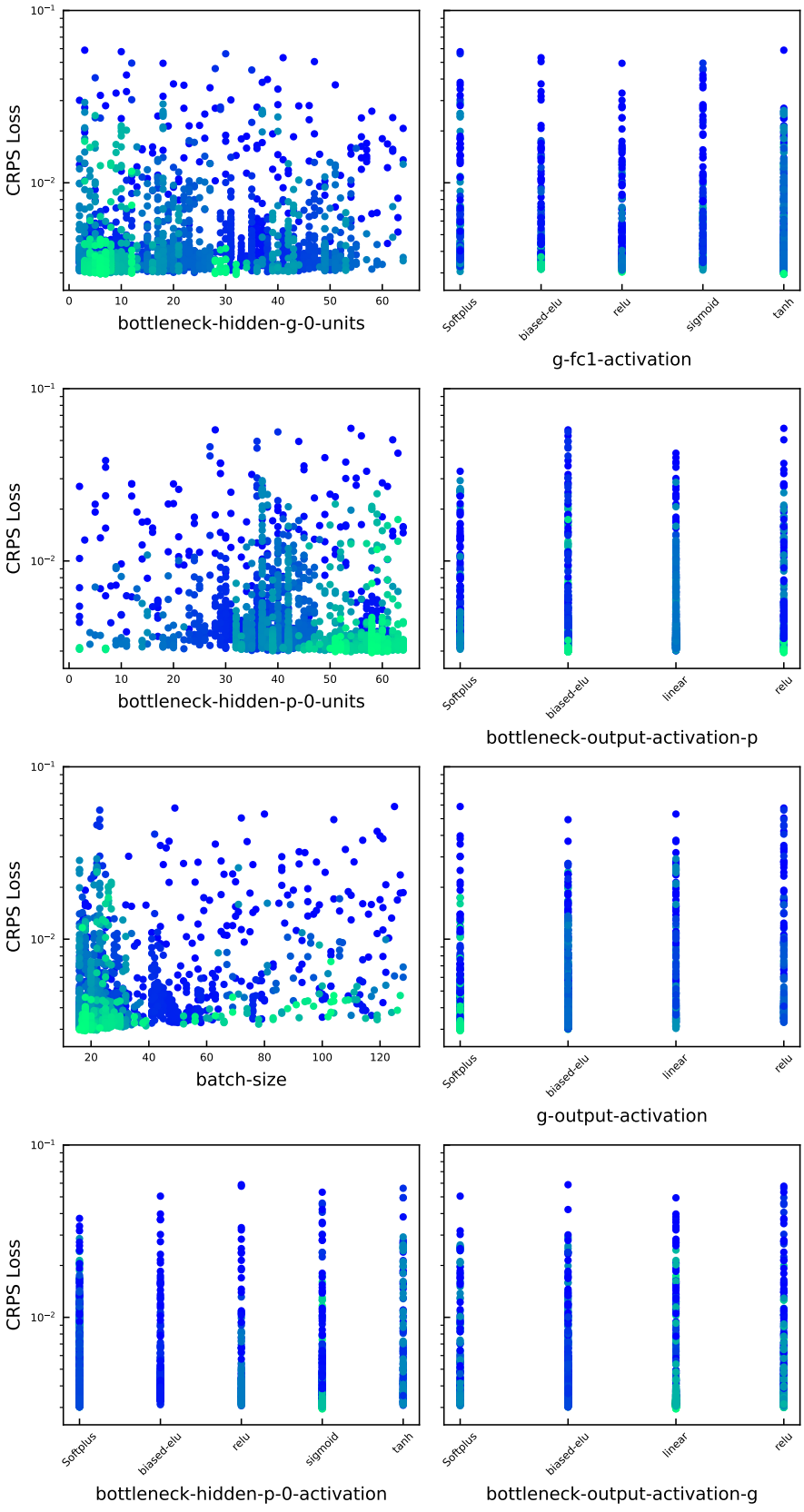

Fig. A.11: Detailed HPO plots for deep ensembles (Part 2)

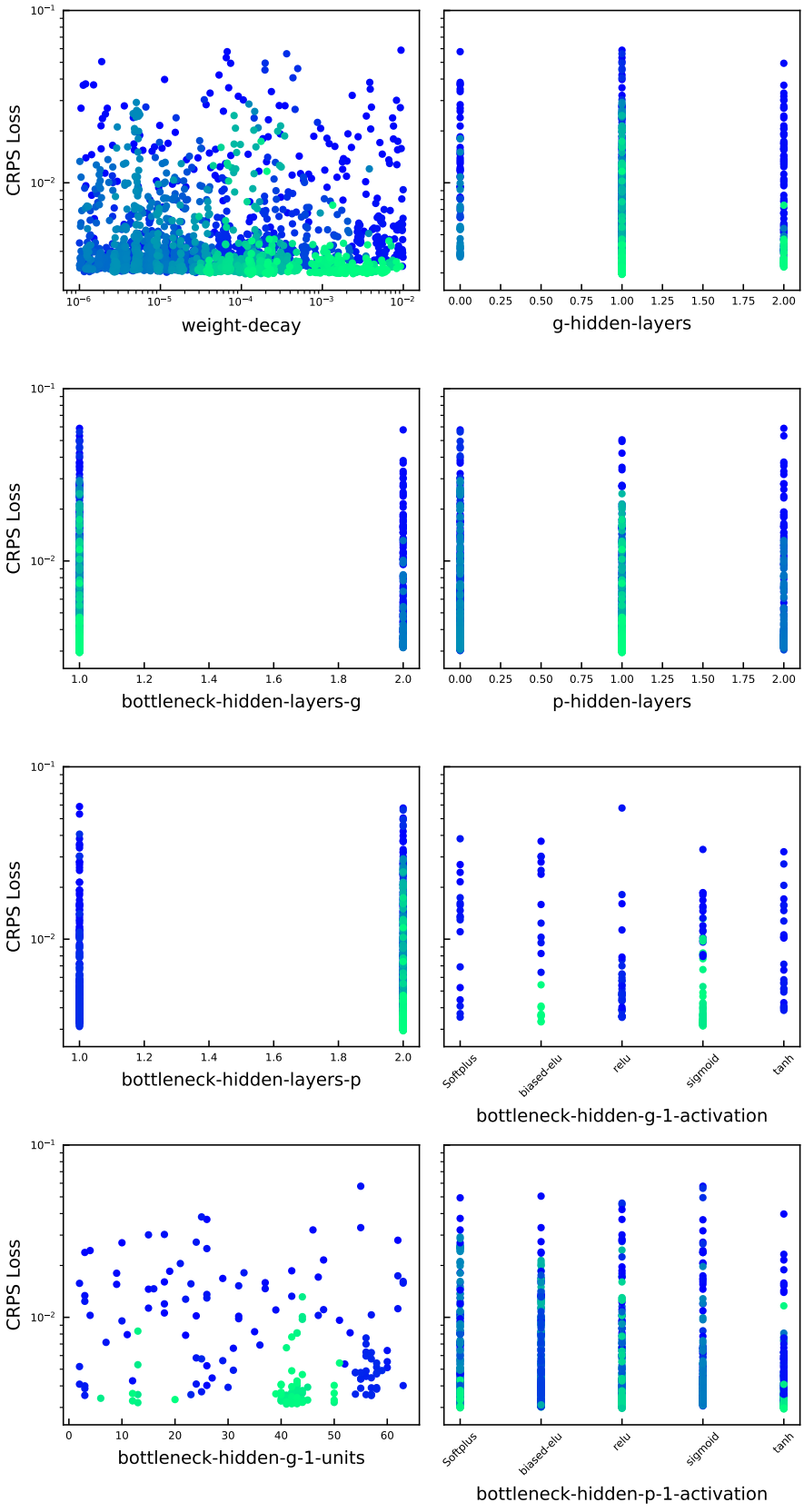

Fig. A.12: Detailed HPO plots for deep ensembles (Part 3)

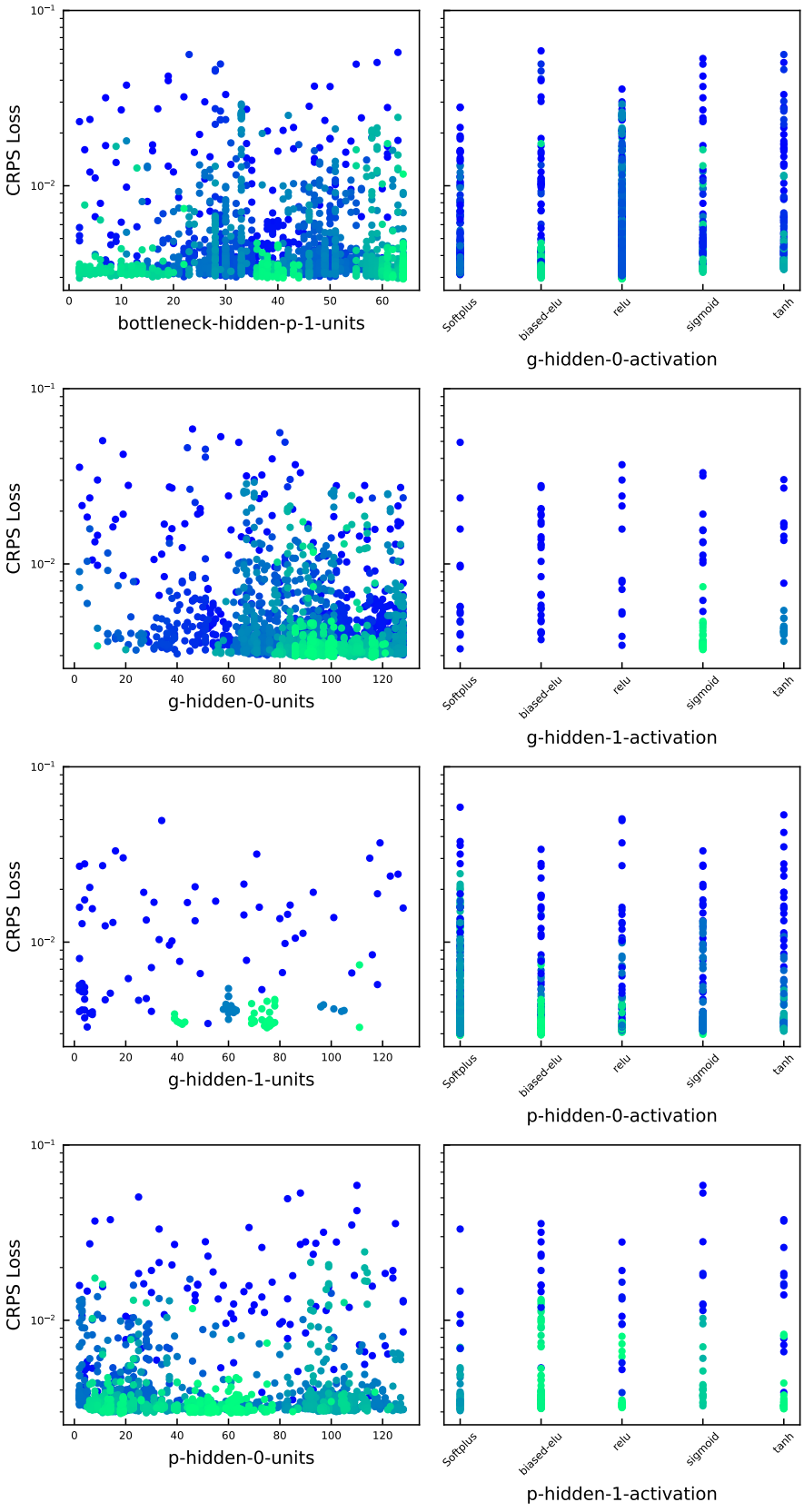

Fig. A.13: Detailed HPO plots for deep ensembles (Part 4)

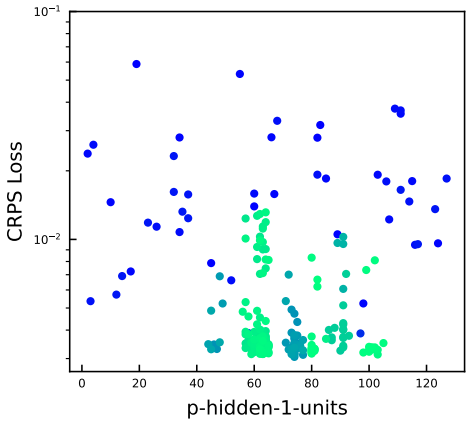

Fig. A.14: Detailed HPO plots for deep ensembles (Part 5)

A.3 HPO plots for EDL (NLL loss)

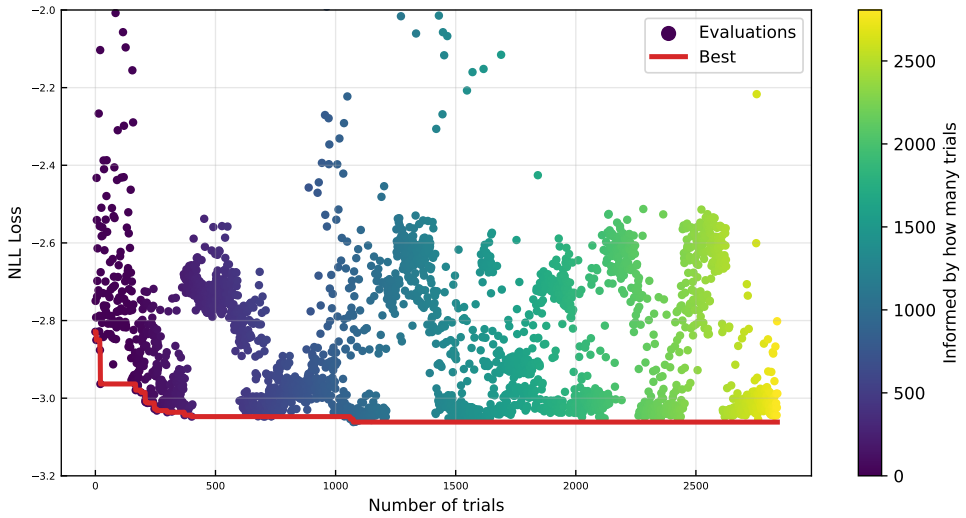

Fig. A.15: HPO progress plot for EDL

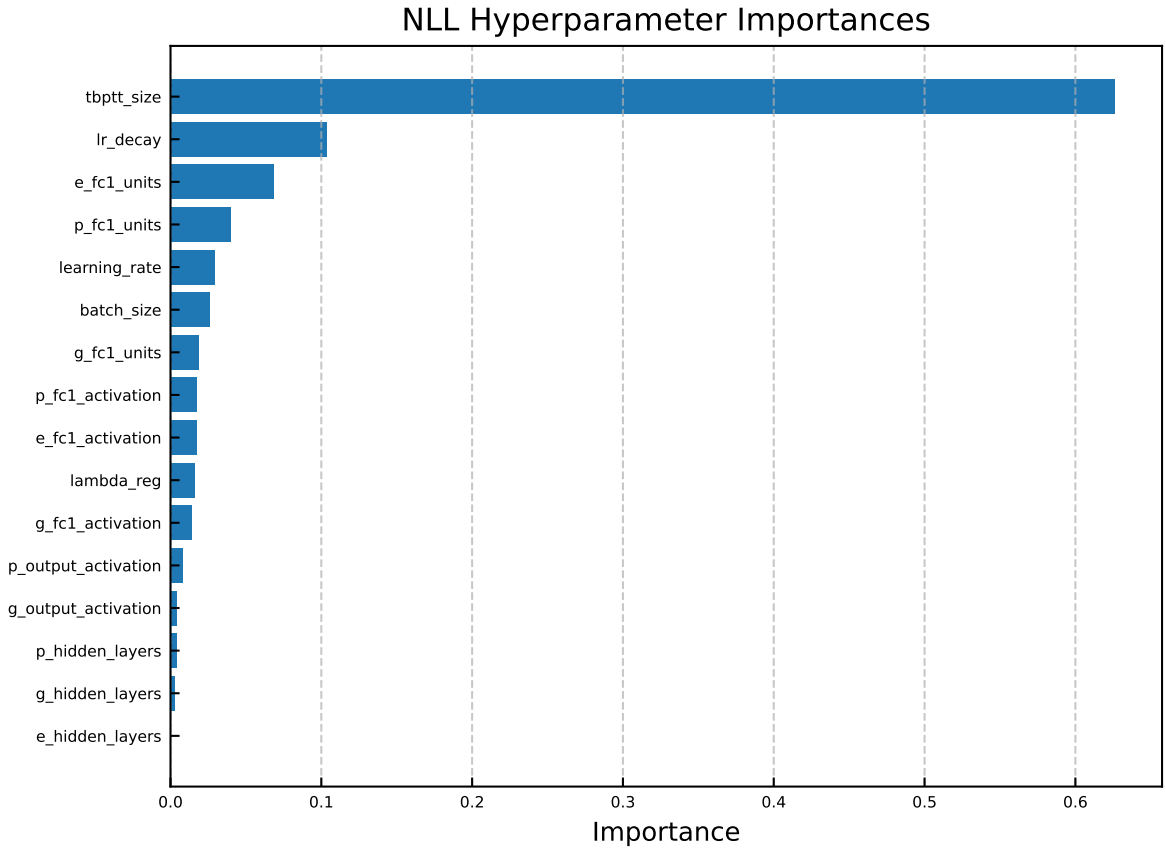

Fig. A.16: Hyperparameter importance plot for EDL

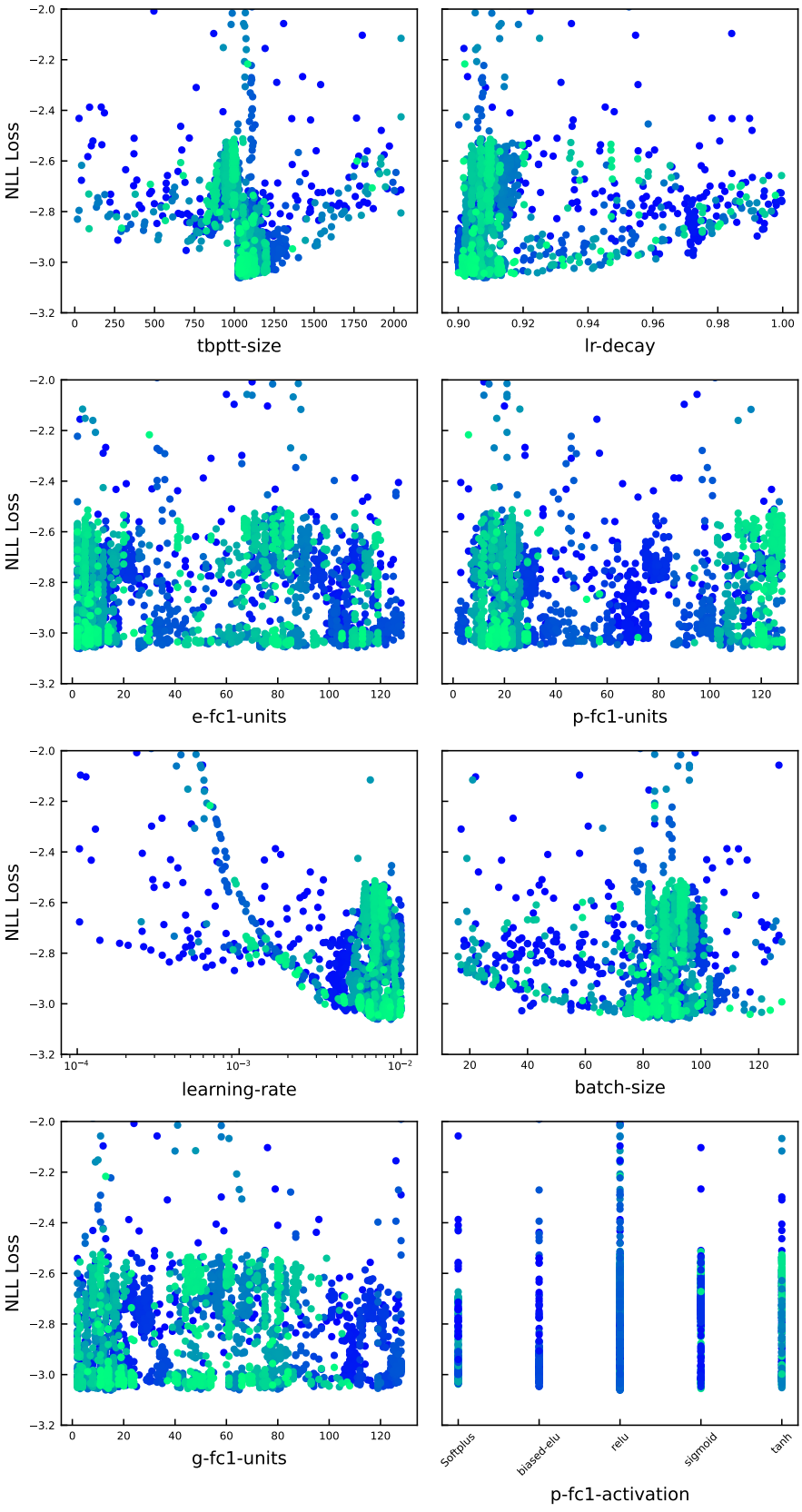

Fig. A.17: Detailed HPO plots for EDL (NLL Loss) (Part 1): Colour indicates the number of trials, blue being the initial trials and green being later trials.

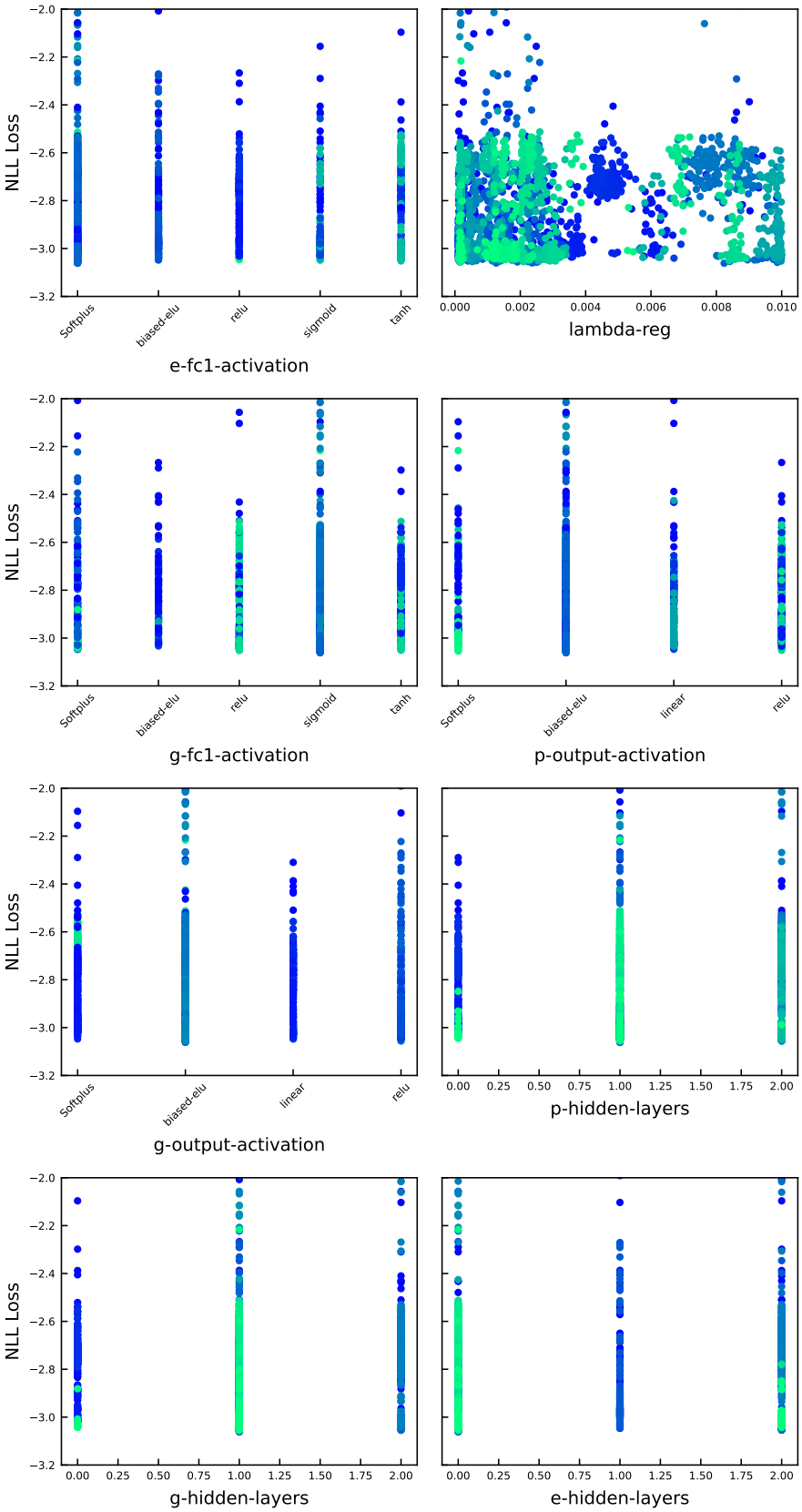

Fig. A.18: Detailed HPO plots for EDL (NLL Loss) (Part 2)

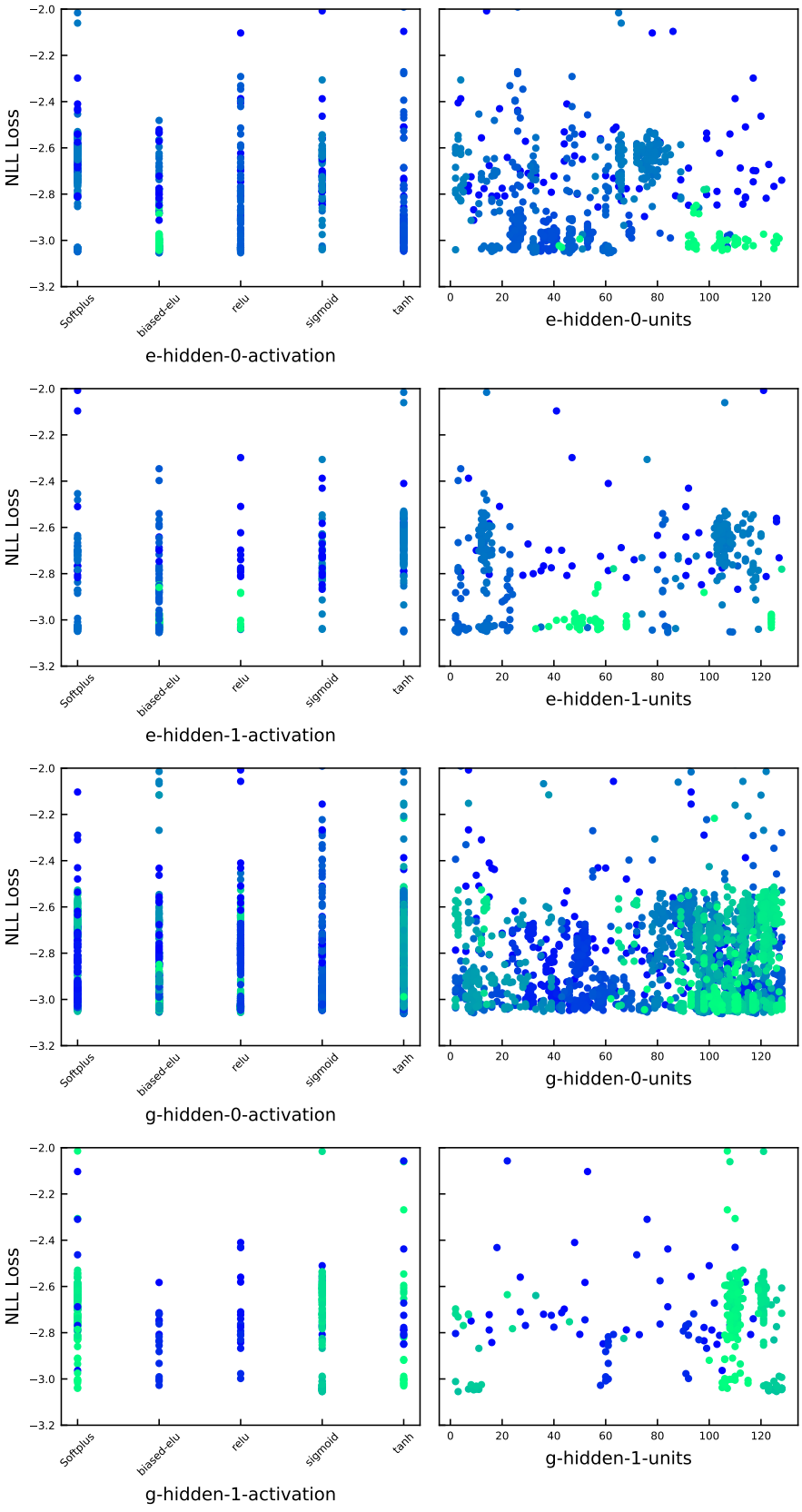

Fig. A.19: Detailed HPO plots for EDL (NLL Loss) (Part 3)

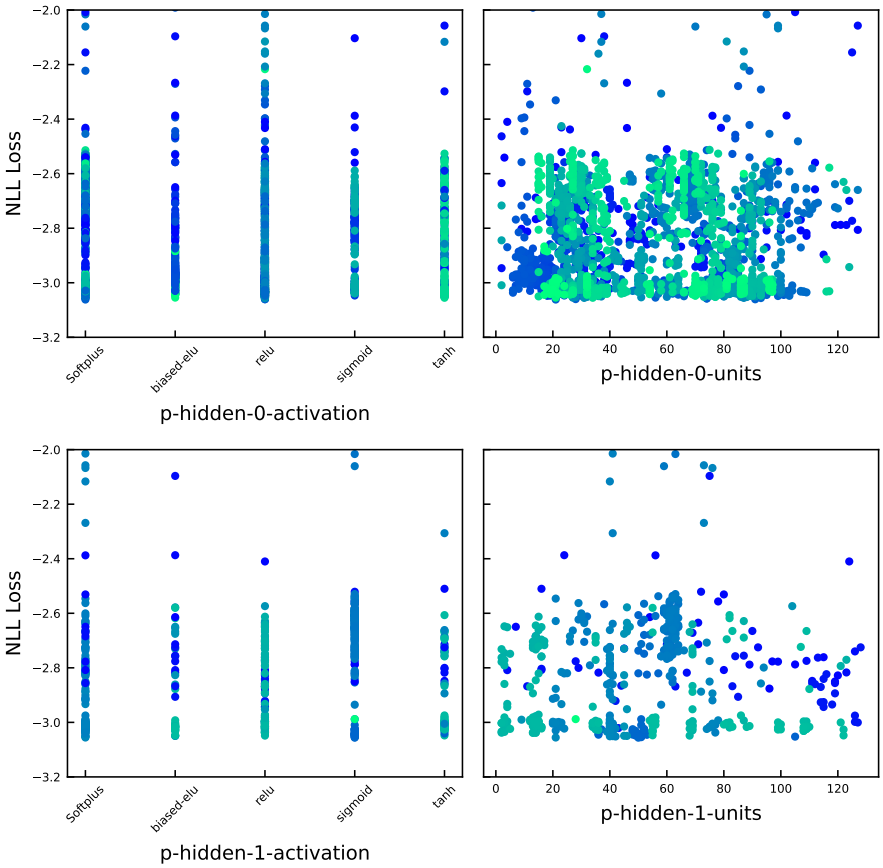

Fig. A.20: Detailed HPO plots for EDL (NLL Loss) (Part 4)

A.4 HPO plots for EDL (CRPS loss)

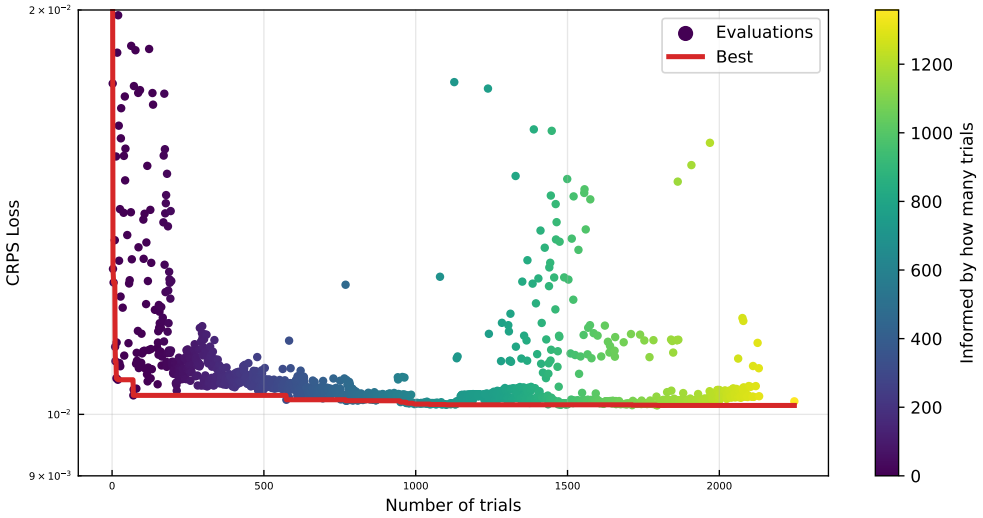

Fig. A.21: HPO progress plot for EDL (CRPS loss)

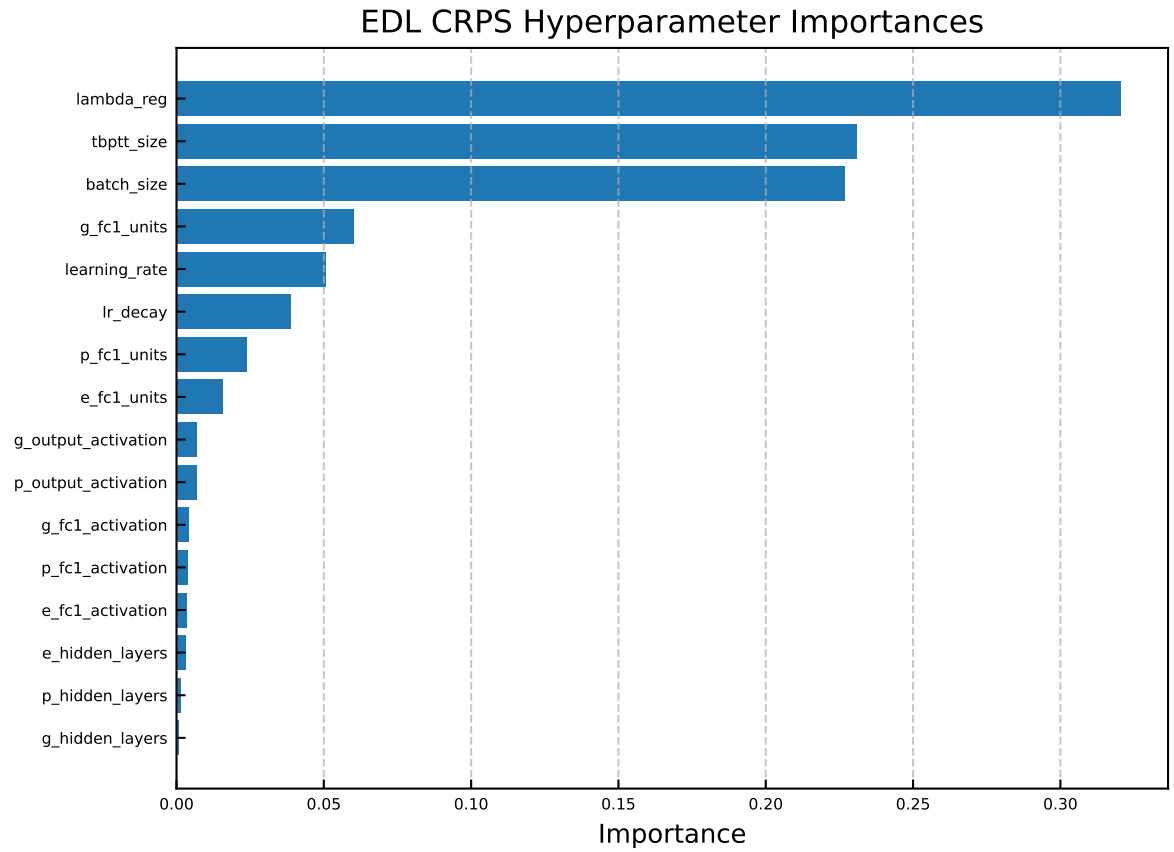

Fig. A.22: Hyperparameter importance plot for EDL (CRPS loss)

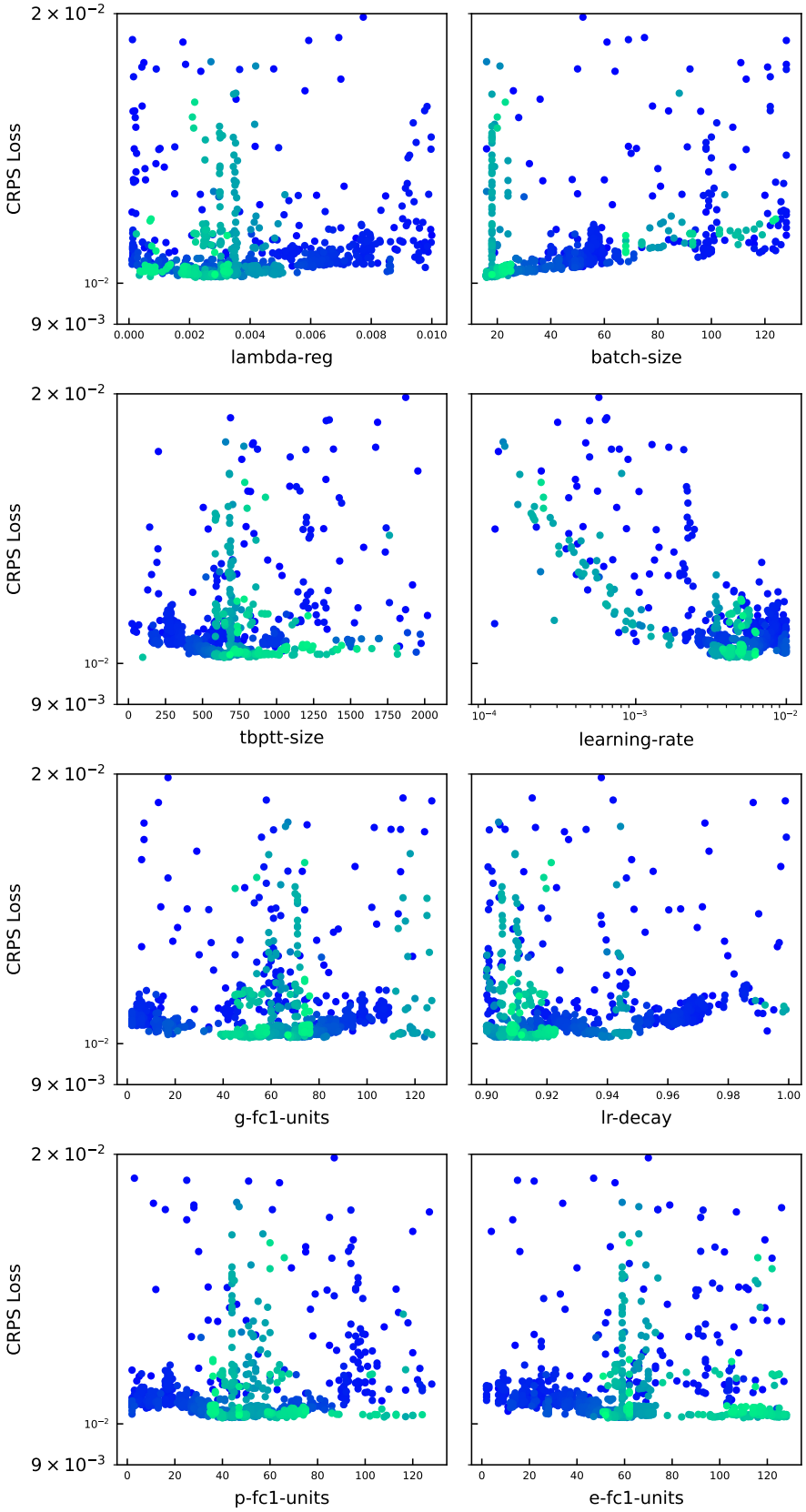

Fig. A.23: Detailed HPO plots for EDL (CRPS loss) (Part 1): Colour indicates the number of trials, blue being the initial trials and green being later trials.

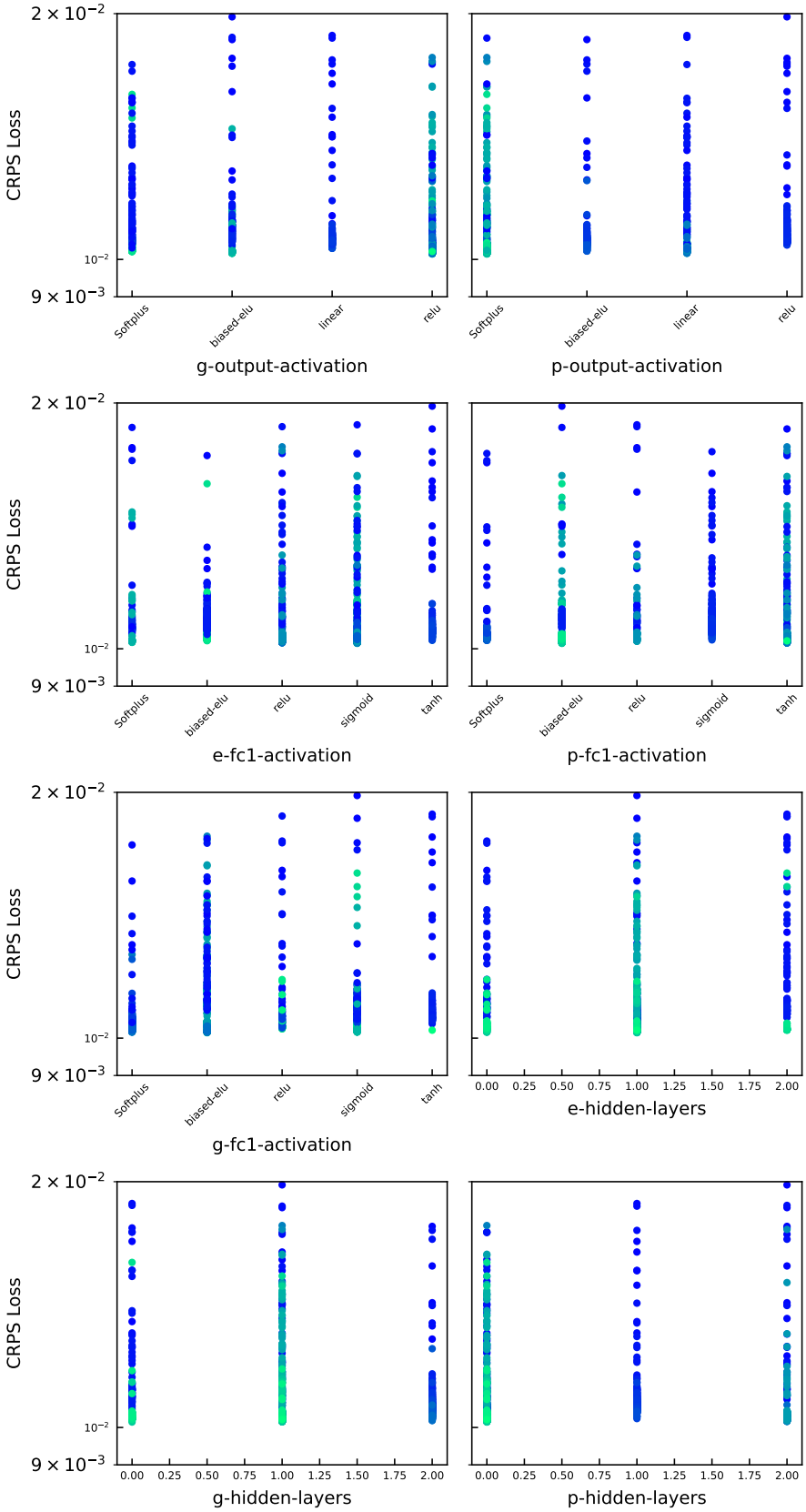

Fig. A.24: Detailed HPO plots for EDL (CRPS Loss) (Part 2)

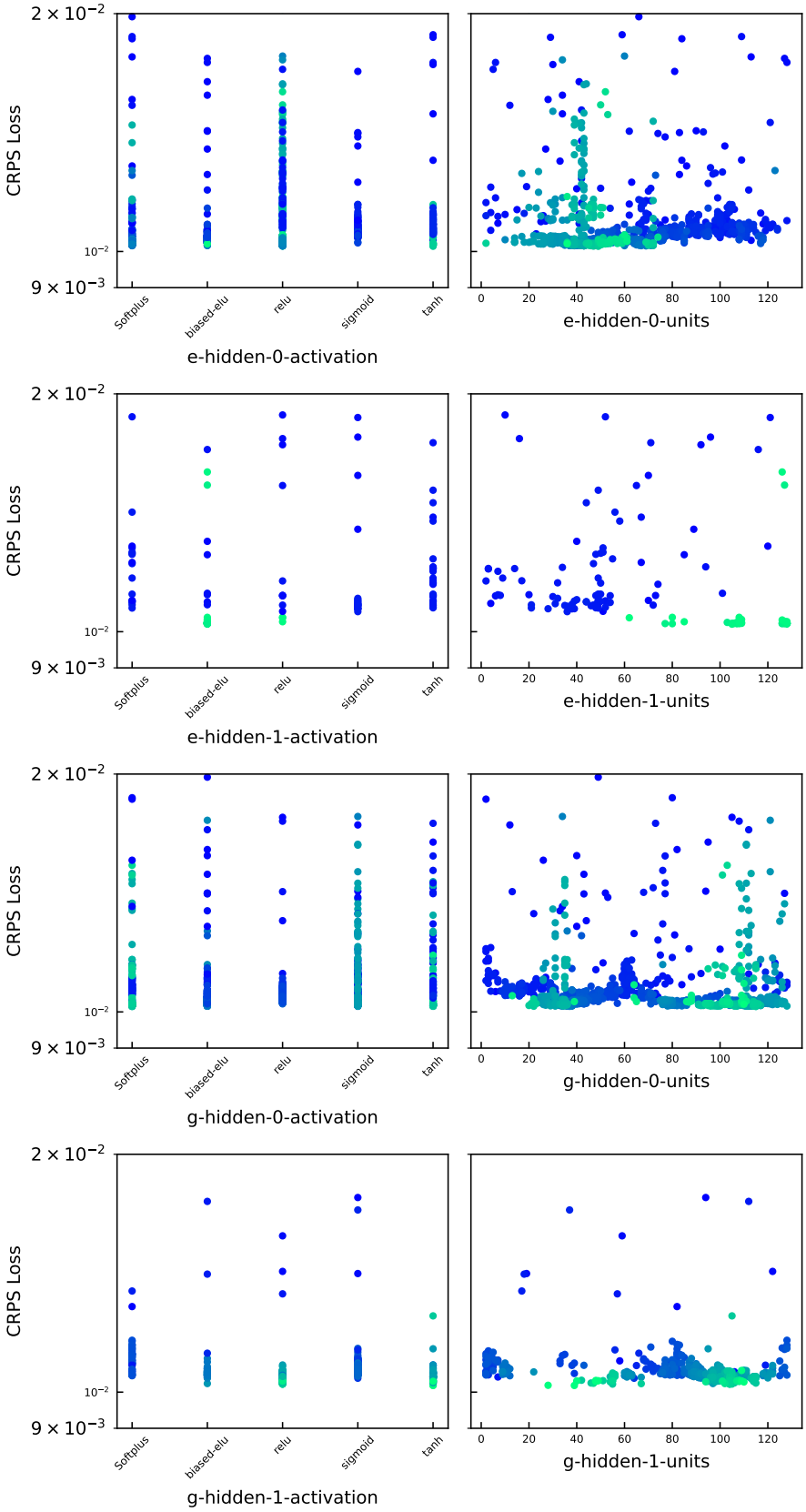

Fig. A.25: Detailed HPO plots for EDL (CRPS Loss) (Part 3)

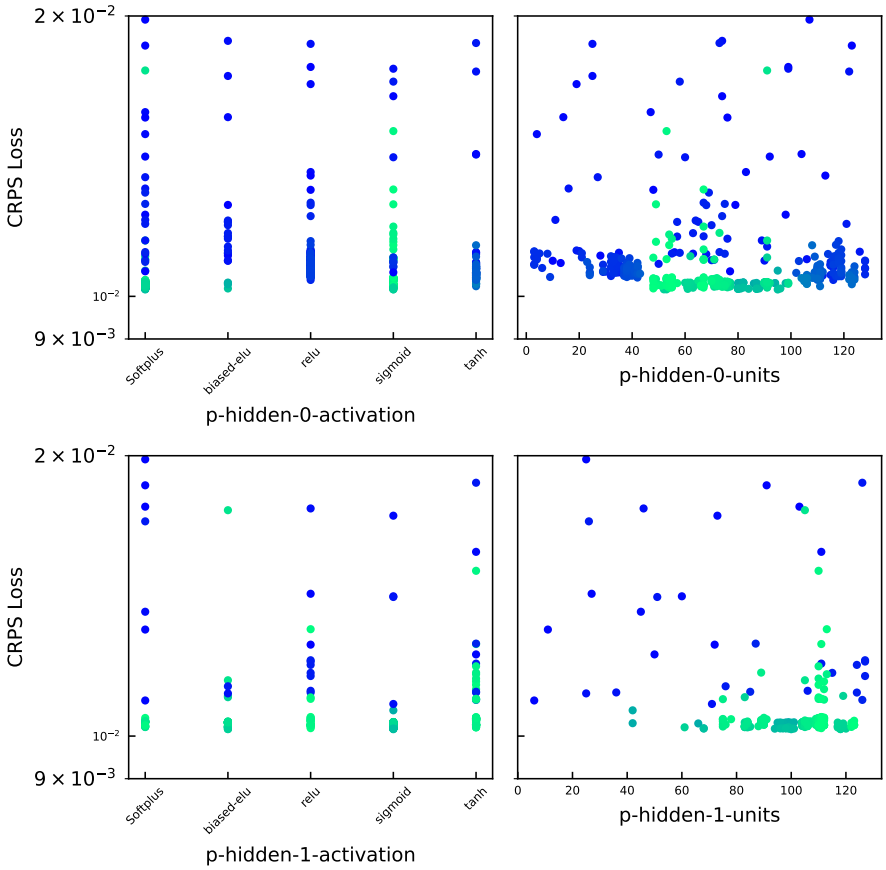

Fig. A.26: Detailed HPO plots for EDL (CRPS Loss) (Part 4)

A.5 Artificial Intelligence use declaration

To ensure transparency and uphold academic integrity, this table lists the artificial intelligence (AI) tools utilized during the preparation of this thesis.

Tool	**Scope**	**Use case**
Grammary 2026 [55]	Entire manuscript	Grammar, and spell checking
Notebooklm [56]	Entire manuscript	Grammar, and spell checking
Gemini 3.1 [57]	Fig. 3.3, 3.4, 3.5 and 3.6	Tikz code generation for these figures based on a hand-drawn picture
Gemini 3.1 [57]	Recalibration function for Student-t distribution, Plotting functions for HPO images, Fig. 2.6 Fig. 2.8,	Code generation/debugging
Gemini 3.1 [57]	Beautification of plot 2.7 and 2.14	Code generation
Gemini 3.1 [57]	Sec. 2.4, 2.7, and 2.1.2	Latex code generation for equations
Gemini 3.1 [57]	Entire HPO code	Code debugging to diagnose numerical errors, nan gradient
Claude Haiku 4.5 [58] and Github Copilot [59]	Github README.md	Readme generation based on abstract and conclusion from this work.

Tab. A.1: AI use declaration

Lists

List of Tables

List of Figures

References

[1] Y. Su, J. Zhao, G. An, W. Jin, S. Li, Y. Nie, and G. Xu, "An overview of recent advances in the online temperature estimation of pmsm in electric vehicle applications," *Electronics*, vol. 15, no. 6, p. 1249, 2026.

[2] S. Singh, S. Singh, and A. Tiwari, "Pmsm drives and its application: An overview," *Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering)*, vol. 16, no. 1, pp. 4–16, 2023.

[3] O. Wallscheid and J. Böcker, "Global identification of a low-order lumped-parameter thermal network for permanent magnet synchronous motors," *IEEE Transactions on Energy Conversion*, vol. 31, no. 1, pp. 354–365, 2015.

[4] O. Wallscheid, "Thermal monitoring of electric motors: State-of-the-art review and future challenges," *IEEE Open Journal of Industry Applications*, vol. 2, pp. 204–223, 2021.

[5] W. Kirchgässner, O. Wallscheid, and J. Böcker, "Thermal neural networks: Lumped-parameter thermal modeling with state-space machine learning," *Engineering Applications of Artificial Intelligence*, vol. 117, p. 105 537, 2023.

[6] Y.-z. Luo and Z. Yang, "A review of uncertainty propagation in orbital mechanics," *Progress in Aerospace Sciences*, vol. 89, pp. 23–39, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376042116301051

[7] 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.

[8] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.

[9] T. A. Brown, "Admissible scoring systems for continuous distributions.," 1974.

[10] J. E. Matheson and R. L. Winkler, "Scoring rules for continuous probability distributions," *Management science*, vol. 22, no. 10, pp. 1087–1096, 1976.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] P. Hennig, "Probabilistic machine learning," University of Tuebingen, lecture course, 2025, https://uni-tuebingen.de/en/180804.

[13] M. A. Woodbury, *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

[14] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[15] K. A. Wang, G. Pleiss, J. R. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson, "Exact gaussian processes on a million data points," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[16] A. Solin and S. Särkkä, "Hilbert space methods for reduced-rank gaussian process regression," *Statistics and Computing*, vol. 30, no. 2, pp. 419–446, 2020.

[17] A. BANERJEE, D. B. DUNSON, and S. T. TOKDAR, "Efficient gaussian process regression for large datasets," *Biometrika*, vol. 100, no. 1, pp. 75–89, 2013. Accessed: Mar. 10, 2026. [Online]. Available: http://www.jstor.org/stable/43304538

[18] A. Wilson and H. Nickisch, "Kernel interpolation for scalable structured gaussian processes (kiss-gp)," in *International conference on machine learning*, PMLR, 2015, pp. 1775–1784.

[19] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," *arXiv preprint arXiv:1309.6835*, 2013.

[20] M. Katzfuss, J. Guinness, W. Gong, and D. Zilber, "Vecchia approximations of gaussian-process predictions," *Journal of Agricultural, Biological and Environmental Statistics*, vol. 25, no. 3, pp. 383–414, 2020.

[21] X. Qiu, E. Meyerson, and R. Miikkulainen, "Quantifying point-prediction uncertainty in neural networks via residual estimation with an i/o kernel," *arXiv preprint arXiv:1906.00588*, 2019.

[22] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on bayesian neural networks—a tutorial for deep learning users," *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29–48, 2022.

[23] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, PMLR, 2016, pp. 1050–1059.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[26] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[27] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 3183–3193.

[28] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, "Deep evidential regression," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Vancouver, BC, Canada: Curran Associates Inc., 2020.

[29] N. Meinert and A. Lavin, "Multivariate deep evidential regression," *arXiv preprint arXiv:2104.06135*, 2021.

[30] A. Jordan, F. Krüger, and S. Lerch, "Evaluating probabilistic forecasts with scoringrules," *Journal of Statistical Software*, vol. 90, pp. 1–37, 2019.

[31] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. US Government printing office, 1948, vol. 55.

[32] M. Juergens, N. Meinert, V. Bengs, E. Hüllermeier, and W. Waegeman, "Is epistemic uncertainty faithfully represented by evidential deep learning methods?" *arXiv preprint arXiv:2402.09056*, 2024.

[33] Y. Wu, B. Shi, B. Dong, Q. Zheng, and H. Wei, "The evidence contraction issue in deep evidential regression: Discussion and solution," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 21 726–21 734.

[34] D. Levi, L. Gispan, N. Giladi, and E. Fetaya, "Evaluating and calibrating uncertainty prediction in regression tasks," *Sensors*, vol. 22, no. 15, p. 5540, 2022.

[35] V. Kuleshov, N. Fenner, and S. Ermon, "Accurate uncertainties for deep learning using calibrated regression," in *International conference on machine learning*, PMLR, 2018, pp. 2796–2804.

[36] M. H. DeGroot and S. E. Fienberg, "The comparison and evaluation of forecasters," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 32, no. 1-2, pp. 12–22, 1983.

[37] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 625–632.

[38] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*, PMLR, 2017, pp. 1321–1330.

[39] Y. Chung, I. Char, H. Guo, J. Schneider, and W. Neiswanger, "Uncertainty toolbox: An open-source library for assessing, visualizing, and improving uncertainty quantification," *arXiv preprint arXiv:2109.10254*, 2021.

[40] C. Bauer, T. Kenter, M. Lass, L. Mazur, M. Meyer, H. Nitsche, H. Riebler, R. Schade, M. Schwarz, N. Winnwa, et al., "Noctua 2 supercomputer," *Journal of large-scale research facilities JLSRF*, vol. 9, no. 1, 2024.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[42] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, vol. 31, 2018.

[43] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[44] W. Kirchgässner, O. Wallscheid, and J. Böcker, *Electric motor temperature*, 2021. [Online]. Available: https://www.kaggle.com/dsv/2161054

[45] W. Kirchgässner, O. Wallscheid, and J. Böcker, "Estimating electric motor temperatures with deep residual machine learning," *IEEE Transactions on Power Electronics*, vol. 36, no. 7, pp. 7480–7488, 2021.

[46] W. Kirchgässner, *Data-driven thermal modeling of a permanent magnet synchronous motor with machine learning*. LibreCat University, 2024.

[47] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022.

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

[49] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373)*, Ieee, 2000, pp. 153–158.

[50] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, 2015.

[51] M. L. Stein, *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 1999.

[52] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[53] I. Loshchilov, F. Hutter, et al., "Fixing weight decay regularization in adam," *arXiv preprint arXiv:1711.05101*, vol. 5, no. 5, p. 5, 2017.

[54] T. P. G. D. Group, - *www.postgresql.org*.

[55] Grammarly, *Grammarly*, version May 2026 version, Computer software / Large language model, 2026. [Online]. Available: https://grammarly.com

[56] Google, *Notebooklm*, Large language model / AI research assistant, 2026. [Online]. Available: https://notebooklm.google

[57] Google DeepMind, *Gemini 3.1*, Large language model / Multimodal AI, 2026. [Online]. Available: https://deepmind.google/models/gemini/pro/

[58] Anthropic, *Claude haiku 4.5*, version 4.5, Large language model. Accessed: 2026-05-17, 2026. [Online]. Available: https://claude.ai

[59] GitHub, *Github copilot*, Computer software. Accessed: 2026-05-17, 2026. [Online]. Available: https://github.com/features/copilot