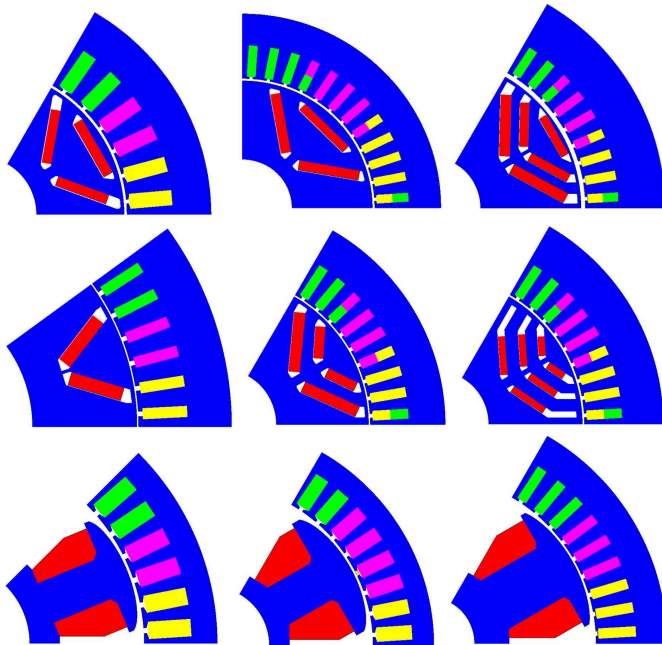


Nuo Chen

Electromagnetic Optimization of Electric Machines Using Machine Learning Metamodeling



Electromagnetic Optimization of Electric Machines Using Machine Learning Metamodeling

Zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN
(Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)
angenommene

DISSERTATION

von
M.Sc. Nuo Chen
geb. in Guizhou, China

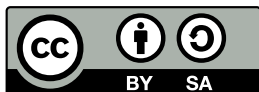
Tag der mündlichen Prüfung:
Hauptreferent:
Korreferent:

23. September 2026
Prof. Dr.-Ing. Martin Doppelbauer
Prof. Dr.-Ing. Sebastian Schöps
(Technische Universität Darmstadt)

Karlsruher Institut für Technologie (KIT)
Elektrotechnisches Institut (ETI)

Zur Erlangung des akademischen Grades eines Doktor-Ingenieurs
von der KIT-Fakultät für Elektrotechnik und Informationstechnik des
Karlsruher Instituts für Technologie (KIT) genehmigte Dissertation
von M.Sc. Nuo Chen, geb. in Guizhou, China

Tag der mündlichen Prüfung: 23. September 2026
Hauptreferent: Prof. Dr.-Ing. Martin Doppelbauer
Korreferent: Prof. Dr.-Ing. Sebastian Schöps
(Technische Universität Darmstadt)



This document - excluding pictures and graphs - is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0):
<https://creativecommons.org/licenses/by-sa/4.0/deed.en>

Preface

This dissertation arises from the research projects I undertook while pursuing my doctorate at the Institute of Electrical Engineering, Karlsruhe Institute of Technology. I would like to take a moment to sincerely thank and extend my warmest wishes to all those who have been by my side during this incredible journey.

Firstly, I would like to express my gratitude to my doctoral supervisor, Prof. Dr.-Ing Martin Doppelbauer. I deeply appreciate his constant trust, which allowed me the freedom to delve into new research paths; his generous support and patient guidance, which enabled me to fully engage in reflection and exploration; and his continual backing and encouragement, which bolstered my confidence and enthusiasm for scientific inquiry. It is thanks to having such an incredible advisor that I am able to successfully complete my doctoral dissertation.

Secondly, I would like to extend my appreciation to Zhiwei Zheng and Dr. Kai Brune from the project partner company AUDI AG for their exceptional support and assistance during my internship, master's thesis, and doctoral project. Their recognition and trust have fueled my boundless motivation and passion to complete each research mission. Additionally, I would like to express my gratitude to Dr. Bastian Heidler and Christian Nörenberg, my corporate mentors at AUDI AG. I am appreciative of the invaluable advice, support, and long-term guidance they provided me during my research ventures.

Additionally, I want to express thanks to all of my colleagues at the institute, with a special emphasis on those from "Campus Ost Cares!" I am grateful to Christian Digel for our scientific collaboration and the exchange of ideas. I am also thankful for the support provided by Martin Bremer and Tobias Schneider. I am truly grateful for the infinite happiness that Lorenz Schoch,

Julius Kesten, and Sophia Beck have brought. I also extend my best wishes to them as they proceed on their doctoral projects. Lastly, I would also like to thank my co-supervisor, Dr. Matthias Brodatzki, for his long-term support and assistance in my work.

Finally, I would like to express my gratitude to my family. I am eternally grateful to my parents for their tireless efforts and unwavering support. They have been an inspiration throughout my life. I also want to express my profound appreciation to my wife, who has consistently supported me, offering comfort during my darkest moments and sharing in my joyous moments. We appreciate the mutual support shared over the last fifteen years, the trust built among one another, and our joint dedication towards each other's success.

Nuo Chen

02.26.2026 Karlsruhe

Kurzfassung

In der Phase der Designoptimierung von elektrischen Maschinen wird datengetriebenes Metamodellieren weit verbreitet eingesetzt, um wichtige Leistungsindikatoren (KPIs) wie Leistung, Drehmoment und Drehmomentwelligkeit vorherzusagen. Es kann rechenintensive Simulationen durch die Erstellung von Metamodellen mit geringeren Rechenkosten ersetzen. Dennoch stehen datengetriebene Methoden vor Herausforderungen, darunter hohe Komplexität bei großer Eingabedimension, zeitaufwändige Datensammlung und reduzierte Vorhersagezuverlässigkeit bei begrenzten Daten. In dieser Dissertation werden erstmals mehrere auf maschinellem Lernen basierende Ansätze und Optimierungsstrategien vorgeschlagen, um die aktuellen Hindernisse zu überwinden. Zunächst wird ein deterministisches neuronales Netzwerk auf Basis von Transferlernen vorgeschlagen, um das Problem der Reparametrisierung zu lösen, das bei der parametrischen Modelltraining auftritt. Dieser Ansatz umgeht die Anforderung, das Modell von Grund auf neu zu trainieren, und erzielt simultan eine gesteigerte Vorhersagegenauigkeit, indem die vortrainierten Gewichte aus dem ursprünglichen Netzwerk dynamisch übertragen werden.

Zweitens wird eine neuartige, auf Unsicherheitsquantifizierung basierende Optimierungsmethode unter Verwendung probabilistischer Metamodellierung vorgestellt. Diese Methode ermöglicht es den Anwendern, optimale Design-KPIs präzise vorherzusagen, indem sie zusätzliche Unsicherheitsinformationen berücksichtigt. Darüber hinaus bildet sie die Grundlage für die probabilistische Vorhersage von KPIs mittels Metamodellierung und die unsicherheitsbasierte Optimierung im Bereich elektrischer Maschinen.

Zur Lösung des Problems des zeitaufwändigen Samplings werden in dieser Dissertation zwei neuartige, auf Unsicherheit basierende, dateneffiziente Lernansätze präsentiert. Zunächst wird ein Rahmenwerk für das Selbsttraining vorgestellt, das die kontinuierliche Generierung von Daten mit Pseudobeschriftungen ermöglicht, um den Datensatz zu erweitern und somit das Modelltraining

zu optimieren. Im Anschluss werden verschiedene Optimierungsstrategien auf Basis des aktiven Lernens entwickelt. Diese Strategien können unter Bedingungen mit kleinen Daten iterativ wichtige Daten zum ursprünglichen Datensatz hinzufügen und dadurch die Vorhersageleistung des Modells steigern.

Abstract

In the design optimization stage of electric machines, data-driven metamodeling has been widely used to predict key performance indicators (KPIs) such as power, torque, and torque ripple. It can replace computationally expensive simulations by creating metamodels with lower computational costs. However, data-driven methods still face challenges, including high complexity in the case of large input dimension, time-consuming data collection, and reduced prediction reliability with limited data. In this dissertation, multiple machine learning-based approaches and optimization strategies have been proposed for the first time to overcome the current obstacles.

First, a deterministic neural network based on transfer learning is proposed to overcome the reparameterization issue inherent in parameter-based model training. It circumvents the necessity of retraining the model from scratch while achieving higher prediction accuracy by dynamically transferring the pretrained weights from the original network.

Second, a novel uncertainty quantification-based optimization method using probabilistic metamodeling is presented. This method allows users to balance optimal design KPIs with predictive accuracy using additional uncertainty information. It also lays the groundwork for probabilistic metamodeling-based prediction of KPIs and uncertainty-based optimization in the field of electric machines.

Finally, to overcome the issue of time-consuming sampling, two novel uncertainty-based data-efficient learning approaches are introduced. Initially, a self-training framework is introduced. It enables continuous data generation with pseudo labels to extend the dataset, thereby enhancing model training. Subsequently, several active learning-based optimization strategies are designed. They can iteratively add key data to the original dataset under small-sample conditions, thereby boosting the model's prediction performance.

List of Symbols

Abbreviations

R^2	coefficient of determination
AC	actor-critic
AC	alternating current
Adam	adaptive moment estimation
AI	artificial intelligence
ANNs	artificial neural networks
AUC	area under curves
BEM	boundary element method
BNNs	Bayesian neural networks
CNNs	convolutional neural networks
DC	direct current
DE	deep ensemble
DL	deep learning
DNNs	deep neural networks
DoE	design of experiments
ELBO	evidence lower bound
ELM	extreme learning machine
EMF	inductive electromotive force
FDM	finite difference method
FEA	finite element analysis
FEM	finite element method
FF	full factorial
FN	false negative
FP	false positive

FPR	false positive rate
GANs	generative adversarial networks
GA _s	genetic algorithms
GD	gradient descent
GNNs	graph neural networks
GPR	Gaussian process regression
GPU _s	graphics processing units
IM _s	induction motors
KL	Kullback-Leibler
KPI _s	key performance indicators
LHS	Latin hypercube sampling
LN	layer normalization
MAE	mean absolute error
MC	Monte-Carlo
MCMC	Markov chain Monte Carlo
ML	machine learning
MLE	maximum likelihood estimation
MLP _s	multilayer perceptrons
MNN _s	modular neural networks
MOO	multi-objective optimization
MSE	mean squared error
NLL	negative log-likelihood
NN	neural network
PINAW	prediction interval normalized average width
PINN _s	physics-informed neural networks
PM	permanent magnet
PMSM _s	permanent magnet synchronous motors
PSA	Pareto search algorithm
PSO	particle swarm optimization
RBF	radial basis function
RL	reinforcement learning
RNN _s	recurrent neural networks
ROC	receiver operating characteristic
SAC	soft actor-critic
SMAPE	symmetric mean absolute percentage
SRM _s	switched reluctance motors
SVM _s	support vector machines

SVR	support vector regression
THD	total harmonic distortion
TL	transfer learning
TN	true negative
TP	true positive
TPR	true positive rate
TPUs	tensor processing units
UI	uncertainty interval
VAEs	variational autoencoders
VI	variational inference
XGBoost	extreme gradient boosting

Symbols

Physical Variables

$\vec{A}$	magnetic vector potential
$\vec{B}$	magnetic flux density
$\vec{D}$	electric displacement field
ϵ_0	vacuum permittivity
ϵ_r	relative permittivity
ϵ	electric permittivity
η	efficiency
$\vec{E}$	electric field intensity
f	frequency
$\vec{H}$	magnetic field intensity
i_d	d-axis current
i_q	q-axis current
$\vec{J}_s$	source current density
$\vec{J}_{\text{tot}}$	total current density
$\vec{J}$	electric current density
L_d	d-axis inductance
l_{Fe}	length in the z-axis direction
L_q	q-axis inductance
$M_{\text{ep,max}}$	maximum torque over one electrical period

$M_{\text{ep,min}}$	minimum torque over one electrical period
$\vec{M}_{\text{pm}}$	magnetization of permanent magnets
M_{ripple}	torque ripple
m	number of phases
μ	magnetic permeability
M	electromagnetic torque
N_{t}	number of turns
n	rotational speed
S	slot cross-sectional area
P_{addl}	additional loss
P_{Cu}	copper loss
P_{eddy}	eddy current loss
P_{el}	input electrical power
P_{Fe}	iron loss
P_{hys}	hysteresis loss
P_{loss}	total loss of machines
P_{mech}	mechanical loss
P_{pm}	permanent magnet loss
P	power
Φ	magnetic flux
Ψ_{d}	d-axis flux linkage
Ψ_{pm}	permanent magnet flux linkage
Ψ_{q}	q-axis flux linkage
Ψ	magnetic flux linkage
p	number of pole pairs
ρ	electric charge density
σ	electrical conductivity
V	electric scalar potential

Mathematical Expressions

a_j^l	output of neuron
α_i	Lagrange multiplier
b_j^l	bias
β	momentum
$\mathcal{D}_s$	source domain
$\mathcal{D}_t$	target domain

$\mathcal{D}$	dataset
δ_j^l	error value
γ	margin
$k(\mathbf{x}, \mathbf{x}')$	covariance (kernel) function of GPR
$\mathcal{L}$	objective or loss function
$m(\mathbf{x})$	mean function of GPR
σ_n	Gaussian noise
$\mathcal{T}_s$	source task
$\mathcal{T}_t$	target task
θ	hyperparameter
w_{ij}^l	weight
$\mathbf{x}$	input vector
$\mathbf{y}$	output vector
z_j^l	input of neuron

Contents

Preface	i
Kurzfassung	iii
Abstract	v
List of Symbols	vii
1 Introduction	1
1.1 Motivation	1
1.2 Related works	3
1.3 Objectives	5
1.4 Outline	6
2 Fundamentals of Electric Machines and Optimization	9
2.1 Brief introduction to electric machines	10
2.2 Electromagnetic analysis and FEM simulation	11
2.2.1 Electromagnetic equations	12
2.2.2 Finite element method	16
2.3 Optimization fundamentals	27
2.3.1 Brief introduction to optimization	27
2.3.2 Optimization methods	28
3 Fundamentals of Machine Learning and Literature Review	39
3.1 Classic machine learning	40
3.1.1 Support vector machine	41
3.1.2 Gaussian process regression	44

3.2	Deep learning	47
3.2.1	Deep neural network	49
3.2.2	Convolutional neural network	54
3.3	State-of-the-art: machine learning applications to electric machines	57
3.3.1	Dataset representation	60
3.3.2	Machine learning paradigms	61
3.3.3	Machine learning metamodels	66
3.4	Contribution	69
4	Deterministic Metamodeling-Based Optimization	71
4.1	Introduction	71
4.1.1	Deterministic metamodel	72
4.1.2	ML-based design optimization framework	74
4.2	Dataset acquisition	76
4.2.1	FEA simulation toolchain	76
4.2.2	Dataset introduction	78
4.3	Methodology and model training procedure	82
4.3.1	Motivation	82
4.3.2	Modular neural network and proposed workflow	82
4.3.3	Training of neural networks	83
4.4	Design optimization and results	91
4.5	Summary and discussion	94
5	Probabilistic Metamodeling-Based Optimization	97
5.1	Introduction and methodology	97
5.1.1	Background introduction	98
5.1.2	Bayesian neural networks	99
5.1.3	Framework	101
5.2	Dataset acquisition	103
5.3	Model training and results	104
5.3.1	Training of Bayesian neural networks	105
5.3.2	Training of Gaussian process regression	107
5.3.3	Results and analysis	108
5.4	Uncertainty-based optimization and results	114
5.4.1	Optimization problem	114
5.4.2	Optimization results	115
5.4.3	Sensitivity analysis	119
5.5	Summary and discussion	123

6	Data-Efficient Learning-Based Optimization using Limited Data	125
6.1	Introduction and motivation	125
6.2	Semi-supervised learning-based method	127
6.2.1	Self-training framework and dataset	128
6.2.2	Training of neural networks	134
6.2.3	Conclusion and limitations	142
6.3	Active learning-based optimization	144
6.3.1	Methodology	145
6.3.2	Dataset acquisition	148
6.3.3	Comparative study of metamodels	151
6.3.4	Active learning-based optimization and results	155
6.3.5	Investigation of optimal dataset sizes	168
6.4	Summary and discussion	173
7	Conclusion and Outlook	175
7.1	Result summary of this dissertation	175
7.2	Outlook	177
8	Appendix	181
8.1	Software and package details	181
8.2	Simulation toolchain supplement	182
8.3	Other results	183
8.3.1	Datasets in Chapter 4	183
8.3.2	Dataset in Chapter 5	186
8.3.3	Training results of 6.3	187
	List of figures	189
	List of tables	193
	Bibliography	195

Chapter 1

Introduction

1.1 Motivation

The electric machine, as an essential component for converting electrical and mechanical energy, has emerged as a pivotal force in contemporary industry, energy transition, transportation electrification, and robotics. In the industrial sector, electric machines serve as the “heart” of manufacturing, accounting for about 60 % of global industrial electricity use [1]. A wide variety of electric machines with specialized features and configurations are selected for various applications [2]. For example, high-precision servo motors (e.g., numerical control machine tools [3]) and high-efficiency permanent magnet synchronous motors (PMSMs) (e.g., compressors [4] and pumps [5]) have significantly improved productivity and energy efficiency, while linear motors in semiconductor manufacturing achieve contactless high-speed positioning, promoting automated production line upgrades [6]. Furthermore, PMSMs have also been extensively utilized in the electric vehicle domain due to their high torque density, wide speed range, compact structure, and excellent control ability [7], [8], [9]. In addition, induction motors (IMs) are frequently employed in household appliances and industrial applications, owing to their low cost, simple design, and outstanding reliability [10]. The essential components of an electric machine comprise the stator, rotor, shaft, bearings, housing, cooling system, and electrical terminal box, each possessing distinct functions and specifications. The rotor and stator, as the active parts, have their materials and geometric design intricately linked to the motor’s performance and efficiency. The stator core is typically made of laminated silicon steel, which possesses high magnetic permeability and low eddy

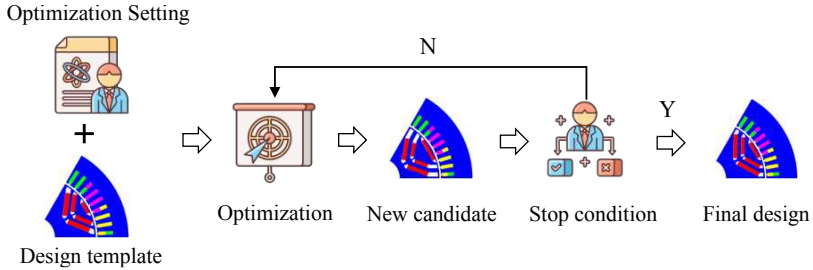


Figure 1.1: Conventional design optimization workflow for electric machines.

current losses, hence reducing energy loss during operation. The stator winding involves copper as the conductive medium. Copper exhibits higher conductivity and is often employed in high-efficiency machines. The rotor core, similarly to the stator core, is typically constructed from laminated silicon steel to reduce losses. IMs feature a rotor designed as a squirrel cage composed of aluminum or copper bars. PMSMs utilize rotors produced from permanent magnet (PM) rare earth materials, including neodymium-iron-boron (NdFeB) or samarium-cobalt (SmCo), which generate a robust magnetic field [11]. However, the utilization of rare earth minerals in electric machine manufacture adversely affects the environment and poses challenges regarding production costs [12]. Therefore, optimizing the geometric parameters of electric machines throughout the design phase is crucial to fulfill specific industrial demands and minimize material usage. A general optimization workflow is depicted in Fig. 1.1. Nevertheless, conventional design optimization methods for electric machines encounter many obstacles:

- 1) Design optimization using finite element analysis (FEA) is a prevalent approach in the industry. The candidate designs obtained during each generation of the optimization process undergo FEA to evaluate the predetermined objective function, typically including the machine's key performance indicators (KPIs) such as maximum torque, maximum power, efficiency, and cost. The computation time for a single design can range from a few minutes to several hours, depending on the varying complexities of different models. In addition, the simulation duration is considerably affected by the computer hardware (e.g., CPUs, RAM). Consequently, the entire optimization process

may endure for days or even months, plagued by high computational complexity and extensive time consumption.

- 2) FEA-based optimization generally does not execute numerous generations, considering limitations in computational time. As the number of input design parameters and output KPIs escalates, the optimization procedure will struggle to effectively and broadly explore the design space. The final optimal solutions may therefore become trapped in a local optimum dilemma. Furthermore, when the dataset utilized for model training is limited, the predicted KPIs generally diverge considerably from the ground truth due to the out-of-distribution problem.
- 3) Analytical methods enable rapid calculation of target KPIs without necessitating advanced hardware assistance, resulting in minimal computational costs. However, these methods typically rely on simplifying assumptions, making it very difficult to obtain analytical solutions for intricate geometries, nonlinear materials, and boundary conditions.

Hence, a computationally efficient alternative method with a certain level of accuracy requires development. Furthermore, it has the ability to explore high-dimensional design space effectively. In this thesis, design optimization for electric machines using machine learning (ML) is proposed to significantly accelerate the optimization process while providing high prediction accuracy.

1.2 Related works

To mitigate the limitations associated with traditional methods, artificial intelligence (AI)-based techniques have drawn raised interest in both academic and industrial domains [13], [14], [15]. ML algorithms generate rapid and precise predictions regarding fresh data by learning the underlying relations between inputs and outputs throughout training datasets. The main difference between FEA-based and ML-based optimization is illustrated in Fig. 1.2. The meta-models used thus far can be broadly classified into three categories: a) classic ML models such as support vector machines (SVMs), boosting algorithms. b) deep learning (DL) models such as multilayer perceptrons (MLPs), convolutional neural networks (CNNs), and generative models. 3) reinforcement learning (RL)-based models. This section outlines briefly the application of various data-driven models throughout diverse scenarios within the domain of electric

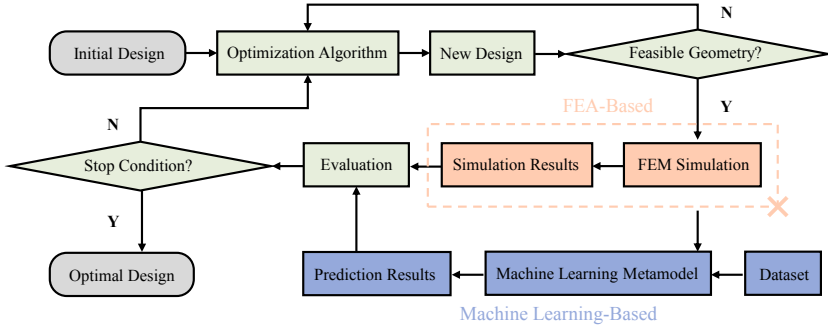


Figure 1.2: Comparison between simulation-based and machine learning-based optimization.

machines. A more detailed literature review will be summarized in Section 3.3.

Metamodeling based on classic ML has been widely utilized in numerous domains [16]. Especially in the realm of rotating electric machines, classic ML models demonstrate their superiority in predicting motor KPIs and computational consumption. In [17], [18], [19], [20], [21], SVMs were applied to predict KPIs for PMSMs or switched reluctance motors (SRMs), such as torque-related parameters, efficiency, PM flux linkage, and total harmonic distortion (THD) of induced voltage. In addition, the extreme learning machine (ELM), XG-Boost models, and Gaussian process regression (GPR) [22] were also employed to compare with SVMs.

Advancements in hardware technology, particularly in graphics processing units (GPUs) and tensor processing units (TPUs), enable deep learning models to handle and train on larger datasets. These hardware components can significantly enhance the capacity for parallel computation and facilitate large-scale data processing, rendering the implementation of deep learning algorithms efficient. The applications of DL in the field of motor design optimization can be categorized into three groups: a) parameter vector-based [23], [24], [25], [26], [27], [28], [29], [30], [31], [32]. b) image-based [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]. c) vector-image combination [46], [47]. In [48] and [49], image-based and parameter vector-based metamodeling were compared, with vector-based approaches necessitating fewer computing

costs and yielding superior prediction accuracy. However, when the inputs undergo modifications, the model must be retrained from scratch. On the other hand, the image-based approaches provide greater flexibility and reusability.

Furthermore, RL-based optimization techniques have received more attention due to their high efficiency and enhanced generalization achieved through interaction with an environment [50], [51], [52]. The basic idea is to employ RL algorithms as a substitute for conventional numerical optimization algorithms. The agent works to optimize long-term rewards through continuous interactions and state transfers until convergence is achieved.

1.3 Objectives

In this work, design optimization methods using ML based on parameter vector inputs for PMSMs are proposed. The benefit is that reduced computational time and resources facilitate a more efficient and extensive exploration of the design space, leading to more optimal designs. The predicted accuracy of data-driven metamodels is one of the main focuses of the research. The excellent performance of a pretrained metamodel on a test set does not always guarantee a high prediction accuracy for the final optimized solutions following the integration of the optimization algorithms. This issue occurs because, with the high dimensionality of input design parameters, the training datasets are inadequate to comprehensively capture the whole design space. After the optimization process, numerous out-of-distribution [53] designs are produced, resulting in the model's inability to accurately forecast them, hence leading to significant prediction errors. Furthermore, the generation of datasets still relies on FEA, indicating that large datasets necessitate considerable time for simulation. A limited initial dataset saves a great deal of time. Yet, insufficient training data may negatively affect the learning capacity of models. Hence, it is a trade-off problem regarding time and the model's generalization capability. To address the above issues, the following works will be conducted.

Firstly, to overcome the necessity to retrain the model from scratch based on parameter vector-based input in case of reparameterization, an adaptive neural network (NN) will be presented. The predictive performance of this deterministic metamodel and the optimization outcomes achieved through the integration with the optimization algorithm will be studied. However, the final optimization results obtained using deterministic models generally

suffer from high average errors, and prediction accuracy is highly dependent on the size of the dataset. Large datasets need prolonged simulation durations. Thus, the subsequent sections will concentrate on achieving the most accurate optimization outcomes achievable given limited data.

Secondly, to assist users in making sound decisions, metamodels must provide further information in addition to the predicted KPIs, several probabilistic metamodels will be proposed. Probabilistic models are able to offer deterministic predicted values while also giving uncertainty over each prediction. Which probabilistic models exhibit superior computing efficiency and are more appropriate for motor design applications? A comprehensive comparative study will be carried out. In addition, it will be investigated whether there is a relation between the uncertainty of the prediction and the final optimization solutions. The combination of uncertainty information with optimization algorithms to guide them to obtain more reliable optimized designs will also be studied.

Thirdly, the metamodels capable of keeping performance with limited sample sizes will be examined. Data-efficient learning techniques [54] have been frequently used in the domain of image processing. Consequently, is it applicable to begin with a limited initial dataset and then enhance the initial dataset with the help of data-efficient learning to rapidly acquire new data? New optimization frameworks based on data-efficient learning will be introduced. Furthermore, to speed up the overall optimization process and obtain optimization results with higher prediction accuracy, how prediction uncertainty can help the data-efficient learning process to achieve these goals will also be explored.

1.4 Outline

This thesis consists of seven chapters as shown in Fig. 1.3.

Chapter 1 introduces the research motivation. Subsequently, the integration of ML in optimizing electric machines is briefly reviewed. Finally, the objectives of the research and the outline of this thesis are summarized.

Chapter 2 starts with the fundamental set of electromagnetic equations in motor design calculations, namely Maxwell's equations. Subsequently, the detailed process of finite element method (FEM) simulation is summarized. Finally, the fundamental optimization problem is introduced. Commonly employed

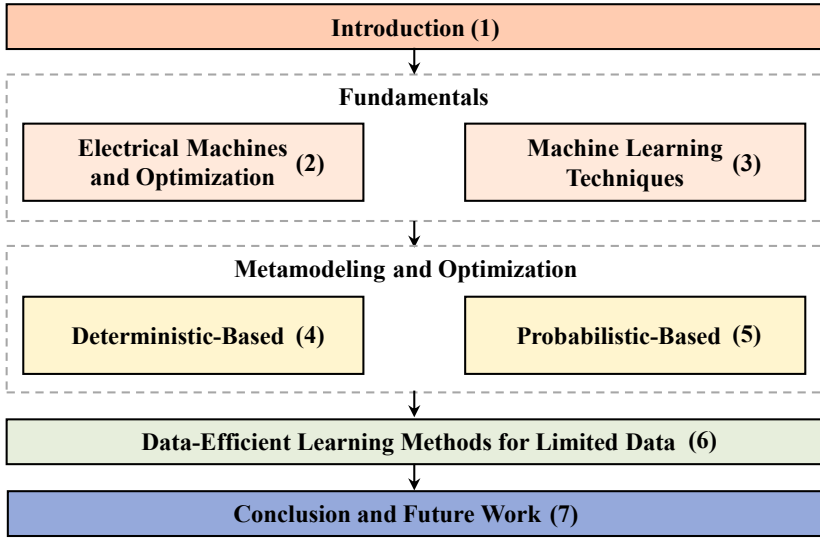


Figure 1.3: Outline of this dissertation.

intelligent algorithms for motor design optimization and optimization methods for updating NN hyperparameters are discussed, respectively.

Chapter 3 mainly provides the principles of many widely used classical machine learning and neural network structures. The use of data-driven techniques in electric machine design and optimization is comprehensively studied and presented. The author's contributions to this research field are also summarized.

Chapter 4 introduces a complete motor optimization process based on deterministic metamodels. First, the concept of deterministic metamodels is explained. The design optimization framework adopted in this thesis is presented. Subsequently, the simulation toolchain used for dataset acquisition is introduced. The neural network employed and its training process are thoroughly explored and detailed. Finally, the advantages and limitations of methods based on deterministic metamodels are discussed.

Chapter 5 presents a novel uncertainty-based optimization method using probabilistic metamodels. First, the definition of probabilistic neural networks is presented, followed by an introduction to Bayesian neural networks. Concurrently, an optimization framework based on probabilistic metamodels is described. Subsequently, the machine learning probabilistic model GPR and the deep learning model Bayesian neural network are trained and compared, respectively. The model demonstrating superior performance is employed for subsequent optimization processes. For optimization, two strategies leveraging uncertainty information from model predictions are proposed. Their respective advantages and disadvantages are discussed. Furthermore, sensitivity analysis is applied to investigate the relationship between prediction uncertainty and robustness. Finally, the superiority of the uncertainty-based optimization approach is summarized, while its remaining shortcomings are also discussed.

Using the probabilistic metamodels introduced in the previous chapter, Chapter 6 proposes two data-efficient learning methods to overcome the time-consuming nature of obtaining simulation data in motor design scenarios. Firstly, a self-training framework based on semi-supervised learning is proposed. A classifier and a predictor are trained separately. They can annotate pseudo labels to a large number of unlabeled data. Then, the model is trained by combining data with pseudo labels and data with true labels. Secondly, four active learning-based optimization strategies are experimentally evaluated and compared. Through active learning, a subset of key data points is selected during each optimization iteration. These key data points are then added back to the initial dataset to enhance model training. Finally, these two approaches are summarized and discussed.

In the end, Chapter 7 summarizes and discusses all results obtained in this thesis. Finally, future applications of data-driven methods in the field of electric motors will be concluded.

Chapter 2

Fundamentals of Electric Machines and Optimization

The design of electric machines includes technological, economic, and environmental goals, deeply influencing energy efficiency, product competitiveness, and technological innovations. As electrification advances (e.g., carbon neutrality objectives, expansion of electric transportation), the significance of motor design will be increasingly emphasized as a key catalyst for industrial development. A proficient electromagnetic design can dictate the fundamental performance of motors, influencing efficiency, power density, torque characteristics, and operational stability, while minimizing losses, reducing material usage, and enhancing economic viability. To obtain multiple candidate designs that fulfill the specifications, motor optimization plays an essential role. By integrating optimization methods, the design space can be effectively navigated, ultimately leading to the identification of appropriate designs.

In this chapter, an overview of electric machines is provided, encompassing the fundamental components and operational principles. Additionally, the general process of electromagnetic computations for electric machines is presented. The mathematical concepts and simulation procedures of commonly used FEM are summarized. Finally, design optimization methods are categorized into gradient-based algorithms and intelligent algorithms for discussion.

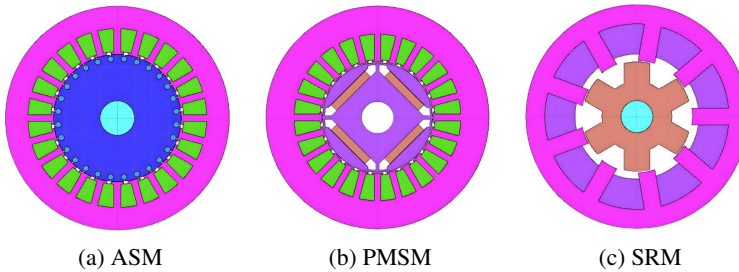


Figure 2.1: Cross-sections of different electric machines. (a) Asynchronous motor. (b) Permanent magnet synchronous motor. (c) Switched reluctance motor.

2.1 Brief introduction to electric machines

Electric machines are devices that convert electrical energy into mechanical energy (motors) or mechanical energy into electrical energy (generators) via electromagnetic interactions. They are composed of two essential components: a stationary part called the stator and a rotating part called the rotor. The stator generally comprises three main elements: the stator core, the stator windings, and the housing. The stator core includes laminated steel sheets to minimize eddy current losses and establish a low-reluctance pathway for magnetic flux. The stator windings, typically constructed from insulated copper or aluminum conductors, are placed in slots within the core and generate a rotating magnetic field when supplied with alternating current (AC) power (120° phasic and positional shifts). The housing secures the stator assembly, offers mechanical support, and frequently assists in heat dissipation. The rotor includes the rotor core, the rotor conductor (squirrel cage bars or wound winding) or the PMs, and the shaft. In PMSMs, the rare-earth magnets provide a constant magnetic field without the need for external excitation (unlike wound rotors), which interacts with the rotating magnetic field of the stator to produce torque and maintain synchronization. In IMs, the most common rotor conductor is the squirrel cage, which is made of conductive bars (aluminum or copper) embedded in a laminated iron core. The current is driven by the stator's rotating magnetic field instead of external excitation. Fig. 2.1 depicts examples of different machine types. In SRMs, the stator has several salient poles, each containing an individual winding. When the windings are energized, they generate a magnetic field that attracts the nearest section of the rotor into alignment. The rotor is composed of protruding poles

manufactured from ferromagnetic material. It does not have windings or permanent magnets; rather, it positions itself to minimize magnetic reluctance, aligning with the active stator poles.

2.2 Electromagnetic analysis and FEM simulation

The electromagnetic analysis is essential for the design, optimization, and performance assessment of electric machines. Precise forecasting of electromagnetic behavior is crucial for ensuring efficiency, reliability, and adherence to design specifications. An electromagnetic analysis can be conducted via two major approaches: analytical methods and numerical methods [55]. Analytical methods involve solving Maxwell's equations or their reduced variants through mathematical formulas. These methods often rely on assumptions such as linear materials, simplified geometries, or particular boundary conditions, facilitating closed-form or semi-analytical solutions. For electric machines, analytical methods are commonly used to derive approximate analytical expressions for electromagnetic quantities such as the internal magnetic field, inductances, and torque ripple. The benefits are evident in the following areas: 1) Enabling fast computations that are ideal for parameter studies and early-stage design. 2) Providing deep physical insights into the behavior of machines. 3) Only requiring basic hardware with minimal memory usage. However, analytical methods suffer from limited accuracy due to simplifying assumptions. They are not suitable for complex geometries or highly nonlinear materials [56]. In contrast, numerical methods address Maxwell's equations over intricate geometries and materials by discretizing the domain. The most common approach is the FEM, although alternatives such as the finite difference method (FDM) and boundary element method (BEM) are also employed. The BEM discretizes only the boundaries of the problem domain, rather than the entire volume as in FEM. It can diminish the problem's dimensionality as it necessitates only meshed boundaries; yet, it faces challenges in handling nonlinear constitutive laws, such as nonlinear magnetic materials [57]. The FDM substitutes derivatives with finite differences on a structured grid. It converts partial differential equations into algebraic equations, which are resolved at discrete grid points. Nevertheless, it struggles to deal with complicated structures and nonlinear problems well [58]. Therefore, the FEM is the preferred numerical method for the electromagnetic analysis of electric machines. The complete machine geometry (stator, rotor, air gap, windings, etc.) is

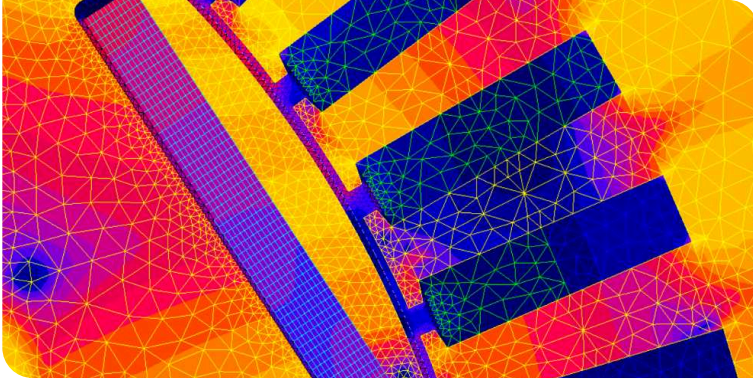


Figure 2.2: Discretization of a 2D field area by a grid of triangles and rectangles.

discretized into small finite elements (e.g., triangles in 2D, tetrahedra in 3D), and Maxwell's equations are solved over these elements to calculate fields, forces, currents, and other quantities. It can proficiently manage complex geometries, including slots, magnets, and skewing, while also handling magnetic saturation and hysteresis [59]. In addition, the FEM is supported and developed by much commercial software (e.g., ANSYS Maxwell, Altair Flux) and open-source tools (e.g., FEMM).

2.2.1 Electromagnetic equations

Maxwell's equations are four fundamental partial differential equations describing the interactions between electric and magnetic fields and their sources, electric charges and currents. Attributed to James Clerk Maxwell, their modern formulation was later developed using vector calculus and constitutes the foundation of classical electromagnetism. Maxwell's equations reveal the energy conversion mechanisms in electric machines, guide the design of magnetic fields and windings, and serve as the theoretical foundation for simulation and optimization. The effective functioning of modern electric motors, the construction of novel structures, and the analysis of electromagnetic performance all depend on the comprehension and implementation of Maxwell's equations. In this section, Maxwell's equations are discussed. The first equation is referred to as Gauss's law for electricity. This equation indicates that electric charge is the

origin of the electric field. The divergence of the electric displacement field $\vec{D}$ is equivalent to the volume charge density ρ , implying that the electric flux through a closed surface is proportional to the charge enclosed within that surface. It can be expressed as below

$$\nabla \cdot \vec{D} = \rho. \quad (2.1)$$

The second equation represents Gauss's law for magnetism, which can be described as follows

$$\nabla \cdot \vec{B} = 0, \quad (2.2)$$

where $\vec{B}$ is the magnetic flux density. This equation shows that magnetic fields are source-free, meaning that there are no magnetic monopoles. Magnetic flux lines are always closed loops. Therefore, the magnetic flux through any closed surface is zero.

The third equation is Faraday's law of induction, which means that a time-varying magnetic field can induce an electric field in the neighboring region. It is described by

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}, \quad (2.3)$$

where $\vec{E}$ represents the electric field, and $\nabla \times$ is the curl operator.

The last equation consists of two parts, known as the Ampère–Maxwell law, which can be written as

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t}, \quad (2.4)$$

where $\vec{H}$ is the magnetic field intensity and $\vec{J}$ is the electric current density. Ampère's law addresses only the magnetic field produced by steady currents, whereas Ampère–Maxwell's law incorporates the concept of displacement currents, allowing the law to account for the generation of magnetic fields by time-varying electric fields.

The material equations are commonly assumed to be linear relationships for simplification:

$$\vec{D} = \epsilon \vec{E}, \quad (2.5)$$

$$\vec{B} = \mu(\vec{H} + \vec{M}_{\text{pm}}), \quad (2.6)$$

$$\vec{J} = \sigma \vec{E}, \quad (2.7)$$

where ϵ is the electric permittivity, which can also be calculated by multiplying the vacuum permittivity (ϵ_0) and the relative permittivity (ϵ_r), namely $\epsilon = \epsilon_0 \epsilon_r$. μ is the magnetic permeability and it can be similarly expressed as $\mu = \mu_0 \mu_r$. It should be noted, that the magnetic material law can generally be nonlinear and tensorial. In particular, for anisotropic magnetic materials, μ may be represented by a permeability tensor and may depend on the magnetic field strength and the operating point. Furthermore, when magnetic hysteresis is taken into account, the material response depends not only on the instantaneous magnetic field but also on the magnetic history of the material. (2.6) therefore represents a simplified formulation of the general magnetic material law. σ is the electrical conductivity. $\vec{M}_{\text{pm}}$ represents the magnetization of permanent magnets. To facilitate following calculations, the magnetic vector potential $\vec{A}$ is introduced herein:

$$\vec{B} = \nabla \times \vec{A}, \quad (2.8)$$

The relationship between magnetic vector potential and flux density is analogous to the relationship between magnetic field intensity and current density, as shown in Fig. 2.3. According to mathematical identities, it can be obtained:

$$\nabla \cdot (\nabla \times \vec{A}) = 0, \quad (2.9)$$

which means that the incorporation of $\vec{A}$ inherently fulfills (2.2). Subsequently, (2.3) can be further calculated using (2.8):

$$\nabla \times \left(\vec{E} + \frac{\partial \vec{A}}{\partial t} \right) = 0. \quad (2.10)$$

In a simply connected region, the vector field of $(\vec{E} + \frac{\partial \vec{A}}{\partial t})$ can be regarded as a conservative field, i.e., it can be represented as a gradient of a scalar potential field. Then

$$\vec{E} + \frac{\partial \vec{A}}{\partial t} = -\nabla V \quad (2.11)$$

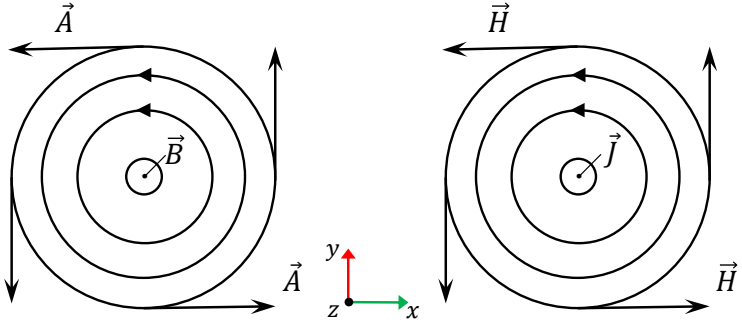


Figure 2.3: Analogy between magnetic vector potential and magnetic field intensity.

is achieved, where V indicates the electric scalar potential. In the quasi-static case, ($\frac{\partial \vec{D}}{\partial t} = 0$), substituting (2.6), (2.7), (2.8), and (2.11) into (2.4) and simplifying yields the following:

$$\vec{J} = \nabla \times \vec{H} \quad (2.12)$$

$$\sigma \vec{E} = \nabla \times \mu^{-1} \vec{B} - \nabla \times \vec{M}_{\text{pm}} \quad (2.13)$$

$$\sigma \vec{E} = \nabla \times (\mu^{-1} \nabla \times \vec{A}) - \nabla \times \vec{M}_{\text{pm}} \quad (2.14)$$

$$\vec{J}_s + \nabla \times \vec{M}_{\text{pm}} = \nabla \times (\mu^{-1} \nabla \times \vec{A}) + \sigma \frac{\partial \vec{A}}{\partial t}, \quad (2.15)$$

where $\vec{J}_s$ is the source current density, which is equal to $-\sigma \nabla V$, and $\nabla \times \vec{M}_{\text{pm}}$ implies current from PMs. The equation (2.15) is referred to as the A-formulation, which serves as the foundation for 2D FEA. For further simplification, $\frac{\partial \vec{A}}{\partial t}$ can be disregarded in a magnetostatic case. In addition, both current terms can be regarded as a total current $\vec{J}_{\text{tot}}$ [60]:

$$\vec{J}_{\text{tot}} = \nabla \times (\mu^{-1} \nabla \times \vec{A}). \quad (2.16)$$

In the case of 2D FEM, the magnetic field $\vec{B}$ usually distributes within the x-y plane, meaning that the direction of $\vec{A}$ is perpendicular to the observation plane,

i.e., along the z-axis. This likewise pertains to the current density $\vec{J}_{\text{tot}}$. Therefore, the above equation can be further streamlined step by step:

$$\vec{A} = A_z(x, y)\hat{z}, \quad (2.17)$$

$$\mu^{-1}\nabla \times \vec{A} = \mu^{-1} \left(\frac{\partial A_z}{\partial y} \hat{x} - \frac{\partial A_z}{\partial x} \hat{y} \right), \quad (2.18)$$

$$J_{\text{tot},z} = -\mu^{-1} \left(\frac{\partial^2 A_z}{\partial x^2} + \frac{\partial^2 A_z}{\partial y^2} \right), \quad (2.19)$$

where μ is assumed to be spatially invariant and isotropic. (2.19) is the so-called Poisson's equation that delineates how currents within the machine generate magnetic potentials, which subsequently produce magnetic fields. It is applicable solely for 2D magnetostatic analysis, as eddy currents are ignored.

In transient analysis, individual discrete time steps are considered. The differential equation is substituted with a difference equation:

$$\frac{\partial A_z}{\partial t} \approx \frac{1}{\Delta t} (A_{z,t+\Delta t} - A_{z,t}), \quad (2.20)$$

which requires extensive computational resources. However, it is appropriate to all two-dimensional situations.

2.2.2 Finite element method

The FEM is a numerical approach employed to address problems with boundary values or partial differential equations in sophisticated physical systems. The process involves partitioning complex geometry or structures into numerous simple, interconnected regions known as finite elements, creating localized mathematical models for each element, and ultimately deriving an approximate solution for the entire system [61]. In this section, the basic process of FEM will be introduced, which is segmented into five phases, as depicted in Fig. 2.4.

The first phase:

In FEM, discretization denotes the breakdown of a continuum (continuous structure or region) into a collection of tiny finite units (elements), hence converting a continuous problem into a computable discrete problem. This enables the

solving of partial differential equations by matrix equations. Furthermore, it enhances the solution's efficiency while preserving sufficient approximation accuracy. In the Fig. 2.5, a PMSM example is illustrated. Initially, a geometric model of the analytic object must be created for defining its shape and scope. This area is referred to as the solution domain. The continuous region is subsequently segmented into a large amount of small elements by meshing. These elements may consist of line segments, triangles, quadrilaterals, tetrahedrons, or hexahedrons, depending on the dimension and configuration of the problem. Each element consists of a bunch of nodes, which are discrete points representing the locations of variables to be solved afterwards, such as displacement, temperature, etc. The mesh density significantly influences the computational accuracy of FEM: a denser mesh yields more precise results; however, it also increases computational cost, resulting in slower computations.

The second phase:

Upon completion of the meshing, it is necessary to specify basis functions, which are mathematical functions used to represent the solution within each element of a discretized domain. In addition, nodes of each element are generally allocated to the corner points (first-order elements). To enhance precision, they may also

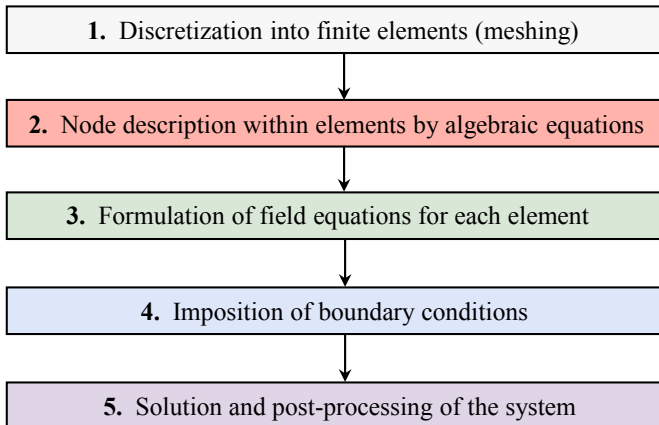


Figure 2.4: General FEM simulation process for electric machines [62].

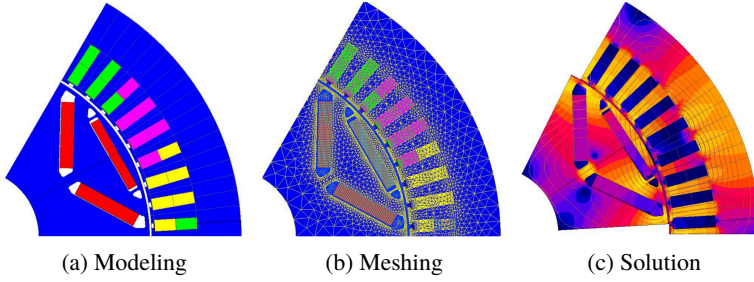


Figure 2.5: Different phases of FEM of a PMSM example.

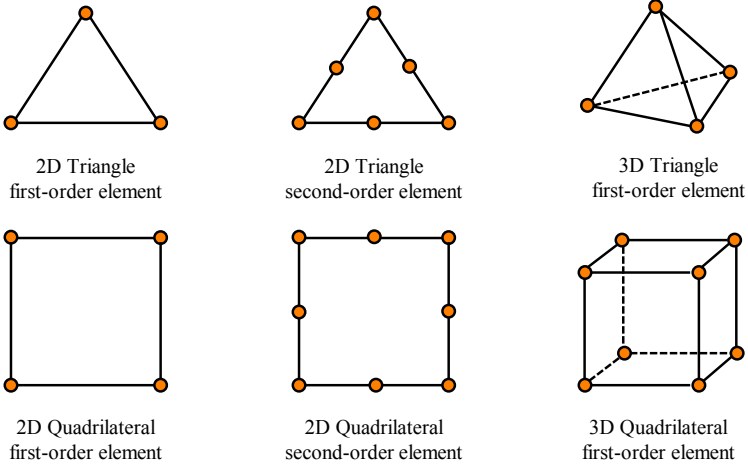
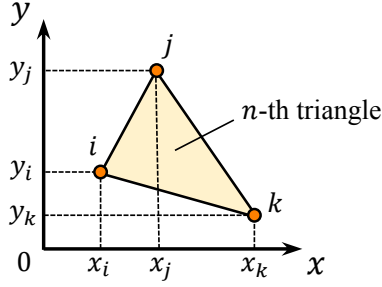


Figure 2.6: Illustration of first-order and second-order elements.

be assigned to the midpoints of the sides (second-order elements), as shown in Fig. 2.6. The following assumes that the vector potential varies linearly in the element. The simplest 2D element is a 3-node triangle (see Fig. 2.6). The basis function of the n -th element can be defined as [61]:

$$A_{z,n} = a_n + b_n x + c_n y, \quad (2.21)$$


 Figure 2.7: Definition of the n -th triangle element.

where a_n , b_n , and c_n depend on the coordinates of the element. For a 3-node triangle, each node is described as follows:

$$A_{z,i} = a_n + b_n x_i + c_n y_i, \quad (2.22)$$

$$A_{z,j} = a_n + b_n x_j + c_n y_j, \quad (2.23)$$

$$A_{z,k} = a_n + b_n x_k + c_n y_k. \quad (2.24)$$

This means three equations and three unknowns, which can be solved to obtain:

$$a_n = \frac{1}{2S_n} (A_{z,i} (x_j y_k - x_k y_j) + A_{z,j} (x_k y_i - x_i y_k) + A_{z,k} (x_i y_j - x_j y_i)), \quad (2.25)$$

$$b_n = \frac{1}{2S_n} (A_{z,i} (y_j - y_k) + A_{z,j} (y_k - y_i) + A_{z,k} (y_i - y_j)), \quad (2.26)$$

$$c_n = \frac{1}{2S_n} (A_{z,i} (x_k - x_j) + A_{z,j} (x_i - x_k) + A_{z,k} (x_j - x_i)), \quad (2.27)$$

$$S_n = \frac{1}{2} (x_j y_k + x_i y_j + x_k y_i - x_j y_i - x_k y_j - x_i y_k), \quad (2.28)$$

where S_n denotes the area of the triangle, which consists of the nodes i, j , and k . Substituting a_n, b_n, c_n , and S into (2.21) yields:

$$\begin{aligned}
 A_{z,n} = & A_{z,i} \left(\frac{(x_j y_k - x_k y_i) + x \cdot (y_i - y_k) + y \cdot (x_k - x_j)}{x_j y_k + x_i y_j + x_k y_i - x_j y_i - x_k y_i - x_i y_k} \right) \\
 & + A_{z,j} \left(\frac{(x_k y_i - x_i y_k) + x \cdot (y_k - y_i) + y \cdot (x_i - x_k)}{x_j y_k + x_i y_j + x_k y_i - x_j y_i - x_k y_i - x_i y_k} \right) \\
 & + A_{z,k} \left(\frac{(x_k y_i - x_i y_k) + x \cdot (y_i - y_j) + y \cdot (x_j - x_i)}{x_j y_k + x_i y_j + x_k y_i - x_j y_i - x_k y_i - x_i y_k} \right).
 \end{aligned} \tag{2.29}$$

This can also be expressed as:

$$A_{z,n} = \sum_v^m N_v(x, y) \cdot A_{z,v} \tag{2.30}$$

$$= N_i(x, y) \cdot A_{z,i} + N_j(x, y) \cdot A_{z,j} + N_k(x, y) \cdot A_{z,k}, \tag{2.31}$$

where $N(x, y)$ is the so-called shape function and m implies the number of nodes within the triangle element. Shape functions enable the interpolation of variable values at the nodes throughout the entire element, thus establishing a local approximation model. They have two properties that hold for most finite element shape functions:

$$\text{Partition of unity:} \quad \sum N_v(x, y) = 1. \tag{2.32}$$

$$\text{Kronecker delta property:} \quad N_v(x_s, y_s) = \begin{cases} 1, & v = s \\ 0, & v \neq s \end{cases}. \tag{2.33}$$

The sum of all the shape functions at any point in the triangle is equal to 1. This is essential for the ability to describe trivial constant solutions exactly. Besides, every shape function is equal to 1 at one of the nodes and equal to 0 at all other nodes.

The third phase:

The field equations for each element can be derived using the weighted residuals, which is an approximate method for solving partial differential equations. It

constructs an approximate solution equation by making the residual zero in a weighted sense [63]. Introduce A_z^* as an approximation and R as a residual:

$$R = \mu^{-1} \left(\frac{\partial^2 A_z^*}{\partial x^2} + \frac{\partial^2 A_z^*}{\partial y^2} \right) + J_{\text{tot},z}, \quad (2.34)$$

The residual is multiplied by a weighting function W and the integral over the computational domain Ω is set to zero:

$$\iint_{\Omega} RW \, dx dy = 0. \quad (2.35)$$

Substituting (2.34) in the equation:

$$- \iint_{\Omega} W \mu^{-1} \left(\frac{\partial^2 A_z^*}{\partial x^2} + \frac{\partial^2 A_z^*}{\partial y^2} \right) dx dy = \iint_{\Omega} W J_{\text{tot},z} dx dy. \quad (2.36)$$

The left side term can be integrated by parts as follows:

$$\begin{aligned} - \iint_{\Omega} W \mu^{-1} \left(\frac{\partial^2 A_z^*}{\partial x^2} + \frac{\partial^2 A_z^*}{\partial y^2} \right) dx dy &= \iint_{\Omega} \mu^{-1} \left(\frac{\partial W}{\partial x} \frac{\partial A_z^*}{\partial x} + \frac{\partial W}{\partial y} \frac{\partial A_z^*}{\partial y} \right) dx dy \\ &\quad - \oint_C W \mu^{-1} \frac{\partial A_z^*}{\partial \hat{n}} dC, \end{aligned} \quad (2.37)$$

where C is the boundary and $\hat{n}$ is the outward unit normal vector. Substituting this result into (2.36) and converting the surface integrals into summations over small elements. Subsequently, the following equation can be obtained by treating A_z^* as A for simplification [61]:

$$\begin{aligned} \sum_{e=1}^N \left[\frac{1}{\mu^e} \iint_{\Omega^e} \left(\frac{\partial W^e}{\partial x} \frac{\partial A^e}{\partial x} + \frac{\partial W^e}{\partial y} \frac{\partial A^e}{\partial y} \right) dx dy \right] - \frac{1}{\mu^e} \frac{\partial A^e}{\partial \hat{n}} \int_C W^e dC \\ = J_{\text{tot},z} \iint_{\Omega^e} W^e dx dy, \end{aligned} \quad (2.38)$$

where N represents the number of linear triangle elements and Ω^e is the domain of the e -th element. Here, the Galerkin approach [64] is utilized for further calculation, which means that the weighted function W is identical to the shape function $N(x,y)$ mentioned in (2.30). Normally, the term $\frac{\partial A^e}{\partial \hat{n}}$ is set to zero

because of the Neumann boundary condition (in the fourth phase). Then, substituting (2.30) in (2.38):

$$\begin{aligned} \sum_{e=1}^N \left[\sum_{s=1}^m \sum_{v=1}^m \left[A_v^e \frac{1}{\mu^e} \iint_{\Omega^e} \left(\frac{\partial N_s^e}{\partial x} \frac{\partial N_v^e}{\partial x} + \frac{\partial N_s^e}{\partial y} \frac{\partial N_v^e}{\partial y} \right) dx dy \right] \right. \\ \left. - \sum_{e=1}^N \left[\sum_{s=1}^m \left[\iint_{\Omega^e} J_{\text{tot},z} N_s^e dx dy \right] \right] \right] = 0, \end{aligned} \quad (2.39)$$

where s denotes the index for the test functions corresponding to each node of the element, and v reflects the index of the nodes used for the approximation within each element. m is the total number of nodes of each element.

The fourth phase:

Boundary conditions in FEA describe the specifications required on the model's boundary when formulating a finite element model to imitate the boundary behavior of a real system. These conditions may include displacement, force, pressure, temperature, etc., which relate to limitations or external loads in a physical system. Without reasonable boundary conditions, the system of equations may either lack a solution or provide outcomes that are incompatible with physical reality. Thus, appropriately establishing boundary conditions not only guarantees the stability and convergence of computations but also directly influences the precision and dependability of the final analytical results [65].

- **Dirichlet boundary condition:** The value of the unknown field quantity is explicitly provided at the boundary. In FEM of electric motors, the Dirichlet boundary condition is often used to directly specify the vector potential at the boundary $C = C_{\text{dir}} \cup C_{\text{neu}}$ of the computational domain

$$\vec{A}_z|_{C_{\text{dir}}} = f_1(x, y). \quad (2.40)$$

If $\vec{A}_z|_{C_{\text{dir}}} = 0$, it is known as a homogeneous boundary condition, otherwise, it is classified as an inhomogeneous condition.

- **Neumann boundary condition:** It is the derivative of the field potential in the normal direction specified at the boundary

$$\frac{\partial \vec{A}}{\partial \vec{n}}|_{C_{\text{neu}}} = f_2(x, y), \quad (2.41)$$

where $\vec{n}$ is the normal unit vector. Similarly, if $f_2(x,y) = 0$, it implies a homogeneous boundary.

- Robin boundary condition: It specifies both the vector potential and its derivative at the boundary. This condition is normally inapplicable to magnetic fields.
- Symmetry and periodic boundary conditions: They are mainly used to minimize the computational domain, enhance computational efficiency, and ensure that the solution satisfies to the geometric and physical principles of the system. Their fundamental concept is to leverage the redundancy of the motor architecture and the distribution of the magnetic field.

The fifth phase:

An assembly equation, comprising the equations from each element, can be described as follows [60]:

$$\sum_{e=1}^N ([\mathbf{K}^e][\mathbf{A}^e] - [\mathbf{T}^e]) = 0, \quad (2.42)$$

where $\mathbf{K}$ is the stiffness matrix. (2.42) can be further expressed

$$K_{v,s}^e = \frac{1}{\mu^e} \iint_{\Omega^e} \left(\frac{\partial N_s^e}{\partial x} \frac{\partial N_v^e}{\partial x} + \frac{\partial N_s^e}{\partial y} \frac{\partial N_v^e}{\partial y} \right) dx dy, \quad (2.43)$$

$$[\mathbf{A}^e] = [A_1^e, \dots, A_N^e]^\top, \quad (2.44)$$

$$T_s^e = \iint_{\Omega^e} J_{\text{tot},z} N_s^e dx dy, \quad (2.45)$$

where $K_{v,s}^e$ denotes the element (v, s) of the matrix $[\mathbf{K}^e]$, A_n^e and T_n^e denote the n -th entry of the vector $[\mathbf{A}^e]$ and the vector $[\mathbf{T}^e]$, respectively. Once the equation for each node is established, the system of equations can be solved, with each node corresponding to a vector potential A_z . The solution phase can be categorized into two types: linear and nonlinear. This mostly relies on material characteristics (especially the B-H curve of the core), boundary conditions, and the occurrence of effects such as saturation, eddy currents, and hysteresis. Upon resolving the system of equations, the parameters of interest can be derived through post-processing.

- Magnetic flux density $\vec{B}$: Assuming the application of first-order triangle elements, where A is linear, the flux density can be calculated as follows:

$$B_{x,n} = \frac{dA_{z,n}}{dy} = \frac{1}{2S_n} [A_{z,i}(x_k - x_j) + A_{z,j}(x_i - x_k) + A_{z,k}(x_j - x_i)], \quad (2.46)$$

$$B_{y,n} = -\frac{dA_{z,n}}{dx} = \frac{1}{2S_n} [A_{z,i}(y_k - y_j) + A_{z,j}(y_i - y_k) + A_{z,k}(y_j - y_i)]. \quad (2.47)$$

- Magnetic flux Φ : It is generally defined as the surface integral of flux density $\vec{B}$ across area Ω and can be rewritten using (2.8):

$$\Phi = \iint_{\Omega} \vec{B} d\Omega = \iint_{\Omega} \nabla \times \vec{A} d\Omega = \oint_C \vec{A} dC. \quad (2.48)$$

In the 2D scenario, the magnetic flux between two points (x_1, y_1) and (x_2, y_2) can be simply represented as the difference between vector potentials:

$$\Phi = l_{Fe} \cdot [A(x_1, y_1) - A(x_2, y_2)], \quad (2.49)$$

where l_{Fe} is the length of the machine in the z direction.

- Flux linkage Ψ : The flux linkage produced by a coil can be determined by calculating the difference between the mean values of the vector potential in the conductive regions on both sides (left and right) of the coil.

$$\Psi = \frac{l_{Fe} \cdot N_t}{S} \cdot \left(\int_{S, \text{left}} \vec{A} d\Omega - \int_{S, \text{right}} \vec{A} d\Omega \right), \quad (2.50)$$

where N_t denotes the number of turns and S denotes the cross-sectional area of the slot.

- Electromagnetic torque M : Multiple approaches available for determining electromagnetic torque, including Maxwell stress tensor [66] and d/q axis transformation methods. In this work, the torque is calculated during the post-processing with the help of variables from d/q axis [67]:

$$M_{\text{avg}} = \frac{1}{2} mp (\Psi_d i_q - \Psi_q i_d) = \frac{1}{2} mp (\Psi_{\text{pm}} i_q + (L_d - L_q) i_d i_q), \quad (2.51)$$

where m is the number of phases and p is the number of pole pairs. In the d/q-coordinate system, which is derived from Park transformation [68], Ψ_d and Ψ_q describe the d-axis and q-axis flux linkages, respectively. i_d , i_q and L_d , L_q denote currents and inductances in the d- and q-axis, respectively. Ψ_{pm} represents the flux linkage from the permanent magnet. It is the conventional average torque expression. More precisely, torque is also related to the rotor's position θ

$$M_e = M_e(\theta, i_d, i_q). \quad (2.52)$$

It is important to note that the torque generated by the motor during operation is not completely smooth but rather exhibits periodic or irregular fluctuations over time or angle, which is known as torque ripple M_{ripple} [69]. The torque ripple mainly comes from cogging torque, non-ideal back inductive electromotive force (EMF) waveforms, magnetic saturation, mechanical factors, etc. [70]. The torque ripple can be described as below:

$$M_{\text{ripple}} = M_{ep,\max} - M_{ep,\min}, \quad (2.53)$$

where $M_{ep,\max}$ and $M_{ep,\min}$ are the maximum and minimum electromagnetic torque over one electrical period, respectively. The traditional d-q equation describes the main component of the electromagnetic torque, while the position-dependent FEM calculation explicitly accounts for the torque ripple.

- Loss P_{loss} : This work primarily discusses the losses associated with PMSMs, which mainly include copper losses P_{Cu} , iron losses P_{Fe} , mechanical losses P_{mech} , and PM loss P_{pm} [71], as depicted in Fig. 2.8. The copper losses come mostly from the Joule heat generated by the current passing the winding, which can be calculated by

$$P_{\text{Cu}} = P_{\text{Cu,DC}} + P_{\text{Cu,AC}}. \quad (2.54)$$

The direct current (DC) copper loss is essentially the loss caused by the winding DC resistance. The AC copper loss relates to the loss caused when the winding is energized by AC, resulting from the skin effect and proximity effect [72], which lead to non-uniform current distribution and a rise in equivalent resistance. It is more sensitive to the current frequency and conductor configuration than DC copper loss [73]. The iron losses mostly consist of eddy current loss P_{eddy} , hysteresis loss P_{hys} , and additional loss P_{addl} due to the influence

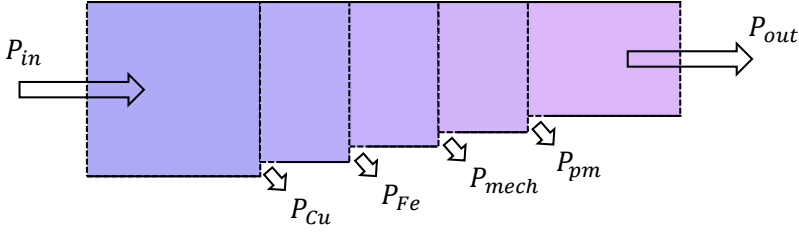


Figure 2.8: Loss diagram of a PMSM.

of an alternating magnetic field. When excitation is sinusoidal, they can be expressed by [74], [75]:

$$P_{Fe} = P_{eddy} + P_{hys} + P_{addl} \quad (2.55)$$

$$= k_e f^2 B_m^2 + k_h f B_m^\alpha + k_a f^{1.5} B_m^{1.5}, \quad (2.56)$$

where k_e denotes the eddy current loss coefficient. k_h and α are the hysteresis loss coefficient and the exponent for silicon steel lamination (1.7~2.0), respectively. k_a is the addition loss coefficient. B_m is the peak value of the flux density and f is the frequency. Mechanical losses refer to energy losses caused by friction in bearings and seals, as well as air resistance generated by the rotation of the rotor during motor operation [76]. As the motor operates at higher speeds, the air friction on the rotor surface grows, leading to an increase in mechanical loss. PM loss mostly is caused by eddy currents generated in the PM by alternating magnetic fields and harmonics, which are transformed into heat. Excessive levels may result in an increased temperature of PMs, leading to demagnetization [77].

- **Power P and efficiency η :** The efficiency of an electric machine indicates its capacity to transform input electrical energy into mechanical energy effectively, serving as an essential indicator of motor performance. It can be calculated in two different ways:

$$\text{Direct method: } \eta = \frac{P_{out}}{P_{in}} = \frac{2\pi \cdot M \cdot n}{P_{el}}, \quad (2.57)$$

where n is the rotation speed of the machine and P_{el} is the input electrical power. η is also expressed by:

$$\text{Indirect method: } \eta = \frac{P_{in} - P_{loss}}{P_{in}} = \frac{P_{el} - P_{loss}}{P_{el}}. \quad (2.58)$$

It is worth noting that the results obtained by these two methods are usually not equal. The indirect method is demonstrably more precise, particularly for highly efficient devices. Nonetheless, the challenge lies in quantifying the losses directly through measurement techniques [78].

2.3 Optimization fundamentals

Electric machines are essential components of new energy vehicles, smart manufacturing, and renewable energy power generation systems. In this context, the performance requirements for motors are becoming progressively complicated and varied, including enhanced power density, improved efficiency, reduced noise and vibration, increased operating reliability, and diminished manufacturing costs. Nonetheless, these performance specifications frequently exhibit mutually restrictive relationships: for instance, boosting power density may result in higher losses and temperature increases, whereas minimizing noise may necessitate complex designs and raised prices. Consequently, multi-objective optimization methods offer a viable solution to this dilemma. Engineers can utilize mathematical modeling, finite element analysis, and powerful algorithms to generate a range of trade-off solutions across several indicators, providing adaptable design alternatives for diverse application contexts. This method not only considerably lowers the R&D cycle but also lessens dependence on costly prototype testing, therefore reducing overall R&D and production expenses. In this section, the standard mathematical formulation of optimization will be presented, accompanied by an overview of commonly used optimization methods.

2.3.1 Brief introduction to optimization

The concept of optimization, often heard in ordinary dialog, describes in mathematical contexts the identification of the combination of selected variables that either maximizes or minimizes the objective function within a certain set of constraints. For example, in the design of industrial motors, variables can include the air gap length, sizes of slots and teeth, PM dimensions, and so on. The objective function is based on various KPIs of the motor, such as torque maximization,

cost minimization, and torque ripple minimization. Constraints may encompass material volume and maximum temperature. A typical optimization problem is normally written as:

$$\min_{x \in \mathbb{R}^n} f(x), \quad (2.59)$$

$$\text{s.t. } g_i(x) \leq 0, \quad i = 1, \dots, m \quad (2.60)$$

$$h_j(x) = 0, \quad j = 1, \dots, p \quad (2.61)$$

$$x \in \mathcal{X}, \quad (2.62)$$

where $x = (x_1, x_2, \dots, x_n)$ is the set of selected variables. $f(x)$ denotes the objective function. $g_i(x)$ and $h_j(x)$ represent an inequality and an equality constraint, respectively. $\mathcal{X}$ is the feasible region for each variable, which means $x_i^L \leq x_i \leq x_i^U$. When multiple objective functions coexist simultaneously:

$$\min_x (f_1(x), f_2(x), \dots, f_k(x)). \quad (2.63)$$

At this point, the solution is not unique, but rather a set of Pareto optimal solutions, which corresponds to a multi-objective optimization (MOO) problem. A Pareto optimal solution (non-dominated solution) is a solution where no other feasible solution exists that is at least as good in all objectives and strictly better in at least one objective [79]. For the MOO problem (2.63), a solution x^* is Pareto optimal if there is no $x \in \mathcal{X}$ satisfies:

$$f_i(x) \leq f_i(x^*), \quad \forall i \quad (2.64)$$

and

$$f_j(x) < f_j(x^*). \quad \text{for at least one } j \quad (2.65)$$

The set of Pareto optimal solutions is known as the Pareto front. For a 2D optimization problem that aims to minimize both f_1 and f_2 simultaneously, its Pareto front is shown in Fig. 2.9.

2.3.2 Optimization methods

Various optimization techniques may be employed for specific obstacles. Bayesian optimization [80] and random search [81] are normally employed for hyperparameter optimization in ML models. For weight updates in NN back-propagation, the most commonly used method is gradient-based optimization [82]. Intelligent optimization approaches are widely applied to motor design

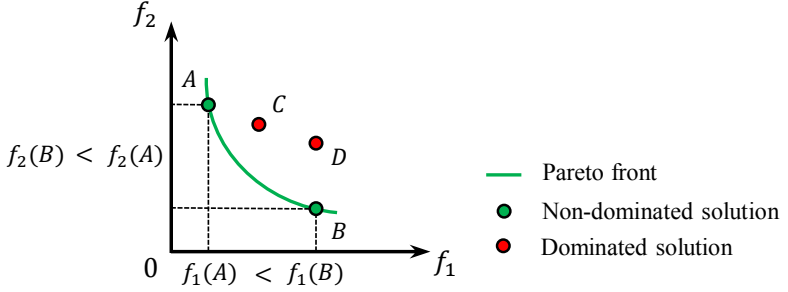


Figure 2.9: An example Pareto front for a 2D-objective optimization with min-min.

optimization [83], [84], [85], as illustrated in Fig. 2.10. The initial design is inputted, followed by the integration of a computed metamodel with optimization methods to iteratively produce new designs until the termination criteria are satisfied. Conventional approaches often use analytical methods [86], [87] and FEM [88], [89], [90], [91] to evaluate each design candidate. Nevertheless, owing to the precision limitations of analytical approaches and the protracted nature of finite element methods, an increasing amount of research is shifting towards surrogate models. Here are several often utilized surrogate models: response surface model [92], [93], [94], [95], Kriging-based models [96], [97], [98], traditional ML models [18], [99], [100], [101], DL models [35], [46], [47], [102]. They are evaluated across four dimensions. DL typically achieves better prediction accuracy; however, it necessitates large amounts of data. In situations involving limited sample sizes, conventional ML tends to be an appropriate choice, as it maintains a degree of prediction accuracy. In the following, several popular optimization techniques will be briefly introduced.

Gradient descent

One simple way to find the minimum value of an objective function is to use gradient descent (GD). It employs gradient information to iteratively modify parameters and identify the optimal target value [103]. In ML and DL, it is the fundamental technique for training model's weights and biases. The basic form of GD is taking a step in the direction of the gradient, with the step length being proportional to the gradient's magnitude, as shown in Algorithm 1.

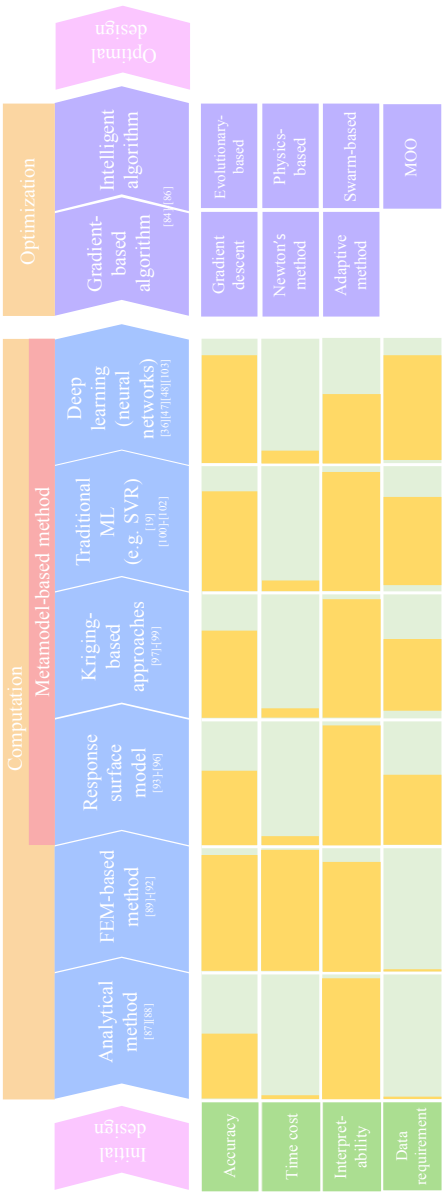


Figure 2.10: Comparison of different optimization models and categorization of generally used optimization methods

Algorithm 1 Gradient descent

Input: Objective function $\mathcal{L}$, learning rate η , number of steps K ,
initial parameter θ

- 1: **for** $k = 0, \dots, K - 1$ **do**
- 2: $\theta_{k+1} \leftarrow \theta_k - \eta \cdot \nabla_{\theta} \mathcal{L}(\theta_k)$
- 3: **end for**

Output: Trained parameter $\theta^* = \theta_K$

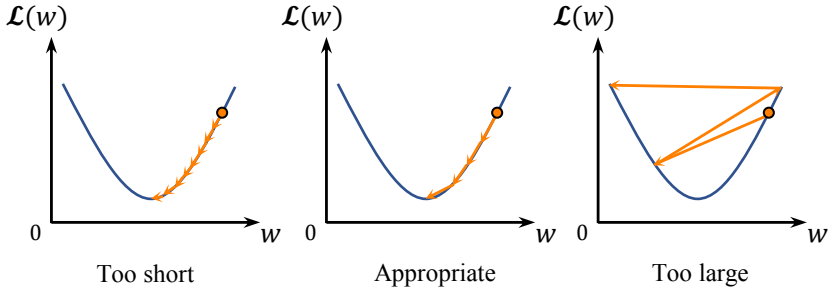


Figure 2.11: Gradient descents with different learning rates.

$\nabla_{\theta} \mathcal{L}(\theta^k)$ describes the current gradient of the objective function. The learning rate η and the number of steps K are two hyperparameters that directly influence optimization convergence, convergence speed, and final performance. A large learning rate and few steps may expedite the process but lack stability, risking non-convergence. A small learning rate combined with numerous steps guarantees stable training; nonetheless, it is time-intensive and may become stuck at saddle points. A moderate learning rate with an appropriate number of steps is generally the most effective strategy, as depicted in Fig. 2.11

To overcome shortcomings of the basic GD, such as large oscillations and difficulty in stably converging to the optimal solution, momentum-based GD has been widely adopted [104]. The momentum method draws inspiration from the concept of inertia in physics. When an object undergoes acceleration, it is influenced not only by the current force but also by its previous velocity. Similarly, parameter updates depend not only on the current gradient but also on the cumulative trend of historical gradients. The concrete calculation process is shown

in Algorithm 2. v^k is the velocity, namely a running average gradients. β denotes the momentum factor between 0 and 1. The introduction of momentum improves convergence and allows parameters to cross saddle points, progressing toward more optimal solutions.

Algorithm 2 Momentum-based gradient descent

Input: Objective function $\mathcal{L}$, learning rate η , number of steps K ,
initial parameter θ , momentum β

```
1: for  $k = 0, \dots, K - 1$  do  
2:    $v_{k+1} \leftarrow \beta \cdot v_k + (1 - \beta) \cdot \nabla_{\theta} \mathcal{L}(\theta_k)$   
3:    $\theta_{k+1} \leftarrow \theta_k - \eta \cdot v_{k+1}$   
4: end for
```

Output: Trained parameter $\theta^* = \theta_K$

Adaptive method

To further address the issues of large gradient fluctuations and unstable convergence, adaptive methods have been suggested. One commonly used adaptive method is called adaptive moment estimation (Adam) [105]. It is an adaptive learning rate optimization algorithm that combines momentum and RMSProp [103]. The momentum implies the first-order moment, which accelerates convergence and reduces oscillations. The RMSProp indicates the second-order moment, which adaptively scales the learning rate to enhance stability. More details can be checked in Algorithm 3.

The first-order decay rate β_1 controls the gradient's exponentially weighted average and it is suggested to be 0.9. The closer it is to 1, the greater the influence of the historical gradients, resulting in smoother updates but slower responses. The second-order decay rate β_2 regulates the exponentially weighted average of the squared gradient and it is suggested to be 0.999. The learning rate changes more consistently, and the smoothness increases as it approaches 1. ϵ serves as a constant to avoid a denominator of 0. Adam's unique advantage lies in its ability to adaptively adjust the learning rate for each parameter, thereby accelerating the model's convergence speed. However, it still has some limitations, including the potential to miss the global optimum and the failure to converge in specific circumstances [106].

Algorithm 3 Adaptive moment estimation

Input: Objective function $\mathcal{L}$, step size η , number of steps K ,
initial parameter θ , exponential decay rates β_1, β_2

- 1: Initialized 1st moment vector $m_0 \leftarrow 0$
- 2: Initialized 2nd moment vector $v_0 \leftarrow 0$
- 3: **for** $k = 0, \dots, K - 1$ **do**
- 4: $g_{k+1} \leftarrow \nabla_{\theta} \mathcal{L}(\theta_k)$ Obtain gradients of the objective function
- 5: $m_{k+1} \leftarrow \beta_1 \cdot m_k + (1 - \beta_1) \cdot g_{k+1}$ Update the first moment vector
- 6: $v_{k+1} \leftarrow \beta_2 \cdot v_k + (1 - \beta_2) \cdot g_{k+1}^2$ Update the second moment vector
- 7: $\hat{m}_{k+1} \leftarrow \frac{m_{k+1}}{1 - \beta_1^{k+1}}$ Compute bias-corrected first moment estimate
- 8: $\hat{v}_{k+1} \leftarrow \frac{v_{k+1}}{1 - \beta_2^{k+1}}$ Compute bias-corrected second moment estimate
- 9: $\theta_{k+1} \leftarrow \theta_k - \eta \cdot \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1} + \epsilon}}$ Update parameters with $\epsilon = 10^{-8}$
- 10: **end for**

Output: Trained parameter $\theta^* = \theta_K$

Genetic algorithm

The genetic algorithms (GAs) are a category of search algorithms derived from the principles of natural evolutionary theory. They may provide superior solutions for search, optimization, and learning by emulating natural selection and reproductive processes. Their similarity to natural evolution allows them to overcome certain challenges faced by conventional search and optimization algorithms, especially in problems that involve many parameters and complex mathematical formulations [107]. The three basic principles of a general GA are: Natural selection - individuals better adapted to their environment are more likely to survive; Inheritance - genes from superior individuals are passed on to the next generation; Mutation - genes periodically undergo mutations, enhancing variety and preventing a plateau in local optima. A general GA is shown in Algorithm 4.

During each generation of the algorithm, individuals are evaluated using the fitness function F , also known as the objective function. Individuals with higher fitness represent better solutions, making them more likely to be selected for reproduction and transmit their attributes to the subsequent generation. As the GA progresses, solution quality improves and fitness increases. It terminates once solutions with satisfactory fitness values are found. Following the evaluation of each individual's fitness within the population, selection identifies the individ-

Algorithm 4 Genetic algorithm

```
1:  $k \leftarrow 0$ 
2: Initialize population  $P(k)$ 
3: Evaluate  $P(k)$  using fitness function  $F$ 
4: while isNotTerminated() do
5:    $P_p(k) \leftarrow P(k).selectParents()$ 
6:    $P_c(k) \leftarrow P_p.crossOver()$ 
7:    $P_m(k) \leftarrow P_c.mutate()$ 
8:   Evaluate  $P_m(k)$  using fitness function  $F$ 
9:    $P(k+1) \leftarrow (P(k), P_m(k)).nextGeneration()$ 
10:   $k \leftarrow k+1$ 
11: end while
```

uals designated for reproduction to obtain the next generation. Subsequently, through crossover, portions of chromosomes from the selected parental samples in the current generation are exchanged. Mutation randomly updates the population, introducing new patterns to chromosomes and encouraging exploration in previously unexplored regions of the solution space. Fig. 2.12 demonstrates the processes of crossover and mutation. GAs have the benefit of improved efficiency in locating global optima, since crossover and mutation processes preserve population variety. It concurrently enables parallel processing to enhance computational performance while continuously learning. However, GAs also require additional hyperparameter selection, such as the mutation rate. Furthermore, large populations may demand substantial computational resources, making the process highly time-consuming before achieving satisfactory results [108].

Particle swarm optimization

In 1995, James Kennedy and Russell Eberhart built a simplified computational model. Following years of research, this model finally developed into the particle swarm optimization (PSO) algorithm [109]. The PSO is inspired by the feeding behavior of flocks of birds. Suppose a flock of birds searches randomly for food in a forest without knowing the exact locations of food sources. Each bird follows its own search path and records food locations. Simultaneously, the entire flock shares its food records to help other birds adjust their search directions. After a period of searching, the entire flock can locate the area with the highest amount of food. The PSO emulates bird behavior in a flock by

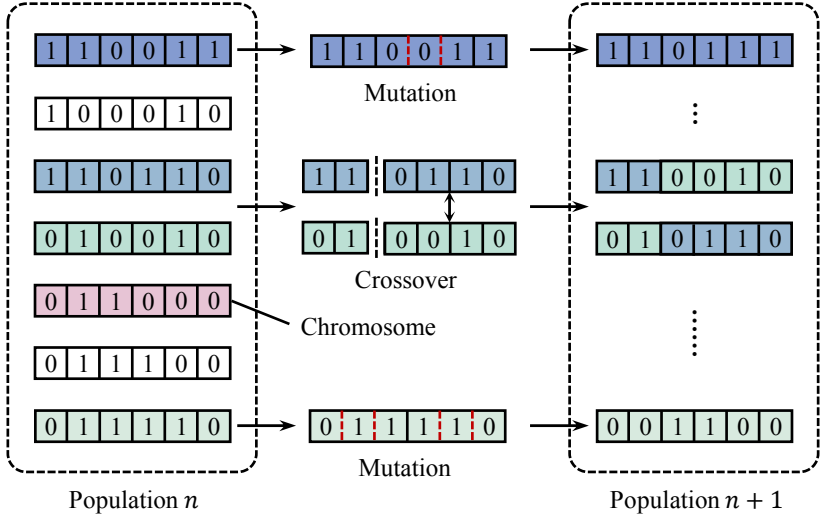


Figure 2.12: Illustration of the genetic algorithm.

creating massless particles characterized only by two attributes: velocity and position. Velocity denotes the velocity and direction of motion. The current position of the particle i in m -dimensional space is represented as a vector $x_i = (x_{i1}, x_{i2}, \dots, x_{im})$ and the velocity is $v_i = (v_{i1}, v_{i2}, \dots, v_{im})$. Each particle in each position possesses a corresponding fitness value determined by the objective function and is aware of both the best position $pbest_i$ discovered so far and its current position x_i . This represents the past experience of each individual particle. In addition, each particle knows the best position $gbest$ discovered by all particles in the entire group so far, which can be regarded as a shared collective experience. In each generation, particles determine their next movement by combining their own experience with the best experience within their group. The detailed algorithm flow is shown in the Algorithm 5.

In the velocity update formula, c_1 and c_2 represent the cognitive coefficient and social coefficient, respectively. c_1 indicates the weight assigned to a particle's subsequent actions, determined by its experience component, while c_2 represents the weight influenced by the experience of other particles. ω denotes the

Algorithm 5 Particle swarm optimization

```
1:  $k \leftarrow 0$ 
2: for each particle  $i$  do
3:   for each dimension  $j$  do
4:     Initialize position  $x_{ij}^{(k)}$  randomly within feasible region
5:     Initialize velocity  $v_{ij}^{(k)}$  randomly within feasible region
6:   end for
7:   Calculate fitness value  $F_i^{(k)}$  using fitness function  $F$ 
8: end for
9: Archive the best position  $pbest_i$  and best fitness value  $F_{i,best}$  for each particle in history
10: Archive the global best position  $gbest$  and global best fitness value  $F_{best}$  among all particles in history
11: while  $isNotTerminated()$  do
12:   for each particle  $i$  do
13:     for each dimension  $j$  do
14:        $v_{ij}^{(k+1)} \leftarrow \omega v_{ij}^{(k)} + c_1 r_1 (pbest_{ij} - x_{ij}^{(k)}) + c_2 r_2 (gbest_j - x_{ij}^{(k)})$ 
        Update particle velocity
15:        $x_{ij}^{(k+1)} \leftarrow x_{ij}^{(k)} + v_{ij}^{(k+1)}$  Update particle position
16:     end for
17:     Calculate fitness value  $F_i^{(k+1)}$  using fitness function  $F$ 
18:     Update the best position  $pbest_i$  and best fitness value  $F_{i,best}$  for each particle in history
19:   end for
20:   Update the global best position  $gbest$  and global best fitness value  $F_{best}$  among all particles in history
21:    $k \leftarrow k + 1$ 
22: end while
```

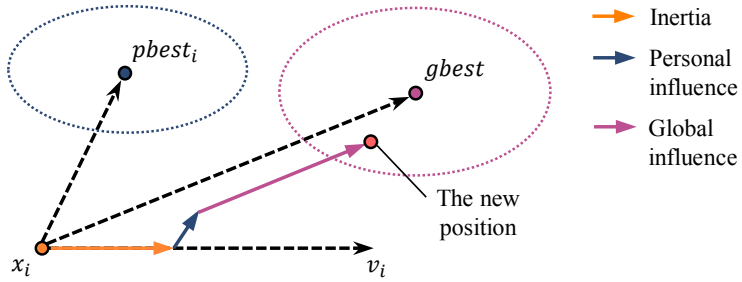


Figure 2.13: Illustration of the process of velocity update.

inertia weight. When ω is large, particles have strong global search skills but display limited local search capabilities. r_1 and r_2 are two random values between 0 and 1, which increase the randomness of the search. Fig. 2.13 describes how to update the velocity for each particle. The PSO approach has benefits like rapid convergence, few parameters, and simplicity in implementation. In high-dimensional optimization problems, it converges to optimum solutions more swiftly than genetic algorithms. Nonetheless, it is sensitive to hyperparameters, requiring good starting parameters. It also has the potential to get trapped in local optima [110].

Chapter 3

Fundamentals of Machine Learning and Literature Review

The design optimization of electric machines with metamodels (surrogate models) is no longer a new concept. More than twenty years ago, the response surface as a metamodel was used with numerical optimization to enhance the design process of electromagnetic devices [111]. This method is simple to implement and computationally fast, making it suitable for low-dimensional problems. However, its accuracy is limited in high-dimensional and nonlinear complex systems [13]. Subsequently, with the advancement of ML, an increasing number of ML models have been applied in the field of motor design optimization, such as the classic radial basis function (RBF) models [112]. Classic ML models, including SVMs and extreme gradient boosting (XGBoost), have also seen extensive application due to their improved accuracy. However, as computing hardware continues to advance, datasets have grown increasingly massive. Classic ML models, constrained by computational complexity and limited expressive capacity, are no longer suitable for large datasets. This has led to the emergence of DL models, such as deep neural networks (DNNs) and CNNs. DL models can learn highly complex functions through a number of nonlinear hidden layers, capturing intricate patterns within high-dimensional data. Big data provides sufficient samples, enabling high-capacity models to learn more generalized features without overfitting. Furthermore, DL models'

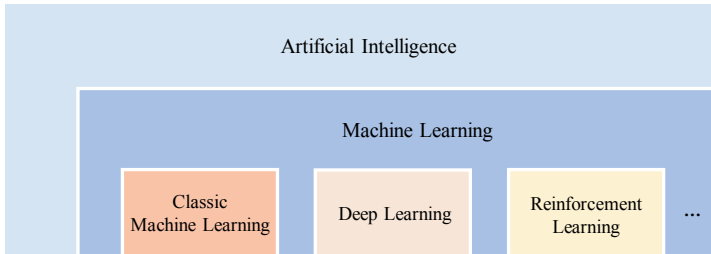


Figure 3.1: The relation among AI, ML, and DL.

training typically leverages GPUs and TPUs, fully utilizing parallel computing and supporting distributed training. This allows ultra-large datasets to be partitioned, with multiple nodes training in parallel, significantly reducing training time [113]. The relationship between ML and DL is illustrated in the Fig. 3.1. The ML, in a broad sense, includes classic ML, DL, RL, and more techniques.

In this chapter, classic ML and DL models will be explained separately. Finally, a review of the application of the state-of-the art ML models in motor design optimization is provided.

3.1 Classic machine learning

Classic ML is an important subset of ML, whereby models generate predictions by discerning patterns from data. Meanwhile, they demonstrate robust interpretability, indicating that the foundational decision rationale is clear and clearly comprehensible to humans. Representative classic ML methods include:

1. *Support Vector Machines (SVMs)* [114]: SVMs are often used for classification and regression tasks. They aim to find a hyperplane in feature space that optimizes the classification margin, maximizing the distance between data points in distinct categories. In cases when data is not linearly separable, kernel functions can be applied to transform the data into a higher-dimensional space, therefore achieving linear separability. The advantages include a robust theoretical framework and exceptional efficiency with limited samples, non-linear data, and high-dimensional datasets. Nevertheless, SVMs demonstrate

considerable training complexity, entail substantial computing expenses with extensive sample sizes, and are acutely sensitive to kernel function choices [13].

2. *Extreme Gradient Boosting (XGBoost)* [115]: XGBoost builds a powerful ensemble model by iteratively training multiple CART decision trees [116], each fitted to the residuals of the previous tree, and then weighting and summing the predictions from all trees. Additionally, XGBoost applies first- and second-order gradients to precisely optimize the loss function, enabling faster model convergence and higher prediction accuracy. This model is also highly efficient in engineering implementation, supporting multi-threading and distributed training, making it capable of handling large-scale structured data tasks. However, XGBoost involves numerous hyperparameters, and different hyperparameter combinations may yield varying results, requiring considerable time and effort for hyperparameter tuning. Additionally, it may consume substantial memory when processing large datasets [117].
3. *Gaussian Process Regression (GPR)* [118]: GPR is a probabilistic, nonparametric Bayesian method used to address regression issues. The fundamental concept states that the distribution of data points follows a Gaussian process, indicating that the joint distribution of function values for every finite set of sample points is a multivariate Gaussian distribution. Through the modeling of training data, GPR can provide a confident area including mean and variance for every prediction, rather than a singular predicted value. This allows it to provide forecasts while also quantifying the uncertainty associated with such predictions. In addition, GPR possesses a solid theoretical foundation, is closely linked to Bayesian methods, and yields results with good interpretability. However, an extensive number of inverse matrices required during the training phase results in a computational complexity of $O(n^3)$ for precise inference when applying GPR on a dataset of size n , which implies that computational costs become prohibitively expensive when dealing with large volumes of data [119].

3.1.1 Support vector machine

As one of the most widely used classic ML models, SVM has also found extensive application in the field of motor optimization. This section will briefly introduce the most fundamental binary classification SVM [15]. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, m; y_i \in \{-1, 1\}\}$. SVM seeks to find a hyperplane

in the sample space based on the training set, thereby separating samples of different classes. In a 2D space, the equation of the hyperplane, which is a straight line, can be expressed as

$$ax_1 + bx_2 + c = 0. \quad (3.1)$$

When extended to multidimensional cases, a hyperplane can be represented in vector form as

$$\boldsymbol{\omega}^T \mathbf{x} + b = 0, \quad (3.2)$$

where $\boldsymbol{\omega} = (\omega_1, \omega_2, \dots, \omega_d)$ is the normal vector and b is the displacement term. The distance from any point $\mathbf{x}$ to a hyperplane $(\boldsymbol{\omega}, b)$ can be expressed as

$$r = \frac{|\boldsymbol{\omega}^T \mathbf{x} + b|}{\|\boldsymbol{\omega}\|}. \quad (3.3)$$

If the hyperplane $(\boldsymbol{\omega}, b)$ can correctly classify all training samples, for every $(\mathbf{x}_i, y_i) \in \mathcal{D}$, the following holds:

$$\begin{cases} \boldsymbol{\omega}^T \mathbf{x}_i + b > 0, & y_i = +1 \end{cases} \quad (3.4)$$

$$\begin{cases} \boldsymbol{\omega}^T \mathbf{x}_i + b < 0, & y_i = -1 \end{cases} \quad (3.5)$$

The training sample points nearest to the hyperplane fulfill the equality in the above equation; they are referred to as support vectors. The interval between two contrasting samples is referred to as the margin γ , as shown in Fig. 3.2. The margin can be described as:

$$\gamma = \frac{2}{\|\boldsymbol{\omega}\|}. \quad (3.6)$$

Within a given space, multiple hyperplanes may exist that can separate training samples. Finding the optimal separating hyperplane is equivalent to finding the maximum margin within the sample space [120]. Hence, this problem can be expressed mathematically as

$$\begin{aligned} & \max_{\boldsymbol{\omega}, b} \quad \frac{2}{\|\boldsymbol{\omega}\|}, \\ & \text{s.t. } \boldsymbol{\omega}^T \mathbf{x}_i + b, \quad i = 1, \dots, m. \end{aligned} \quad (3.7)$$

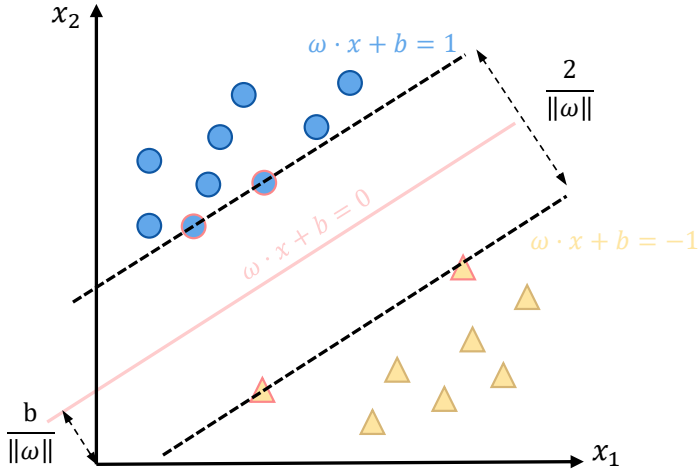


Figure 3.2: Illustration of SVM principle.

The above equation is equivalent to

$$\begin{aligned} \min_{\omega, b} \quad & \frac{1}{2} \|\omega\|^2, \\ \text{s.t.} \quad & \omega^T x_i + b, \quad i = 1, \dots, m, \end{aligned} \quad (3.8)$$

which implies a convex quadratic programming [121]. In constrained optimization problems, the Lagrange duality is often employed to transform the primal problem into a dual problem, with the solution to the dual problem yielding the solution to the primal problem. This approach is taken because the complexity of the dual problem is typically lower than that of the primal problem. Therefore,

the optimization problem can be reformulated as a dual problem for solution, which is expressed as [120]

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) \quad (3.9)$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \dots, n, \quad (3.10)$$

$$\sum_{i=1}^n \alpha_i y_i = 0, \quad (3.11)$$

where α_i denotes the Lagrange multiplier. After solving for α using sequential minimal optimization [122], calculate ω and b to obtain the final model:

$$f(\mathbf{x}) = \omega^T \mathbf{x} + b = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b. \quad (3.12)$$

3.1.2 Gaussian process regression

In this dissertation, GPR serves as an important metamodel in the motor optimization process. The following content will focus on the mathematical principles of GPR. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \dots, N\}$, where $\mathbf{x}$ is the input and $\mathbf{y}$ is the target. GPR utilizes a Gaussian process as a prior and conducts Bayesian inference directly within the function space. It can be described by its mean function and covariance function [123]:

$$f(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad (3.13)$$

where $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ represent the mean and covariance (kernel) functions, respectively. They can be calculated by

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \quad (3.14)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \quad (3.15)$$

The mean value $m(\mathbf{x})$ is generally set to zero since the dataset is normalized with a mean of zero during pre-processing. The kernel function $k(\mathbf{x}, \mathbf{x}')$ represents the relationship between the different input points, whereby Gaussian process can predict unknown point values from point-to-point correlations based on the

training data. In a regression scenario, a prior distribution of output $\mathbf{y}$ on a given dataset $\mathcal{D}$ can be expressed as

$$\mathbf{y} \sim \mathcal{N}(0, K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}), \quad (3.16)$$

where σ_n is the Gaussian noise. $K(\mathbf{X}, \mathbf{X})$ is a Gram matrix and can be calculated by

$$K(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \quad (3.17)$$

The selection of kernel function greatly influences a model's predictive performance, hence kernel functions are often regarded as hyperparameters and tuned appropriately [124]. Here, a commonly used kernel function example, namely RBF, is expressed by

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma_l^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}'), \quad (3.18)$$

$$\delta(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \mathbf{x} = \mathbf{x}' \\ 0 & \mathbf{x} \neq \mathbf{x}' \end{cases}, \quad (3.19)$$

where σ_f , σ_l , and σ_n are hyperparameters, which can be treated as Θ as a whole. The σ_f defines the vertical span of the function, while the σ_l denotes the pace at which the correlation between two points decreases as distance increases. The amplitude of the prediction function increases as σ_f increases, and the fitted curve also exhibits a greater range of variation. If σ_l is large, the kernel function diminishes at a slow rate, yielding a very smooth function and varies gradually. The σ_n indicates unexpected noise in observational data. If σ_n is larger, the model will tolerate more data bias, and predictions will be smoother [125]. These hyperparameters Θ can be updated through training process by the log marginal likelihood:

$$\begin{aligned} \Theta^* &= \arg \max_{\Theta} \log p(\mathbf{y} | \mathbf{X}, \Theta) \\ &= \arg \max_{\Theta} -\frac{1}{2} \mathbf{y}^\top (\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \\ &\quad - \frac{1}{2} \log |\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}| - \frac{N}{2} \log 2\pi. \end{aligned} \quad (3.20)$$

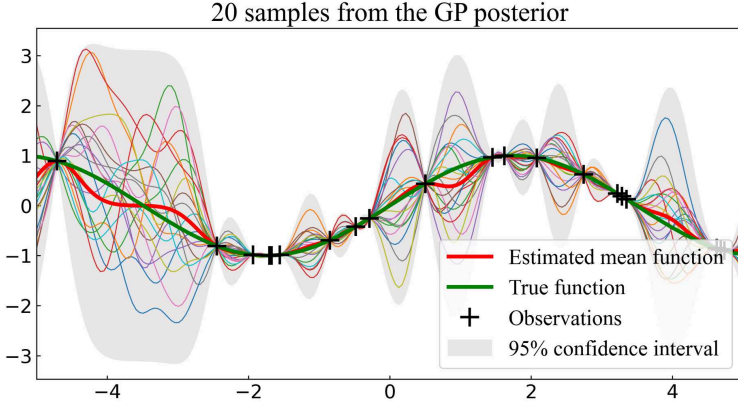


Figure 3.3: Illustration of an example GPR model.

The trained model now predicts outcomes of a new observation, including inputs $\mathbf{X}^*$ and outputs $\mathbf{y}^*$. Assume that the training data and the new observation follow a joint distribution, which can be expressed as follows [123]:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} & K(\mathbf{X}, \mathbf{X}^*) \\ K(\mathbf{X}^*, \mathbf{X}) & K(\mathbf{X}^*, \mathbf{X}^*) \end{bmatrix}\right). \quad (3.21)$$

Using the conditional distribution formula of multivariate normal distribution, the posterior distribution given the observation $\mathbf{y}$ can be obtained [126]:

$$\mathbf{y}^* | \mathbf{X}^*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\bar{\mathbf{y}}^*, \text{cov}(\mathbf{y}^*)), \quad (3.22)$$

where $\bar{\mathbf{y}}^*$ and $\text{cov}(\mathbf{y}^*)$ can be calculated by:

$$\bar{\mathbf{y}}^* = K(\mathbf{X}^*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{y}, \quad (3.23)$$

$$\begin{aligned} \text{cov}(\mathbf{y}^*) &= K(\mathbf{X}^*, \mathbf{X}^*) \\ &\quad - K(\mathbf{X}^*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} K(\mathbf{X}, \mathbf{X}^*). \end{aligned} \quad (3.24)$$

The $\bar{\mathbf{y}}^*$ represents the average predicted value (point estimation) and the $\text{cov}(\mathbf{y}^*)$ is a variance matrix to provide the uncertainty for each estimated point. Fig. 3.3 illustrates a regression example of GPR [125]. The green line is the true

function, while the red line is the estimated mean function using GPR based on the black observations. The other 20 different colored lines represent 20 possible prediction functions, and the gray areas denote the predicted confidence interval.

3.2 Deep learning

Deep learning (DL) is an algorithm that emulates the operation of the human brain's neural networks, allowing computers to autonomously recognize patterns from extensive datasets and consequently perform tasks such as perception, recognition, and prediction. The human brain has multiple linked neurons that transfer information during data processing. These neurons communicate by electrical impulses and chemical agents, transmitting information across various brain areas [127]. In this regard, artificial neural networks (ANNs) are the fundamental architecture of DL. They replicate this biological behavior via artificial neurons made up of software components known as nodes. These nodes transmit and relay information using numerical calculations instead of the chemical impulses present in the brain [128]. Fig. 3.4 illustrates the composition of a typical ANN. DL can be understood as a kind of flowchart, starting from the input layer and ending at the output layer. Sandwiched between these two layers are hidden layers, responsible for processing information at different levels and adjusting their weights as they continuously receive new data. DL models may comprise hundreds of hidden layers, each playing a part in detecting links and patterns within the dataset. The input layer comprises numerous nodes representing various input parameters. The trajectory of data through each layer is dictated by the computational rules established for each node. As data advances through each layer, the model derives observations or significant insights, culminating in the final output or analytical results [128].

The main DL models include convolutional neural networks (CNNs), recurrent neural networks (RNNs), transformers, generative adversarial networks (GANs), and variational autoencoders (VAEs), etc. CNNs were initially employed in image processing, demonstrating capability in extracting spatial characteristics across varying scales from local to global. As a result, they are extensively used in applications like image classification, object identification, autonomous driving perception systems, and video analysis [129]. Concurrently, CNNs have been widely used in motor optimization. Cross-sectional diagrams of electric motors are often used as inputs, which will be further elaborated upon in Section 3.3. RNNs are adept at handling sequential data. They can preserve temporal

information over time intervals, making them highly effective in applications like voice recognition, text production, and time series forecasting [130]. The transformer model is one of the most significant architectures in recent years. It utilizes self-attention techniques to identify relationships between any points within sequences, hence avoiding the need for recurrent structures. This method provides exceptional training efficiency and robust performance. The transformer has emerged as a fundamental framework in natural language processing, driving huge language models like ChatGPT and BERT. It has also been widely used in images, speech, and multi-modal applications [131]. GANs consist of a generator and a discriminator, enabling the model to learn to generate realistic data through mutual competition. GANs have found prominent applications in image generation, style transfer, and data-efficient learning [132]. VAEs are unsupervised learning models that compress data using an encoder and reconstruct it using a decoder. They are employed for tasks such as feature extraction, dimensionality reduction, denoising, and anomaly detection [133].

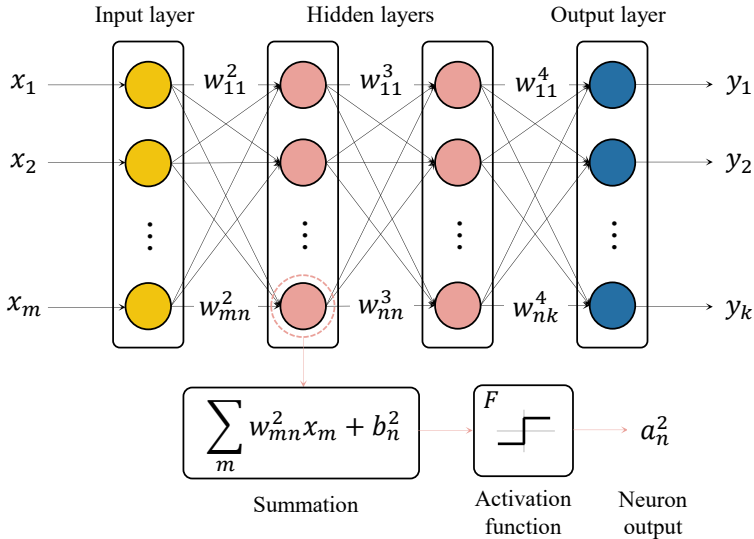


Figure 3.4: An example architecture of an artificial neural network.

This chapter will mainly discuss the two most prevalent DL metamodels utilized in motor optimization: DNNs and CNNs. They are each relevant to two separate scenarios: CNNs are used for motor image inputs, whilst DNNs are applied for parameter vector inputs.

3.2.1 Deep neural network

Deep NNs are a type of ML model composed of multiple layers of neurons, designed to automatically learn complex patterns in data. Fig. 3.4 is a four-layer NN, including one input layer, one output layer, and two hidden layers. x represents the input data with m features and y denotes the output ground truth with k targets. The number of neurons for each hidden layer is n . w_{mn}^2 is the weight from the m -th neuron in the first (input) layer to the n -th neuron in the second layer. b_n^2 is the bias of the n -th neuron in the second layer. Based on Fig. 3.4, the fundamental working principles of NNs will be derived below.

Forward propagation

Assume that the maximum layer depth is L . w_{ij}^l is the weight from the i -th neuron in the $(l-1)$ -th layer to the j -th neuron in the l -th layer. b_j^l is the bias of the j -th neuron in the l -th layer. z_j^l represents the input of the j -th neuron in the l -th layer and a_j^l represents the output of the j -th neuron in the l -th layer. F is the activation function. Then, the equations for any given neuron can be obtained:

$$z_j^l = \sum_i w_{ij}^l a_i^{l-1} + b_j^l, \quad (3.25)$$

$$a_j^l = F\left(\sum_i w_{ij}^l a_i^{l-1} + b_j^l\right). \quad (3.26)$$

The above can be rewritten as:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l, \quad (3.27)$$

$$\mathbf{a}^l = F(\mathbf{z}^l) = F(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l). \quad (3.28)$$

Forward propagation is the process by which NNs perform inference. It enables NNs to derive outputs through layer-by-layer computations from inputs while providing the error information required for training, serving as the foundation for both prediction and learning.

Backpropagation

Backpropagation is a method used in the training of NNs to calculate gradients. The error between the model's prediction and the real label is propagated backward through each layer, starting with the output layer. Utilizing the chain rule, it accurately determines each parameter's contribution to the overall error, thereby acquiring the gradients necessary for weight updates. The loss function $\mathcal{L}$ quantifies the discrepancy between model predictions and actual targets, providing an optimizable metric that allows the network to evaluate the accuracy of its predictions [134]. During training, forward propagation computes predictions based on existing parameters and produces the loss. Backpropagation calculates parameter gradients from the loss, and optimization algorithms modify parameters appropriately. Collectively, they provide the fundamental process that allows NNs to autonomously learn and perpetually enhance their performance [135]. In backpropagation, root mean square is a commonly used loss function, which can be expressed as:

$$\mathcal{L} = \sum_j \frac{1}{2} (y_j - a_j^L)^2, \quad (3.29)$$

where y_j is the ground truth value δ_j^l and a_j^L is the output value of the j -th neuron in the output layer L . $\frac{\partial \mathcal{L}}{\partial w}$ and $\frac{\partial \mathcal{L}}{\partial b}$ are the target values that need to be computed for backpropagation. Initially, establish an error value of the j -th neuron in the l -th layer:

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}. \quad (3.30)$$

The error value of the j -th neuron in the last layer L can be obtained:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} F'(z_j^L). \quad (3.31)$$

(3.31) can be rewritten as:

$$\delta^L = \nabla_a \mathcal{L} \odot F'(z^L), \quad (3.32)$$

where $\odot$ is the Hadamard product. Further calculations on δ_j^l can be obtained:

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}. \quad (3.33)$$

In particular:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial(\sum_j w_{jk}^{l+1} a_j^l + b_k^{l+1})}{\partial z_j^l} = \frac{\partial(\sum_j w_{jk}^{l+1} F(z_j^l) + b_k^{l+1})}{\partial z_j^l} = w_{jk}^{l+1} F'(z_j^l). \quad (3.34)$$

Substituting (3.34) into (3.33) yields:

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{jk}^{l+1} F'(z_j^l). \quad (3.35)$$

(3.35) can be rewritten as matrix form:

$$\delta^l = ((\mathbf{W}^{l+1})^T \delta^{l+1} \odot F'(\mathbf{z}^l)). \quad (3.36)$$

From (3.36), it can be seen that the error value at any layer can be recursively obtained from the error at the next layer. Therefore, the error for the last layer δ^L can be calculated first, then traced back sequentially to $\delta^{L-1}, \delta^{L-2}, \dots$, until reaching the first layer. Furthermore, the partial derivatives of the loss function with respect to weights and bias can be calculated:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l} = \delta_j^l \frac{\partial(\sum_i w_{ij}^l a_i^{l-1} + b_j^l)}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}, \quad (3.37)$$

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \frac{\partial(\sum_i w_{ij}^l a_i^{l-1} + b_j^l)}{\partial b_j^l} = \delta_j^l. \quad (3.38)$$

The explicit algorithmic form can be summarized in the Algorithm 6.

Algorithm 6 Backpropagation

- 1: Training dataset
 - 2: *Feedforward*:
 - 3: $\mathbf{z}^l \leftarrow \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l, \mathbf{a}^l \leftarrow F(\mathbf{z}^l)$
 - 4: *Calculate* the error value of the last layer L :
 - 5: $\delta^L \leftarrow \nabla_a \mathcal{L} \odot F'(\mathbf{z}^L)$
 - 6: *Calculate* the error value of any layer l :
 - 7: $\delta^l \leftarrow ((\mathbf{W}^{l+1})^T \delta^{l+1} \odot F'(\mathbf{z}^l))$
 - 8: *Output* the gradient of the loss function:
 - 9: $\frac{\partial \mathcal{L}}{\partial w_{ij}^l} \leftarrow \delta_j^l a_i^{l-1}, \frac{\partial \mathcal{L}}{\partial b_j^l} \leftarrow \delta_j^l$
-

Activation function

In DL, activation functions are nonlinear operations applied to the outputs of each layer of neurons in a NN. Their primary goal lies in introducing nonlinearity to the model, enabling the NN to express and learn complex nonlinear relationships. Without activation functions, regardless of the number of layers, the entire network would essentially be a stack of linear transformations. It would ultimately remain equivalent to a simple linear model, incapable of handling complex tasks like image recognition or natural language understanding. The role of activation functions can be summarized in several aspects [136]:

- Activation functions enable the model to fit complex patterns in the real world, making NNs truly “deep”.
- By limiting the output range (e.g., sigmoid outputs values between 0 and 1), activation functions help stabilize the model and prevent numerical overflow.
- Certain activation functions can improve gradient propagation. For instance, ReLU (Rectified Linear Unit) maintains a constant gradient of 1 in the positive range, mitigating the vanishing gradient problem.
- Appropriately chosen activation functions can accelerate network convergence, enhancing training efficiency.

Fig. 3.5 summarizes the equations, graphs, and application scenarios of various commonly used activation functions. In this dissertation, ReLU and ELU (Exponential Linear Unit) are the two most commonly used activation functions. Within the positive interval, the derivative of ReLU remains constant at 1. Unlike Sigmoid or Tanh, its gradient does not become extremely small in high saturation regions, making it more conducive to backpropagation and resulting in faster training speeds. Additionally, directly outputting 0 for negative values causes many neurons to remain inactive, which increases the model’s sparsity. This helps reduce overfitting while enhancing representational capacity [137]. However, ReLU may cause certain neurons to never update their weights again. Compared to ReLU, ELU’s primary advantage lies in its handling of negative values. ELU outputs the same result as ReLU for positive values. For negative values, it uses an exponential function to gradually approach a negative constant, thereby avoiding ReLU’s tendency to become zero in the negative range. This brings is a reduction in the occurrence of “dead ReLUs”. Since the negative range still possesses a non-zero gradient, parameters can continue to update, preventing neurons from getting stuck in a state where they perpetually output

zero [138]. Nevertheless, the cost of ELU involves more computing overhead than ReLU since it requires the calculation of exponentials. Concurrently, ELU may provide large negative outputs in the negative range, affecting numerical stability.

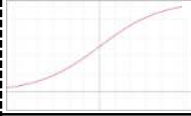
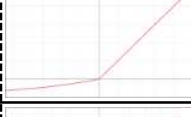
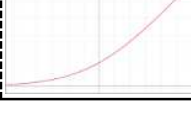
Name	Diagram	Equation	Application
Linear		$F(x) = x$	Linear Regression
Sigmoid		$F(x) = \frac{1}{1 + e^{-x}}$	Binary Classifier
Tanh		$F(x) = \frac{2}{1 + e^{-2x}} - 1$	RNN LSTM
ReLU		$F(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	ANN CNN
Leaky ReLU		$F(x) = \begin{cases} ax & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	GAN CNN
ELU		$F(x) = \begin{cases} a(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	DNN CNN
Softplus		$F(x) = \log_e(1 + e^x)$	VAE MLP

Figure 3.5: Summary table of commonly used loss functions with their typical applications.

3.2.2 Convolutional neural network

The origins of CNNs can be traced back to neuroscience research conducted between the 1960s and 1980s. In 1962, Hubel and Wiesel discovered during their study of the feline visual cortex that neurons in this region exhibit sensitivity to localized areas within images, possessing a “receptive field” structure. This discovery laid the biological foundation for the concept of convolution: vision is a process where complex perceptions are progressively constructed through a hierarchical mechanism of localized sensitivity. Inspired by this, Kuniyiko Fukushima proposed the neocognitron in 1982 [139]. This early model incorporated convolutional concepts and hierarchical structures, serving as the prototype for CNNs. In 1998, Yann LeCun and colleagues introduced LeNet-5, applying CNNs to handwritten digit recognition (MNIST) [140]. Their successful achievement of industrial-grade performance brought CNNs into widespread view for the first time as trainable deep structures.

The fundamental components of CNNs include three principal operations: convolutional layers, pooling layers, and fully connected layers. Convolutional layers execute convolution operations on input data utilizing sliding filters to extract features from localized regions; pooling layers downsample feature maps, diminishing dimensions and augmenting the model’s translation invariance; fully connected layers correlate high-dimensional features at the network’s conclusion to specific tasks such as classification or regression. With the progression of study, conventional pooling layers have progressively been supplanted by more adaptable convolutional or attention processes in several contemporary CNN designs. Nonetheless, convolution persists as a fundamental activity and maintains its essential significance.

Convolutional layer

The convolutional layer serves as the key element of CNNs, designed to autonomously extract local features from input data. The procedure relies on a fundamental assumption: important information in spatially structured data, such as images and sounds, is often found inside localized areas. Consequently, rather than necessitating that each neuron perceive the whole input, it concentrates only on small local regions. The convolutional layer sweeps a group of learnable convolutional kernels (or filters) over the input, executing dot-product accumulation on each localized area to produce feature maps. Subsequently, a nonlinear activation function is often added, allowing the model to represent

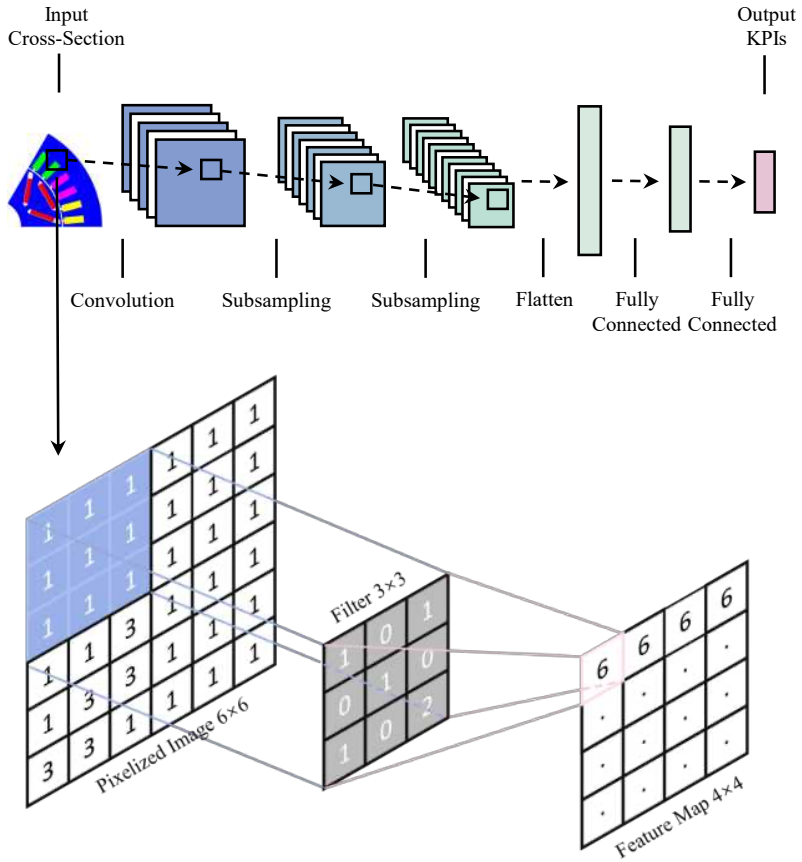


Figure 3.6: Visualization of a CNN architecture with different layers and computation process.

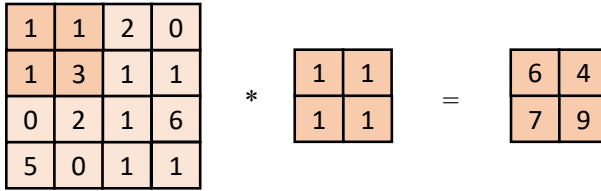
complex nonlinear relations. Each convolutional kernel is tasked with extracting a particular feature, such as edges, textures, or corners in images. The aggregation of several kernels enhances feature representation capabilities. Fig. 3.6 illustrates the convolutional computation process. The convolution operation could be mathematically articulated as [128]:

$$b(i, j) = (a * w)(i, j) = \sum_u \sum_v a(i + u, j + v)w(u, v), \quad (3.39)$$

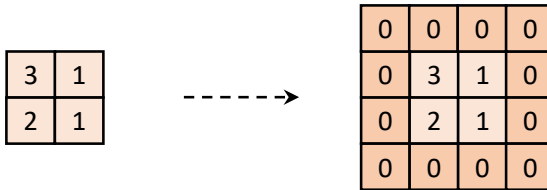
where $b(i, j)$ is the output feature map, $a(i + u, j + v)$ is the input feature map, and $w(u, v)$ is the filter. The example in the Fig. 3.6 can be calculated to obtain the value of the output pixel:

$$1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 = 6. \quad (3.40)$$

Additionally, stride and padding are as shown in Fig. 3.7. Stride refers to the distance the convolution kernel moves across the input feature map at each step. By default, stride equals 1, meaning the kernel slides pixel-by-pixel, resulting in a larger output feature map that retains more information. When stride equals 2, the kernel moves two pixels at a time, reducing the output feature map size and effectively downsampling the input. Padding means artificially adding extra



(a) Input feature map 4×4 , filter 2×2 , stride = 2, padding = 0, output feature map 2×2



(b) Input feature map 2×2 , padding = 1, output feature map 4×4

Figure 3.7: Demonstration of stride and padding.

pixels (typically filled with zeros) around the edges of the input. This is done to control the spatial dimensions of the output feature map after convolution or to better preserve boundary information.

Pooling layer

In CNNs, the pooling layer serves to downsample feature maps. Its purpose is to reduce spatial dimensions, lessen processing demands, and concurrently enhance the model's resilience to positional shifts. Pooling layers do not learn parameters; rather, they use set mathematical procedures to get representative statistics from local areas, thereby condensing the initially large feature maps into more compact representations. The pooling operations involves dividing the input feature map into smaller regions (e.g., 2×2 , 3×3) and then applying an aggregation function to each region. The two most common pooling methods are max pooling and average pooling. Max pooling selects the maximum value within a local region to represent its features. Average pooling calculates the average of all values within a region, resulting in smoother and more holistic features [141].

Fully connected layer

Several fully connected layers are generally placed in the network's backend, characterized by each neuron being interconnected with all outputs of the previous layer. The major role is to combine the high-dimensional information obtained from the previous convolutional and pooling layers, transferring them to the final decision space for classification, regression, or other tasks. Convolutional layers extract local and hierarchical characteristics, while fully linked layers use these features to provide the final conclusion. Many research in electric motors have investigated the use of CNNs for nonparametric topology optimization. This work mainly investigates design and optimization with parameter vectors instead of images. The following section will provide a comprehensive comparison of the two methodologies.

3.3 State-of-the-art: machine learning applications to electric machines

AI-based technologies have emerged as a new research trend to enhance the design optimization of electric motors efficiently and precisely. This section

discusses how ML is utilized in the realm of electromagnetic optimization for electric motors. Additionally, the core principles of various meta-models and methodologies, as well as their respective benefits and drawbacks, are discussed in depth. Ultimately, the exploration of future research challenges and possible avenues for enhancement in this area is undertaken [O1].

Conventional methods for optimizing machine design exhibit numerous shortcomings. For example, while the analytical method allows for quick calculations, it often depends on simplified assumptions, which restrict its accuracy and make it inappropriate for complex machine architectures. Moreover, while FEA can provide high accuracy in modeling, its computational cost is significant, rendering it impractical for large-scale use in optimization algorithms, particularly when iterations occur frequently. To address the challenges associated with traditional physics-based approaches, data-driven ML metamodeling has found extensive application in both academic and industrial sectors for optimizing motor design [6]. This method gathers data to allow metamodels to learn the fundamental relationships and patterns, resulting in high accuracy, computational efficiency, and suitability for complex nonlinear situations [14]. Fig. 3.8 illustrates a typical workflow for electromagnetic design optimization of electric machines using ML. It can be summarized in three steps:

1. *Data Acquisition:* The motor for optimization is initially chosen. Design of experiments is conducted on the design space, utilizing common approaches such as full factorial (FF) design and Latin hypercube sampling (LHS). FEA is subsequently utilized to annotate the data, namely the output parameters associated with each machine, which can also be referred to as KPIs, such as maximum torque and power. Finally, the collected data are divided into a training set, a validation set, and a test set to train metamodels.
2. *Model Training:* Various classical machine learning models or deep learning models are evaluated and selected based on different tasks, prediction targets, and data volume size. The models with the best prediction performance are employed in the final optimization algorithm.
3. *Optimization:* The optimization objective is first defined, and appropriate constraints are added. The trained metamodels are combined with the optimization algorithm to predict KPIs for each individual. The obtained final optimization solutions are verified using FEA to determine the optimization errors.

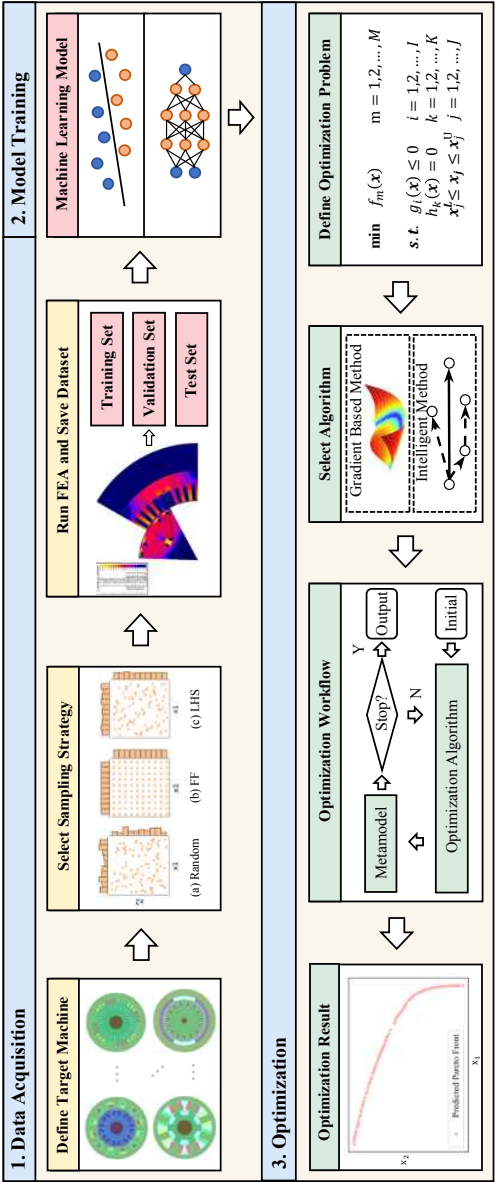


Figure 3.8: General design optimization workflow for electric machines using ML metamodels.

This section reviews and categorizes the application of ML in the electromagnetic optimization of electric machines from three perspectives: 1) Input object; 2) Machine learning paradigm; 3) Machine learning models.

3.3.1 Dataset representation

Datasets play a crucial role in AI, containing massive quantities of information and knowledge that are vital for training and improving the effectiveness of ML algorithms. In motor optimization, datasets are often derived from FEA, which can be divided into two primary categories: cross-sectional images and parameter vectors. The difference between them is illustrated in 3.9. The input based on the parameter vector only outlines the established features, which may encompass geometric or additional parameters. The advantages

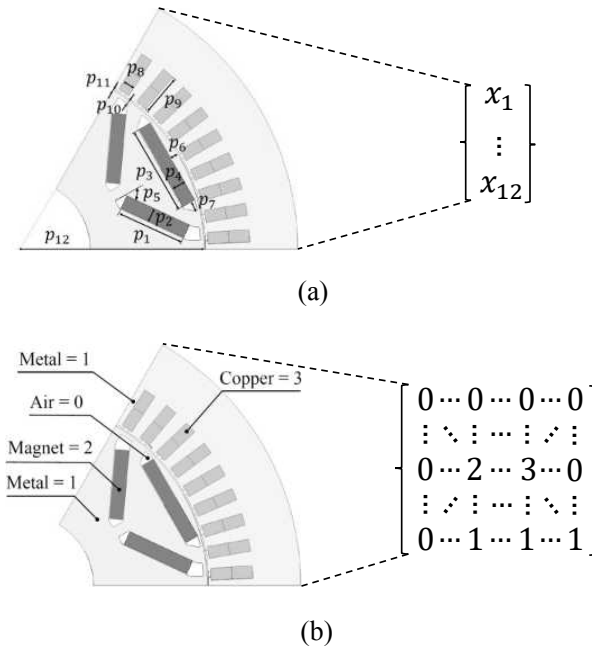


Figure 3.9: Difference between (a) parameter vector- and (b) image-based inputs.

include minimal input dimensions and quick model training. When it comes to reparametrization, gathering new datasets and training the model from scratch is essential. The data derived from images comprises numerous numbers of pixel points, with various values typically allocated to distinct materials to differentiate the various structures of electric machines. Models based on images offer enhanced reusability and flexibility; however, they face challenges with high input dimensions and prolonged training times [48].

In [O2], [17], [18], [19], [20], geometric vector inputs were utilized to train metamodels aimed at achieving precise predictions. In [33], [35], [36], [37], image-based methods were utilized for the topology optimization of electric machines. Other studies [48], [49] compared these two forms of inputs in terms of time efficiency and prediction accuracy. Methods based on vectors consistently provide enhanced prediction accuracy and lower time expense. Additionally, a combination of images and vectors was explored in [46], [47]. Motor cross sections and current vectors are combined as multiple inputs to the metamodels to improve learning effectiveness.

3.3.2 Machine learning paradigms

ML consists of four main learning approaches: supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning, as shown in Fig. 3.10. Applications of ML methods based on these four paradigms are summarized as follows.

Supervised learning

Supervised learning employs labeled datasets to train algorithms, allowing them to accurately classify data or predict outcomes [142]. In the field of electric motors, the most common training approach employed to predict motor KPIs is supervised learning. For example, in [20], a total of 5924 geometric vectors were gathered to forecast flux linkages and inductances. Metamodels were trained and compared, including support vector regression (SVR), ridge regression, and XGBoost. In [33], a dataset of 6000 images along with their respective KPIs (average torque and torque ripple) was created for the purpose of training NNs.

Additionally, supervised learning techniques can be divided into deterministic and probabilistic models according to their predictions. Deterministic models

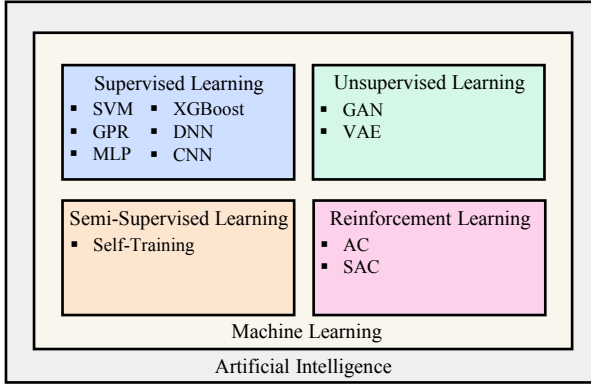


Figure 3.10: Structure of ML paradigms used for optimization of electric machines.

typically produce only point predictions, such as deterministic predictive values, as seen in ANNs. In contrast, probabilistic models offer additional insights into the confidence of the forecast alongside the point prediction. The study [O3] introduced two probabilistic metamodels: Gaussian process regression (GPR) and Bayesian neural networks (BNNs), aimed at improving the optimization process. The proposed algorithm incorporated predictive uncertainty information to exclude individuals with high uncertainty. The prediction error in relation to the true value tends to increase as forecast uncertainty grows. A comprehensive study of uncertainty quantification-based design optimization of electric machines will be presented in Section 5.

Semi-supervised learning

Semi-supervised learning merges a small amount of labeled data with a large amount of unlabeled data to train a model. It lies in the realm between supervised and unsupervised learning, aiming to improve learning while reducing dependency on expensive manual labeling [143]. Self-training is a well-established method in semi-supervised learning and has been widely applied. The fundamental concept is to enhance the training process by allowing the pre-trained model to produce “pseudo labels” for the unlabeled samples [144]. A self-training framework for electric machines was introduced in [O4] to enhance an

initial dataset, as shown in Fig. 3.11. The initial dataset with KPI labels by FEA was used to train a predictor and a classifier. LHS generated a substantial unlabeled dataset. Subsequently, the classifier decided the unlabeled input as either geometrically feasible or infeasible. The predictor subsequently pseudo-labeled the feasible data. In the end, these pseudo-labeled data were incorporated into the original dataset for expansion. The findings suggested that a more precise model could be developed with the enhanced dataset. Nonetheless, the method has certain limitations, notably a strong reliance on the pre-trained model. Additionally, as the data grows to a large size, the effectiveness of model training declines. Overabundant data with pseudo-labeling results in a significantly contaminated dataset, meaning the model training will produce considerable bias. This approach will be introduced and discussed in detail in Chapter 6.

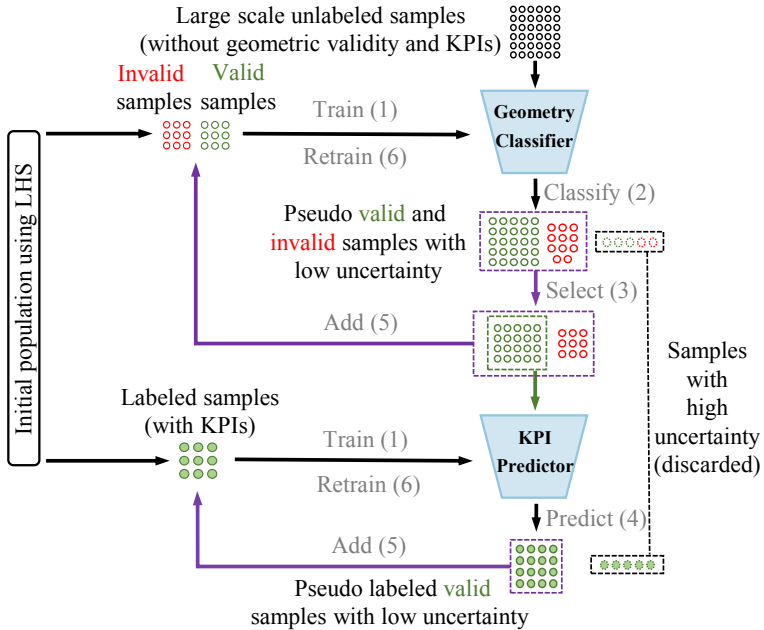


Figure 3.11: Proposed self-training framework based on semi-supervised learning.

Unsupervised learning

Unsupervised learning is a ML approach that allows algorithms to independently discover structures, patterns, or distributions in data without the need for specifically assigned labels. In the realm of motor optimization, generative adversarial networks (GANs) serve as a classic instance of unsupervised learning [145]. Figure 3.12 illustrates the overall algorithm of GANs. A GAN consists of a generator G and a discriminator D . The generator produces artificial images by inputting random noise, with the goal of tricking the discriminator into believing its generated samples are real. The discriminator aims to accurately identify whether the input is “real” or “fake”. After enduring persistent challenges, the generators ultimately produce samples that are remarkably realistic. The loss function for the generator can be represented as

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z} [\log D(G(z))], \quad (3.41)$$

where z is the random noise. It can be viewed that G aims for the output of D to approximate 1 as closely as possible on the fake samples it produces. The loss function of the discriminator can be described as

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] - \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))], \quad (3.42)$$

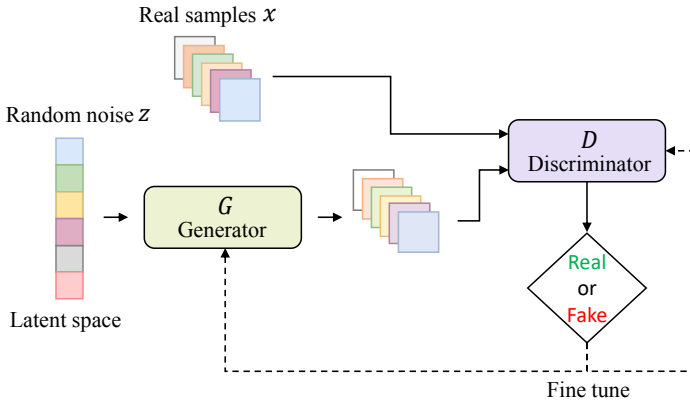


Figure 3.12: Basic workflow of GAN.

where x represents the real samples. It prompts D to assess real samples as true (1) and fake samples as false (0) to the greatest extent possible. In [42], GAN was utilized to accurately produce a wide range of rotor topologies for three distinct PMSMs. Various rotor images were generated by repeatedly creating new latent variables through the NSGA-II method [146]. The produced images were then input into a CNN for KPI prediction, aiming to optimize the maximum torque and reduce the volume of the PM.

Reinforcement learning

The adoption of RL [147] is primarily evident in intelligent design decisions and multi-objective optimization. Particularly in traditional optimization methods, such as FEA combined with genetic algorithms, which exhibit high computational costs and low search efficiency, RL facilitates a more efficient and intelligent motor design process through interaction with the design environment and continuous learning of optimization strategies. A basic architecture of the actor-critic (AC) algorithm in RL is shown in Fig. 3.13. An actor can be treated as a policy network that directs the agent to execute an action a in a certain state s . A NN $\pi(a|s; \theta)$ can be used to approximate the policy function $\pi(a|s)$. Similarly, a critic can be a value network $q(s,a; \mathbf{w})$, which does not control agent movement, only scores the agent. Another network is employed to approximate

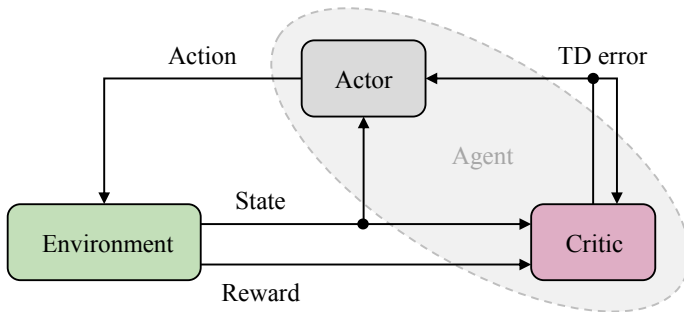


Figure 3.13: Architecture of the actor-critic RL algorithm.

the value function $Q_\pi(s, a)$, where θ and w represent trainable parameters of the networks. Then, the state-value function can be expressed as:

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s, a) \approx \sum_a \pi(a|s; \theta) \cdot q(s, a; w). \quad (3.43)$$

In [51], the AC method was implemented to optimize the rotor topology of a synchronous reluctance motor. To update the policy and the value networks, the temporal difference and the policy gradient were used, respectively. The results indicated that despite the extensive computational duration of the training, once the training was over, the agent needed much less computational cost compared to conventional optimization approaches.

3.3.3 Machine learning metamodels

In machine learning-based optimization methods, the choice of model has a direct impact on the precision of the optimization results. Metamodeling approaches currently can be classified into three main groups according to different scenarios: classic ML, DL, and RL. The following content provides an overview of these three types of models as they are applied in the field of motor design.

Classic machine learning models

Classic ML is a category of conventional methods, including linear regression, decision trees, and SVMs, etc. They provide robust interpretability, are appropriate for small to medium-sized datasets, exhibit good training efficiency, and are frequently employed in classification and regression tasks. In [17], [19], [21], SVMs were employed to forecast machine torque. In [18] and [20], SVMs were compared with extreme learning machine and XGBoost. Moreover, GPR was trained for multi-optimization of a switched reluctance machine [22]. In [O3], GPR was compared with BNNs, revealing a correlation between prediction accuracy and uncertainty.

Deep learning models

DL is mostly utilized to develop efficient surrogate models, predict motor performance, extract intricate nonlinear feature correlations, and accelerate design space exploration. It provides an alternative to expensive finite element simulations, allowing for quick assessment of how physical variables affect performance. Fig. 3.4 depicts a fundamental configuration of a NN and its

principle is thoroughly derived in Section 3.2.1. In [23], [24], [28], geometric parameters of the stator and rotor were used as inputs, and DNNs were trained to predict machines' torque characteristics. In the study [29], an automatic optimization framework considering multiphysics characteristics was proposed. An additional geometry checker was trained to filter out the infeasible designs during the optimization process. In another work [30], a hybrid approach using a DNN and a physics-based model was presented. The DNN was trained to predict intermediate parameters rapidly. Subsequently, the intermediate parameters combined with system parameters were post-processed by the physics-based model to obtain the final machine KPIs and performance curves.

CNNs yield superior outcomes when the inputs are images of machines' cross-sections. In [38], [39], [41], CNNs were adopted to estimate KPIs of PMSMs rapidly with adequate accuracy. Furthermore, generative modeling was also extensively used for the creation of additional data from existing image datasets. In [40], a GAN combined with a CNN was developed to generate and validate fake images. In [44], a GAN was trained to obtain different rotor shapes, and a CNN was used to predict the hysteresis loss and eddy current loss under various speeds and current conditions. In addition, another generative model known as variational autoencoder (VAEs) was employed to recover parameter vectors [26], [27]. The workflow is depicted in Fig. 3.14. Generated latent vectors were passed through a predictor for KPI forecasting. The optimization solutions were ultimately transformed into actual physical parameters by the decoder. Another work [43] focused on topology optimization with image inputs based on VAEs and NNs.

Additionally, physics-informed neural networks (PINNs) is an approach that embeds prior physics knowledge (e.g., electromagnetic field equations, heat conduction equations) directly into the neural network training process. In the study [32], a novel PINN was presented to predict the electromagnetic responses of a PMSM, such as the magnetic flux density and the electric field.

Reinforcement learning models

RL models are applied in motor design optimization to automatically investigate optimal design solutions. The agent employs motor structural features as actions, receives rewards based on performance metrics (such as efficiency, maximum torque, etc.), and continuously improves the design through iterative interactions with the environment. Unlike traditional optimization methods, RL can more

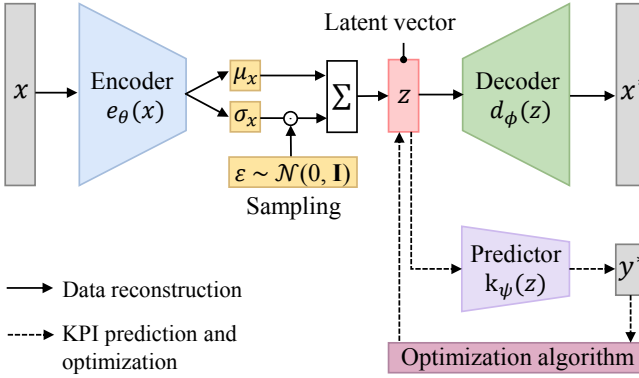


Figure 3.14: Workflow of VAE-based optimization.

effectively tackle high-dimensional, nonlinear, and multi-objective problems. An automatic design method that integrates RL and evolutionary optimization was proposed in [50]. A tree search process was created to explore the machine shape variables. Another study [51] applied the AC algorithm for pixel-wise optimization of the rotor topology, demonstrating greater efficiency compared to the genetic algorithm. Additionally, a refined soft actor-critic (SAC) method was utilized to boost torque while simultaneously minimizing torque ripple [52]. Unlike AC, SAC incorporates an entropy regularization component to enhance the exploration capacity of the algorithm, hence facilitating policy learning and diminishing the probability of the policy converging to inefficient local optima [148].

In summary, Table 3.1 offers a detailed overview of the implementation of various metamodels alongside different motors and the amount of data used for training. Various metamodels in different task scenarios are suitable for distinct electric machines, each with specific advantages and limitations. Hence, the applications are categorized according to the form of input data, the methodology of model training, and the type of model employed. Generally, CNNs exhibit superior feature extraction capabilities when the inputs are images. DNNs are more appropriate when the inputs consist of parameter vectors. Generative models like VAEs or GANs are optimal selections when it is necessary to produce additional

data. Moreover, RL models could serve as effective alternatives to conventional optimization methods. Nevertheless, the training process is very time-intensive.

3.4 Contribution

This thesis mainly explores design and optimization methods for electric motors based on ML. In contrast to current research methodologies, several novel techniques have been proposed for achieving more efficient and accurate optimization outcomes.

- Initially, a complete optimization procedure using deterministic metamodels was investigated. To address the challenge of metamodel's reparameterization, a new transfer learning-based approach was proposed and higher prediction accuracy was achieved. However, due to inherent model limitations, relatively large datasets remained necessary to achieve improved prediction capability. The final optimization results showed considerable deviations from FEA-verified outputs owing to out-of-distribution problems.
- To enable metamodels to provide more information for users to make better decisions, a novel machine design optimization method based on probabilistic metamodels was proposed for the first time. The relationship between the uncertainty of model predictions and actual FEA verification has been proven. Specifically, higher prediction uncertainty tends to indicate greater deviation between prediction and FEA results.
- Generating datasets inevitably involves high computing expenses from FEA. A semi-supervised self-training framework has been presented to swiftly enhance datasets and mitigate this problem. Nonetheless, it has limitations: the addition of a large amount of data with pseudo labels might contaminate the original dataset, resulting in diminished model performance.
- To accelerate the overall optimization process, a novel active learning-based motor optimization framework has been developed, assuring models obtain high prediction accuracy with a limited amount of data. The results demonstrate that even with just hundreds of samples, the final optimization solutions achieve great accuracy while substantially decreasing computing time.

Table 3.1: Studies on data-driven models for electric machines.

Model category	Reference	Metamodel	Motor	Input	KPI	Data volume
Classic machine learning	[17]	SVM	SRM	Parameter vector (1 x 3)	Phase current torque	1196
	[18]	ELM, SVM	PMSLM	Parameter vector (1 x 4)	THD, average thrust thrust ripple	625
	[19]	SVM	PMSM	Parameter vector (1 x 4)	torque, torque ripple current angle	200
	[20]	SVM, XGBoost	PMSM	Parameter vector (1 x 12)	PM flux linkage dq-axis inductances	5924
	[22]	GPR	SRM	Parameter vector (1 x 5)	torque, torque ripple efficiency	300
	[21]	SVM	PMSM	Parameter vector (1 x 5)	THD, efficiency average torque	200
Deep learning	[23]	DNN	FSM	Parameter vector (1 x 8)	torque, torque ripple cogging torque	256
	[24]	MLP	PMSM	Parameter vector (1 x 5)	torque, torque ripple factor	1196
	[25]	PINN	-	Parameter vector (1 x 2)	Electromagnetic analysis	-
	[26]	VAE, CNN, DNN	PMSM	Parameter vector (1 x 32)	max. torque, max. power max. torque ripple, cost	28278
	[27]	VAE, CNN, DNN	PMSM, ASM	Parameter vector (1 x 52)	max. torque, max. power cost	101919
	[28]	DNN	PMSM	Parameter vector (1 x 8)	air-gap radial flux density	10000
	[29]	MLP, SVM, GRNN	PMSM	Parameter vector (1 x 11)	torque, torque ripple cost, loss	800
	[30]	DNN	PMSM	Parameter vector (1 x 35)	iron loss, torque flux linkage	44877
	[31]	MLP	PMSM	Parameter vector (1 x 12)	THD, cogging torque back EMF, PM volume	1000
	[32]	PINN	PMSM	Parameter vector (1 x 8)	electromagnetic field flux density	24000000
	[33]	CNN	PMSM	Image (-)	torque torque ripple	6000
	[34]	CNN	PMSM	Image (400 x 400 x 4)	magnetic field distribution	45000
	[35]	CNN	PMSM	Image (-)	torque torque ripple	168000
	[36]	CNN	PMSM	Image (-)	torque torque ripple	4848
	[37]	CNN	SyncRM	Image (32 x 48 x 1)	torque	-
	[38]	CNN	PMSM	Image (56 x 56 x 1)	torque curve	4000
	[39]	CNN	PMSM	Image (224 x 224 x 3)	torque torque ripple	6400
	[40]	CNN, GAN	PMSM	Image (240 x 180 x 1)	Back EMF	56400
	[41]	CNN	PMSM	Image (-)	Flux linkage torque	12900
	[42]	CNN, GAN	PMSM	Image (256 x 256 x 3)	PM flux linkage dq-axis inductances	165000
	[43]	VAE, DNN	PMSM	Image (224 x 224 x 1)	Flux linkage, flux density PM area	100
	[44]	CNN, GAN, MLP	PMSM	Image (256 x 256 x 3)	Hysteresis loss eddy current loss	85184
	[45]	AE	SyncRM	Image (100 x 100 x 1)	torque torque ripple	10000
	[46]	CNN, RNN	PMSM	Parameter vector + image	dq-axis flux linkages	2900
	[47]	CNN, DNN	PMSM	Parameter vector + image	torque, torque ripple-order components	38604
	[48]	CNN, DNN	PMSM	Parameter vector vs. image	motor performance material costs	68099
	[49]	CNN, DNN	SynRM	Parameter vector vs. image	Torque related parameters efficiency	961
Reinforcement learning	[50]	ST	LIM	Parameter vector (1 x 12)	Line voltage, frequency rated rotor speed	-
	[51]	DNN-based AC	SynRM	Image (12 x 12 x 4)	Average torque	-
	[52]	DNN-based SAC	PMSM	Parameter vector (1 x 4)	torque torque ripple	625

Chapter 4

Deterministic Metamodeling-Based Optimization

This chapter presents the optimization of PMSMs with deterministic NNs. Moreover, transfer learning is used to tackle the challenge of reparameterization in parameter-based metamodels. Section 4.1 provides the definition of deterministic metamodels. In addition, the ML-based design optimization framework employed in this thesis is demonstrated. Section 4.2 describes the dataset acquisition process, especially showing the features of an automatic simulation toolchain. The generated datasets are also displayed. Section 4.3 alleviates the reparameterization issue of parameter-based metamodels. The suggested NN architecture using transfer learning is then introduced. Training results are analyzed at the end of this section. The following Section 4.4 deals with motor design optimization, detailing the optimization approach and its outcomes. Finally, the limitations of deterministic metamodels are discussed to guide subsequent research in Chapter 5 and Chapter 6.

4.1 Introduction

The increasing demands for electrification, intelligentization, and high-performance control made current motor systems much more complicated in structure, control accuracy, and operating environment variety compared to older

applications. Conventional motor design and optimization techniques mostly depend on physical models, empirical equations, and FEA. Although such approaches are theoretically robust and highly interpretable, they often need several modeling assumptions and fail to comprehensively and effectively characterize complex nonlinear phenomena, including material characteristics, temperature effects, magnetic saturation, and eddy current losses. Moreover, FEA entails high processing expenses, rendering it unsuitable for the rapid iterative optimization and cross-scenario flexibility required in actual engineering. These methods rely significantly on specialist knowledge and involve extensive adjustment of parameter procedures, often facing limitations when tackling more intricate multi-objective optimization challenges (e.g., efficiency, torque, power, and costs). Furthermore, in the context of extensive design spaces, the extended duration of FEA becomes especially evident, making the numerical optimization of electric motors very intricate. As a result, the optimization of motor design with data-driven metamodels has attracted heightened interest in recent years [13]. As shown in Fig. 3.9, training data for data-driven models can typically be categorized into parameter-based and image-based datasets. This dissertation utilizes parameter-based datasets for experimentation to achieve reduced data dimensionality and model complexity. In addition, due to the diversity of motor types, PMSMs for electric vehicles are the primary focus of this study. The proposed methods can be extended to other motor types; however, the variety and structure of metamodels will differ depending on the specific motor, necessitating independent model selection and optimization. Fig. 4.1 illustrates many common PMSM rotor topologies. Design optimization depends on one of these motor templates. Input parameters may include PM dimensions, skew angle, stator slot specifications, among others.

4.1.1 Deterministic metamodel

A deterministic metamodel is a type of surrogate model whose behavior and output are entirely determined by the model itself, given specific inputs and initial conditions. Such metamodels contain no random factors or probabilistic variables. Consequently, identical inputs will yield identical outputs regardless of how many times the model is run. Deterministic metamodels typically describe systems through explicit mathematical relationships, such as algebraic equations, differential equations, or deterministic mapping functions, which characterize the causal relationship between input variables and output variables [149]. The mathematical representation can be articulated as follows: Set the input space as

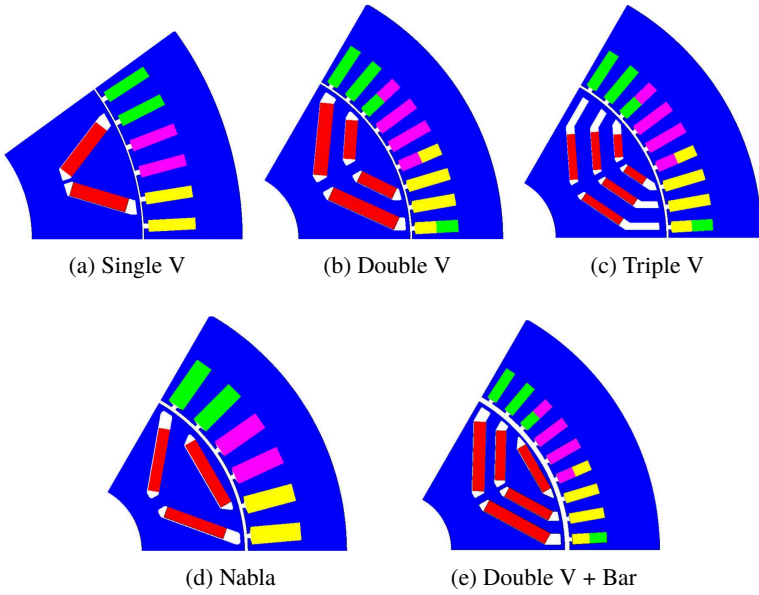


Figure 4.1: Different rotor topologies of PMSMs.

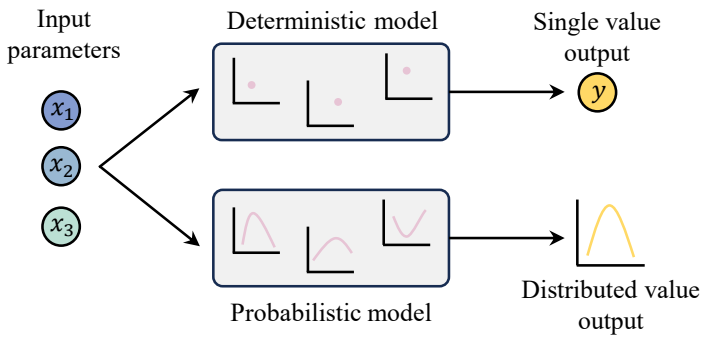


Figure 4.2: Comparison between deterministic and probabilistic metamodels.

$\mathcal{X} \subseteq \mathbb{R}^n$ and the output space as $\mathcal{Y} \subseteq \mathbb{R}^m$. A deterministic model is characterized as a deterministic function that establishes a mapping:

$$f : \mathcal{X} \rightarrow \mathcal{Y}, \quad (4.1)$$

enables the model for any given input $\mathbf{x} \in \mathcal{X}$ to output

$$\mathbf{y} = f(\mathbf{x}), \quad (4.2)$$

which is deterministic and unique, independent of the number of runs.

Therefore, a core characteristic of deterministic metamodels is the uniqueness and reproducibility of their outcomes. These models contain no random perturbations or noise terms. This makes the model's behavior easy to analyze and verify. Fig. 4.2 illustrates a comparison between deterministic and probabilistic metamodels. The probabilistic metamodel and its application in motor design optimization will be introduced in the next chapter.

4.1.2 ML-based design optimization framework

The optimization framework based on data-driven metamodels proposed in this dissertation can be divided into three parts: dataset acquisition, model training and tuning, optimization and verification. Fig. 4.3 summarizes the entire process. During the data acquisition phase, an in-house automated simulation toolchain is applied, which will be introduced in Section 4.2. Initially, a target machine template is selected, corresponding to a specific topology. Input design parameters and output KPIs are then defined. An appropriate sampling method is chosen to generate the data space, and a geometric feasibility analysis is performed for each candidate data point. Finally, the simulation toolchain progressively simulates and post-processes each candidate data point. The final simulation results are organized into a model-readable training data format. In the phase of model training, there's the option to select from classic ML metamodels like GPR and SVMs, or choose deep learning metamodels. The parameters within the model are optimized to attain enhanced prediction performance. The optimized metamodel undergoes evaluation using the test set. Finally, the metamodel combined with optimization algorithms serves as a prediction model for the optimization objectives. The final optimization results are compared with FEA verification.

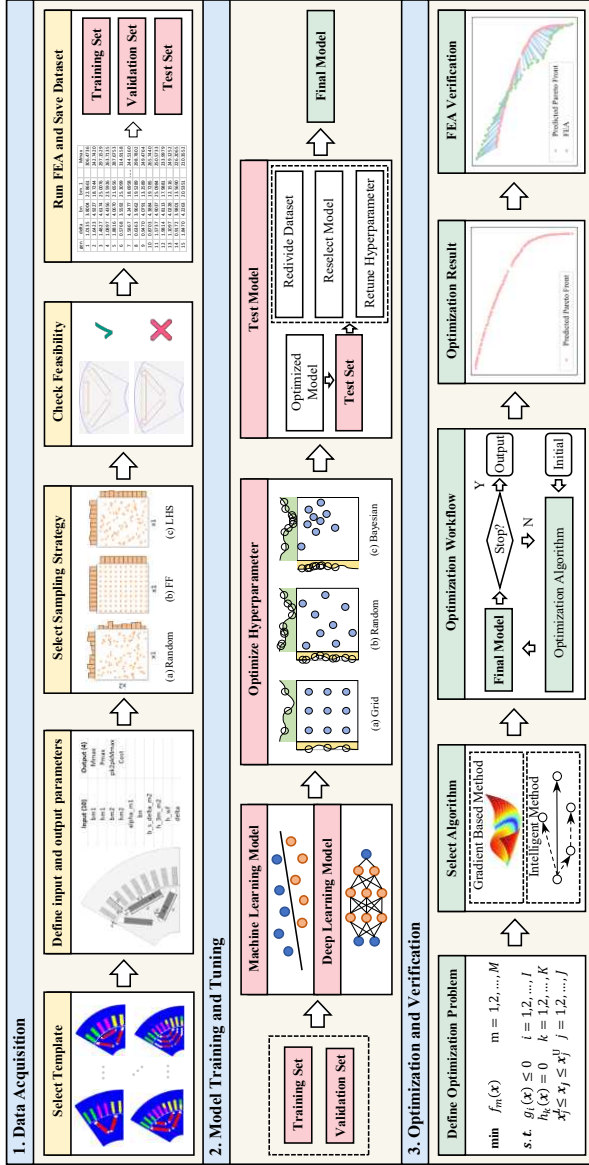


Figure 4.3: Proposed data-driven design optimization workflow for electric motors.

4.2 Dataset acquisition

Data serves as both the knowledge source and performance ceiling for ML. Its quality, scale, and distribution directly determine a model's learning capability, generalization performance, and reliability. Without high-quality data, even the most complex models struggle to produce trustworthy results [150]. In the realm of motor design optimization, data is generally acquired not through experimental collection but through FEA simulation. Consequently, the results obtained from the simulation are regarded as ground truth. To reduce the need for manual intervention throughout the simulation process, the use of automated simulation tools is crucial. Section 4.2.1 presents the essential concepts of the toolchain. Section 4.2.2 provides a comprehensive overview and thorough analysis of the data utilized for the training of the model.

4.2.1 FEA simulation toolchain

From an engineering practice standpoint, motor simulation toolchains have the potential to greatly lower research and development expenses and timelines. Conventional approaches that depend on iterative prototyping and evaluation frequently consume significant time and resources. Conversely, simulation toolchains facilitate rapid assessment of various structural designs, parameter combinations, and control strategies in the initial design stage. This facilitates the prompt recognition and enhancement of possible challenges, minimizing the possibility of revisions further along in the procedure. At the same time, the results from simulations offer measurable proof for aspects such as structural design, electromagnetic design, thermal management, and the adjustment of control parameters, thereby rendering design choices more systematic and verifiable.

In the realm of data-driven motor design optimization, simulation toolchains serve an essential role. They offer reliable, manageable, and expandable data sources for methods driven by data. By utilizing parametric modeling and batch simulation, a substantial array of design samples and their corresponding performance metric data can be automatically produced, taking into account diverse structural dimensions, material parameters, operating conditions, and control strategies. This method tackles the significant expenses, prolonged timelines, and restricted scope linked to obtaining real experimental data. The data produced through simulation serves as the basis for training and validating ML and intelligent optimization algorithms. This allows for the

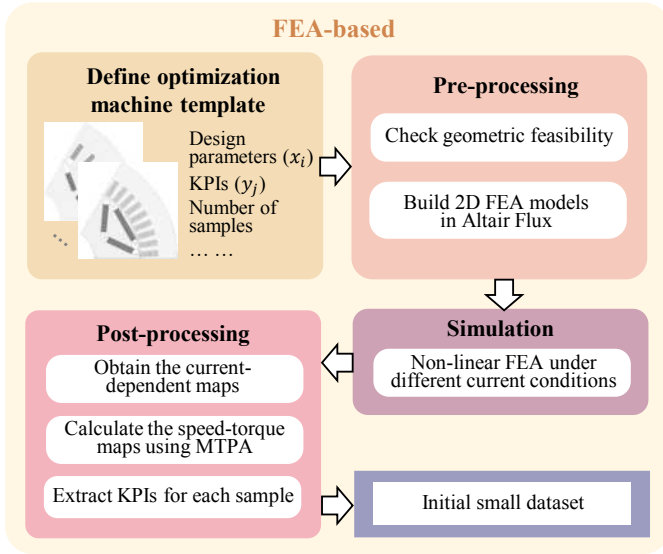


Figure 4.4: Flowchart of the in-house FEA-based simulation toolchain.

implementation of data-driven methods to conduct modeling and forecasting, effectively encapsulating the intricate nonlinear behaviors of electric motors.

Fig. 4.4 shows the workflow of the used FEA-based toolchain. After selecting the motor template, the target design parameters, output KPIs, and required data volume must be defined. During the preprocessing stage, each data point generated through the design of experiments (DoE) undergoes geometric checking. Models with conflicting design structures are eliminated. Each feasible model is generated and automatically configured within the commercial software Altair Flux, including adding materials, setting boundary conditions, and configuring solvers. FEA is then performed on each design for 36 current conditions consisting of currents I_d (from -500 to -0.1 A) and I_q (from 0 to 500 A). In the post-processing, an interpolation to interpolate the current-dependent maps will be adopted. Subsequently, the speed-torque maps are generated using maximum torque per ampere optimization to derive the final KPIs for each machine de-

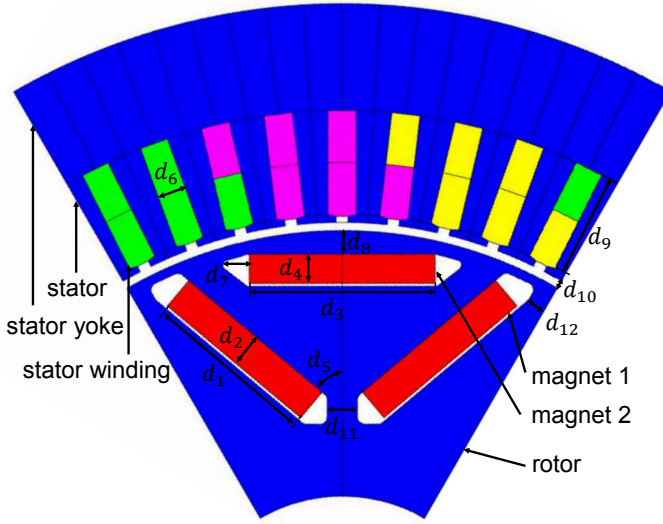


Figure 4.5: Reference PMSM geometry and design parameters.

sign, considering the maximum voltage constraint. More information about the toolchain is summarized in Section 8.2.

4.2.2 Dataset introduction

Three main datasets that have slightly different input design parameters and the same outputs are generated. The sampling procedure is carried out using Latin hypercube sampling [151]. The reference machine is shown in Fig. 4.5. Dataset I contains six design parameters ($d_1, d_2, d_3, d_4, d_5, d_6$) and a total of 968 samples. Dataset II is formed by removing one parameter from dataset I and adding three new design parameters. As a result, dataset II includes eight design parameters ($d_1, d_2, d_3, d_4, d_5, d_8, d_{11}, d_{12}$) and 2000 samples. Dataset III has 2951 samples and consists of ten design parameters ($d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}$). Table 4.1 summarizes information on all relevant parameters. The upper and lower boundaries for each design parameter are determined empirically.

Table 4.1: Details of reference machine parameters.

Symbol	Design parameter	Min.	Max.	Unit
d_1	Magnet 1 width	10.0	28.0	mm
d_2	Magnet 1 height	2.0	8.0	mm
d_3	Magnet 2 width	15.0	43.0	mm
d_4	Magnet 2 height	2.0	8.0	mm
d_5	Magnet inclination angle	15.0	60.0	deg
d_6	Slot width	3.5	5.0	mm
d_7	Flux barrier width	1.0	6.0	mm
d_8	Width between magnet 2 and air gap	0.5	4.0	mm
d_9	Slot depth	13.0	18.0	mm
d_{10}	Air gap	0.5	2.0	mm
d_{11}	Distance between magnets	1.0	4.0	mm
d_{12}	Distance between poles	4.0	6.5	mm
Constant parameter		Value		
c_1	Number of poles	6	-	
c_2	Number of stator slots	54	-	
c_3	Maximum rotational speed	16800.0	rpm	
c_4	Magnet temperature	75.0	°C	
Indicator				
o_1	Maximum torque			N·m
o_2	Maximum power			kW
o_3	Torque ripple at maximum torque			N·m
o_4	Material cost			Euro

In the analysis of sampling results from DoE, the significance of the Pearson correlation coefficient [152] lies in quantitatively describing the linear relationship between factors and responses, as well as among factors themselves. By calculating the Pearson coefficient between an individual factor and the response, one can quickly determine the direction and strength of the factor's linear influence on the response. A larger absolute value of the coefficient indicates a more pronounced linear main effect of that factor on the response within the current experimental space; a coefficient close to zero suggests a weak or non-existent linear influence of the factor on the response. On the other hand, the quality of the DoE sampling itself can be assessed by analyzing the Pearson correlation coefficients between

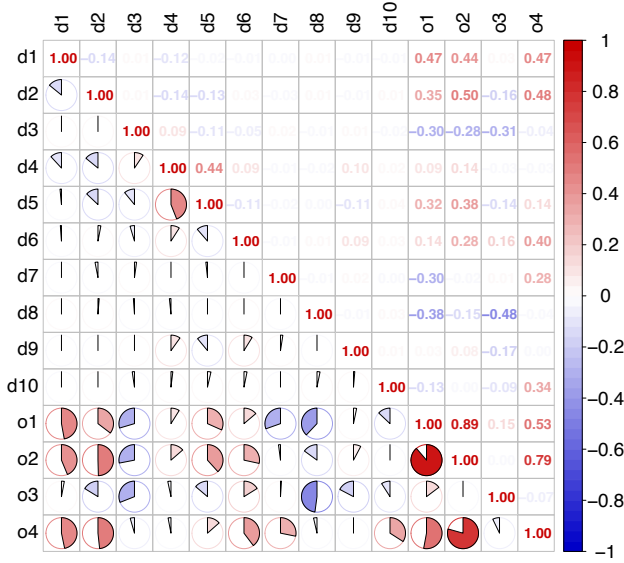


Figure 4.6: Correlation between design parameters and KPIs for dataset III.

factors. In an ideal experimental design, factors should be as independent as possible. If large Pearson correlation coefficients appear between factors, it often indicates collinearity issues within the sampling space. This may lead to unstable regression models or biased parameter estimates. Therefore, the Pearson correlation coefficient is applied to the three collected datasets. The following discussion focuses solely on Dataset III, while results for the other datasets are presented in Section 8.3.1. The Pearson correlation coefficients among all parameters can be calculated by:

$$r = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^N (y_i - \bar{y})^2}} \quad (4.3)$$

where x_i and y_i denote different values of variables in a sample. $\bar{x}$ and $\bar{y}$ represent sample means. The results are visualized in Fig. 4.6. The correlation between the design input parameters is generally low, except for the width of

magnet 2 and the V-magnet angle. This is due to restrictions on their feasible domains. The design parameters exhibited a strong correlation with KPIs, suggesting a good sampling process.

In order to obtain pre-trained NN, separate experimental designs are performed for each design parameter individually. A group of small subdatasets is then generated, each containing around 50 samples. Fig. 4.7 demonstrates rotor images in the subdataset for the design parameters d_1 , d_3 , d_4 , and d_5 . Why subdatasets are needed will be explained in the next section, which covers transfer learning technique.

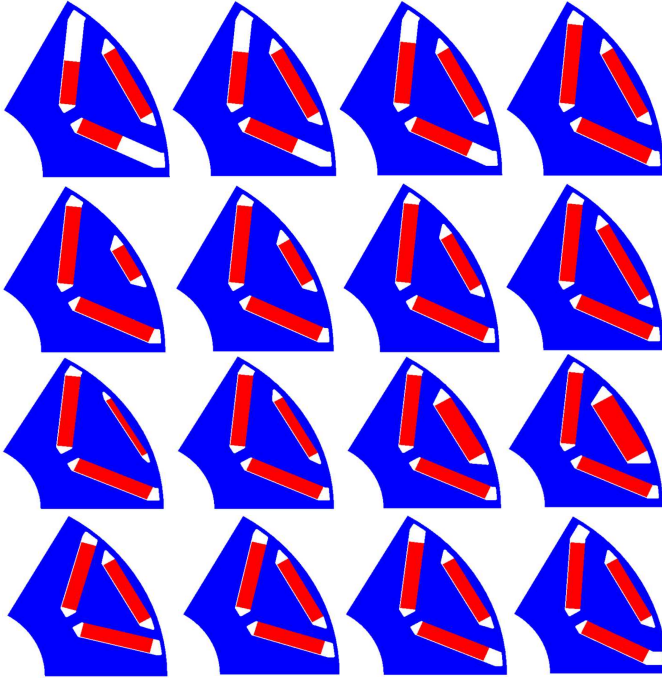


Figure 4.7: Example rotor images of subdatasets.

4.3 Methodology and model training procedure

This section presents an in-depth optimization process for motor design grounded in a deterministic framework. The framework for model training and the chosen architecture of the neural network are outlined. Finally, the optimization algorithm has been incorporated with the trained model. The outcomes of the optimization process and the limitations of this method will be discussed.

4.3.1 Motivation

So far, the inputs for NN-based surrogate models can be categorized into three groups: parameter-based (1D vector) [48], [26], [43], image-based (2D or 3D matrix) [35], [42], and mixed input, which includes both vectors and images [46]. In real-world scenarios, parameter-based datasets are more accessible to obtain without additional storage of images. However, when the reference motor is reparameterized, the NN that was previously trained becomes incompatible for further usage due to alterations in the feature composition of parameter vectors. Retraining new NN from scratch is demanded for every alteration, thereby leading to the discarding of previously trained parameters.

In the following sections, parameter-based modular neural networks (MNNs) are proposed through transfer learning (TL) to get over the issue of the networks requiring to be retrained from scratch after reparameterization [O2]. TL is an important technique in the field of ML that facilitates the training of NN by transferring previously learned knowledge to a new task [153]. One of the most common ways to reuse knowledge is through the transfer of NN. This method makes it possible to save part of the old networks' weights and use them to help the new NN update its weights [154]. In the field of design optimization of electric motors, TL-based CNN has been employed in [36]. An advantage of this proposed approach was that the CNN was effectively trained with a limited amount of input data using TL. Nevertheless, it relied on images but failed to address the issue of reparameterization.

4.3.2 Modular neural network and proposed workflow

Assume a task $\mathcal{T}_t$ in the target domain $\mathcal{D}_t$ is given, and an assistance can be obtained from a source task $\mathcal{T}_s$ coming from the source domain $\mathcal{D}_s$. By transferring latent knowledge from $\mathcal{T}_s$ to the target task $\mathcal{T}_t$, TL intends to boost the performance of the prediction function $\mathcal{F}_t$, where $\mathcal{F}_t$ is a representation of a deep NN

[153]. A workflow example for the proposed method is depicted in Fig. 4.8. Assuming an initial learning task based on three input design parameters (d_1 , d_2 , d_3) and four output performance parameters (o_1 , o_2 , o_3 , o_4) is given. Subsequently, the input space is expanded, and the whole process is described as follows:

- *Stage I*: A separate training session is performed for each of the input design parameters. Every individual multi-layer NN is called a base module, which consists of m dense layers with n neurons in each layer.
- *Stage II*: All the base modules are subsequently aggregated to the target NN by transferring the respective $m - 1$ dense layers, namely the layers with colors. A multi-branched MNN is applied, with each branch representing an independent input parameter. The quantity of input design parameters directly matches the quantity of branches in the target MNN.
- *Stage III*: The input space is altered by eliminating the design parameter d_3 and introducing the new design parameters d_4 and d_5 . As the input space expands, the NN's architecture goes through an adjustment whereby the layers associated with the branches of d_1 and d_2 stay unchanged while the branch linked to d_3 is removed. Another two basic modules (pink and purple) with d_4 and d_5 as inputs are trained independently to provide pre-trained models resembling Stage I. The required layers are then transferred to various branches. Only the first and last dense layers (white) necessitate retraining from scratch.

A classic MNN consists of three parts: the task-dividing layer, sub-network layer, and output-combining layer [155]. The idea of modularity is derived from biological inspiration. Within a biological brain, distinct information is typically processed in designated regions to carry out certain activities [156]. Inspired by the above approaches, every input design parameter can be regarded as a distinct sub-task that corresponds to a branch in the proposed MNN. Through this method, the target MNN exhibited the capability to adapt in response to variations in the input parameters. Meanwhile, the knowledge learned from the previous task can be transferred to the new task.

4.3.3 Training of neural networks

General setting

In ML, datasets are often standardized before training. This process enhances training stability, accelerates model convergence, and improves generalization

performance. Since different features typically possess varying dimensions and numerical ranges, feeding raw data directly into the model allows features with larger numerical scales to dominate gradient calculations. This masks the influence of other features, leading to suboptimal parameter updates. Standardizing

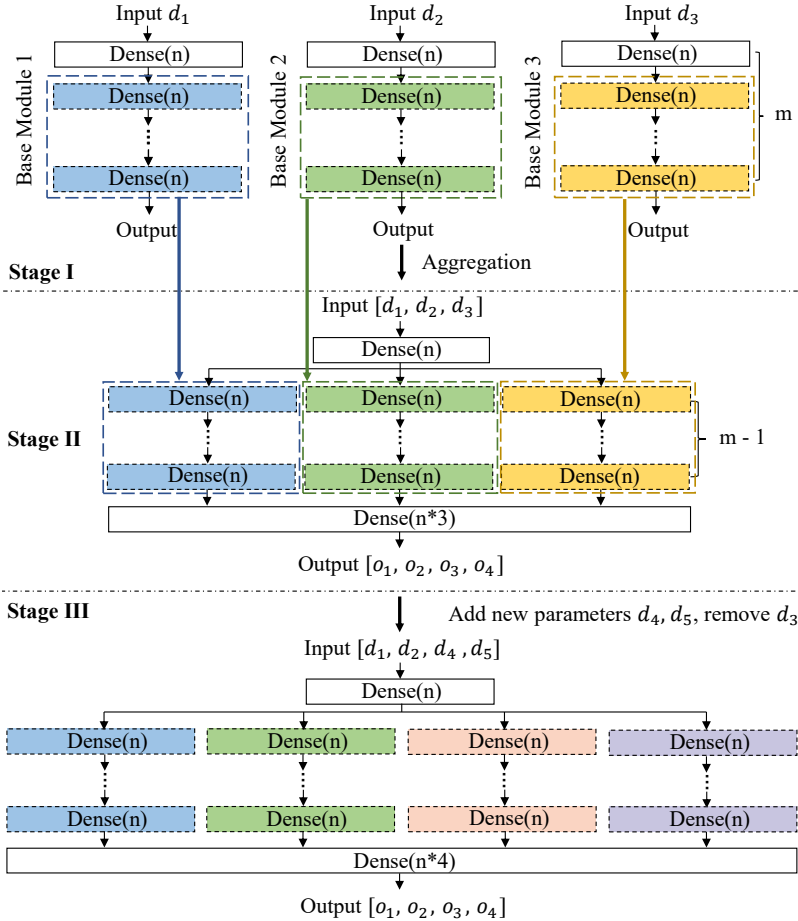


Figure 4.8: Proposed workflow for aggregating pre-trained NNs.

data maps features to similar numerical scales, enabling the model to utilize feature information more equitably during training. Data normalization also significantly improves numerical conditions in the optimization process, making the loss function smoother in parameter space. This reduces risks of oscillation, gradient explosion, or gradient vanishing during gradient descent, allowing the optimizer to find suitable update steps more readily and converge faster [157]. Therefore, all samples are standardized by their mean and variance using z-score normalization before being fed to NN:

$$z_i = \frac{x_i - \mu}{\sigma}, \quad (4.4)$$

where z_i reflects the standardized result, x_i represents the original data, μ is the mean value and σ is the standard deviation.

All experiments, including model comparison, are conducted using 5-fold cross-validation [158], with loss calculated through the mean squared error (MSE) between predictions and ground truth. The complete dataset is split into five equal folders. Four folders for training are used, and one fold for testing purposes is reserved. Training is subsequently carried out on five occasions, as shown in Fig. 4.9. It is important to point out that the final model is examined by choosing the fold that exhibits the lowest loss during cross-validation for the outcomes. To ensure that comparisons among different models relied on identical dataset divisions and to enhance the reproducibility of the results, random seeds are established. In addition, the early stopping method [159] is implemented during

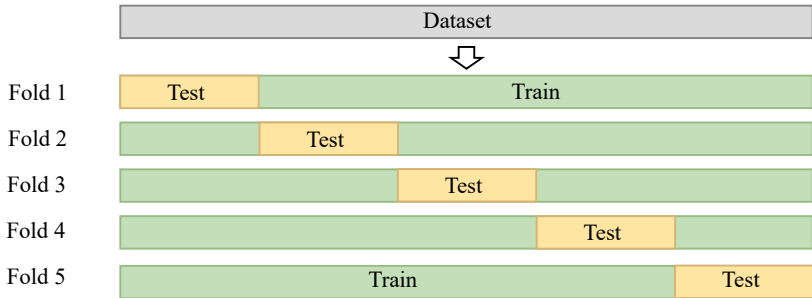


Figure 4.9: Illustration of five-fold cross validation.

the training process to avoid overfitting.

Evaluation metrics

After training a neural network, it is essential to assess the results. Evaluation metrics act as necessary tools for objectively and quantitatively measuring model performance on data that is separate from the training process. Throughout the training process, the model simply optimizes its parameters by reducing the loss function, which mainly aids in computational optimization and gradient flow—it does not inherently indicate the model's true performance in practical applications or commercial contexts. Consequently, depending exclusively on training loss metrics does not provide a clear indication of whether the model has truly acquired valuable patterns or exhibits robust generalization abilities. Introducing evaluation metrics allows for a more intuitive and interpretable assessment of model performance, considering aspects like accuracy, stability, and robustness. They help detect challenges such as overfitting or underfitting, while also offering a consistent benchmark for evaluating various models or parameter settings. In this dissertation, three different evaluation metrics are employed: the mean absolute error (MAE), symmetric mean absolute percentage (SMAPE), and coefficient of determination (R^2). The MAE enables the measurement of the absolute error between the prediction value y_i^* and the true value y_i :

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - y_i^*|. \quad (4.5)$$

The SMAPE aims to address the problem of the mean absolute percentage error producing overly large values when the real value is quite small:

$$\text{SMAPE} = \frac{100\%}{N} \sum_{i=1}^N \frac{|y_i - y_i^*|}{(|y_i| + |y_i^*|)/2}. \quad (4.6)$$

The R^2 measures how well the prediction model fits the data. It can be calculated as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - y_i^*)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}, \quad (4.7)$$

where N represents the total number of test samples. y_i^* describes the prediction value. $\bar{y}$ is the average true value of test data.

Table 4.2: Hyperparameter details of MNNs.

Hyperparameter	Range	Optimized value
Batch size	[2, 32]	8
Number of neurons per hidden layer	[16, 256]	128
Number of hidden layers per branch	[1, 6]	3
Learning rate (transferred layers)	[0, 10^{-3}]	10^{-3}
Learning rate (non-transferred layers)	[0, 10^{-1}]	1.2×10^{-2}
Optimizer	[Adam, SGD, AdaGrad]	Adam
Activation function	[ReLU, ELU, Tanh]	ELU
Loss function	[MAE, MSE]	MSE
Early stopping patience	[10, 30]	10

Network architecture and training

As previously stated in 4.3.2, the quantity of design input parameters in the dataset is equivalent to the number of branches in the MNN. Thus, the MNNs (MNN I, MNN II, and MNN III) corresponding to the three main datasets have 6, 8, and 10 branches, respectively. For MNN I, six individual MLPs with three hidden layers ($m = 3$) and 128 neurons in each layer ($n = 128$) for every design parameter are optimized and built. Each MLP is then trained using subdatasets. Subsequently, the last two hidden layers of these six MLPs are transferred directly to the six branches in MNN I. For MNN II and MNN III, simply eliminate the corresponding branches, add the new ones, and perform the transfer once more. Regarding transferred neurons, there are two options: freezing (learning rate equal to 0), or fine-tuning (learning rate lower than that of non-transferred neurons) [160]. The learning rate for the first and last hidden layers (non-transferred layers) is optimized and set to 0.001. A test of the learning rate of transferred neurons regarding dataset III is then conducted, as shown in Fig. 4.10. The ranges for other hyperparameters are reduced firstly by trial and error. The final optimized hyperparameters are determined using random search to conserve optimization time, such as optimizer (Adam), activation function (ELU), and batch size (8) are established. More details are summarized in Table 4.2. In addition, layer normalization (LN) [161] between each two hidden layers is added. This technique may increase the network's stability and convergence speed since the suggested network has several branches and a large number of trainable parameters.

ters. In particular, LN normalizes the data and determines the mean and standard deviation of the input data for every layer:

$$\mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l, \quad \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}, \quad (4.8)$$

where μ^l and σ^l denote the mean and standard deviation in the l^{th} hidden layer. a_i^l is the input of the i^{th} neuron, and H is the number of neurons in the l^{th} hidden layer. The normalized value can be computed:

$$\tilde{a}^l = \frac{a^l - \mu^l}{\sqrt{(\sigma^l)^2 + \epsilon}}, \quad (4.9)$$

where ϵ is a very small value to prevent the denominator from being 0. Two trainable parameters g (gain) and b (bias) are included in the LN for the purpose of

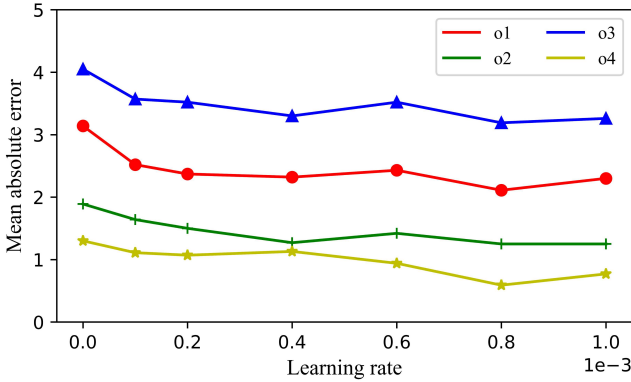


Figure 4.10: Evaluation of prediction performance based on learning rate of transferred neurons under 10 sets of random seeds.

Table 4.3: Evaluation of all datasets.

Performance indicators	Dataset I		Dataset II		Dataset III	
	MAE	R^2	MAE	R^2	MAE	R^2
o_1 (max. torque)	1.63	0.9969	1.60	0.9947	2.01	0.9944
o_2 (max. power)	0.98	0.9985	0.95	0.9972	1.04	0.9981
o_3 (torque ripple at max. torque)	1.17	0.9527	2.39	0.8580	3.19	0.6835
o_4 (material cost)	0.60	0.9991	0.58	0.9985	0.59	0.9987

maintaining the distribution of the original data as much as possible. Assuming the activation function is f , the output of the LN is

$$\mathbf{h}^l = f(\mathbf{g}^l \odot \tilde{\mathbf{a}}^l + \mathbf{b}^l), \quad (4.10)$$

where $\odot$ indicates element-wise multiplication.

The entire training process is accomplished through the NVIDIA Quadro P4000 GPU and took approximately ≈ 3 minutes to complete a whole 5-fold cross-validation. Following cross-validation using ten distinct random seeds, the model with the minimum loss is selected for qualitative analysis. Subsequently, this model is also employed in the following optimization procedure.

Training results

Table 4.3 summarizes the performance of the proposed MNNs on three main datasets. Dataset III exhibited the lowest overall prediction accuracy. This is because it possesses the largest dimension of input design parameters and lacks sufficient data. Since MNN III is constructed on top of MNN I and MNN II, only the results of MNN III are discussed below. Fig. 4.11 provides evaluation details for each KPI and more results of dataset I and dataset II are shown in 8.3.1 The proposed TL-based MNN demonstrated superiority in predicting maximum torque, maximum power, and material cost. The color bar indicates the density of test data. The error in the predicted torque ripple is noteworthy. Insufficient sample size could be a potential explanation for this phenomenon. 2956 samples are relatively few for a dataset containing ten design parameters.

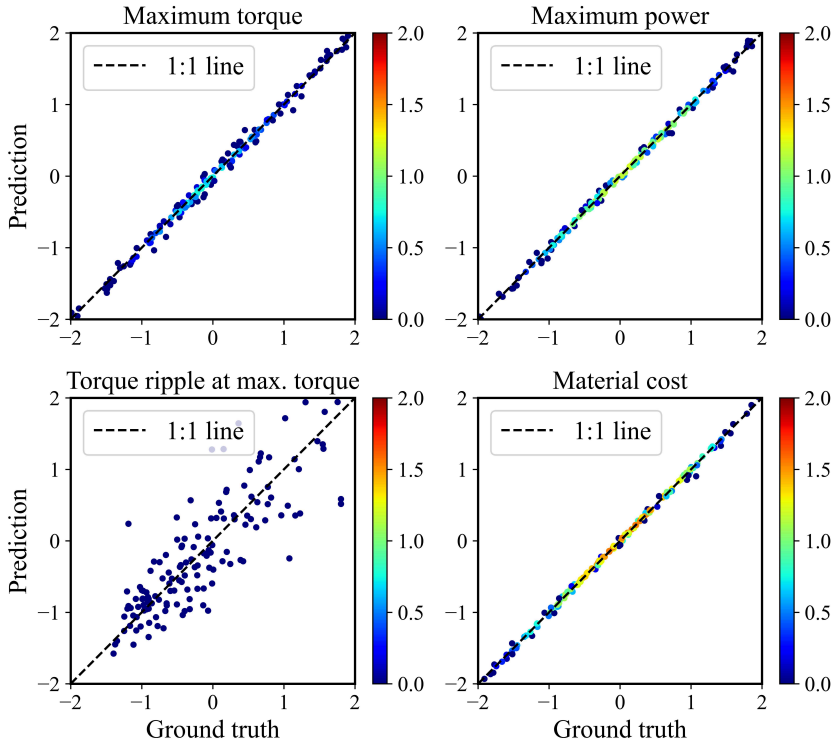


Figure 4.11: Prediction performance of MNN III based on dataset III.

Fig. 4.12 demonstrates a comparison of the proposed MNN III (multi-branched MNN with transferred layers), MNN III without TL (the architecture is completely identical to the former but without the transferred layers, so all the neurons need to be trained from scratch), and traditional MLPs based on dataset III. Ten random seeds are employed to ensure that different models are compared across ten different divisions of the dataset. It can be observed that the MNN III outperformed the MNN III without pre-trained transferred layers, which indicates transfer learning is able to boost the network to obtain improved prediction performance. The primary rationale behind this is that the network with transferred layers can leverage prior knowledge to more effectively determine the optimum.

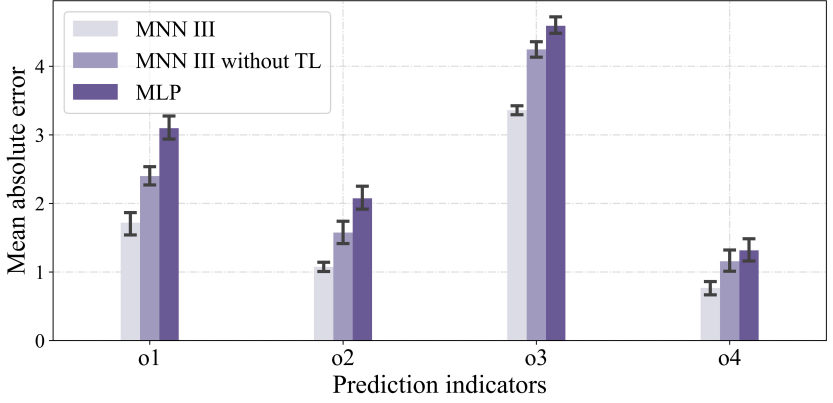


Figure 4.12: Prediction performance comparison among MNN III, MNN III without TL, and MLP.

Besides, the multi-branched architecture offered superior predictive capability in comparison to MLP. In addition, a comparison between MNN III with and without LN is conducted, and the outcome shows a $\approx 31\%$ improvement in prediction accuracy with the inclusion of LN.

4.4 Design optimization and results

The proposed MNN III is applied to speed up the optimization process. A comparative experiment with two common and widely used MOO algorithms is conducted, namely NSGA-II [146] and NSGA-III [162]. The implementation is carried out by the framework Pymoo [163]. The objective functions are maximizing maximum torque and maximum power. The associated generalized optimization problem is formulated as follows:

$$o_1 \rightarrow \max, \quad o_2 \rightarrow \max \quad (4.11)$$

$$\text{s.t.} \quad d_i^L \leq d_i \leq d_i^U, \quad i = 1, \dots, n_{\text{input}} \quad (4.12)$$

$$c_j \leq 0, \quad j = 1, \dots, n_{\text{cons}} \quad (4.13)$$

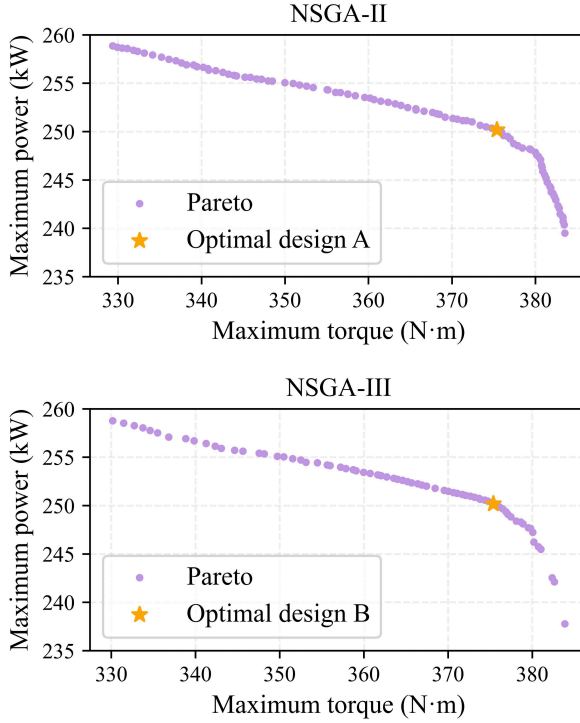


Figure 4.13: Pareto fronts for maximum torque and maximum power based on two different optimization algorithms NSGA-II and NSGA-III.

where d_i indicates input design parameters and d_i^L and d_i^U are predefined bounds. c_j represents constraints for the optimization process. The number of iterations (200), population size (100), crossover probability (0.9), and polynomial mutation (15) are included.

The obtained Pareto fronts are displayed in Fig. 4.13. FEA is performed by choosing a candidate solution from each Pareto front. The selection process for verification using the following rules: Normalize o_1 and o_2 using min-max

Table 4.4: FEA verification of the optimal designs from Pareto fronts.

KPI	Unit	Optimal design A B		
		FEA	Prediction	SMAPE
o_1	N·m	375.38 375.19	379.31 379.89	1.04 % 1.24 %
o_2	kW	247.18 247.30	245.54 245.12	0.67 % 0.89 %
o_3	N·m	23.86 23.07	18.21 18.48	26.86 % 22.09 %
o_4	Euro	343.21 344.00	344.72 345.98	0.46 % 0.57 %

normalization, respectively, then assign them equal weights. Finally, select the design with the highest score:

$$score = \frac{1}{2} \times \tilde{o}_1 + \frac{1}{2} \times \tilde{o}_2, \quad (4.14)$$

where $\tilde{o}_1$ and $\tilde{o}_2$ are the min-max-normalized values. Table 4.4 summarizes the FEA and predicted values for the selected designs. It can be seen that the predictions for o_1 , o_2 and o_4 are in high agreement with the FEA. A minimum SMAPE of 0.46 % in predicting material cost can be achieved. However, the prediction accuracy for torque ripple is relatively low (over 20 %). As previously stated, inadequate data results in inefficient model training, hence causing the prediction outcomes to be considerably biased. Table 4.5 summarizes the optimal design parameter values of selected design A and design B. It can be seen that the highest-scoring optimized designs obtained through different optimization algorithms are very similar, demonstrating the robustness of the optimization results. Then, both optimization approaches are executed 10 times each in order to calculate the average optimization time. The results indicate that it takes approximately 12 seconds to perform NSGA-III, whereas NSGA-II can be finished in only about 8 seconds. At last, to verify the overall prediction accuracy, five non-repeating groups are randomly sampled from the 100 optimized results obtained via NSGA-II for FEA validation, with each group comprising 10 samples. The results are shown in Table 4.6. Compared to individual optimization designs A and B, the overall prediction accuracy decreases and exhibits large fluctuations. It can be observed that the proposed MNN method demonstrates unstable prediction accuracy, achieving high precision on certain samples while still exhibiting a certain degree of prediction bias overall.

Table 4.5: Optimized design parameter values of selected design A and design B.

Design parameter	Design A	Design B	Unit
Magnet 1 width	24.99	25.00	mm
Magnet 1 height	5.00	5.00	mm
Magnet 2 width	31.98	31.96	mm
Magnet 2 height	5.00	5.00	mm
Magnet inclination angle	54.61	54.90	deg
Slot width	3.51	3.50	mm
Flux barrier width	1.00	1.00	mm
Width between magnet 2 and air gap	2.03	2.21	mm
Slot depth	14.30	14.26	mm
Air gap	0.50	0.50	mm

Table 4.6: FEA verification of five group sampling from the results of NSGA-II using SMAPE.

KPI	Group 1	Group 2	Group 3	Group 4	Group 5
o_1	2.69 %	1.87 %	1.41 %	2.82 %	2.23 %
o_2	0.97 %	1.02 %	0.79 %	0.85 %	0.80 %
o_3	20.21 %	13.12 %	27.87 %	24.43 %	17.32 %
o_4	0.40 %	0.51 %	0.23 %	0.55 %	0.49 %

4.5 Summary and discussion

In this chapter, a TL-based MNN for the design optimization of the PMSMs is proposed. This approach can be adapted dynamically in response to changes in the parameter-based input space. By transferring the pre-trained neurons to the new NN as initial weights, the previous knowledge can be reused. After reparametrization, it is no longer needed to retrain the new NN from scratch. The experiments demonstrate that the prediction accuracy of MNN is higher than that of classical MLPs. Furthermore, the TL-based MNN outperformed the normal MNN, which is attributed to the employment of transferred neurons.

The MOO results showed high accuracy in predicting maximum torque, maximum power, and cost, with the optimization completed in around 12 seconds. However, accurate estimation of torque ripple is still a challenge.

It is worth noting that the utilized deterministic metamodel demonstrates considerable fluctuations in its predictive precision, with a probable contributing factor being out-of-distribution challenges [164]. This indicates that throughout the optimization process, the designs produced within the design space diverge from the distribution of the training data. As a result, the model faces challenges in effectively predicting these samples that fall outside the distribution. Furthermore, deterministic models can only provide singular information, namely, point predictions. They are not able to directly yield probability distributions or confidence intervals for outcomes, making it difficult to quantify risks and probabilities in complex scenarios. Therefore, in the following chapters, probabilistic metamodels will be introduced to address the inherent limitations of the deterministic models itself and the issue of overall large errors in optimization results. Multiple optimization strategies based on probabilistic metamodels will also be explored.

Chapter 5

Probabilistic Metamodeling-Based Optimization

This chapter delves into the details of motor design and the optimization processes informed by probabilistic metamodels. The employment of probabilistic metamodels offers supplementary decision-making insights to steer the optimization process. The measurement of uncertainty can influence the precision of optimization outcomes. Section 5.1 describes the motivation for introducing probabilistic models and introduces a probabilistic NN. Section 5.2 presents a new dataset collected for subsequent model training. In Section 5.3, the structure of the probabilistic NN is introduced, followed by its training process. Furthermore, the relationship between model prediction uncertainty and actual prediction bias is explored. Finally, the trained probabilistic metamodel is integrated with optimization algorithms, providing additional uncertainty information for decision-making during the optimization process. Two optimization strategies are proposed. Furthermore, the relationship between sensitivity analysis and model prediction uncertainty is also investigated.

5.1 Introduction and methodology

In the design optimization stage of an electric machine, metamodeling has been widely used to calculate KPIs such as electromagnetic torque and torque ripple.

It is a data-driven approach that replaces computationally expensive simulations by creating metamodels with lower computational costs. However, in practical scenarios, there exists inherent uncertainty in the modeling process. Current methods with metamodels offer deterministic estimations, thereby failing to provide further information to support the optimization. Hence, investigating and developing models that tackle the problem of uncertainty is fundamental.

5.1.1 Background introduction

Bayesian neural networks (BNNs) obtain uncertain prediction intervals by adding probability units to traditional NN [165]. In the field of power engineering, an approach using BNNs to predict short-term wind power generation is proposed in [166]. With the help of predictive uncertainty, irrelevant features can be identified and removed, thereby improving prediction accuracy. Another study [167] offers guidelines for power system dispatch by employing BNNs to estimate the load of electric vehicle charging stations. Recently, this metamodel has been utilized for solar-integrated distribution systems [168]. Combining with a statistical post-processing approach, the distribution systems are optimized in terms of power planning and supply. In addition, BNNs have also been successfully applied to many other research fields [169], such as fault diagnosis [170], economics [171], and biomedical image segmentation [172]. Moreover, another powerful ML probability-based regression model is called Gaussian process regression (GPR) [118]. GPR has the capability to not only output single prediction values but also offer correlated covariance. This covariance can be utilized to quantitatively assess the uncertainty associated with the predicted results. In [173], a method employing the weighted GPR is presented to estimate photovoltaic power, which outperforms other models for multi-input data. Another work [174] introduces a design space exploration algorithm for powertrain optimization. With the inherent uncertainty information, GPR enables the determination of candidate points to be assessed for each iteration. Recent research in [175] proposed a GPR-based adaptive sampling approach for design optimization of electric machines. The original data set is expanded by adding new samples generated by GPR. Furthermore, GPR is also extensively employed in various fields, for instance, fault detection [176], battery state estimation [177], and temperature modeling [178]. To date, the comparison of these two probabilistic models to the design optimization of electric machines has not been conducted. To the author's best knowledge, there are no suggested approaches for using supplementary prediction uncertainty information to

support motor optimization in the literature.

In this chapter, a new design optimization method for electric machines based on uncertainty quantification is introduced [O3]. Initially, BNNs and GPR are employed which can estimate the electric machine KPIs and come up with predictive uncertainty. After establishing the parameters of the models via the training process, the point prediction is derived by sampling a substantial quantity of values and subsequently computing their average. Both metamodels also calculate the standard deviation or correlated covariance, enabling the determination of the corresponding uncertainty interval (UI) for each prediction. A detailed comparison between the two approaches GPR and BNNs is conducted. Finally, a MOO is applied using the more advantageous model as an alternative to FEA. An investigation is conducted to look into the association between predictive uncertainty and prediction accuracy. A sensitivity analysis for the optimized designs reveals information about the relationship between predicted uncertainty and the robustness of design solutions.

5.1.2 Bayesian neural networks

A NN parameterized by network weights $\mathbf{w}$ can be conceived as a conditional distribution model $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$. The process of learning a NN can be viewed as a maximum likelihood estimation (MLE) [179]:

$$\mathbf{w}^{\text{MLE}} = \arg \max_{\mathbf{w}} \log p(\mathcal{D}|\mathbf{w}). \quad (5.1)$$

The goal is to find the optimal setting of weights $\mathbf{w}^{\text{MLE}}$ to most likely model the training data. However, MLE does not make any assumptions regarding the prior probability of $\mathbf{w}$. It treats all possible values of $\mathbf{w}$ as having equal chances of occurring. When NN incorporate a probability distribution over their weights, they are referred to as BNNs. Fig. 5.1 demonstrates the difference between classical NN and BNNs. A BNN introduces a prior distribution $p(\mathbf{w})$, which is considered as the initial distribution of weights, and calculates the posterior distribution $p(\mathbf{w}|\mathcal{D})$ depending on the training data. It can be expressed through Bayes' theorem as

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}. \quad (5.2)$$

Here $p(\mathcal{D}|\mathbf{w})$ and $p(\mathcal{D})$ denote likelihood and evidence, respectively. The distribution over the network parameters is updated by (5.2) when new data are

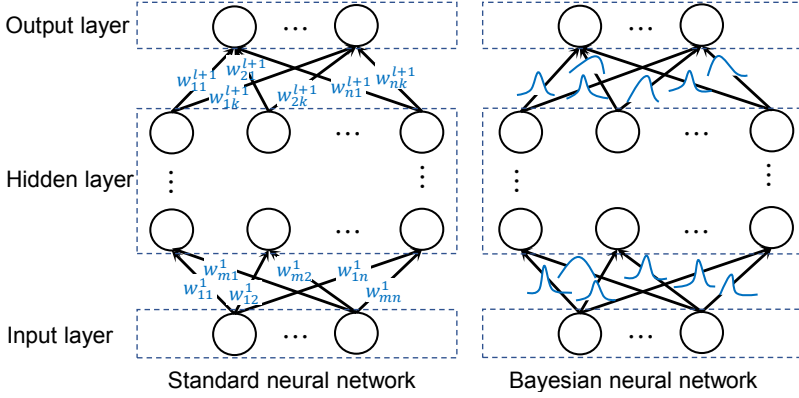


Figure 5.1: Left: each weight is a deterministic value. Right: each weight is associated with a probability distribution.

observed. The exact calculation of $p(\mathcal{D})$ is generally infeasible, necessitating the use of approximate solutions in practical scenarios. In this dissertation, variational inference (VI) is employed to approximate the true posterior distribution over the model parameters $p(\mathbf{w}|\mathcal{D})$ by using a surrogate function $q_\theta(\mathbf{w})$, which is parameterized by θ [180]. The task of determining the posterior distribution is reformulated as an optimization problem, wherein the objective is to search for the optimal set of θ that minimizes the Kullback-Leibler (KL) divergence between the two distributions [179], i.e.,

$$\begin{aligned}
 \theta^* &= \arg \min_{\theta} \text{KL}(q_\theta(\mathbf{w}) \| p(\mathbf{w}|\mathcal{D})) \\
 &= \arg \min_{\theta} \int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{p(\mathbf{w}|\mathcal{D})} d\mathbf{w} \\
 &= \arg \min_{\theta} \underbrace{\int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{p(\mathbf{w})} d\mathbf{w}}_{\text{KL}(q_\theta(\mathbf{w}) \| p(\mathbf{w}))} \\
 &\quad - \underbrace{\int q_\theta(\mathbf{w}) \log p(\mathcal{D}|\mathbf{w}) d\mathbf{w}}_{\mathbb{E}_{q_\theta(\mathbf{w})} [\log p(\mathcal{D}|\mathbf{w})]} + \log p(\mathcal{D}).
 \end{aligned} \tag{5.3}$$

Since the KL divergence is non-negative, it can be further obtained:

$$\log p(\mathcal{D}) \geq \underbrace{\mathbb{E}_{q_\theta(\mathbf{w})}[\log p(\mathcal{D}|\mathbf{w})] - \text{KL}(q_\theta(\mathbf{w})\|p(\mathbf{w}))}_{\text{ELBO}}, \quad (5.4)$$

where the right-hand expression is called evidence lower bound (ELBO). Consequently, the optimization problem shifts from minimization of KL divergence to maximization of ELBO:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{q_\theta(\mathbf{w})}[\log p(\mathcal{D}|\mathbf{w})] - \text{KL}(q_\theta(\mathbf{w})\|p(\mathbf{w})). \quad (5.5)$$

In this manner, the intractable evidence $p(\mathcal{D})$ is avoided. The previously defined prior distribution $p(\mathbf{w})$ and the likelihood $p(\mathcal{D}|\mathbf{w})$ are the only things that need to be handled, as they are more manageable in nature. As a result, the inference issue is reduced to a tractable optimization problem.

5.1.3 Framework

The proposed BNN-based optimization workflow is depicted in Fig. 5.2. Initially, a new motor dataset is collected for model training. Subsequently, the GPR and BNN metamodels are trained separately. The better-performing model is then employed for MOO. Unlike the deterministic models used in the previous chapter, this chapter leverages the uncertainty information in the model outputs to assist the optimization process. Two uncertainty-based optimization strategies are introduced and compared in Chapter 5.4.

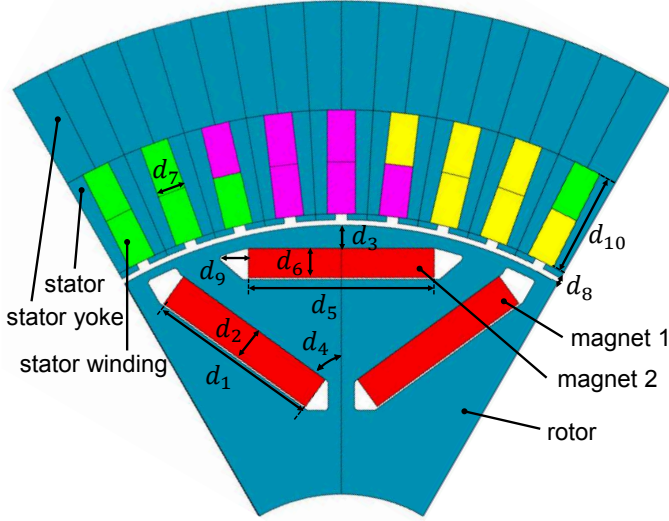


Figure 5.3: Geometry of the reference PMSM.

5.2 Dataset acquisition

In order to facilitate performance comparisons with deterministic models, it is necessary to utilize the same dataset as was employed in Chapter 4. The only difference is that the order of variables in the design parameters has been altered to facilitate subsequent research. The reference machine is shown in Fig. 5.3 with a V-magnet and a bar magnet. The sampling process is illustrated in Fig. 5.2, wherein a set of $k = 10$ input design parameters and four KPIs are considered. Each input parameter is predefined with a range. In Table 5.1, all input design parameters that capture essential machine geometry information and four KPIs are listed. The hairpin winding technique is employed for the stator winding. There are two parallel paths and eight layers per slot, which remain constant regardless of the geometrical parameter d_7 . The dimension of each layer of winding varies in accordance with the dimensions of the slots. In order to cover the design space efficiently, LHS is applied to generate the initial population. The DoE process and all relevant simulations are executed using the in-house toolchain, which is introduced in Section 4.2.1. Following the above sampling procedure, a dataset

Table 5.1: Details of reference machine parameters.

Symbol	Design parameter	Min.	Max.	Unit
d_1	Magnet 1 width	10.0	28.0	mm
d_2	Magnet 1 height	2.0	8.0	mm
d_3	Width between magnet 2 and air gap	0.5	4.0	mm
d_4	Magnet inclination angle	15.0	60.0	deg
d_5	Magnet 2 width	15.0	43.0	mm
d_6	Magnet 2 height	2.0	8.0	mm
d_7	Slot width	3.5	5.0	mm
d_8	Air gap	0.5	2.0	mm
d_9	Flux barrier width	1.0	6.0	mm
d_{10}	Slot height	13.0	18.0	mm
KPI				
o_1	Maximum torque			N·m
o_2	Maximum power			kW
o_3	Torque ripple at maximum torque			N·m
o_4	Material cost			Euro

including $n_d = 2951$ samples is acquired. The samples are then pre-processed using z-score normalization to convert data into a distribution with a mean of 0 and a standard deviation of 1. After normalization, the units and dimensions of the data no longer affect model training, which can improve the stability and convergence speed of model training. In Fig. 5.4, the distribution of the collected dataset in the output space is displayed. Overall, a large proportion of the samples are concentrated in the middle section, whereas there are relatively fewer examples in the tail. It can be seen that the split test and validation sets properly cover the scope of the training set.

5.3 Model training and results

In this section, BNNs and GPR are trained on the same dataset. Details regarding the training process are provided. Finally, the training results are compared, and the better metamodel is selected for subsequent optimization procedures.

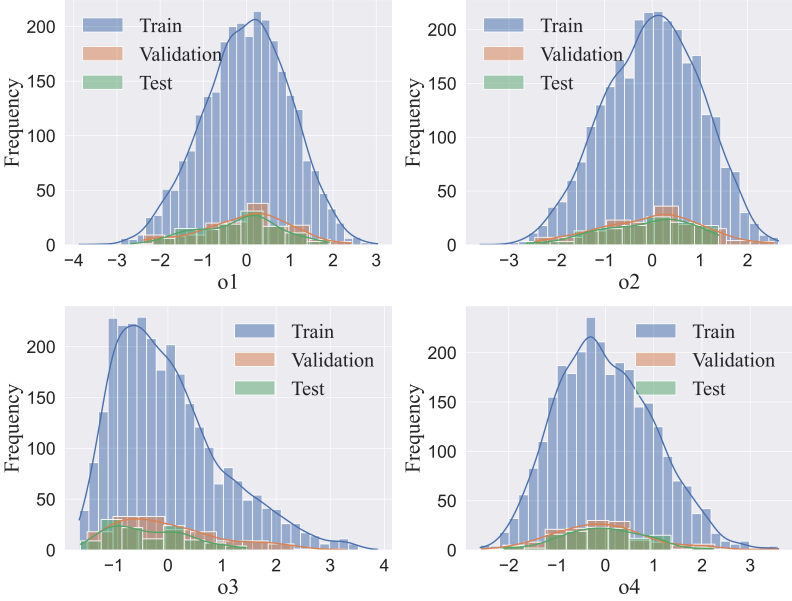


Figure 5.4: Sample distribution visualization of output parameters.

5.3.1 Training of Bayesian neural networks

A BNN can be viewed as a fusion of a conventional NN architecture and probability distribution weights, as illustrated in Fig. 5.1. In this work, a parameter-based multi-branch DNN is employed due to its excellent performance [27], [O2], as shown in Fig. 5.2. The depth of the applied NN is n_H . Following the first hidden layer, the network underwent expansion, resulting in a multi-branch structure. The determination of branches relies on the quantity of input parameters k . Specifically, there are ten branches. Each dense block consisted of n_L neurons. The combination of all branches leads to the last hidden layer containing $k \times n_L$ neurons. Four single-output BNNs to predict different KPIs are trained. To determine the aforementioned and other hyperparameters, every hyperparameter is initially narrowed down for each BNN by trial and error. Specifically, over ten distinct configurations are validated to reach the reduced ranges of hyperparameters. Based on the reduced ranges,

Table 5.2: Details of BNN hyperparameters.

Hyperparameter	BNN- o_1	BNN- o_2	BNN- o_3	BNN- o_4
n_H	4	4	4	3
n_L	10	12	12	9
n_E	3000	3500	4000	2100
n_S	40	80	60	40
F	ELU	ELU	ELU	ReLU
Opt	Adam	Adam	SGD	Adam
Lr	$1.4 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$

n_H : Depth of NN

n_L : Number of neurons per dense block

n_E : Number of epochs

n_S : Number of samples drawn from predictive distribution

F : Activation function

Opt : Optimizer

Lr : Learning rate

the final hyperparameters are identified using random search [81] via 5-fold cross-validation to conserve optimization time. The average prediction accuracy is set as the objective function. The optimization results are summarized in Table 5.2. The number of samples n_S drawn from predictive distribution indicates n_S times forward propagations performed during the prediction phase.

In contrast to the classical DNN, the BNN integrated with Bayesian theory uses a collection of random variables to replace the weights and biases. Before starting the training process, it is important to assign prior distributions for the weights and biases of the neural network. However, precisely determining them poses a challenge due to their lack of actual physical meaning. In practice, the handling of this issue can be achieved by the utilization of standard parametric distributions [181], [182]. Therefore, the prior distribution is specified as the Gaussian distribution with a zero mean and a diagonal covariance $\mathbf{I}$ on the coefficients $p(\mathbf{w}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. As stated previously in Section 5.1.2, the evidence $p(\mathcal{D})$ can not be calculated accurately. To this end, the stochastic variational inference [183] is used, which is a common approach belonging to the VI, to approximate the true posterior $p(\mathbf{w}|\mathcal{D})$. The variational posterior

$q_{\theta}(w)$ is also required to be predefined, which is similarly specified as a normal distribution with a diagonal covariance.

After the training process, the mean and variance of the probability distribution for each weight stop undergoing further updates. In the prediction phase, point predictions are obtained by averaging the predicted values after sampling from the predictive distribution. Furthermore, additional information namely UIs, can be calculated with the help of the standard deviation of each prediction. The data are pre-processed by using the scikit-learn library [184] and the pandas library [185], which enables a more effective generalization of the NN. In addition, an open-source programming language called Pyro [186] for Bayesian inference and probabilistic modeling in ML is applied.

5.3.2 Training of Gaussian process regression

A GPR model is comprised of a set of random variables. The variables follow a Gaussian distribution, which is fully determined by the mean function and the covariance function [118]. Hence, the initial step involves selecting the suitable mean function, covariance function, and noise. The mean to be zero is defined because the dataset pre-processing is done using the z-score normalization. Kernel function holds particular meaning as it conveys prior knowledge regarding the problem to be dealt with. Different kernels should be chosen based on the specific applications.

In this work, the hyperparameters of each GPR model are determined using Bayesian hyperparameter optimization [187]. The objective function is the average prediction accuracy determined via 5-fold cross-validation. The optimized hyperparameters included the kernel function, basis function, and initial value for sigma. The lower bound for sigma ($1 \cdot 10^{-3}$) is set at ten percent of the default value. All other parameters are left as default values. The optimized hyperparameters for each model are displayed in Table 5.3. It can be seen that, depending on the objective, the squared exponential kernel and the Matern kernel 5/2 with automatic relevance determination (ARD) exhibit advantageous properties. The squared exponential kernel is one of the most commonly used covariance functions for GPR, and it is defined as

$$k_{SE}(x, x') = \sigma_f^2 \exp\left[-\frac{1}{2} \sum_{i=1}^k \frac{(x_i - x'_i)^2}{\sigma_i^2}\right], \quad (5.6)$$

Table 5.3: Details of GPR Hyperparameters.

Model	Basis function	Kernel function
GPR- o_1	Constant	ARD Matern 5/2
GPR- o_2	Linear	ARD Matern 5/2
GPR- o_3	Constant	ARD Squared Exponential
GPR- o_4	Constant	ARD Squared Exponential

where σ_f is the signal standard deviation and σ_i is the separate length scale for each input parameter. The Matern kernel 5/2 is a generalization of the radial basis function. It has an additional parameter to control the smoothness of the resulting function

$$k_{Matern}(x, x') = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp(-\sqrt{5}r), \quad (5.7)$$

$$r = \sqrt{\sum_{i=1}^k \frac{(x_i - x'_i)^2}{\sigma_i^2}}. \quad (5.8)$$

Next, the GPR model is trained to repeatedly update the hyperparameters σ_f and σ_i in the kernel function in order to maximize the prediction performance. Once the kernel function has been identified, the matrix K is calculated using (3.17). The computation of the joint distribution of y and y^* is subsequently performed by (3.21). Ultimately, the conclusive prediction is derived from (3.23) and (3.24).

5.3.3 Results and analysis

In this section, both of the proposed probabilistic metamodels, BNN and GPR, are evaluated based on their accuracy in forecasting and predictive uncertainty using different evaluation criteria.

Evaluation criteria

To evaluate the results of the proposed BNN and GPR models, the SMAPE and the R^2 are used as evaluation metrics to measure the quality of the point pre-

dictions, which have been introduced in Section 4.3.3. The SMAPE addresses the issue of the mean absolute percentage error yielding excessively large values when the true value is relatively small. R^2 measures how well the prediction model fits the data, and the closer R^2 is to one, the better the model fits. Before proceeding with the evaluation of the uncertainty predictions, the upper and lower bounds of UIs are calculated for each prediction value as follows:

$$[y_i^L, y_i^U] = [\bar{y}_i^* - Z_{\alpha/2} \cdot \frac{\sigma_i}{\sqrt{n}}, \bar{y}_i^* + Z_{\alpha/2} \cdot \frac{\sigma_i}{\sqrt{n}}], \quad (5.9)$$

where $Z_{\alpha/2}$ denotes the confidence coefficient and σ_i reflects the standard deviation. n is the number of samples. In order to compare the degree of uncertainty between the different metamodels, the negative log-likelihood (NLL) [188] and the prediction interval normalized average width (PINAW) [189] are introduced as evaluation metrics. A smaller NLL results in a better model at generating the observed training dataset, and it can be calculated as below:

$$\text{NLL} = \frac{1}{2} \log 2\pi + \frac{1}{N} \sum_{i=1}^N \log \sigma_i + \frac{1}{2N} \sum_{i=1}^N \frac{(y_i - \bar{y}_i^*)^2}{\sigma_i^2}, \quad (5.10)$$

while PINAW assesses the quality of predictive intervals. It is expressed as follows:

$$\text{PINAW} = \frac{1}{N \cdot R} \sum_{i=1}^N (y_i^U - y_i^L), \quad (5.11)$$

where R represents the range of actual values.

Evaluation of BNN and GPR

In the experiment, the dataset divisions available for BNN and GPR are identical. Fig. 5.5 and Fig. 5.6 exhibit the deviation of the predicted values from the true values for both models. The 90 % and 95 % UIs commonly used in statistics applications are calculated to evaluate the predictive uncertainty. It can be observed that the prediction obtained by GPR demonstrates higher accuracy and less fluctuation for tasks o_1 , o_2 , and o_4 . The smaller interval width of GPR means it offers more reliability than BNN. In predicting torque ripple, the UIs of GPR are much larger than those of BNN, but the fluctuation of the point predictions is still less. One potential explanation for this phenomenon is that during hyperparameter optimization, achieving higher prediction accuracy may affect the model generalization, resulting in a rise in predictive uncertainty.

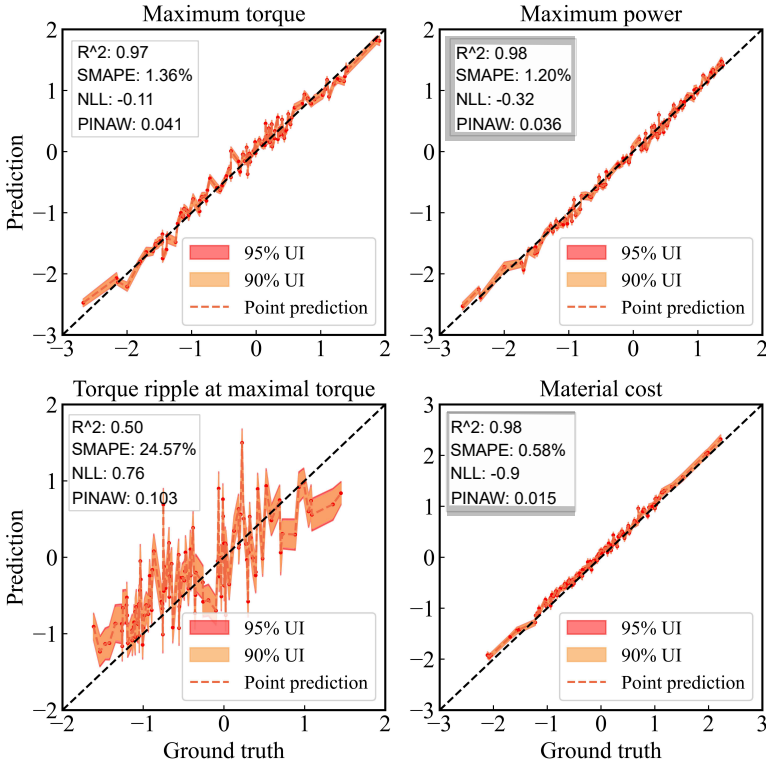


Figure 5.5: Prediction performance of BNN.

Both models achieved high prediction accuracy and relatively low uncertainty on the tasks of predicting o_1 , o_2 , and o_4 . BNN has better performance in predictive uncertainty regarding torque ripple prediction due to its lower NLL and PINAW compared to GPR.

Considering the better performance of GPR over a wide range of tasks, GPR is selected as the final metamodel to engage in the design optimization. Fig. 5.7 illustrates the predictive uncertainty statistics using GPR and their prediction percentage errors for the test samples. Most forecasts for KPIs o_1 , o_2 , and o_4 exhibit a concentration in the area characterized by low predictive uncertainty. This is

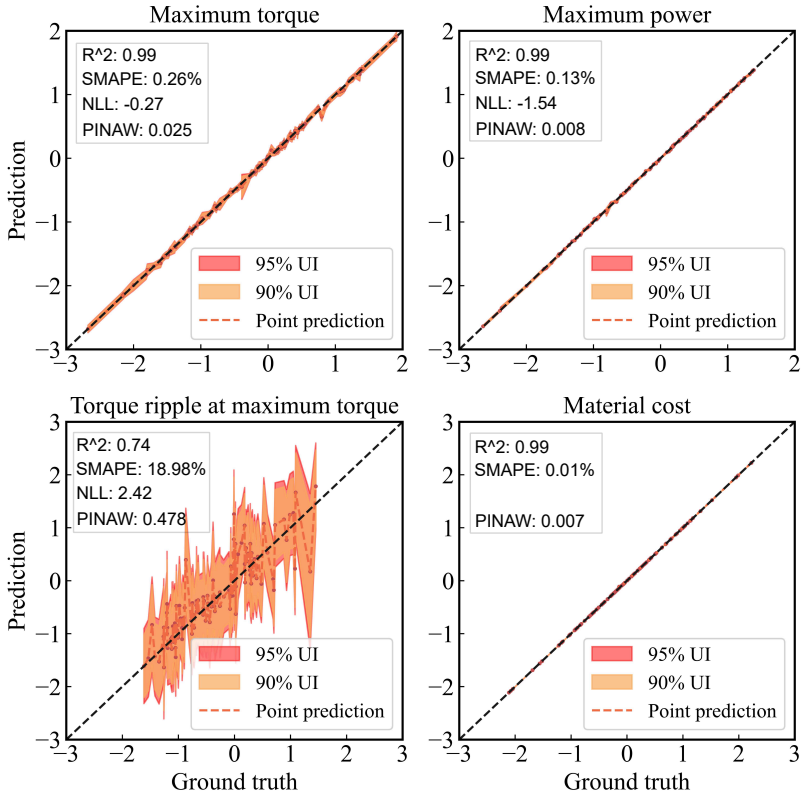


Figure 5.6: Prediction performance of GPR.

due to the fact that the majority of the test samples are distributed in the middle of the output space, as shown in Fig. 5.4. These samples characterize moderate machine performance, and thus the predictions for them are less uncertain. In relation to KPI o_3 , the standard deviations of the predictions are mainly clustered in the area close to the center. This phenomenon may be explained by the insufficient prediction accuracy of o_3 . In particular, by observing the SMAPE curves, it can be seen that the prediction error exhibits a general tendency to grow as

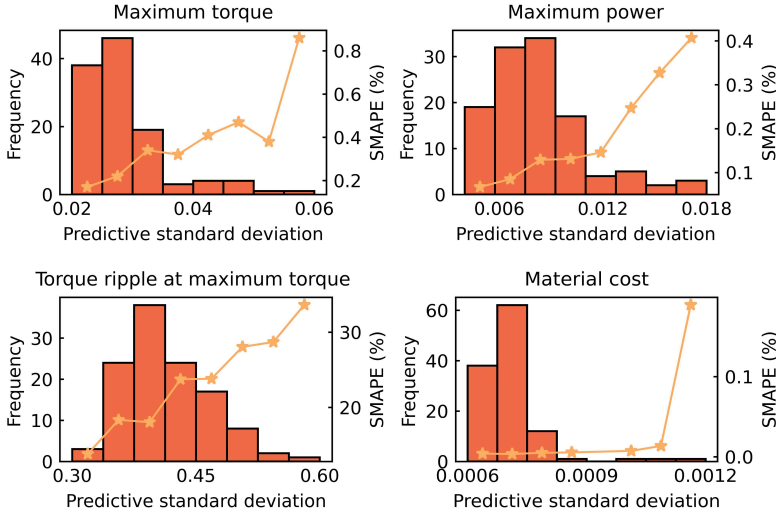


Figure 5.7: Frequency statistics of predictive uncertainty for each KPI using GPR and associated prediction error.

predictive uncertainty rises. It implies that an increase in predictive uncertainty tends to be followed by a decline in prediction accuracy.

Discussion of Probabilistic and Deterministic Approaches

The frequently used DNNs as deterministic metamodels have shown their superiority in the prediction of KPIs [48]. However, they are only able to output point estimations. The probabilistic metamodels possess the capability to provide point estimates along with their associated uncertainty, which may give a wider search range, assisting in locating the optimums. In the following, the predictive performance among BNN, GPR, and DNN are compared. To ensure a more equitable comparison, the same multi-branch structure of DNN as BNN is used. Fig. 5.8 displays the outcomes of these approaches. The results show that GPR has the best prediction accuracy. In comparison to BNN, DNN performs slightly better.

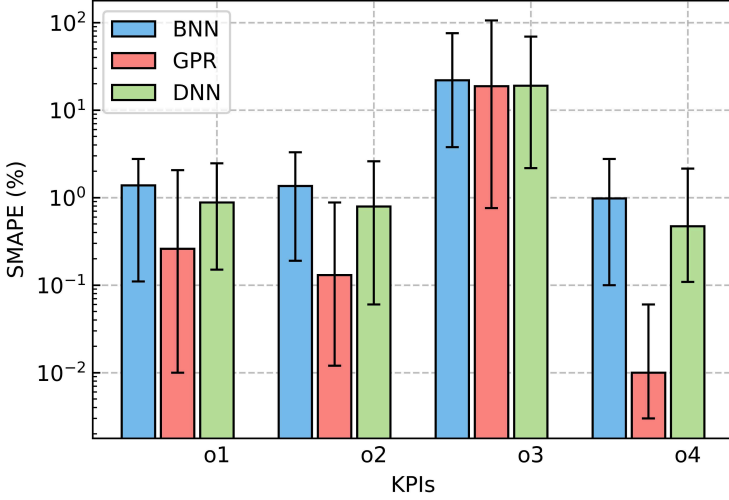


Figure 5.8: KPI prediction performance comparison between probabilistic and deterministic metamodels.

In terms of computational efficiency, it takes a total of ≈ 12 minutes to train four BNNs, while four GPR models and DNNs take ≈ 6 minutes and ≈ 3 minutes, respectively. This effect is caused by probability distributions characterizing the weights and biases in BNNs, which double the parameter pool. For GPR, it needs to “memorize” the complete dataset for training and prediction. Due to the large number of inverse matrices involved in the training process, when applying GPR to a dataset of size n , the computational complexity of exact inference is $O(n^3)$ [74]. As a result, probabilistic metamodels are more computationally expensive and time-consuming than deterministic models. Both BNN and GPR are not suitable for large datasets. In addition, the performance of GPR depends strongly on the selection of a kernel function. However, there is currently no complete theoretical foundation guiding the design of a kernel function customized for a specific circumstance. It can also be seen that the prediction accuracy of the BNN is slightly lower than that of the DNN with the same network architecture. This variation may be attributed to the demand for prior knowledge in the con-

struction of BNNs. Inaccurate or insufficient prior knowledge may result in large deviations in the model output.

5.4 Uncertainty-based optimization and results

The combination of the GPR model with probabilistic prediction and MOO is demonstrated in this section. Two separate optimization strategies are proposed:

1. Strategy I: The point predictions and lower and upper confidence bound values are directly used in the optimization.
2. Strategy II: Individuals with high uncertainty are constrained.

5.4.1 Optimization problem

In this section, MOO is performed for two KPIs: maximize maximum torque and minimize material cost. The optimization problem is defined as follows:

$$\max o_1, \quad \min o_4 \quad (5.12a)$$

$$\text{s.t. } d_i^L \leq d_i \leq d_i^U, \quad i = 1, \dots, 10 \quad (5.12b)$$

$$l_j \leq 0, \quad j = 1, \dots, n_{cons} \quad (5.12c)$$

where d_i denotes input parameters in Table 5.1 and l_j represents constraints for highly uncertain individuals. The Pareto search algorithm (PSA) [190] is adopted as the optimization algorithm, which is based on the direct multi-search principle [191].

In strategy I, point estimations of the GPR as well as their upper and lower boundary values are applied at the 95 % UI case, which are calculated using (5.9), resulting in three individual Pareto fronts. Each input design parameter is further limited by the d_i^L and d_i^U to reduce geometrically invalid designs before optimization. Predictions with a high degree of uncertainty are not constrained. The optimization workflow is displayed in Fig. 5.2.

In strategy II, individuals with high uncertainty are penalized until they are pulled back within the specified feasible region. The penalty function is defined as below:

$$f(\mathbf{d}^{(n)}) = f(\mathbf{d}^{(n)}) + \text{Inf} \cdot \max(\sigma_n - \sigma_{lim}, 0), \quad (5.13)$$

where $f(\mathbf{d}^{(n)})$ is the GPR's point estimation for the n -th individual. σ_n denotes the predictive standard deviation and σ_{lim} is the selected threshold. PSA assigns an infinite penalty value to any point whose estimated standard deviation exceeds the constraint. Only the designs that have predictive standard deviations below the predetermined thresholds are exempt from penalties throughout the optimization process. Using (5.13), a connection between the predictions and their level of uncertainty can be established. This allows reducing highly uncertain designs while generating more confident ones during optimization.

5.4.2 Optimization results

The PSA hyperparameters are set by prior experience, such as Pareto set size (200) and constraint tolerance ($1e-6$). The optimization incorporates limitations on input parameter boundaries in order to reduce the creation of invalid machine designs.

Results of strategy I

The evaluation of the optimization results with regard to strategy I is firstly discussed, which is shown in Fig. 5.9. The Pareto front derived from lower boundary values is on the innermost layer, while the outermost Pareto front is based on the upper boundary values. In the region of low cost and low torque, the three Pareto fronts tend to coincide. The design with the lowest torque is generated by the innermost front. The three Pareto fronts gradually diverge as torque and cost keep rising. The outermost Pareto front yields the greediest machine designs with the highest maximum torque and lowest material cost. To compare the prediction accuracy of designs on different Pareto fronts, fifty designs from each Pareto front are verified and compared with the FEA. The results are summarized in Table 5.4.

From the innermost front, the selected optimized machines have the largest average error in predicting maximum torque ripple (34.18 %), which is possibly due to the large range of the UI. The designs obtained through the upper bound values have the best prediction values with the highest torque and power. However, the actual FEA is lower, resulting in the largest SMAPE for o_1 (1.09 %), o_2 (0.65 %), and o_4 (0.12 %). Designs derived based on point prediction have moderate performance, with the highest average prediction accuracy (o_1 : 0.83 %, o_2 : 0.48 %, o_3 : 30.33 %, o_4 : 0.05 %). The PINAW for each KPI of each case is also

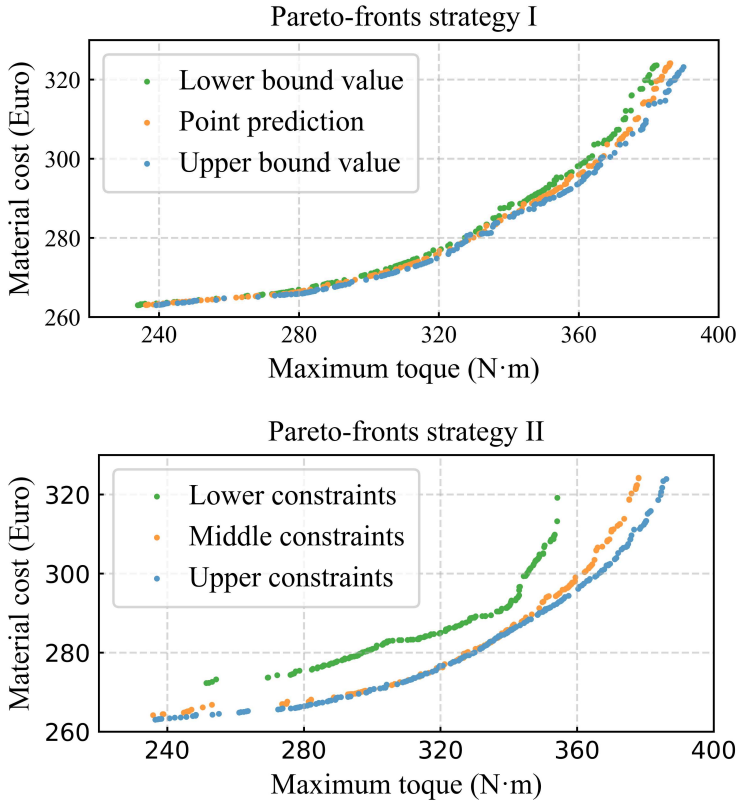


Figure 5.9: Pareto front results regarding two strategies.

calculated to evaluate its degree of uncertainty, as shown in Table 5.4. Point prediction has the lowest PINAW, while the upper bound prediction has a higher PINAW than the lower bound prediction on most tasks. The reason for this trend is that when optimization is performed via upper bound prediction, the resulting designs tend to exhibit greedier behavior and possess a higher degree of uncertainty. The geometrical validity of all the designs is checked on each Pareto front, as shown in Table 5.4. Although the input parameters are initially defined with certain ranges, around 10 % of the designs generated turn out to be invalid.

Table 5.4: Comparison of machine designs obtained by different optimization strategies.

	Strategy I		
Prediction type	Lower bound	Point estimation	Upper bound
Constraint	-	-	-
PINAW o_1	0.0435	0.0425	0.0442
SMAPE o_1	0.98 %	0.83 %	1.09 %
PINAW o_2	0.0161	0.0160	0.0172
SMAPE o_2	0.50 %	0.48 %	0.65 %
PINAW o_3	0.9100	0.8510	0.8637
SMAPE o_3	34.18 %	30.33 %	31.07 %
PINAW o_4	0.0012	0.0011	0.0012
SMAPE o_4	0.09 %	0.05 %	0.12 %
Invalid designs	20	18	19

	Strategy II		
Prediction type	Point estimation	Point estimation	Point estimation
Constraint	C_1	C_2	C_3
PINAW o_1	0.0249	0.0345	0.0418
SMAPE o_1	0.23 %	0.47 %	0.80 %
PINAW o_2	0.0076	0.0118	0.0150
SMAPE o_2	0.15 %	0.34 %	0.42 %
PINAW o_3	0.5901	0.6480	0.7881
SMAPE o_3	5.94 %	22.73 %	29.74 %
PINAW o_4	0.0009	0.0010	0.0010
SMAPE o_4	0.01 %	0.01 %	0.03 %
Invalid designs	0	2	6

$C_1 : \sigma_{L1} = 0.025, \sigma_{L2} = 0.0007$

$C_2 : \sigma_{M1} = 0.040, \sigma_{M2} = 0.0008$

$C_3 : \sigma_{U1} = 0.055, \sigma_{U2} = 0.0009$

Results of strategy II

In optimization strategy II, points with considerable uncertainty are subject to restrictions. Initially, three distinct sets of thresholds are introduced: the maximum and minimum predictive standard deviations for the maximum torque and material cost are firstly calculated. The three constraint sets ($\sigma_{L1} = 0.025$, $\sigma_{L2} = 0.007$), ($\sigma_{M1} = 0.040$, $\sigma_{M2} = 0.008$), and ($\sigma_{U1} = 0.055$, $\sigma_{U2} = 0.009$) are then chosen between the maximum and minimum values. As the σ value decreases, the constraints become more stringent. A decline in uncertainty of the resulting designs is observed following the implementation of constraints on uncertainty, as shown in Table 5.4. Specifically, as the thresholds of the constraints drop, there are associated decreases in the PINAW of the optimized designs. The PINAW of o_1 , o_2 , o_3 , and o_4 is reduced by 41.4 %, 52.5 %, 31.7 %, and 18.2 % compared to the point estimation of strategy I, respectively.

Fig. 5.9 depicts the distinct Pareto fronts corresponding to various sets of constraints. Under the stronger (lower) constraints with σ_{L1} and σ_{L2} , the maximum optimized torque in all optimized solutions is the smallest, and the minimum optimized torque is the largest compared to the results under the other two constraints. The machine design with the highest maximum torque is achieved by the optimization using σ_U (weaker constraints). In the same manner, the designs derived from the σ_U exhibit a higher maximum material cost and a lower minimum material cost compared to the other two cases. This phenomenon is proved by performing FEA validation of the optimized solutions at both ends of each Pareto front. Relaxing the constraints yields superior outcomes, although it amplifies the risk owing to higher model uncertainty. The explanation stems from the direct relationship between the level of limits imposed on uncertainty and the risk of design being discarded in accordance with a more aggressive option. The greediest designs are found on the outermost Pareto front, which is also the area with the highest degree of uncertainty. The lowest predictive uncertainty is achieved from the innermost designs, although they are the most conservative.

The frequency statistics of predictive uncertainty for each KPI after the optimization based on strategy II are displayed in Fig. 5.10. The distribution of all optimized designs over the output spaces o_1 and o_4 in terms of predictive standard deviation followed predefined constraints, e.g., in the case of o_1 Fig. 5.10 (a) and o_4 Fig. 5.10 (d) with lower constraints, the green dots (optimized designs) on the 1D o_1 -axis and o_4 -axis are distributed in areas less than σ_{L1}

$= 0.025$ and $\sigma_{L2} = 0.0007$, respectively. In addition, the uncertainty of the optimized designs over o_2 and o_3 is also decreased, even though o_2 and o_3 are neither optimization objectives. This is because a lot of designs may have more than one highly uncertain predicted KPIs concurrently. When the designs with highly uncertain predictions on o_1 and o_4 are excluded, a portion of highly uncertain o_2 and o_3 predictions is concomitantly eliminated. All optimized machines are split into eight bins regarding uncertainty for each KPIs. One design for every bin is randomly chosen to perform FEA, and then the corresponding SMAPE is calculated. The outcomes are shown in Table 5.4. It can be observed that prediction percentage error tended to rise broadly with increasing predictive uncertainty.

Moreover, the validity of all optimal solutions is assessed, and the results are listed in Table 5.4. The validity of the generated designs is greatly improved as the limiting thresholds of the optimized parameters are reduced. This is because uncertainty increases with distance to samples. Extreme designs typically have an excessive degree of uncertainty. They can be filtered out with the help of restricting uncertainty. The invalidity of designs is reduced by 100 %, 89 %, and 67 % in the case for three different constraint sets. Regarding the duration of the optimization, it is performed for a population size of 200 based on different strategies 50 times each. The optimization without constraints has the shortest average duration of ≈ 11 seconds, while the optimization with constraints σ_L takes ≈ 207 seconds.

In summary, the uncertainty degree and prediction error of the final optimized machines are reduced by constraining the predictive standard deviation of the generated designs during the optimization process. The application of ML probabilistic metamodels allows users to evaluate the quality of predictions and determine if the level of confidence associated with such predictions is satisfactory. With the proposed design optimization based on predictive uncertainty, the optimized solutions are not entirely dependent on the algorithm's choice. Instead, further information is used to assist in analyzing the results and taking further steps if needed.

5.4.3 Sensitivity analysis

In addition to showing excellent performance, a well-optimized design should also possess robustness, meaning it should not be excessively affected by minor variations in its design parameters [192], [193]. To investigate the impact of

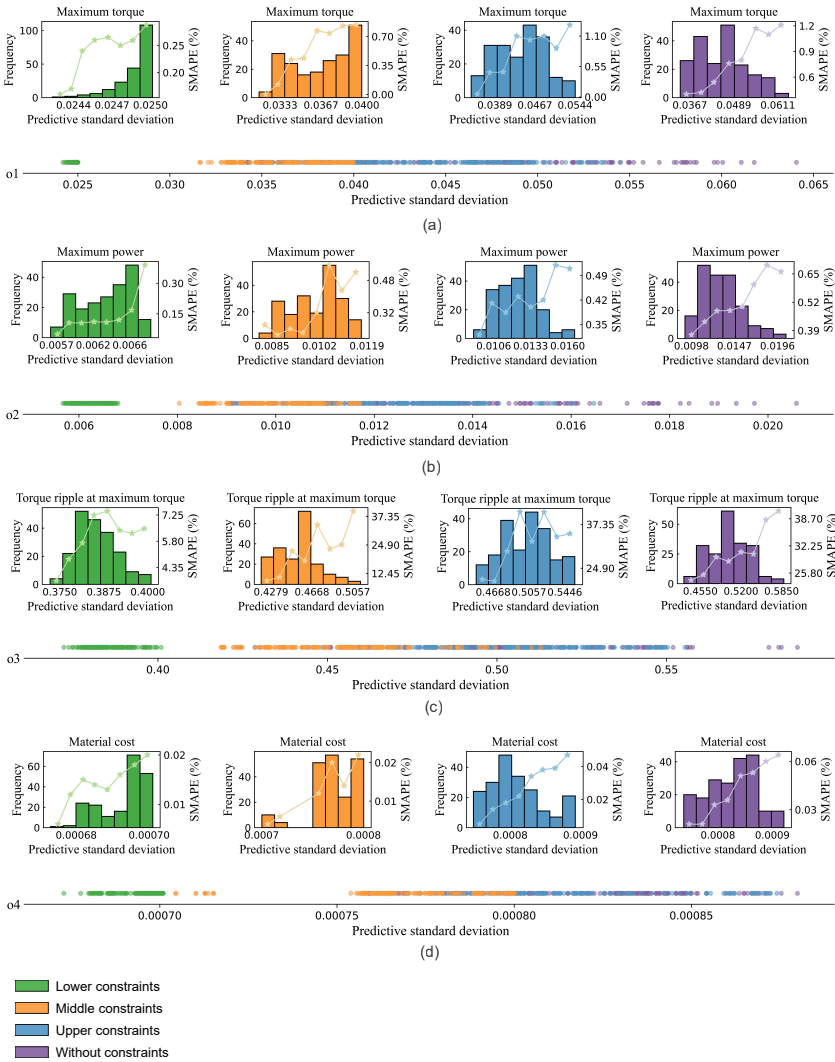


Figure 5.10: Frequency statistics and prediction accuracy for each KPI based on strategy II optimization.

the uncertainty-related constraints added in strategy II on the design robustness, a global sensitivity analysis [194] is conducted on optimized solutions under different constraints. After optimization, the optimized design parameter vectors $\mathbf{x}_i = \{x_{i1}, x_{i2}, \dots, x_{i10}\} (i = 1, 2, \dots, N_{opt})$ and corresponding target parameters (KPIs) with four objectives $\mathbf{y}_i(\mathbf{x}_i) = \{y_{i1}(\mathbf{x}_i), y_{i2}(\mathbf{x}_i), y_{i3}(\mathbf{x}_i), y_{i4}(\mathbf{x}_i)\}$ are obtained. Several feasible deviation intervals are applied for each design variable using relative percentage, e.g., for the design variable x_{i1} , the range is $[x_{i1} - x_{i1} \cdot p\%, x_{i1} + x_{i1} \cdot p\%]$. LHS is performed to generate 1000 “neighboring” designs for each optimized solution from Pareto fronts. The KPIs of these “neighboring” designs are then computed using GPR. The average global sensitivity of the target variable $y_{ij}(\mathbf{x}_i)$ can be calculated as below [194]:

$$\bar{\Delta}_{glob}(y_j) = \frac{1}{N_{opt}} \sum_{i=1}^{N_{opt}} \frac{\tilde{x}_i \cdot (y_{ij}^{max} - y_{ij}^{min})}{y_{ij} \cdot (\tilde{x}_i^{max} - \tilde{x}_i^{min})}, \quad (5.14)$$

where $\tilde{x}_i$ is the design variable value of the i th optimized individual showing the largest relative deviation associated with two “neighboring” designs that possess y_{ij}^{min} and y_{ij}^{max} . In addition, $\tilde{x}_i^{min}$ and $\tilde{x}_i^{max}$ represent the minimum and maximum values of $\tilde{x}_i$ inside these two “neighboring” designs, respectively. y_{ij} is the target variable value of the i th optimized individual. N_{opt} is the number of optimal solutions on the Pareto fronts. Moreover, another two evaluation metrics of global sensitivity are calculated:

$$\bar{y}_j^{loss} = \frac{1}{N_{opt}} \sum_{i=1}^{N_{opt}} \frac{(y_{ij}^{max} - y_{ij})}{|y_{ij}|}, \quad (5.15)$$

$$\bar{y}_j^{gain} = \frac{1}{N_{opt}} \sum_{i=1}^{N_{opt}} \frac{(y_{ij} - y_{ij}^{min})}{|y_{ij}|}. \quad (5.16)$$

They can be used to measure how much the target values of “neighboring” designs change compared to the base optimal values of the Pareto solutions [194].

The calculations for four optimization scenarios are performed, and the results regarding global sensitivities are summarized in Table 5.5. The KPI o_3 is highly sensitive, while o_1 , o_2 , and o_4 are slightly sensitive. The average global sensitivity $\bar{\Delta}_{glob}(y_j)$ of every target under lower uncertainty constraints is the smallest, implying that optimization under the lower (stronger) constraints produces machine designs with the highest average robustness. As the constraints weaken, the

Table 5.5: Global sensitivity results for the design solutions under different constraints.

Indicator		Sensitivity analysis targets			
		o_1	o_2	o_3	o_4
Lower constraints	$\bar{y}^{loss}$	0.74 %	0.84 %	4.91 %	0.49 %
	$\bar{y}^{gain}$	0.81 %	0.80 %	4.41 %	0.53 %
	$\bar{\Delta}_{glob}$	0.82	0.84	4.96	0.55
Middle constraints	$\bar{y}^{loss}$	0.80 %	0.84 %	5.72 %	0.50 %
	$\bar{y}^{gain}$	0.79 %	0.80 %	5.21 %	0.53 %
	$\bar{\Delta}_{glob}$	0.89	0.85	5.82	0.55
Upper constraints	$\bar{y}^{loss}$	0.82 %	0.90 %	7.17 %	0.51 %
	$\bar{y}^{gain}$	0.86 %	0.97 %	7.42 %	0.52 %
	$\bar{\Delta}_{glob}$	0.93	1.01	7.73	0.57
Without constraints	$\bar{y}^{loss}$	0.93 %	0.90 %	7.15 %	0.54 %
	$\bar{y}^{gain}$	0.86 %	0.88 %	6.96 %	0.49 %
	$\bar{\Delta}_{glob}$	0.94	0.95	7.30	0.57

sensitivity of each KPI increases. It can be seen that stronger constraints result in a slightly improved average robustness of the generated designs. However, the average sensitivities of KPIs o_2 and o_3 for designs generated through upper constraints are slightly higher than for designs generated without constraints. This variance is probably due to the fact that only the uncertainties for o_1 and o_4 are constrained. Furthermore, the boundary set by the upper constraints might not be sufficient to have some effect on robustness for every target. The proposed strategy demonstrates the correlation between prediction uncertainty and design parameter uncertainty, which can be utilized to generate more robust optimized solutions.

5.5 Summary and discussion

This section presents a novel uncertainty-based optimization method for electric machines. The results are summed up below.

- Two probabilistic metamodels are introduced, which are trained to produce point estimations of KPIs while also providing associated uncertainty. GPR outperforms BNNs and DNNs in the case with high forecast accuracy after validation by FEA. The results indicate that increased uncertainty in prediction tends to lead to a drop in prediction accuracy.
- Two strategies for optimization with the help of uncertainty quantification are presented. The results in terms of strategy I indicate that the Pareto front obtained based on the upper boundary values has the greediest machine designs, but it leads to higher prediction errors and degrees of uncertainty. In contrast, the designs in the middle of Pareto front have the highest average prediction accuracy and the lowest uncertainty.
- The proposed strategy II restricts the standard deviation of designs during the optimization. As the constraints become more stringent, there is a tendency to exclude machine designs that possess greedier performance, but the resulting optimized designs have a lower degree of uncertainty and higher prediction accuracy. In addition, it can also assist in reducing geometrically invalid designs.
- A sensitivity analysis is performed to study the correlation between predictive uncertainty and the robustness of design solutions. The results show that the stronger constraints lead to a slightly higher average robustness of the optimized solutions.

The uncertainty information enables users to strike a balance between optimized design KPIs and prediction accuracy. This research establishes the foundation for probabilistic metamodeling-based prediction of KPIs and uncertainty-based optimization in the field of electric machines.

This chapter continues to utilize a relatively large dataset comprising 2951 data points. The data collection is based on FEA, meaning that generating the dataset consistently requires significant time investment. The research in this chapter primarily focuses on proposing the implementation of probabilistic metamodels and uncertainty information to assist the motor optimization process. However, no corresponding solution has been provided for the issue of excessive dataset

generation time. Therefore, the next chapter will discuss data-efficient learning methods. The focus will be on how models can be effectively trained with minimal data while maintaining a certain level of prediction accuracy.

Chapter 6

Data-Efficient Learning-Based Optimization using Limited Data

This chapter emphasizes the quick expansion of existing datasets through data-efficient learning techniques to enhance forecast accuracy. At first, a semi-supervised learning-based method is introduced in Section 6.2. A developed self-training framework is able to rapidly increase the dataset size, thereby improving the prediction performance of trained metamodels. The advantages and limitations of this method are also discussed. Additionally, a novel data-efficient learning method based on active learning is proposed in Section 6.3. Multiple probabilistic metamodels are introduced and compared. Subsequently, four distinct optimization strategies are designed based on the active learning approach. Furthermore, the optimization of the initial dataset size is explored using the best optimization strategy. Finally, both methods are summarized and discussed.

6.1 Introduction and motivation

In recent years, a considerable number of studies have concentrated on motor optimization utilizing deterministic metamodels. Hence, Chapter 5 presents an

innovative optimization method supported by probabilistic models, addressing the gap in probabilistic metamodels in the domain of ML for motor design optimization. Thus far, the methods introduced have focused on the models and the optimization phase. Investigations concerning datasets, particularly the efficient usage of data and the optimization under limited data conditions, remain incomplete.

In the field of ML-based motor design optimization, the fundamental reason why data acquisition consumes significant time lies in the high cost of generating each data sample. Unlike domains such as image or text where data can be rapidly collected directly from the real world, motor design data primarily relies on high-precision finite element simulations. A single electromagnetic simulation typically requires consideration of magnetic saturation, transient effects, and multi-condition operation, with computation times often ranging from minutes to hours. When multi-physics coupling, such as temperature fields and structural fields, is further introduced, the computational cost per sample increases significantly. Additionally, motor design spaces are high-dimensional with strong parameter coupling. To effectively cover the design space and obtain representative data for ML models, a large number of samples must be generated. This process also faces the issue of invalid samples. If experimental data is used, prototype manufacturing and testing cycles are even longer. Furthermore, influenced by assembly errors and environmental conditions, repeated experiments are required to ensure data reliability. Moreover, motor optimization is highly sensitive to prediction accuracy, making low-precision data inadequate. Consequently, each sample requires rigorous computation and post-processing, further increasing the time cost of data acquisition and organization. Collectively, the high computational cost of high-fidelity simulation, the multidimensional complexity of the design space, and stringent data quality requirements combine to make data acquisition in this field highly time-consuming [195].

Therefore, to alleviate the time-consuming nature of dataset generation, data-efficient learning methods are the primary emphasis of this chapter. The central approach involves rapidly expanding existing datasets with new data. In subsequent sections, two novel data-efficient learning approaches are proposed. The first method in Section 6.2 rapidly generates additional training data from an initial dataset. The second method in Section 6.3 iteratively expands a small dataset by iteratively generating new key data points. A comparison between

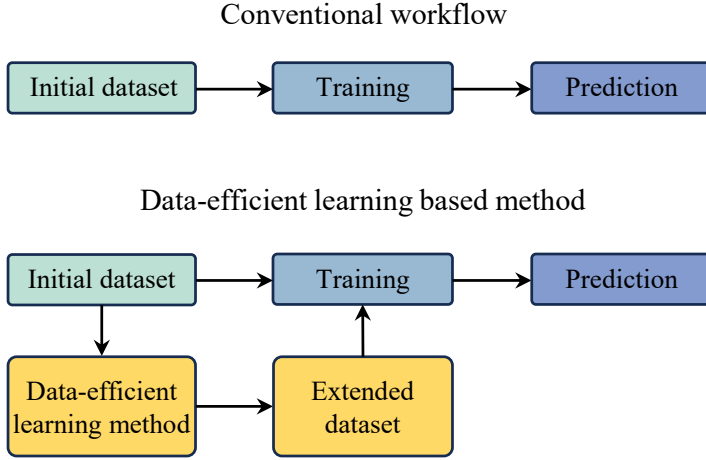


Figure 6.1: Workflow of data-efficient learning method.

conventional methods and data-efficient learning methods is illustrated in Fig. 6.1.

6.2 Semi-supervised learning-based method

So far, most of the studies are based on supervised learning [196]. It requires a large quantity of well-labeled samples to train the metamodel properly in order to obtain good generalization ability. Nevertheless, in the field of design optimization for electric machines, if a large dataset is desired for metamodels, the time required to collect samples using FEA is substantial. To solve this issue, a semi-supervised method is proposed for efficiently using data from a relatively small dataset obtained by FEA. Semi-supervised learning is a recent type of ML technique developed over the past decade. The idea aims to address the problem of reduced performance in conventional supervised learning due to a lack of training data by using unlabeled data during model training with limited labeled data [197]. It has been widely used in various fields of research, such as object detection[198], fault diagnosis [199], medical diagnosis [200],

and power engineering [201].

In the following sections, a self-training framework utilizing semi-supervised learning is presented to label unlabeled data to expedite the sampling procedure [O4]. In the beginning, a classifier (geometry checker) and a regression model (predictor) are trained using initial data labeled by FEA. The classifier examines design feasibility, while the regression model predicts KPIs of the feasible designs, such as maximum torque and maximum power. Next, pseudo-labels are created by applying a confidence score to filter out certain predictions made by the trained models on unlabeled data, as not all predictions are expected to be trusted. The probabilistic metamodel, namely the Monte-Carlo (MC) Dropout [202], is introduced to offer uncertainty information. Subsequently, the filtered pseudo-labels are integrated with the initial labeled data, and retraining is conducted on the combined dataset. The entire process can be repeated many times until convergence is reached.

6.2.1 Self-training framework and dataset

Self-training is an important learning technique in semi-supervised learning [144]. The main idea is to find a way to enlarge the initially labeled dataset with unlabeled datasets. Fig. 6.2 describes the general workflow of self-training methods.

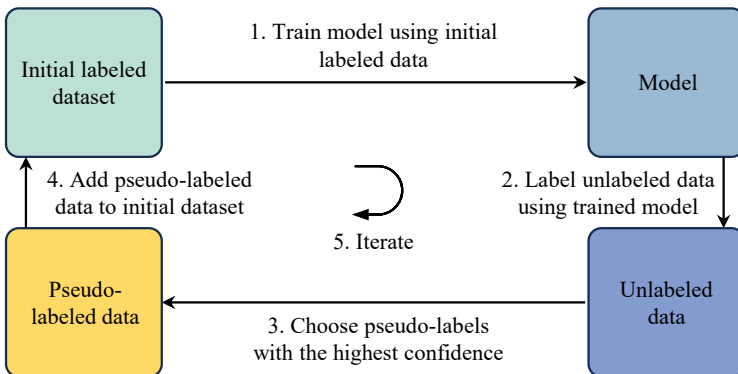


Figure 6.2: General workflow of self-training methods.

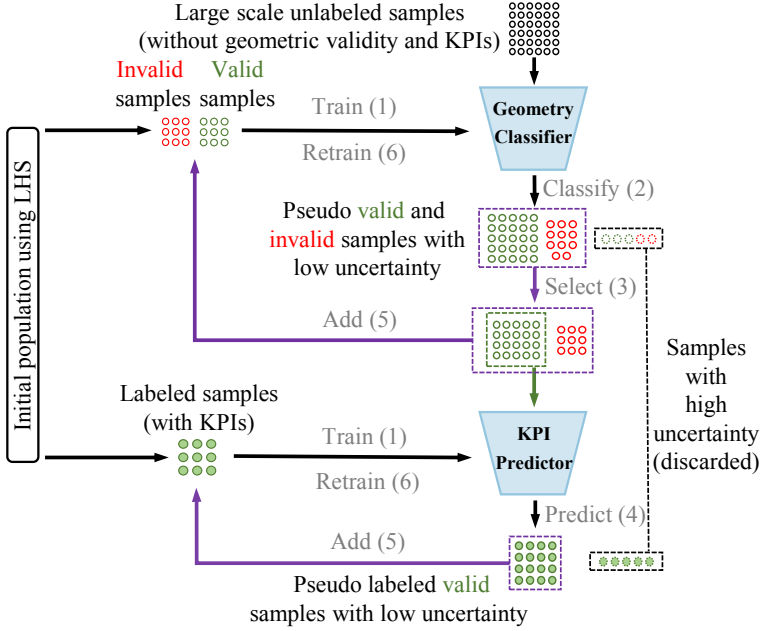


Figure 6.3: Self-training framework based on semi-supervised learning for electric machine design.

Proposed self-training workflow

Based on the concept of self-training, a self-training framework for electric machine design is proposed, which is illustrated in Fig. 6.3. First of all, LHS is utilized to generate data vectors due to its efficient coverage of design parameter space. The data are then identified as geometrically valid and invalid designs. Valid designs are calculated via FEA to obtain corresponding KPIs. All of the above is accomplished through the simulation toolchain. At this point, two initial datasets are obtained: One containing only labels about whether the design is valid, is used to train the classifier, and the other with simulated KPIs is used to train the predictor that only predicts KPIs for valid samples. Subsequently, the detailed steps are described as follows:

1. *Train*: The dataset, which includes whether motor geometries are feasible, is used to train a geometry checker as a classifier initially. The other dataset with simulated KPIs is fed to a regression metamodel to predict KPIs.
2. *Classify*: A large amount of new data with only design parameters is generated via LHS. The geometry checker is then employed to separate these data into pseudo-valid or invalid ones. Further categorization is done based on the confidence scores from the probabilistic classifier. The data with high uncertainty are then discarded.
3. *Select*: Due to the fact that the initially trained classifier has a certain number of classification errors, the Mahalanobis distance [203] is applied to calculate the similarity of the data with pseudo-labels to the initial dataset in order to remove the outliers from the data. The Mahalanobis distance can be seen as a modification of the Euclidean distance, correcting the problem that the dimensions are scale-inconsistent and correlated, as shown in Fig. 6.4. It can be expressed as follows:

$$D_M = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{S}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}, \quad (6.1)$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in})$ is the n -dimensional data point with pseudo-label and $\mathbf{x}_j = (x_{j1}, x_{j2}, \dots, x_{jn})$ is the data point from the initial dataset. $\mathbf{S}$ is the covariance matrix, which characterizes the correlation of features across dimensions. Combine each dimension of m data points into a separate dimensional vector $\mathbf{v}_1 = (x_{11}, x_{21}, \dots, x_{m1})$, $\mathbf{v}_2 = (x_{12}, x_{22}, \dots, x_{m2})$, ..., $\mathbf{v}_n = (x_{1n}, x_{2n}, \dots, x_{mn})$. The average value for each dimensional vector can be obtained by

$$\bar{v}_i = \frac{1}{m} \sum_{j=1}^m v_{ji}. \quad (6.2)$$

The covariance matrix between two dimensional vectors can be calculated by

$$\text{Cov}(\mathbf{v}_i, \mathbf{v}_j) = \frac{1}{n-1} \sum_{k=1}^m (v_{ki} - \bar{v}_i)(v_{kj} - \bar{v}_j). \quad (6.3)$$

Since the sample mean $\bar{v}_i$ is used instead of the true overall mean when calculating the covariance, the sample covariance is divided by $n-1$ rather than n , making it an unbiased estimate of the population covariance. After calcu-

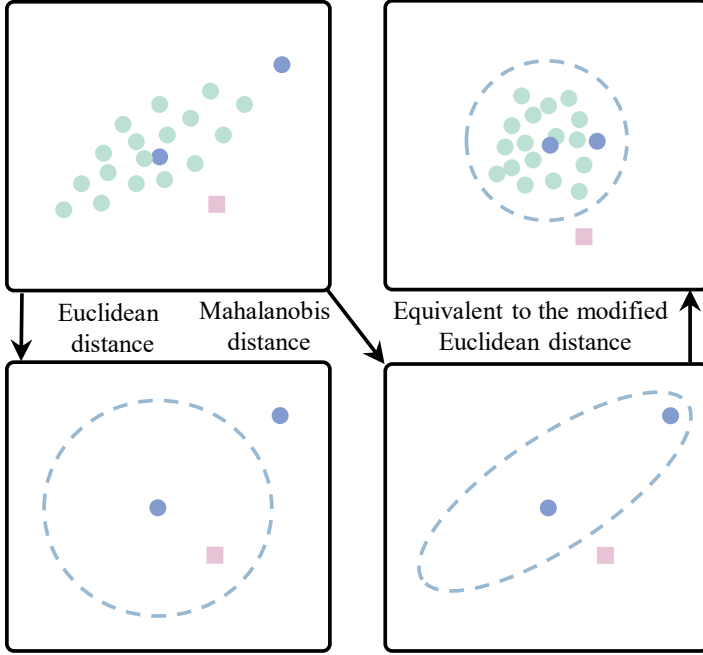


Figure 6.4: Difference between Mahalanobis distance and Euclidean distance.

lating the covariance between all dimension vectors in pairs, the covariance matrix S can be achieved:

$$S = (s_{ij})_{n \times n} = \begin{bmatrix} \text{Cov}(\mathbf{v}_1, \mathbf{v}_1) & \text{Cov}(\mathbf{v}_1, \mathbf{v}_2) & \dots & \text{Cov}(\mathbf{v}_1, \mathbf{v}_n) \\ \text{Cov}(\mathbf{v}_2, \mathbf{v}_1) & \text{Cov}(\mathbf{v}_2, \mathbf{v}_2) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(\mathbf{v}_n, \mathbf{v}_1) & \text{Cov}(\mathbf{v}_n, \mathbf{v}_2) & \dots & \text{Cov}(\mathbf{v}_n, \mathbf{v}_n) \end{bmatrix}. \quad (6.4)$$

After obtaining the Mahalanobis distance for each unlabeled data point, a threshold is set. If the Mahalanobis distance is less than this threshold, the data point is selected for the subsequent step.

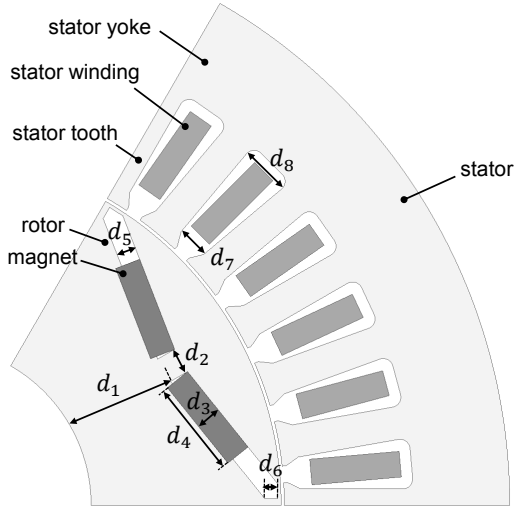


Figure 6.5: Reference PMSM geometry with partial design parameters.

4. *Predict*: The trained predictor is then adopted to predict KPIs for the selected pseudo valid data. Up until this point, new data with pseudo-labels regarding predicted KPIs are obtained.
5. *Add*: Following the selection process, pseudo valid and invalid data are incorporated into the initial dataset, which includes the data for classification. In addition, the data with predicted KPIs are added to the other initial dataset, which includes data for regression.
6. *Retrain*: With the help of the new extended datasets, the classifier and predictor are retrained to achieve better prediction performances.

This approach can be repeated iteratively until the metamodel's performance no longer improves with new added data.

Dataset introduction

The machine template is a PMSM with V-magnets, as shown in Fig. 6.5. A total of 13 input design parameters and four KPIs are included in the dataset. Table 6.1

Table 6.1: Details of reference machine parameters.

Symbol	Design parameter	Min.	Max.	Unit
d_1	Distance from duct bottom to shift	2.0	24.0	mm
d_2	Duct opening width	1.0	5.0	mm
d_3	Magnet thickness	3.0	6.5	mm
d_4	Magnet width	26.0	48.0	mm
d_5	Duct thickness	2.0	5.5	mm
d_6	Rib height	0.5	2.0	mm
d_7	Slot wedge maximum width	3.8	7.0	mm
d_8	Slot body bottom width	6.6	9.8	mm
d_9	Rib width	1.0	15.0	mm
d_{10}	Minimum distance between side magnets	1.0	22.0	mm
d_{11}	Slot opening width	1.8	3.2	mm
d_{12}	Slot opening height	0.3	1.6	mm
d_{13}	Slot wedge height	1.5	4.2	mm
KPI				
o_1	Maximum torque			N·m
o_2	Maximum power			kW
o_3	Torque ripple at maximum torque			N·m
o_4	Efficiency at corner point			%

shows the parameters about the reference machine. First, the dataset for training the classifier is collected. The labels for the data only indicate whether the model is valid or invalid: if valid, the label is 1; if invalid, the label is 0. The initial training set for the classifier includes 3002 samples (1637 valid designs, 1365 invalid designs). Then, finite element simulations are performed on these 1637 data points to obtain the KPI results for each. The data points labeled with KPIs are provided to the predictor for training. Finally, these two datasets are split into training sets and test sets. While the initial training sets have 2836 and 1555 samples, respectively. The number of samples in the two test sets is 166 and 82, respectively.

6.2.2 Training of neural networks

A new probabilistic metamodel is implemented to acquire uncertainty information for predictions that facilitates data filtering.

Monte-Carlo Dropout

Unlike traditional deterministic (point) prediction, probabilistic metamodels provide additional information on prediction uncertainty along with point prediction. MC Dropout is a DL approach used to assess the prediction uncertainty of NN. It involves adding Dropout throughout both the training and inference stages to produce several predictions. The model's confidence is then evaluated by analyzing the variance among these predictions. In [204], it has been proven that a NN with Dropout layers before each hidden layer is mathematically close to a Gaussian process. Dropout is a regularization method in which a portion of the neurons in a NN are randomly deactivated during the training process [205], as showed in Fig. 6.6. In a standard forward process of a NN, the l^{th} and $(l+1)^{th}$ hidden layers can be calculated by the following equations:

$$\mathbf{z}_i^{l+1} = \mathbf{w}_i^{l+1} \mathbf{a}^l + \mathbf{b}_i^{l+1}, \quad (6.5)$$

$$\mathbf{a}_i^{l+1} = F(\mathbf{z}_i^{l+1}), \quad (6.6)$$

where $\mathbf{z}_i^{l+1}$ represents the weighted sum of the i^{th} neuron in the $(l+1)^{th}$ hidden layer. $\mathbf{a}^l$ indicates the output vector of the l^{th} hidden layer. $\mathbf{w}_i^{l+1}$ and $\mathbf{b}_i^{l+1}$ are the weight and bias vectors to the i^{th} neuron of the $(l+1)^{th}$ hidden layer. F

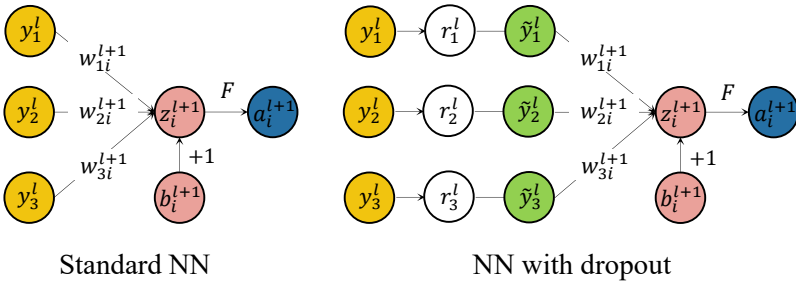


Figure 6.6: Comparison of a standard NN and a NN with Dropout layer.

means the activation function. For the NN with a Dropout layer, the propagation can be described as below:

$$r_j^l \sim \text{Bernoulli}(p), \quad (6.7)$$

$$\tilde{\mathbf{a}}^l = \mathbf{r}^l * \mathbf{a}^l, \quad (6.8)$$

$$\mathbf{z}_i^{l+1} = \mathbf{w}_i^{l+1} \tilde{\mathbf{a}}^l + \mathbf{b}_i^{l+1}, \quad (6.9)$$

$$\mathbf{a}_i^{l+1} = F(\mathbf{z}_i^{l+1}), \quad (6.10)$$

where $\mathbf{r}^l$ is a vector with the same size as the number of neurons in its corresponding hidden layer, each of whose values is independent of each other and has probability p of 1 and probability $1 - p$ of 0. $\tilde{\mathbf{a}}^l$ implies the output after the Dropout layer. The key insight of the MC Dropout is that doing many stochastic forward passes with varying Dropout masks allows us to approximate the posterior. An example is displayed in Fig. 6.7.

Below is the key for implementing the MC Dropout method: Add a Dropout layer with a suitable Dropout rate before each hidden layer, no matter if it is the first or last hidden layer. The work [202] suggests using l_2 regularization when the objective is to enhance uncertainty as it moves away from the data. Furthermore, it is also important that the Dropout layers in a traditional NN are activated during training and deactivated during prediction. However, for a probabilistic

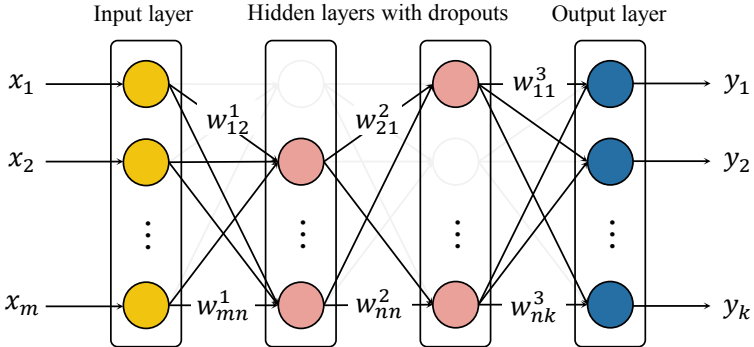


Figure 6.7: Illustration of MC Dropout architecture with Dropout layers.

NN using MC Dropout, the Dropout layers remain active during both the training and prediction phases, which means that each forward pass yields a slightly different prediction. The mean and standard deviation of n times prediction values can be considered as the final point estimation and uncertainty quantification, respectively [165]. They can be calculated as follows:

$$\bar{y}_i^* = \frac{1}{N} \sum_{j=1}^N y_{ij}^*, \quad (6.11)$$

$$\sigma_i^* = \sqrt{\frac{\sum_{j=1}^N (y_{ij}^* - \bar{y}_i^*)^2}{N}}, \quad (6.12)$$

where N is the number of repetitions. y_{ij}^* is the single prediction of the i^{th} sample in the j^{th} repetition. $\bar{y}_i^*$ and σ_i^* are the mean and standard deviation of N predictions for the i^{th} sample, respectively.

NN architecture and training results

In this work, a deep NN with residual blocks [206] is employed for both the classification and regression model. Fig. 6.8 shows the architecture of the NN with two residual blocks. As the depth of the network continues to increase, model training becomes difficult, and the network degrades. Therefore, this issue can be well solved by adding residual blocks [206]. A basic residual block consists of two dense layers and a residual connection. The input and output dimensions of these two layers remain consistent. To achieve uncertainty, each dense layer begins with a Dropout layer. A normalization layer after each dense layer and before the activation function is added to enhance the stability and convergence speed of the NN [161]. All the data from initial datasets are preprocessed using z-score normalization to eliminate the effect of scale between different feature data. The performance of NNs is greatly influenced by the selection of hyperparameters, such as learning rate, the quantity of neurons per dense layer, optimizer, and so on. Furthermore, since Dropout layers and residual blocks are introduced, additional hyperparameters namely Dropout rate and the number of residual blocks, need to be determined before training. To lessen the computational burden, trial and error is performed at the beginning to reduce the range of each hyperparameter. The random search optimization is then executed to determine the final hyperparameter values. A 5-fold cross-validation is utilized to ensure reliability. In addition, the early

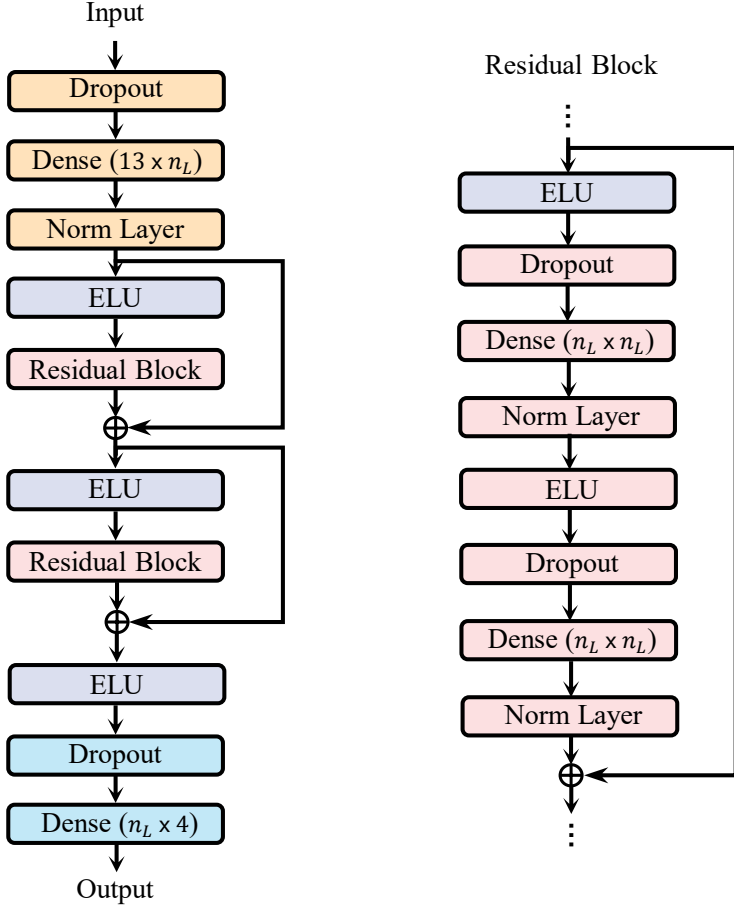


Figure 6.8: MC Dropout neural network architecture with two residual blocks.

stopping method is also applied during the training process to avoid overfitting. The determined hyperparameters are summarized in Table 6.2.

In order to evaluate the prediction accuracy of KPI predictor, the evaluation metrics, namely SMAPE and R^2 are applied, which are introduced in Section 4.3.3.

Table 6.2: Hyperparameter details of the NN with MC Dropout

Hyperparameter	Optimized values	
	Classifier	Predictor
Learning rate	$1.0 \cdot 10^{-2}$	$0.4 \cdot 10^{-2}$
Dropout rate	$1.0 \cdot 10^{-1}$	$0.5 \cdot 10^{-1}$
Activation function	ELU	ELU
Number of neurons	32	106
Number of residual blocks	1	2
Optimizer	Adam	Adam

To assess the degree of uncertainty, the PINAW is employed, which assesses the quality of predictive intervals and is introduced in Section 5.3.3. To evaluate the quality of classification, two evaluation indicators are used: the true positive rate (TPR) and the false positive rate (FPR). They can be expressed by:

$$TPR = \frac{TP}{TP + FN}, \quad (6.13)$$

$$FPR = \frac{FP}{FP + TN}. \quad (6.14)$$

The meaning of true positive (TP), false negative (FN), false positive (FP), and true negative (TN) are demonstrated in Table 6.3. In classification, TPR and FPR characterize the model's discrimination behavior toward positive

Table 6.3: Confusion matrix

		Predicted	
		1	0
Actual	1	TP	FN
	0	FP	TN

and negative samples from different perspectives. TPR, also known as recall, reflects the model's ability to "capture positive samples". A high TPR typically indicates the model's ability to detect as many genuinely important or high-risk samples as possible. FPR reflects the model's tendency to produce false positives. A higher FPR suggests the model is more prone to misclassifying normal samples as positive. These two metrics typically involve a trade-off, which is precisely what the receiver operating characteristic (ROC) curve captures. The ROC curve is essentially derived by continuously adjusting the classification threshold to observe the relationship between TPR and FPR under different discrimination criteria [207]. To compute a ROC curve, a binary classification model that outputs a score for each sample is needed. Next, sort all samples by their model-generated scores from highest to lowest. Treat each of these scores as a classification threshold. For any given threshold, samples with scores greater than or equal to that threshold are classified as positive, while the rest are classified as negative. A ROC curve closer to the upper-left corner indicates that the model achieves a higher TPR while maintaining a low FPR, signifying better performance. In practice, the ROC curve is often used in conjunction with the area under curves (AUC) [208]. It refers to the area under the ROC curve, which summarizes the overall discrimination capability of a binary classification model across all possible decision thresholds using a single scalar value.

After initially training the classifier and predictor, LHS is utilized to create 10000 candidate designs for pseudo-label generation. These 10000 candidates contain only the input design parameters, i.e., they do not have any simulation information. Moreover, they encompass both feasible and infeasible designs. The geometry checker initially eliminates all infeasible designs. Next, the KPI predictor is used to forecast KPIs for the remaining feasible designs. A total of 2852 data points with low uncertainty are obtained after passing through the classifier. The average of the largest and smallest standard deviations in the predictions is taken as the boundary to distinguish between low and high uncertainty. Subsequently, 162 samples are further filtered out using Mahalanobis distance. A total of 2690 samples with valid or invalid pseudo-labels are obtained, which are added to the initial dataset for the classifier. Furthermore, the samples with valid pseudo-labels are then fed to the KPI predictor. Finally, 948 samples with low uncertainty are acquired with KPI pseudo-labels, which are also integrated into the initial dataset for the predictor. The dataset size for each iteration is listed in Table 6.4. The prediction performance of the KPI predictor based on the initial dataset and the expanded dataset after the first iteration is

demonstrated in Fig. 6.9. The commonly used 95 % UI are calculated. It can be seen that the prediction accuracy of the metamodel obtained by training on the extended dataset increases and the uncertainty decreases. The evaluation metrics are shown in Table 6.4. The classification performance of the classifier based on the initial dataset and expanded datasets is demonstrated in Fig. 6.10. As new data are added, the ROC curve of the classifier shifts progressively toward the upper left, indicating an improvement in the model's classification performance. The AUC is also calculated to evaluate the performance of the classifier. In the case of the trained classifier based on the expanded dataset after iteration 1, the AUC is larger, which indicates better classification ability.

Table 6.4: Evaluation summary of KPI predictor based on different datasets

Dataset	KPI	R^2	SMAPE	PINAW
Initial (1555 samples)	o_1	0.971	2.053	0.263
	o_2	0.963	3.209	0.259
	o_3	0.823	10.37	0.344
	o_4	0.585	0.569	0.279
Iteration 1 (2503 samples)	o_1	0.982	1.794	0.209
	o_2	0.971	2.879	0.192
	o_3	0.860	9.645	0.245
	o_4	0.614	0.539	0.226
Iteration 2 (4042 samples)	o_1	0.991	1.605	0.167
	o_2	0.980	2.713	0.157
	o_3	0.876	9.237	0.194
	o_4	0.652	0.537	0.155

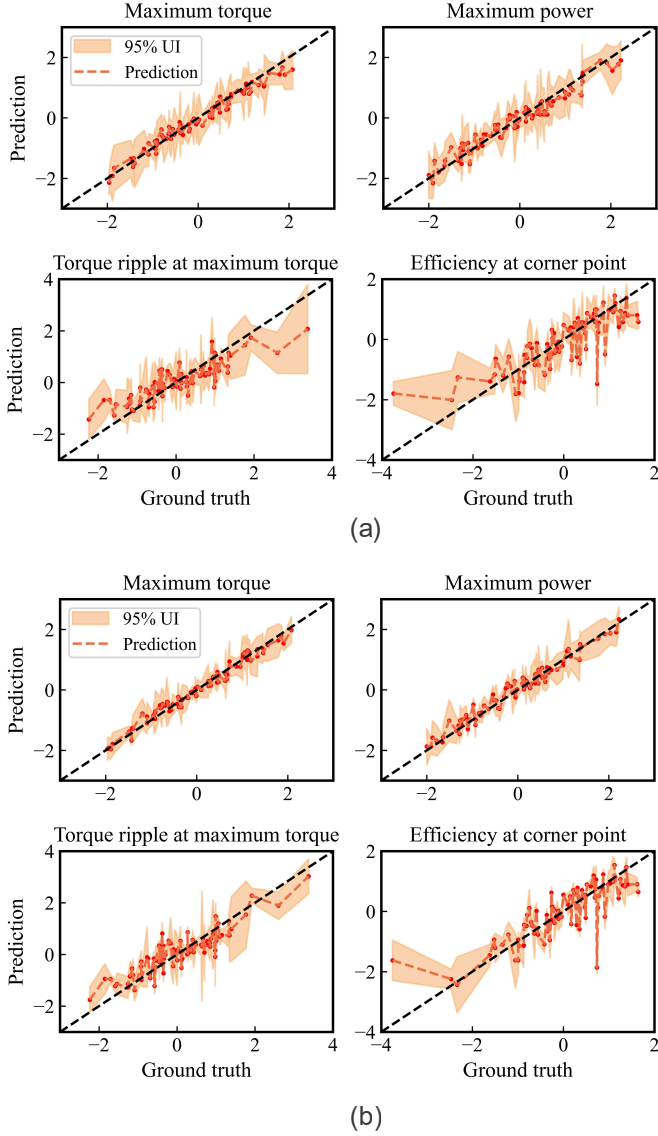


Figure 6.9: Prediction performance of KPI predictor based on initial dataset and expanded dataset after first iteration. (a) Initial dataset. (b) Expanded dataset.

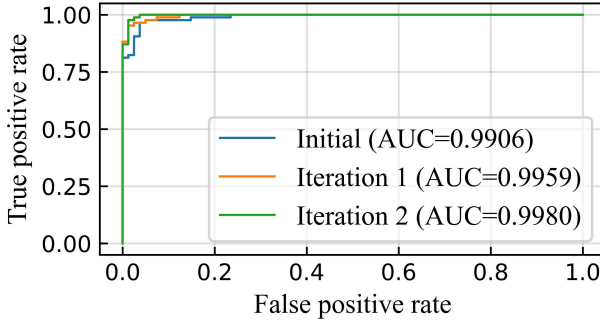


Figure 6.10: ROC curves of geometry classifier.

A second iteration is executed to further confirm the effectiveness of the self-training framework. The whole second iteration procedure mirrors the first iteration. Following the selection algorithm, 4248 samples with low uncertainty are obtained for the expanded classifier dataset. After the KPI prediction phase, 1539 samples with KPIs predicted by the metamodel are added to the dataset. Based on the expanded dataset after the second iteration, the classifier and the predictor are retrained once more. The detailed predictor performance is listed in Table 6.4. As the dataset size rises, the prediction accuracy of the metamodel improves. Furthermore, the associated predictive uncertainty diminishes. The confusion matrix with specific threshold 0.5 is displayed in Fig. 6.11. In this case, the classifier trained based on the dataset obtained after the second iteration has the best classification performance.

6.2.3 Conclusion and limitations

In this work, a self-training framework based on semi-supervised learning is presented to enlarge initial datasets. Two metamodels, namely the geometry classifier and KPI predictor, collaborated to produce pseudo-labels for unlabeled samples. The probabilistic metamodel MC Dropout is implemented to enhance the quality of the pseudo-labeled data. It not only offers point predictions but also provides the uncertainty associated with those predictions. The results show that each iteration of this approach quickly produces pseudo-labeled samples to extend the original dataset. Increasing the dataset size enhances

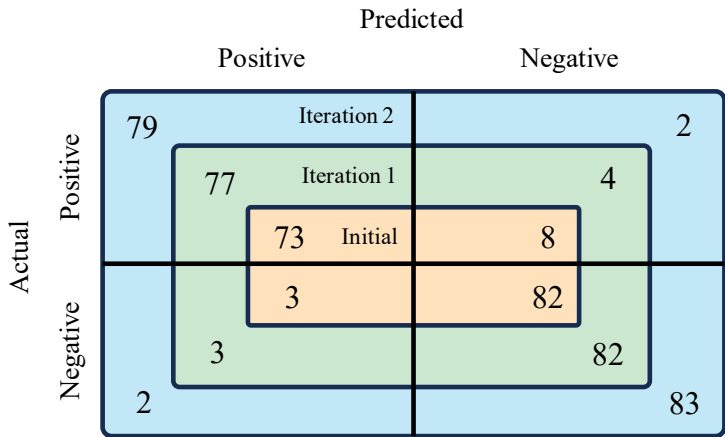


Figure 6.11: Confusion matrix based on initial dataset and expanded datasets with threshold 0.5.

the performance of the classifier and predictor. The uncertainty of the forecast outcomes is also reduced.

However, since the new data are obtained through model predictions, they deviate from the real data. They can contaminate the original dataset. As a result, a two-step selection procedure is designed to eliminate as many unreliable predictions as possible. This process involved initially assessing the predictions based on their uncertainty, followed by applying Mahalanobis distance to remove any remaining outliers. Nevertheless, it is impractical to completely ensure the elimination of all data that drastically deviates from the ground truth. Hence, once the extended dataset reaches a specific threshold of noise, the metamodel's performance will stop increasing and may even decline. Additionally, the original dataset has to fulfill certain requirements, namely, the initially trained metamodels need to possess a certain level of prediction accuracy. If the metamodel is initially inaccurate, the subsequently generated data will progressively diverge more and more from the true values.

6.3 Active learning-based optimization

In Chapter 5, the literature review shows that probabilistic metamodeling has been well studied and widely applied in many research areas. In the field of electromagnetic design optimization, this dissertation pioneers an optimization method based on probabilistic metamodels, incorporating uncertainty information as constraints to yield more reliable solutions throughout the optimization process [O3]. Nevertheless, there are still three major issues that need to be addressed:

1. Which probabilistic metamodels are more appropriate for predicting KPIs and better for optimization? A comprehensive comparative study of various probabilistic metamodels for the design optimization of electric machines has not been published.
2. Since FEA remains necessary throughout the data acquisition phase, large datasets typically come with a massive computing burden. As a result, there is a lack of studies on how to effectively use predictive uncertainty information to shorten the duration of the entire optimization framework.
3. Furthermore, in the case of small datasets, minimizing the error between the final optimization solutions predicted by the probabilistic metamodels and the FEA verification is another challenge that has to be overcome.

To resolve the aforementioned three issues, this section proposes a new optimization framework with an iteratively expanded dataset using probabilistic metamodels to improve the prediction accuracy of optimal solutions [O5]. In the beginning, a relatively small dataset is generated with the help of the automatic simulation toolchain. Based on the initial dataset, a comparative study on six metamodels, including probabilistic models (MC Dropout, deep ensemble (DE), GPR, VI, and Markov chain Monte Carlo (MCMC)), as well as the MLPs, is conducted. They are evaluated in three aspects: prediction accuracy, uncertainty degree, and time consumption. Then, four distinct optimization strategies are introduced in order to improve the accuracy and reliability of the optimal outcomes. The fundamental idea is to continually produce new data through iterative optimization and add them into the original dataset, which is also known as active learning. Active learning is a data-efficient learning method where the most valuable data are chosen and labeled by active selection [209], as shown in Fig. 6.12. The aim is to optimize the performance of a model by utilizing a minimal number of high-quality samples for labeling.

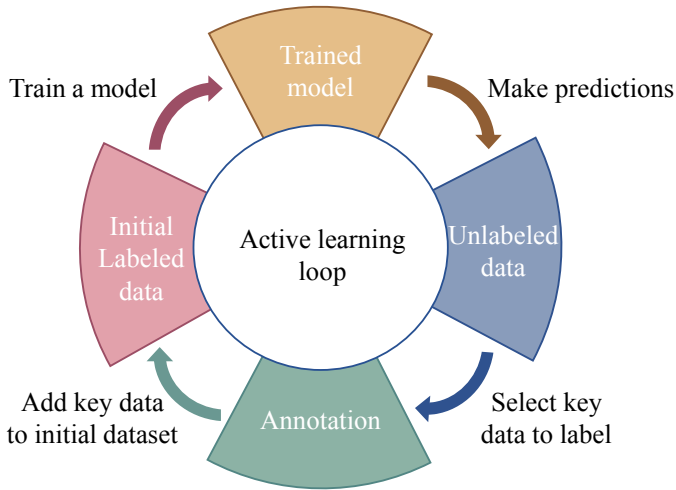


Figure 6.12: Flowchart of active learning process.

6.3.1 Methodology

This section presents the fundamentals of five distinct probabilistic metamodels that will be utilized in the following research.

Monte Carlo Dropout

MC Dropout is a method used in DL to estimate model uncertainty. Its core idea is to extend the Dropout technique, which originally applied only during training, to the testing phase, approximating Bayesian inference through multiple random forward passes. In standard NN training, Dropout randomly “drops” a portion of neurons with a certain probability to reduce overfitting. However, Dropout is typically disabled during testing, transforming the network into a deterministic model. In contrast, MC Dropout remains active during testing, performing multiple forward passes on the same input. Each pass yields slightly different model outputs due to the varying neurons randomly dropped. By statistically analyzing these multiple outputs (e.g., calculating mean and variance), the degree of variation among the outputs allows us to quantify the model’s uncertainty in addition

to obtaining the average forecast as the final output. More details are given in Section 6.2.2.

Deep ensemble

MC Dropout operates by repeatedly performing forward propagation with the same input. Additionally, each forward pass corresponds to a slightly different network structure because of the dropout. In a similar way, MC dropout could also be viewed as an ensemble of multiple NN, which are suggested with distinctly initialized weights but the same structure to approximate the Bayesian inference [210]. Fig. 6.13 demonstrates the workflow of DE. This method involves training multiple NN independently and then combining their predictions to improve the overall performance of the model and obtain predictive uncertainty. For a same input sample x_i , t NN with different initial weights need to be trained. Then, the mean prediction $\bar{y}_i^*$ and corresponding standard deviation σ_i^* can be calculated using (6.11) and (6.12) based on t prediction results.

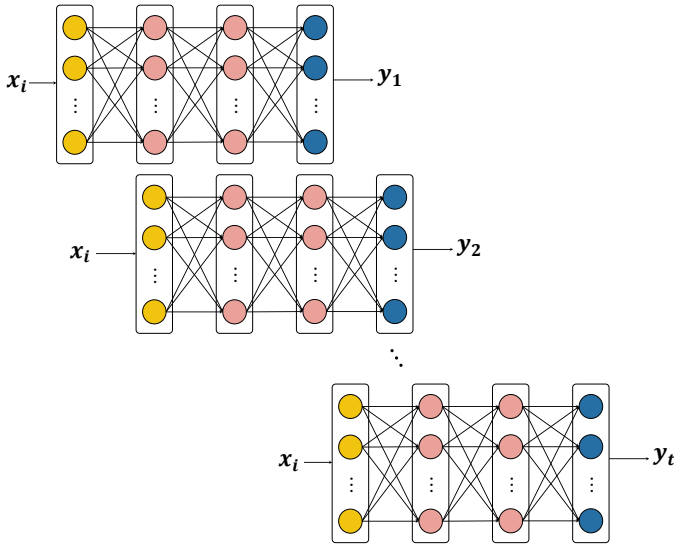


Figure 6.13: Illustration of DE with multiple inference processes.

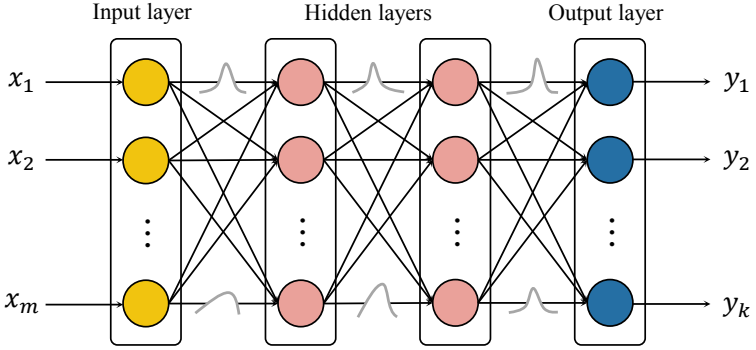


Figure 6.14: Illustration of BNN with probability distributed weights.

Variational inference

Variational inference (VI) converts a Bayesian inference problem into an optimization problem. It attempts to approximate the posterior distribution of BNN's weights. In BNNs, instead of having a single deterministic set of weights, the weights are set as random variables with a distribution that reflects uncertainty about their values, as displayed in Fig. With the help of VI, a solvable optimization problem without $p(\mathcal{D})$ can be obtained. The only task is to handle the prior distribution $p(\mathbf{w})$ and the likelihood $p(\mathcal{D}|\mathbf{w})$. Consequently, the inference problem is simplified to a manageable optimization task. More details can be found in Section 5.1.2.

Gaussian process regression

As introduced in Section 3.1.2, GPR is a nonparametric regression method based on Bayesian theory that performs prediction by defining probability distributions over the function space. Unlike traditional regression methods that directly estimate model parameters, GPR assumes the target function itself follows a Gaussian process. This means that at any finite set of input points, the function values obey a joint Gaussian distribution. This Gaussian process is fully characterized by a mean function and a covariance function, where the mean function describes the overall trend of the function, while the covariance function captures the correlation and similarity between different input points. By selecting

an appropriate kernel function, assumptions about the function's smoothness, periodicity, or other structural features from the prior can be incorporated into the model. Given training data, GPR employs Bayesian inference to obtain the posterior distribution of the target function. This enables it to provide not only a predicted mean as the regression result for new input points but also a predicted variance to measure uncertainty.

Markov chain Monte Carlo

Another powerful approach for probabilistic predictions is the Markov chain Monte Carlo (MCMC). The basic MC method relies on repeated random sampling to obtain numerical results. A Markov chain is a sequence of random variables where the probability of each variable depends only on the state of the previous variable. Combining these methods with other algorithms, MCMC enables an approximation of the desired posterior distribution of model parameters given data [211]. Fig. 6.15 describes the workflow of MCMC. An initial sample is generated in the state space. It moves to a new state in the next step based on a transition rule that depends only on the current state. Subsequently, a decision is made regarding the acceptance or rejection of the new state, taking into consideration a criterion that frequently involves the desired distribution. The transition and acceptance processes are iteratively performed, resulting in a chain of samples that eventually resembles the posterior distribution. A commonly used MCMC method is the Metropolis-Hastings algorithm [213]. Another improved algorithm, namely Gibbs sampling, is more efficient due to directly sampling new states from their conditional distributions [214]. This eliminates the requirement for the rejection step. In this work, an advanced MCMC algorithm known as no-u-turn sampler is employed due to its automatic parameter tuning and faster convergence. The no-u-turn sampler is capable of effectively managing high-dimensional spaces and intricate correlations by considering the joint distribution of all variables. In addition, it can reduce autocorrelation across samples by making larger, informed moves throughout the state space [215].

6.3.2 Dataset acquisition

The electric machine template used in the following work is a PMSM, as displayed in Fig. 6.16. A total of 12 input design parameters are defined that can characterize the machine template decently. Four output KPIs are selected as prediction targets. Table 6.5 summarizes more details of the parameter definition. Due to the high dimensionality of the inputs and the broad range

of each input, LHS is applied to generate a widely covered design space. Geometrically infeasible designs are eliminated, and the KPIs are computed through the automatic simulation toolchain. At the end, a preliminary dataset with 500 samples is collected.

To ensure the dependability of the dataset, the Pearson correlation coefficient between each of two parameters is calculated, as shown in Fig. 6.17. The Pearson coefficient quantifies the strength and direction of the linear relationship between two variables. In general, there is a strong correlation between the design parameter and the KPI, but a weak correlation between the design parameters themselves. The high coefficient between design parameters d_3 and d_5 is noteworthy, since it arises from the inherent geometric limitations between the inclination angle of the V-magnet and the length of the bar magnet.

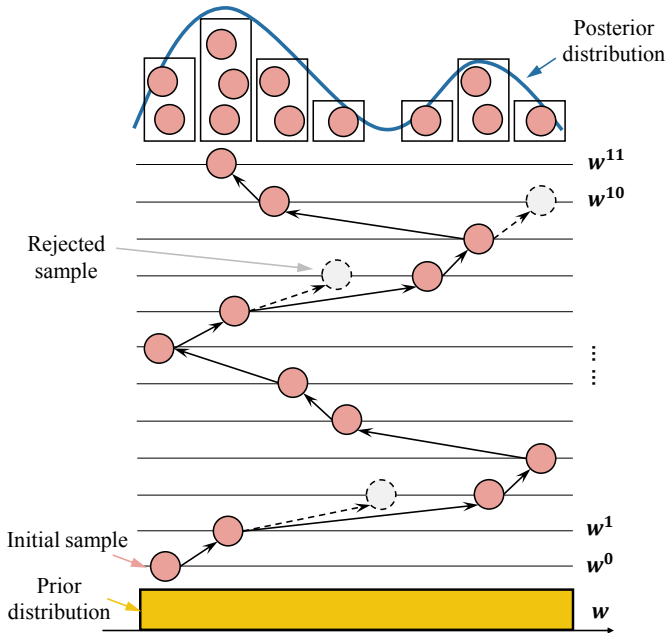


Figure 6.15: Illustration of MCMC's workflow adapted from [212].

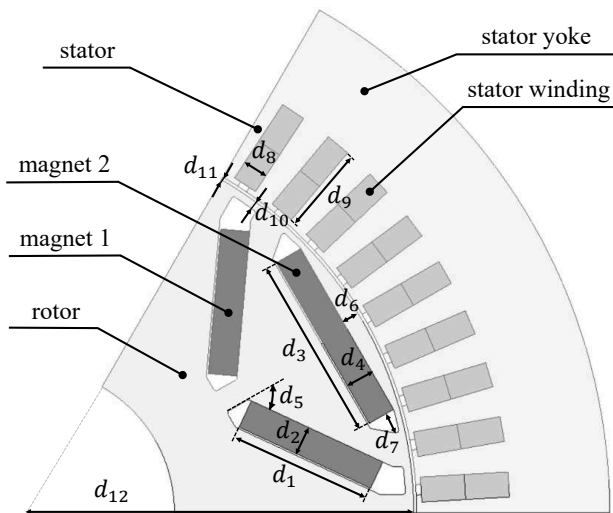


Figure 6.16: Geometry of the reference PMSM.

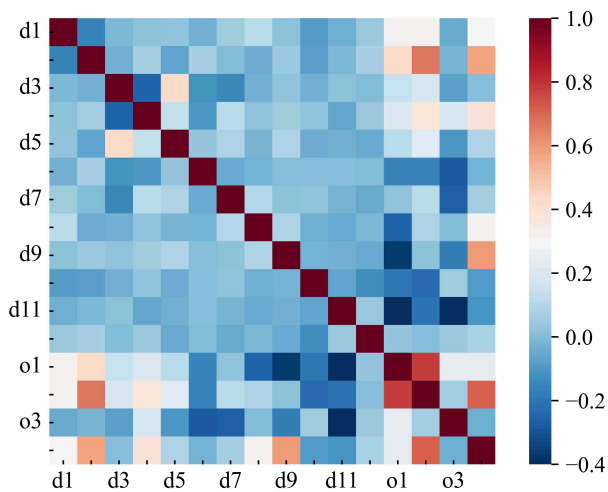


Figure 6.17: Correlation map of varied input and output parameters.

Table 6.5: Details of reference machine template parameters

Symbol	Design parameter	Min.	Max.	Unit
d_1	Magnet 1 width	15.0	28.0	mm
d_2	Magnet 1 height	2.0	8.0	mm
d_3	Magnet 2 width	20.0	40.0	mm
d_4	Magnet 2 height	2.0	8.0	mm
d_5	Magnet inclination angle	30.0	56.0	deg
d_6	Width between magnet 2 and air gap	1.0	3.0	mm
d_7	Flux barrier width	1.0	6.0	mm
d_8	Slot width	3.5	5.0	mm
d_9	Slot height	10.0	20.0	mm
d_{10}	Width between flux barrier and air gap	1.0	6.0	mm
d_{11}	Air gap	0.5	2.0	mm
d_{12}	Rotor outer diameter	130.0	140.0	mm
KPI				
o_1	Maximum torque			N·m
o_2	Maximum power			kW
o_3	Torque ripple at maximum torque			N·m
o_4	Material cost			Euro

6.3.3 Comparative study of metamodels

In this section, the prediction performance of six different metamodels are compared, namely, MLP, MC Dropout, DE, VI, GPR, MCMC.

Hyperparameter optimization

The dataset is first split into two portions, with 10 % for testing and 90 % for training. Each metamodel is trained to predict one KPI, aiming to improve prediction accuracy and minimize model complexity. To determine optimal hyperparameters for each metamodel, the range of each hyperparameter by trial and error is reduced, e.g., the learning rate range of the MLP for predicting maximum torque is narrowed to 0.001-0.005 instead of 0.001-0.1. Subsequently, a 9-fold cross-validation is conducted, where the training set is further partitioned into

nine parts, with eight parts used for training and one part used for validation, resulting in a total of nine cycles. The 9-fold division ensures the same amount of validation and test data. Moreover, the random search is used to find optimal hyperparameters to minimize the 9-fold cross validation error. Table 6.6 displays more details of the general hyperparameters for each metamodel with different KPI predictions. As MC Dropout, DE, and VI all rely on the NN structure, the identical network structure parameters are utilized as MLP, including 3, 3, 4, and 2 hidden layers and 72, 64, 30, and 98 neurons for o_1 , o_2 , o_3 , and o_4 , respectively. In general, MCMC and ANNs are two different methods in their foundations and purposes. However, advanced methods can implement ANNs in an MCMC framework to estimate complex probability distributions [216]. In this work, an optimized MLP structure is also adopted in MCMC. Since both VI and MCMC involve BNN inference, it is crucial to set up prior distributions for the weights and biases of the networks in advance. Nevertheless, accurately de-

Table 6.6: Details of hyperparameters for each metamodel in form of $o_1 \mid o_2 \mid o_3 \mid o_4$

	Number of epochs	Learning rate
MLP	110 125 50 70	0.003 0.003 0.0008 0.002
MC Dropout	70 80 40 85	0.005 0.003 0.001 0.002
DE	-	0.003 0.003 0.0008 0.002
VI	4500 4000 2000 3000	0.012 0.01 0.011 0.01
GPR	40 130 50 190	0.4 0.8 0.5 0.5
MCMC	-	-
	Activation function	Repetition
MLP	ELU ELU ReLU ReLU	-
MC Dropout	ELU ELU ELU ReLU	10
DE	ELU ELU ReLU ReLU	40
VI	ELU ELU ELU ReLU	50
GPR	-	-
MCMC	ELU ELU ReLU ReLU	-

terminating them is an issue due to their absence of physical relevance. To address this issue in practice, the standard Gaussian distribution is employed for every weight and bias, namely $p(\mathbf{w}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ [181]. Furthermore, two key hyperparameters of MCMC are specified: the number of samples (optimized results: 1000, 1000, 1200, and 800 for o_1 , o_2 , o_3 , and o_4 , respectively) and the number of warm-up steps (optimized results: 200, 200, 300, and 100 for o_1 , o_2 , o_3 , and o_4 , respectively). The warm-up phase is used to adjust the parameters and ensure the chain has reached the stationary distribution. Following the warm-up phase, the number of samples corresponds to the desired quantity of samples to be extracted from the posterior distribution. As the number of samples increases, the approximation of the posterior distribution becomes more precise. Yet, an increased number of samples necessitates more computing resources. For GPR, the selection of the kernel function is directly related to the performance of the model. Hence, the kernel functions with different length scales are optimized (30, 1, 1, and 3 for o_1 , o_2 , o_3 , and o_4 , respectively). The result indicates that the Matern kernel (o_1 and o_4), rational quadratic kernel (o_2), and radial basis function kernel (o_3) show superiority in different tasks.

Prediction results and analysis

Fig. 6.18 exhibits the test prediction results of MLP. It can be seen that the MLP possesses high prediction accuracy in predicting the maximum torque, maximum power, and cost. But there is a large error in the forecast of torque ripple at maximum torque. One possible reason is the insufficiency of training data. The prediction performance of GPR is demonstrated in Fig. 6.19. GPR exhibits considerable predictive uncertainty when forecasting torque ripple. Predictions of maximum power and material cost show high accuracy. The prediction error for maximum torque is relatively larger, with a slightly higher degree of uncertainty. Table 6.7 summarizes the evaluation details of various metamodels. It can be seen that MCMC exhibited superior performance in terms of both accuracy in point prediction and predictive uncertainty, as reflected by the markings in bold and shown in Fig. 6.20. However, the computational time required for the MCMC training was almost a thousand times longer compared to other metamodels. GPR provided great computational efficiency, requiring only a few seconds to train a model. Furthermore, it surpassed other metamodels except MCMC in both prediction accuracy and uncertainty degree.

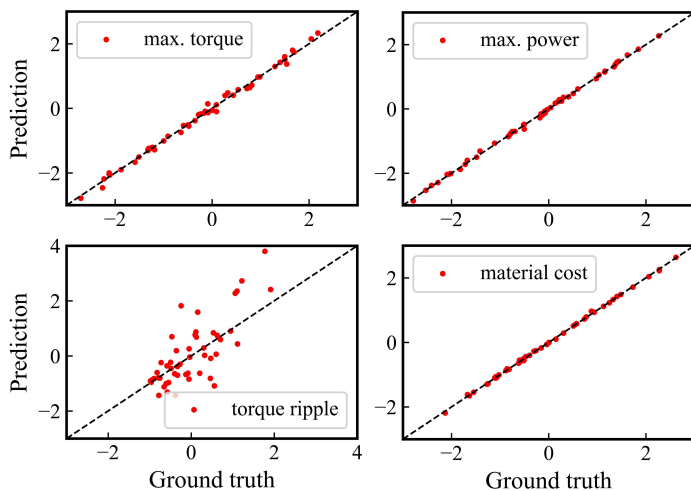


Figure 6.18: KPI prediction performance of MLP over test set.

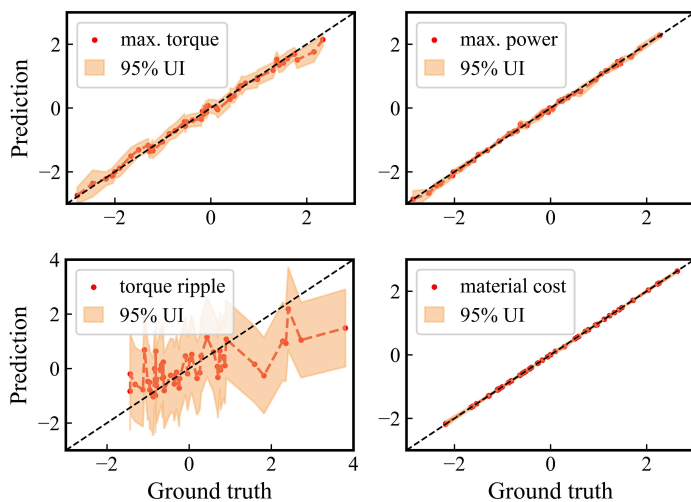


Figure 6.19: KPI prediction performance with uncertainty intervals of GPR over test set.

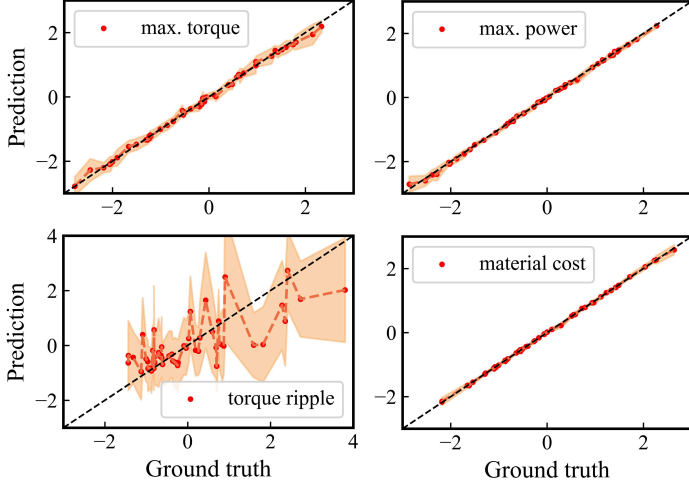


Figure 6.20: KPI prediction performance with uncertainty intervals of MCMC over test set.

6.3.4 Active learning-based optimization and results

In this section, four different optimization strategies based on active learning are proposed. Their corresponding optimization results are also presented and analyzed.

Optimization problem and strategies

A MOO in terms of maximum torque and maximum power is performed. The optimization problem is defined by

$$\max \rightarrow o_1, \quad \max \rightarrow o_2 \quad (6.15a)$$

$$\text{s.t. } d_i^L \leq d_i \leq d_i^U, \quad i = 1, \dots, 10 \quad (6.15b)$$

$$h_j \leq 0, \quad j = 1, \dots, n_{cons} \quad (6.15c)$$

where d^L and d^U are the predefined lower and upper bounds of design parameters. h_j reflects the constraints during optimization. The NSGA-II is adopted

Table 6.7: KPIs evaluation of six metamodels in form of $o_1 \mid o_2 \mid o_3 \mid o_4$

	MLP				MC Dropout			
R^2	0.9910	0.9986	0.1785	0.9993	0.9903	0.9951	0.1563	0.9985
SMAPE	1.13 %	0.58 %	35.94 %	0.19 %	1.26 %	1.00 %	36.69 %	0.26 %
PINAW	-				0.09	0.07	0.16	0.06
Training time	8 s	10 s	5 s	12 s	8 s	11 s	4 s	13 s
	DE				VI			
R^2	0.9890	0.9984	0.1701	0.9987	0.9869	0.9957	0.0410	0.9935
SMAPE	1.38 %	0.64 %	36.63 %	0.28 %	1.51 %	1.02 %	40.98 %	0.51 %
PINAW	0.08	0.04	0.16	0.04	0.08	0.05	1.06	0.05
Training time	80 s	100 s	50 s	120 s	104 s	87 s	73 s	66 s
	GPR				MCMC			
R^2	0.9903	0.9987	0.991	0.9993	0.9966	0.9991	0.2650	0.9997
SMAPE	1.16 %	0.55 %	36.52 %	0.04 %	0.72 %	0.48 %	37.85 %	0.13 %
PINAW	0.09	0.03	0.87	0.02	0.07	0.03	0.60	0.02
Training time	2 s	5 s	4 s	13 s	116 min	110 min	133 min	88 min

because of its computational efficiency and accuracy in solving multiobjective problems. MOO hyperparameters are set empirically, such as the number of generations (1000), population per generation (100), and mutation probability (0.9). As stated in Section 6.3, the aim is to decrease the time required for the entire set of MOO and improve the accuracy of the final predicted solutions compared to the FEA verification. To achieve the goals, four MOO strategies are proposed to compare their results, as illustrated in Fig. 6.21. These strategies are based on the idea of active learning, where the model actively selects the most informative data points to be labeled by a human annotator to improve its learning efficiency. The annotation in this study is realized by FEA. Active learning is especially beneficial in situations when acquiring labeled data is costly.

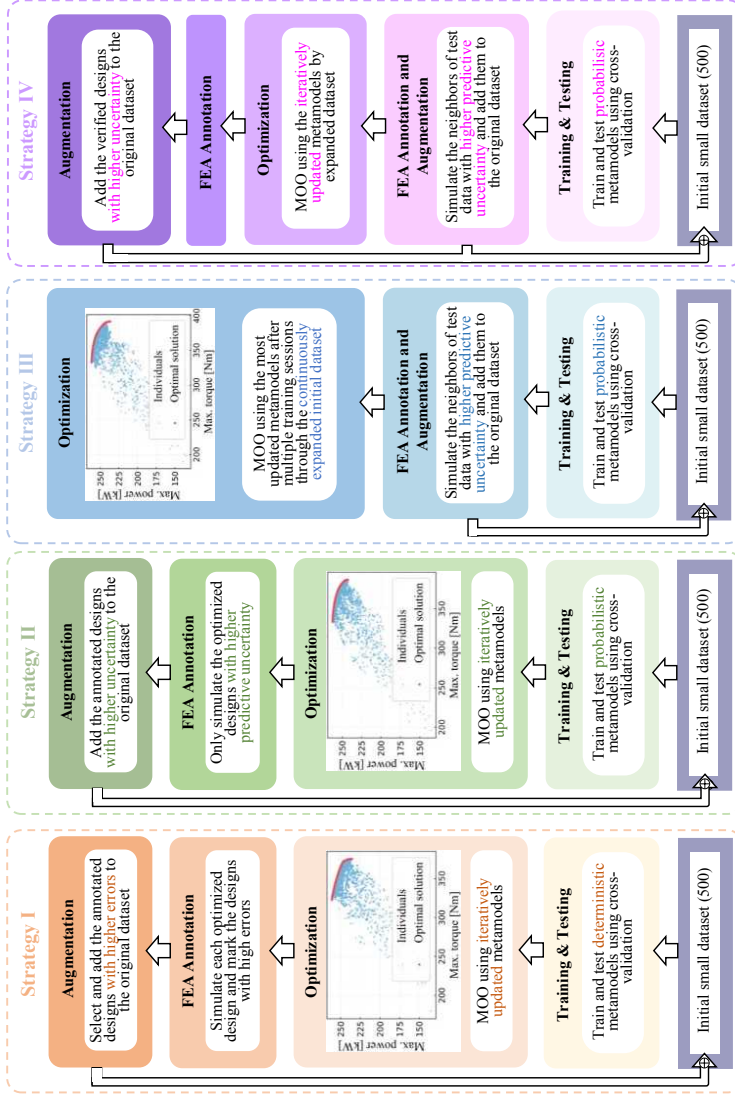


Figure 6.21: The overall framework based on active learning for electric machine optimization.

By picking the most informative data, the model can achieve improved performance with a few labeled instances. Therefore, the number of key data selected for each iteration is a critical parameter that must be determined in advance. A number of n key data with the highest uncertainty are simulated using FEA. The key data with simulated KPIs are added into the initial dataset. The expanded dataset is utilized to retrain metamodels iteratively to obtain improved prediction ability and repeated the same procedure. As n increases, the duration of the annotation and the required number of iterations increase. However, the key data acquired becomes more beneficial for further training. Therefore, a comparative study for different numbers of key data is carried out, as shown in Fig. 6.22. First, for each iteration, selecting only 10 key data points, a total of eight iterations are completed, with the final model training results and time consumption recorded. Subsequently, for each iteration, selecting only 20 key data points, a total of four iterations are completed, as this approach maintains a consistent total number of selected key data points. Similarly, for each iteration, 30 key data are selected, with a total of three iterations; for each set of 40, 50, and 60 key data, there are a total of two iterations. It can be seen that the lowest prediction errors are obtained by adding 10 key data per iteration for a total of 8 iterations and by adding 60 key data per iteration for a total of 2 iterations. This also indicates that they need the greatest amount of time. However, the time required for adding 40 key data each iteration across two iterations is the least, although it is related to the highest prediction errors. Overall, 40 key data points are chosen eventually for each iteration as the standard. More details of the proposed strategies are described below:

1. *Strategy I*: At the beginning (iteration 0), the initial dataset with 500 samples is utilized to train two deterministic MLPs for o_1 and o_2 . Following that, the machine design optimization is carried out with the assistance of the trained MLPs. There are 100 solutions in the final Pareto front. Every solution undergoes FEA annotation, which also means labeling, to determine the absolute error between simulation and prediction. To ensure a small selection of key data while still being sufficient for the models to learn new features, the 40 labeled solutions with the highest errors are chosen and added to the initial dataset. In iteration 1, MLPs are retrained (updated) on the augmented dataset with 540 samples to achieve better prediction performance. Then, the updated MLPs are again employed in the optimization process. After obtaining a new set of 100 optimized designs, FEA annotation is again conducted. Likewise, an additional 40 key samples with the highest errors are included in the dataset, and so forth for the remaining iterations.

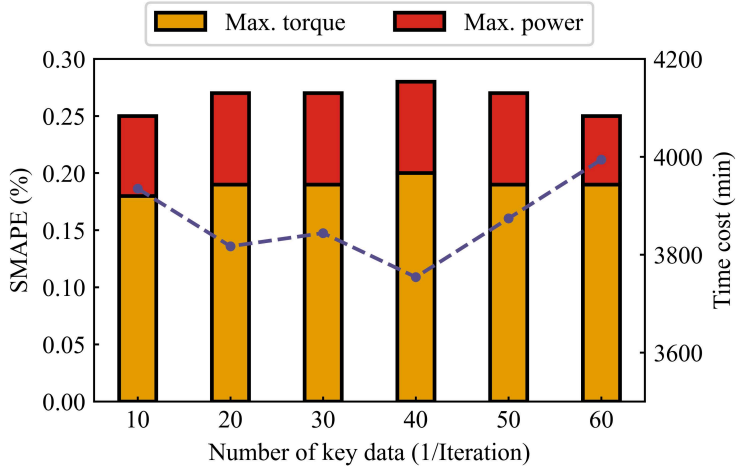


Figure 6.22: Comparison of computation time and KPI prediction errors for different numbers of key data.

2. *Strategy II*: The model training followed Strategy I, with the sole exception that the probabilistic metamodels are trained. After optimization, the proposed approach is modified by verifying only 40 key predicted designs that exhibit the highest degree of uncertainty, as opposed to all designs as done in Strategy I. This step enables a more efficient annotation process. The same procedure is repeated as above for the remaining iterations.
3. *Strategy III*: In this case, the probabilistic metamodels are trained as in Strategy II. In the next step, 20 test data samples with the highest predictive uncertainty are selected. For each of the sample points, two neighbors are simulated. 40 new simulated data samples are then added to the initial dataset. After several iterations, the last trained metamodel is developed in the optimization algorithm and the final Pareto front is achieved. Here, the post-sampling loop is brought forward to the testing session, no longer after each optimization.
4. *Strategy IV*: It is a combination of Strategies II and III. In each iteration, 20 key data from testing and 20 key data from optimization depending on the

level of uncertainty are verified via FEA. At the end of each iteration, key data are added to the initial dataset.

In the experiment, a total of two active learning iterations are predefined, and 40 key data samples are added to the initial dataset during each iteration. As a result, the dataset is expanded to 580 samples and used to update metamodels in iteration 2. Theoretically, the dataset could be expanded through an infinite number of iterations. But this would result in an increase in time consumption. This section mainly focuses on the effectiveness of different strategies. The trade-off issues of how many iterations are most suitable and how many key data to select for annotation in each iteration will be introduced in the next section. For Strategy I, MLPs are implemented as metamodels. GPR is adopted for Strategies II, III, and IV because of its overall great abilities in terms of prediction accuracy, predictive uncertainty, and time consumption.

Optimization results and analysis

The iteratively predicted optimization results based on Strategy I are illustrated in Fig. 6.23. The Pareto fronts obtained by model prediction are in red, while the actual Pareto fronts verified using FEA are in green. The blue line delineates the absolute error between the prediction and FEA. Compared to the FEA verification, the optimal Pareto front predicted by the metamodels trained on the initial dataset has a large error, particularly in the frontal and rear sections of the Pareto front. In iteration 1, the predicted solutions in the head of the Pareto front receive a significant reduction in error, whereas the tail still exhibits a large error. The overall optimized designs yield the maximum prediction accuracy after the second iteration. Fig. 6.23 demonstrates that the accuracy of the optimization results has continuously improved with the addition of a small volume of key data after each iteration.

Regarding Strategy II, the initial optimization results from GPR are generally overpredicted. After iteration 0, 40 key data are chosen and added to the initial dataset. With this expanded dataset, the metamodel is retrained and used in the optimization process. It can be seen that the overall error is significantly reduced, especially in the low torque and high power area. There is still a relatively high error in the high torque area. After two iterations of model retraining, the estimated Pareto front is closely aligned with the FEA validation, as shown in Fig. 6.24. Furthermore, the predictive uncertainty decreases as the iterations progress.

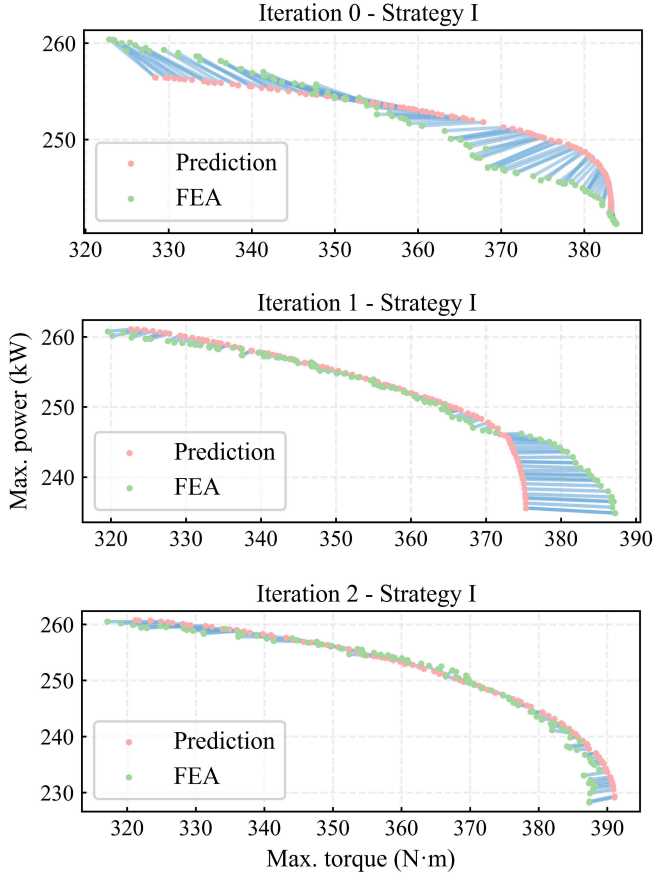


Figure 6.23: Predicted Pareto fronts using Strategy I compared with FEA verification.

The result is achieved according to Strategy III, as seen in Fig. 6.25. The initial dataset is expanded iteratively by simulating crucial data based on the test set's prediction outcomes, and the optimization procedure is carried out only during the final iteration. It can be observed that despite two rounds of data expansion, GPR still overestimate the KPIs for each optimized design. The uncertainty

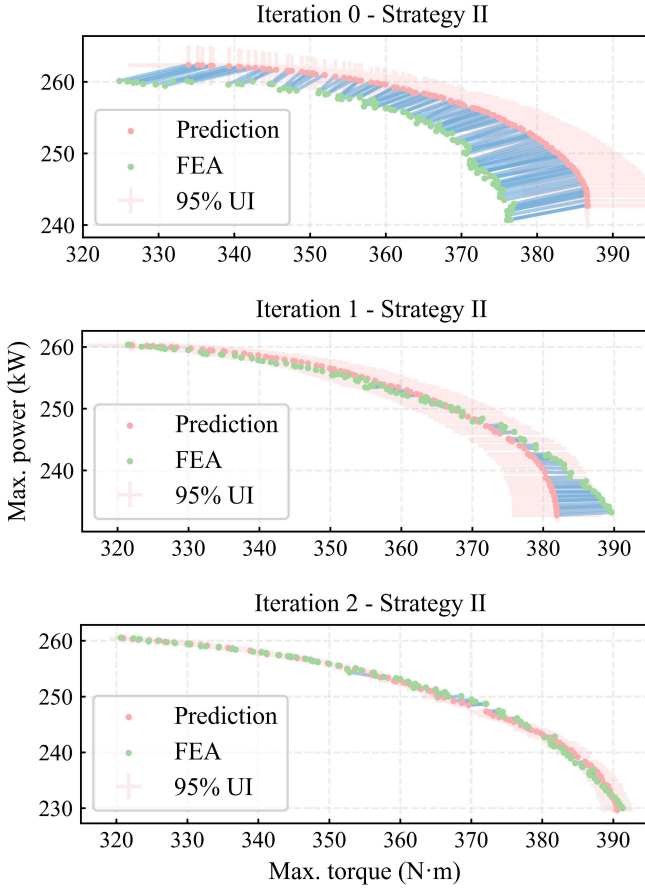


Figure 6.24: Predicted Pareto fronts using Strategy II compared with FEA verification.

of the prediction is also not very clearly reduced. One possible reason is that the acquired key data based on the test set are global rather than local optimal regions. In Strategies I and II, the selection of key data is dependent on the optimization findings, ensuring that the key data are always inside the optimal region. Within each iteration, the metamodel's prediction capacity for the

points in the optimal zone continues to strengthen, resulting in iteratively more accurate optimal designs.

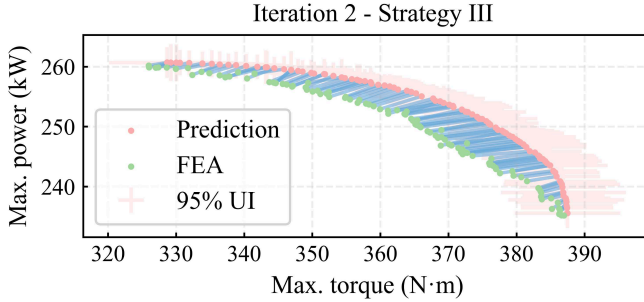


Figure 6.25: Predicted Pareto fronts using Strategy III compared with FEA verification.

Fig. 6.26 displays the optimization results obtained using Strategy IV. In this case, 50 % of the key data are derived from the simulated samples around test set, while the remaining 50 % are obtained from the optimal region. After the first round of retraining, the model's predictive capability is comprehensively corrected, particularly in the high-torque, low-power region. Following two rounds of iteration, the final model achieves high predictive accuracy, with the vast majority of predictions aligning with FEA results. Only a slight discrepancy persists at the very end.

Table 6.8 lists the FEA verification of each iteration's optimized solutions. It can be seen that Strategy IV greatly decreases the prediction error of the final optimization results after including certain key data from the optimal zone. Strategy II exhibits the most accurate design optimization regarding maximum torque and maximum power, with SMAPE of 0.20 % and 0.08 %, respectively. In addition, to compare the proposed strategies, a base group is introduced, i.e., 580 data obtained directly through sampling. The results show that the SMAPE with the initial 580 samples is lower than the other strategies at iteration 0, as described in w/o Strategy in Table 6.8. But it is inferior to the final performance of the suggested strategies. Since Strategy I involves deterministic metamodels, it is necessary to verify all optimal solutions through simulation to quantify the absolute errors. In contrast, probabilistic

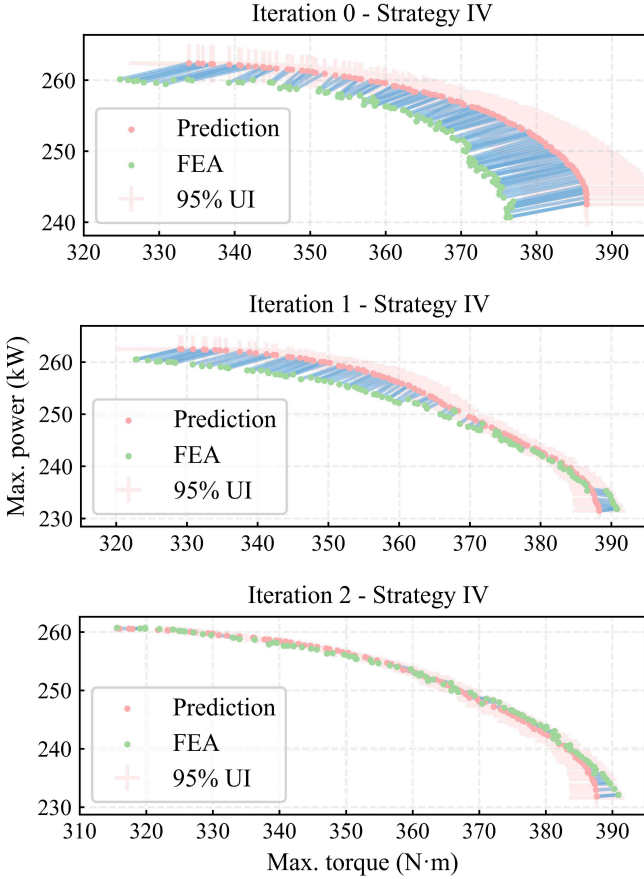


Figure 6.26: Predicted Pareto fronts using Strategy IV compared with FEA verification.

metamodeling allows us to make point predictions and determine the level of uncertainty associated with each forecast. Which solutions can be selected based on the uncertainty would serve as key data for annotation. To better understand the distinction between picking key data based on absolute error and uncertainty, the chosen data for each iteration of Strategy I and Strategy II are plotted in Fig. 6.27. Strategy I primarily focuses on choosing designs with

Table 6.8: SMAPE between FEA verification and optimal solutions in form of $o_1 \mid o_2$

	Iter. 0	Iter. 1	Iter. 2
Strategy I	1.60 % $\mid$ 0.83 %	0.83 % $\mid$ 0.14 %	0.73 % $\mid$ 0.12 %
Strategy II	1.87 % $\mid$ 0.85 %	0.50 % $\mid$ 0.12 %	0.20 % $\mid$ 0.08 %
Strategy III	-	-	1.51 % $\mid$ 0.55 %
Strategy IV	1.87 % $\mid$ 0.85 %	0.77 % $\mid$ 0.56 %	0.30 % $\mid$ 0.11 %
w/o Strategy	1.56 % $\mid$ 0.66 %	-	-

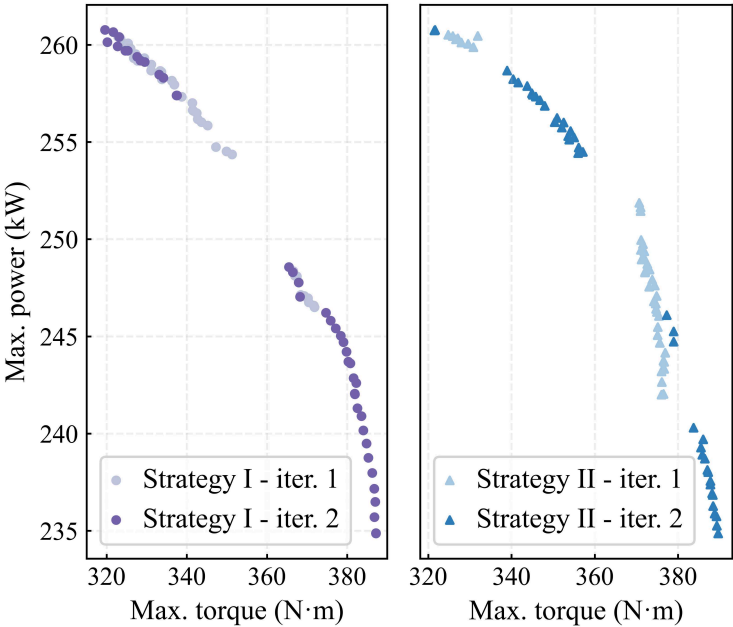


Figure 6.27: The chosen key data of each iteration regarding Strategies I and II.

high power and low torque while only including a limited number of designs with middle power and torque (light purple). Strategy II selects a few solutions with high power and low torque while picking many solutions with moderate KPIs (light blue). In the second round, Strategy I shifts to the designs with high torque and low power (dark purple), whereas Strategy II concentrates on the machines that featured either high torque or high power (dark blue). Therefore, it can be inferred that the selected strategy greatly impacts the distribution of the selected key data, subsequently influencing the retraining of the metamodels.

This research considers the time consumption differences between various strategies as an important aspect. Table 6.9 summarizes the time required for each step in each iteration. Each iteration comprises three processes: 1) Metamodel training (two metamodels for predicting maximum torque and maximum power, respectively) combined with hyperparameter optimization; 2) Machine design optimization; 3) Selection of key data points, simulation via FEA, and addition to the dataset. In the toolchain, the average simulation time for each design is 6 minutes. It is evident that the collection of initial data and the annotation of key data are the two most time-consuming parts, as they involve simulation. Strategy I consumes the longest time, due to the fact that FEA is performed for each optimal design. Strategies II, III, and IV require approximately the same amount of time, resulting in a time savings of around 23 % compared to Strategy I. Despite Strategy II requiring almost 1 hour more computing time than without strategy, it leads to significant improvements of around 82 % and 89 % for o_1 and o_2 in the prediction accuracy of the optimization results.

It should be remarked that in this study only one type of motor template is discussed, i.e., PMSM with one V-magnet and one bar magnet, as shown in Fig. 6.16. Based on this template, Strategy II exhibits superior performance in terms of prediction accuracy and temporal efficiency. In fact, there is no guarantee that Strategy II yields the best outcomes for all machine templates. The main goal of this work is to extend the dataset's size by a data-efficient learning method, namely, active learning. Of the four proposed strategies, the first one uses the conventional absolute error as an indication, although it requires the most time. The other three strategies use uncertainty as a criterion for identifying key data, therefore saving time significantly. All four strategies are based on the idea of data augmentation to iteratively improve the optimization results. As to which of these strategies is more effective for which machine template, it depends on how to balance the accuracy of the final prediction results and the total time cost.

Table 6.9: Time balance of different optimization strategies

	Strategy I	Strategy II	Strategy III	Strategy IV	w/o Strategy
Initial dataset	500 x 6 min	500 x 6 min	500 x 6 min	500 x 6 min	580 x 6 min
Iteration 0: Training with HO	237 min + 296 min	60 min + 143 min	60 min + 143 min	60 min + 143 min	65 min + 151 min
FEA annotation after testing	-	-	40 x 6 min	20 x 6 min	-
Optimization	9 min	9 min	9 min	9 min	9 min
FEA annotation after optimization	100 x 6 min	40 x 6 min	-	20 x 6 min	-
Iteration 1: Training with HO	24 min + 31 min	7 min + 15 min	8 min + 16 min	7 min + 14 min	-
FEA annotation after testing	-	-	40 x 6 min	20 x 6 min	-
Optimization	9 min	9 min	9 min	9 min	-
FEA annotation after optimization	100 x 6 min	40 x 6 min	-	20 x 6 min	-
Iteration 2: Training with HO	23 min + 29 min	7 min + 15 min	7 min + 15 min	7 min + 15 min	-
FEA annotation after testing	-	-	-	-	-
Optimization	9 min	9 min	9 min	9 min	-
FEA annotation after optimization	-	-	-	-	-
Total	81 h 07 min	62 h 34 min	62 h 36 min	62 h 45 min	61 h 33 min

6.3.5 Investigation of optimal dataset sizes

In the previous section, four active learning-based optimization strategies (Fig. 6.21) are explored and compared, with Strategy II yielding the final solutions exhibiting the lowest error relative to the FEA. Its principle involves filtering a certain number of key data based on predicted uncertainty, then adding them to the initial dataset and retraining the model to achieve higher prediction accuracy. In this context, a dataset of 500 samples is used, with 40 key data points selected for each iteration. The question arises: if the initial sample size is further reduced, can reliable predictions still be obtained? Therefore, the benchmark is established using the case in the previous section, namely, 500 initial data points plus two iterations, resulting in a final dataset of 580 samples.

In this section, the same machine template and parameter setting as Table 6.5 remain. Another four independent small datasets, including 100, 200, 300, and 400 samples, are collected. The focus of the investigation is how many initial data points combined with how many iterations can achieve or exceed the benchmark results [O6]. Is it possible to achieve highly accurate optimization solutions even with limited datasets, while significantly reducing overall time costs?

First, the training process remains consistent with the previous section, including the same dataset partitioning and hyperparameter optimization. Next, follow the process of optimization strategy II, metamodels are trained using five different sizes of datasets, respectively, as shown in Fig. 6.28. Table 6.10 summarizes the test results based on the initial datasets. It can be seen that the prediction accuracy is strengthened by increasing the size of the initial dataset. This is because the model can learn from more diverse samples, thus improving its generalization ability instead of memorizing. In addition, the uncertainty of the prediction correlates with the size of the dataset. An increase in data quantity results in a reduction of prediction uncertainty. Then, the MOO is conducted to obtain optimization solutions. A total of 1000 generations and a population size of 100 are established as in the previous section. The iterative optimization results of the benchmark dataset 500 are shown in Fig. 6.24. Theoretically, data augmentation can be executed for countless iterations, but it also implies longer time consumption. Hence, the results of 580 samples are treated as the benchmark, and iterations on other datasets are stopped until their results exceed the benchmark. Table 6.11 summarizes the SMAPE between predictions and FEA verifications based on all datasets after each iteration. The dataset 100 attains

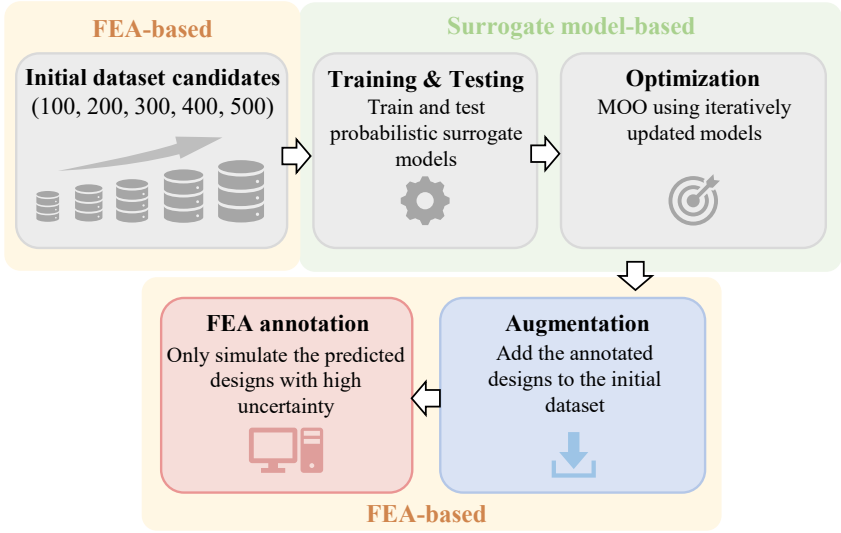


Figure 6.28: Workflow of the optimization strategy II.

Table 6.10: Training results based on five initial datasets

	Dataset - 500		Dataset - 400		Dataset- 300	
	o_1	o_2	o_1	o_2	o_1	o_2
R^2	0.9903	0.9987	0.9900	0.9965	0.9798	0.9945
SMAPE	1.16 %	0.55 %	1.21 %	0.60 %	1.31 %	0.77 %
PINAW	0.09	0.03	0.10	0.05	0.12	0.06
	Dataset - 200		Dataset - 100			
	o_1	o_2	o_1	o_2		
R^2	0.9795	0.9934	0.9452	0.9874		
SMAPE	1.42 %	0.86 %	1.91 %	1.29 %		
PINAW	0.18	0.09	0.55	0.22		

Table 6.11: SMAPE between predicted optimal designs and FEA verification after each iteration

	Dataset - 500		Dataset - 400		Dataset - 300	
	o_1	o_2	o_1	o_2	o_1	o_2
Iteration 0	1.87 %	0.85 %	2.63 %	0.95 %	3.22 %	1.16 %
Iteration 1	0.50 %	0.12 %	0.47 %	0.18 %	0.81 %	0.12 %
Iteration 2	0.20 %	0.08 %	0.28 %	0.08 %	0.21 %	0.07 %
Iteration 3	-	-	0.14 %	0.03 %	0.11 %	0.04 %
	Dataset - 200		Dataset - 100			
	o_1	o_2	o_1	o_2		
Iteration 0	3.67 %	2.06 %	3.96 %	3.84 %		
Iteration 1	0.72 %	0.28 %	1.20 %	0.76 %		
Iteration 2	0.21 %	0.05 %	0.35 %	0.17 %		
Iteration 3	0.15 %	0.04 %	0.21 %	0.05 %		
Iteration 4	-	-	0.07 %	0.03 %		

the best prediction accuracy after four iterations. Fig. 6.29 illustrates the gradual resolution of Pareto fronts for dataset 100. The initial predicted Pareto front is very narrow and overall overpredicted. This is due to the low predictive ability of the model resulting from only 100 data points for training. As key data are continuously added to the original dataset, the Pareto front shifts toward the lower left, indicating a progressive convergence toward the true maximum torque and maximum power values. After four iterations - adding a total of 160 key data points, the results stabilizes near the FEA true values, achieving exceptionally high prediction accuracy. A comparison of the final optimization Pareto fronts based on five different datasets is displayed in Fig. 6.30. It can be observed that the final Pareto fronts from datasets 100 and 300 contained machine designs with better performance, especially in the high torque and low power region. This may be attributed to the impact of the key data in directing model training, surpassing that of the initial data, and the different starting population of optimization algorithm. An increase in initial data enhances the model's prediction capability inside the global search area. However, key data is more beneficial to the model in specific regions, such as those around the Pareto fronts. Thus, it is

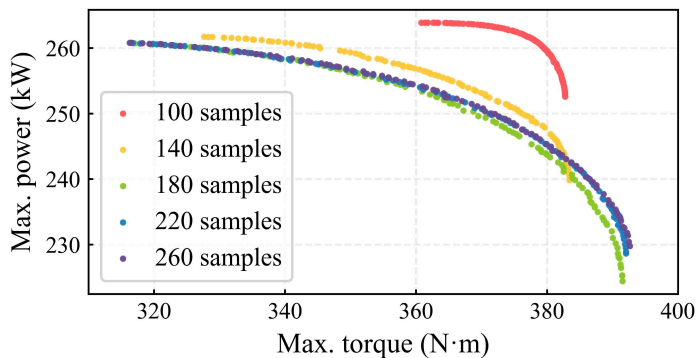


Figure 6.29: Comparison of different Pareto fronts of each iteration based on dataset 100.

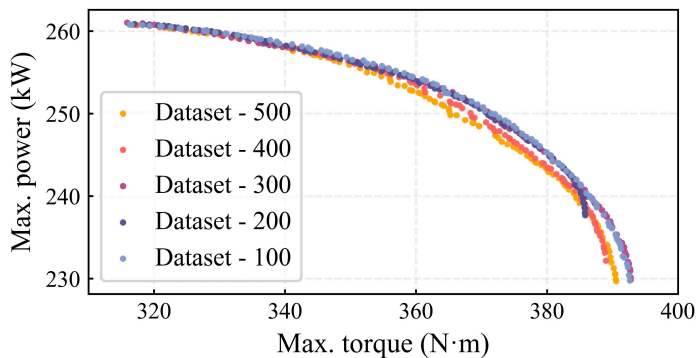


Figure 6.30: Comparison of final Pareto fronts based on five datasets.

a trade-off between global and local optimization. The Pareto front from dataset 200 is obviously narrower than the other. The reason may be the optimization algorithm’s inability to search for more superior solutions, instead concentrating on minimizing the prediction error based on the existing Pareto front. Alongside forecast accuracy, the duration of the whole optimization process is also a critical indicator. Table 6.12 demonstrates the total time cost for each dataset. In

this work, each simulation using the tool chain takes 6 minutes. Each iteration includes a hyperparameter optimization. For all iterations except the last one, FEA is executed for annotation of selected key data. Hence, every last iteration has the least amount of time cost. A bigger starting dataset enhances the prediction capacity of the initially trained model and reduces the number of needed iterations. However, it signifies the greatest expenditure of time. Conversely, dataset 100 has the minimal initial data but achieves the maximum prediction accuracy and the lowest time loss. In addition, FEA is a major portion of the total optimization. Therefore, enhancing the simulation speed for each design can substantially decrease the whole optimization duration and expedite the process.

Table 6.12: Time cost based on five different initial datasets

	Dataset - 500	Dataset - 400	Dataset - 300
Initial Dataset	500 x 6 min	400 x 6 min	300 x 6 min
Iteration 0	452 min	441 min	436 min
Iteration 1	271 min	279 min	267 min
Iteration 2	31 min	278 min	267 min
Iteration 3	-	30 min	28 min
Iteration 4	-	-	-
Total	62 h 34 min	57 h 08 min	46 h 38 min
	Dataset - 200	Dataset - 100	
Initial Dataset	200 x 6 min	100 x 6 min	
Iteration 0	429 min	421 min	
Iteration 1	268 min	267 min	
Iteration 2	268 min	267 min	
Iteration 3	28 min	268 min	
Iteration 4	-	29 min	
Total	36 h 33 min	30 h 52 min	

6.4 Summary and discussion

This chapter employs probabilistic metamodeling combined with active learning to improve the design optimization for the PMSM. The achievements can be summarized as follows. A comprehensive comparison among five probabilistic metamodels and the MLP in terms of prediction accuracy, predictive uncertainty, and training time is conducted. The comparison shows that MCMC offers the greatest prediction performance but suffers more than one hour of training time. GPR emerges as the second-best alternative. After identifying the most suitable probabilistic model, four different active learning-based optimization strategies are investigated. Strategy II shows the best performance. In the case of a limited dataset, the strategies suggest adding key data to augment the initial dataset. With the iteratively expanded dataset, the prediction accuracy of the optimal solutions is substantially improved. Compared to deterministic metamodels, probabilistic metamodels can provide additional uncertainty, which is utilized to guide the direction of selecting key data. Additionally, it is shown that Strategy II achieved a significant time savings compared to Strategy I using MLPs. This work mitigates the challenges of data-driven optimization for electric machines in practical applications. It also provides a guideline for the implementation of probabilistic metamodeling. However, the proposed method relies on FEA, indicating that large datasets still necessitate considerable simulation time.

The second study focuses on investigating the size of the initial data and the number of iterations. Therefore, a comparative analysis of five datasets of varying sizes is done, and they are optimized via iterative processes. The results indicate that an increase in starting data correlated with a reduction in the number of required repetitions, while the overall time consumption is maximized. This effect is related to the large part of computing time occupied by FEA. In contrast, dataset 100 is initially optimized with the lowest accuracy. However, after four iterations, the maximum prediction accuracy compared to other sizes of datasets is achieved. Furthermore, the total time loss associated with dataset 100 is reduced to 50 % of that of dataset 500.

It is worth noting that while the optimization strategy proposed in this chapter can significantly reduce the required simulation time, an initial dataset that is too small may constrain the optimization space. This could lead to key data being repeatedly selected within a local region, potentially causing the optimization results to converge only to a local optimum.

Chapter 7

Conclusion and Outlook

In this chapter, the application of ML methods in the electromagnetic design optimization of electric motors is summarized, including the use of deterministic metamodels, probabilistic metamodels, and data-efficient learning techniques. Finally, a brief perspective is provided regarding the future application of data-driven methods in the field of electric machines.

7.1 Result summary of this dissertation

In this dissertation, ML metamodels serve as surrogate models, replacing finite element simulations in traditional optimization algorithms, thereby significantly accelerating the overall optimization process. Consequently, this thesis comprehensively explores research on ML metamodels as its focal point. A variety of metamodels are utilized, and multiple novel optimization strategies have been proposed. Based on the work performed, the most important results and conclusions are presented as follows.

Firstly, to address the issue that parameter-based design optimization requires retraining the trained metamodels from scratch after altering the input design space, a TL-based approach has been proposed. When the parameter-based input space changes, this method can be dynamically modified. The prior information is reused by feeding the pre-trained neurons into the new NN as initial weights. Retraining the new NN from scratch is no longer necessary after reparametrization. The experiments show that MNNs outperform classical MLP in terms of prediction accuracy. Additionally, because transferred neurons

are used, the TL-based MNNs perform better than the standard MNN. The optimization scans 20000 designs within only around 12 seconds, and the MOO results demonstrate good accuracy in predicting maximum torque, maximum power, and cost. However, this method is based on a deterministic metamodel, meaning that each prediction yields only the predicted value as the final result. If the training data is insufficient, the model cannot be trained effectively, and there is no possibility of obtaining additional information to aid decision-making.

Secondly, to overcome the limitations of deterministic metamodels, a novel uncertainty-based optimization method is introduced. Two probabilistic metamodels are used, which are trained to produce motor KPIs together with the corresponding uncertainty. In the scenario, GPR performs better than BNN and DNN with excellent forecast accuracy following FEA validation. The findings show that prediction accuracy typically declines as prediction uncertainty rises. With the aid of uncertainty quantification, two optimization strategies are developed. According to the results of Strategy I, Pareto fronts derived from the upper boundary values have the most greedy machine designs, but they also result in higher degrees of uncertainty and prediction errors. The designs in the center of the Pareto front, on the other hand, have the lowest uncertainty and the highest average forecast accuracy. In addition, the proposed Strategy II restricts the standard deviation of designs throughout the optimization process. As the constraints tighten, there is a tendency to eliminate machine designs that exhibit more greedy performance, yet the resulting optimized designs demonstrate reduced uncertainty and enhanced prediction accuracy. Furthermore, it can aid in minimizing geometrically invalid designs. Finally, a sensitivity analysis is conducted to examine the relationship between predictive uncertainty and the robustness of design solutions. The outcomes indicate that more stringent constraints result in a marginally increased average robustness of the optimized solutions.

Finally, to tackle the issue of time-consuming acquisition of motor datasets, two novel data-efficient learning methods are proposed. First, a self-training framework based on semi-supervised learning is proposed. Using an initial dataset, a classifier and a predictor are trained. Subsequently, the classifier and predictor are employed to assign labels to a large volume of unlabeled data. The data containing pseudo-labels is then merged with the data that has simulated labels for additional model training, and this procedure is carried out iteratively. The results indicate that every iteration of this method rapidly generates pseudo-labeled samples to enhance the original datasets. Expanding the dataset

size improves the training of the classifier and predictor. The unpredictability of the forecast results is likewise diminished. Subsequently, four optimization strategies based on active learning are proposed. The core idea involves selecting a certain number of key data points from the optimization results based on uncertainty after each optimization iteration, annotating them with FEA, and then adding them back to the original dataset. This approach achieves extremely high prediction accuracy even when the initial dataset contains only 100 samples.

Overall, this thesis not only focuses on research into metamodel training but also explores data expansion and various efficient optimization strategies to achieve two goals: process efficiency and prediction accuracy.

7.2 Outlook

Data collection, model training, and design optimization strategy are currently the three phases that make up data-driven motor design optimization. Based on these three points, the future development direction of data-driven approaches in the field of electric motors is discussed.

Data-driven multi-physical optimization

First, at the data level, it is necessary to establish a high-quality multi-physics data repository. By parameterizing motor geometry, material properties, and operating conditions, coupled with automated finite element simulation tools, electromagnetic field, temperature distribution, mechanical stress, and displacement coupling results can be generated in bulk. Key performance metrics such as efficiency, power density, temperature rise, loss distribution, stress, and vibration can then be extracted. Therefore, a multiphysics simulation toolchain is indispensable, serving as one of the core components at the data level.

At the model level, metamodels oriented toward multi-physics field mapping relationships can be constructed. For high-dimensional input parameters, DNNs or graph neural networks (GNNs) can be employed to learn the nonlinear relationships between motor structure and field distribution, enabling direct prediction of multi-field coupled performance metrics from design variables. Simultaneously, further exploration of physics-informed NNs is warranted. These networks embed electromagnetic equations, heat conduction equations, and structural me-

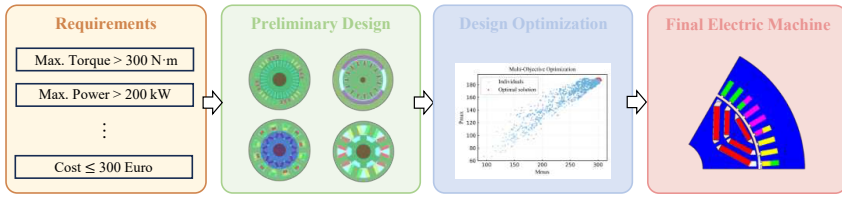


Figure 7.1: Design process for electric machines

chanical constraints as loss functions within the model, ensuring prediction results satisfy fundamental physical conservation laws. This approach enhances the model's generalization capability and interpretability.

Large language model-based preliminary design

The starting point for this research concept is that designing different electric motors for different requirements often requires a considerable amount of time in advance to develop one or more preliminary designs that correspond to the specified scenarios [O7], [O8]. Fig 7.1 illustrates the entire motor design process. In this context, the preliminary design of the electric motor acts as a link between the design requirements and the subsequent optimization. Consequently, the efficient generation of various candidates tailored to different requirements is a critical step. The core idea that enables accelerated initial design essentially comprises two aspects:

1. Large language models [217] are able to simultaneously extract different motor types and topologies from extensive data sets that meet the design requirements, rather than relying on manual calculations by human actors.
2. The chat bot can interact directly with data analysis tools and simulation software, eliminating the need for human intervention.

Therefore, large language models can effectively handle the preliminary design phase of electric motors. Based on different motor design requirements, they can rapidly generate candidate motor designs for selection, laying the groundwork for subsequent design optimization.

AI-aided FEA for electric machines

Currently, finite element simulation as a method for data acquisition has consistently faced the challenge of being time-consuming. Therefore, directly applying AI models to solve the physical equations involved in the finite element method will be a key research focus.

In this scenario, GNNs [218] can be employed to characterize the complex geometric topology and mesh structure information of electric motors. Since finite element models are inherently composed of mesh nodes and their connectivity relationships, future research may directly represent finite element meshes as graph structures. By utilizing GNNs to learn field coupling relationships between nodes, this approach enables rapid mapping from geometric topology to multiphysics responses. Building upon this foundation, future work may further explore fusion strategies between physics-informed NNs and GNNs. For instance, GNNs could extract geometric and mesh features, while physics-informed NNs constrain outputs to satisfy physical equations, thereby constructing hybrid models that combine physical consistency with topological generalization capabilities. Such models could be embedded as rapid simulation engines within optimization workflows, supporting multi-objective design searches and enabling the co-optimization of motor efficiency, power density, temperature rise, and structural reliability. Ultimately, integrating AI-assisted finite element models with experimental data and operational monitoring information holds promise for establishing an intelligent modeling and optimization framework spanning the entire motor design lifecycle. This approach offers a novel technical pathway for the rapid design and iterative refinement of high-performance electric motors.

Chapter 8

Appendix

8.1 Software and package details

For dataset generation:

- All datasets were generated using a in-house FEA toolchain. The integrated simulation software is Altair Flux 2025. The toolchain was developed using Java version 1.8.0 and Python version 3.7.16.

For model training (partially):

- pytorch 1.11.0 - deep learning framework
- gpytorch 1.11.0 - Gaussian process implementation
- pyro-ppl 1.8.4 - Bayesian neural network implementation
- numpy 1.21.5, pandas 1.3.5 - data processing
- matplotlib 3.5.3, seaborn 0.12.2 - data visualization
- optuna 3.1.1 - hyperparameter optimization
- pymoo 0.6.0.1 - optimization algorithm implementation

8.2 Simulation toolchain supplement

In this section, additional information regarding simulation toolchain is provided. It is comprised of four components: the main function programmed by Java, Python as the auxiliary language, a database for reference design, and a simulation base model. In the application, the main function controls the entire simulation and optimization processes. Additionally, it reads or writes the database, opens or modifies the base model, and invokes the Python-based scripts. All commands and settings are visualized and integrated into the user interface. Fig 8.1 depicts the graphical user interface. This tool incorporates many functions, among which the “Parameterstudien” can be used to simulate multi-parameter combinations. When all simulations are complete, all results will be saved in a specific file. More precisely, each valid simulation corresponds

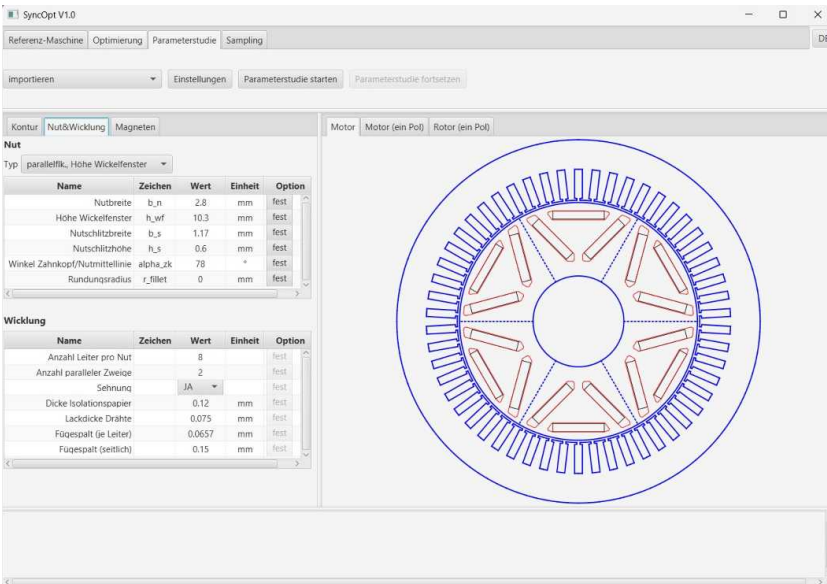


Figure 8.1: User interface of the toolchain.

Table 8.1: General settings of the simulation toolchain

Parameter	Value	Unit	Parameter	Value	Unit
CPU cores	14	-	Battery voltage	640	V
Numeric memory	2500	Mb	Parallel branches	2	-
Skew angle	0	°	Conductors per slot	8	-
Star or delta connection	Star	-	Maximum current	350	A
Magnet temperature	75	°C	Maximum rotational speed	16800	rpm

to a result file, in which all outcomes are saved in tabular form and can be accessed by a user.

Table 8.1 lists some basic motor simulation settings for this thesis. These settings can be modified according to different motor types.

8.3 Other results

8.3.1 Datasets in Chapter 4

Chapter 4 involves three datasets in total. Dataset III has already been introduced; here, the other two datasets, I and II, are supplemented. Fig. 8.2 and Fig. 8.3 show the correlations between design parameters d and KPIs o .

Dataset I: d_1 : magnet 1 width, d_2 : magnet 1 height, d_3 : magnet 2 width, d_4 : magnet 2 height, d_5 : magnet inclination angle, d_6 : slot width.

Dataset II: d_1 : magnet 1 width, d_2 : magnet 1 height, d_3 : magnet 2 width, d_4 : magnet 2 height, d_5 : magnet inclination angle, d_8 : width between magnet 2 and air gap, d_{11} : distance between magnets, d_{12} : distance between poles.

The prediction results based on dataset I and dataset II are shown in Fig. 8.4 and 8.5, respectively.

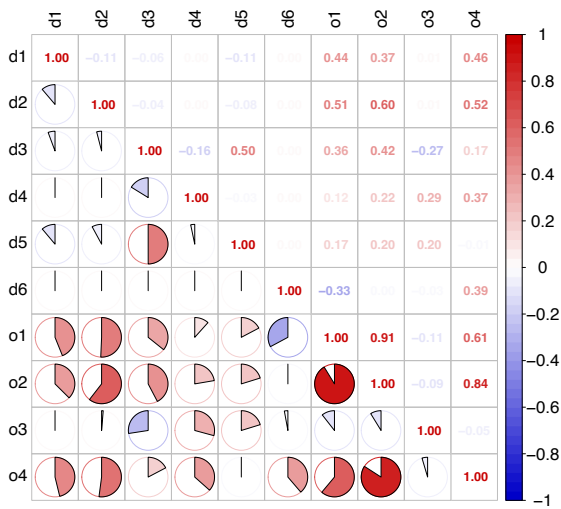


Figure 8.2: Correlation between design parameters and KPIs for dataset I.

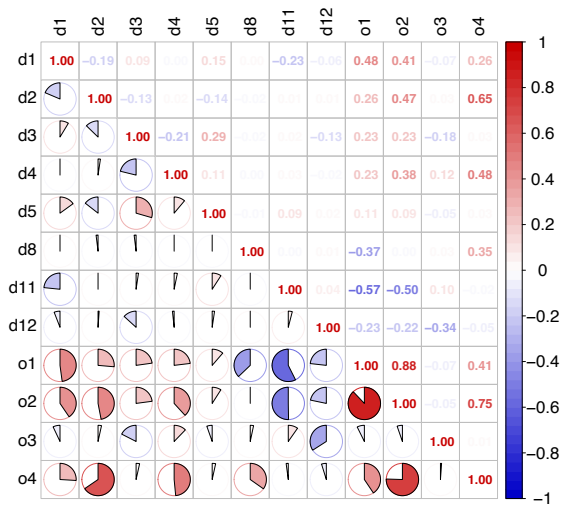


Figure 8.3: Correlation between design parameters and KPIs for dataset II.

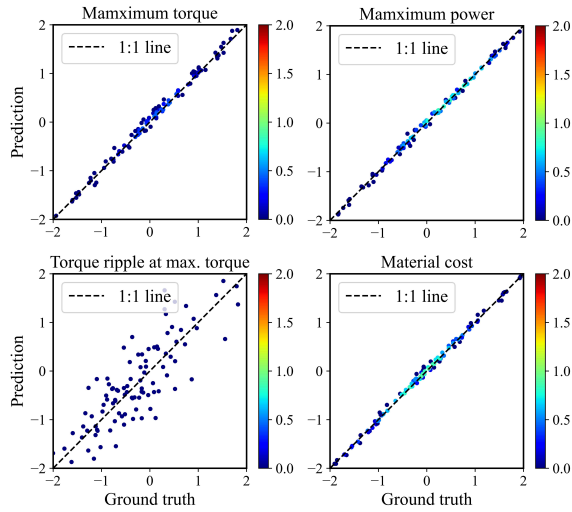


Figure 8.4: Prediction performance based on dataset I.

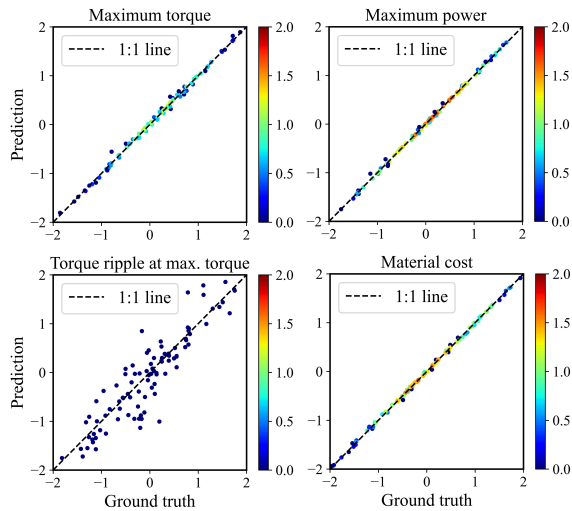


Figure 8.5: Prediction performance based on dataset II.

8.3.2 Dataset in Chapter 5

Dataset III is used in Chapter 5: d_1 : magnet 1 width, d_2 : magnet 1 height, d_3 : width between magnet 2 and air gap, d_4 : magnet inclination angle, d_5 : magnet 2 width, d_6 : magnet 2 height, d_7 : slot width, d_8 : air gap, d_9 : flux barrier width, d_{10} : slot height. Fig. 8.6 demonstrates the distribution of the collected dataset in the input space. The input parameters d_1 , d_4 , and d_5 are particularly

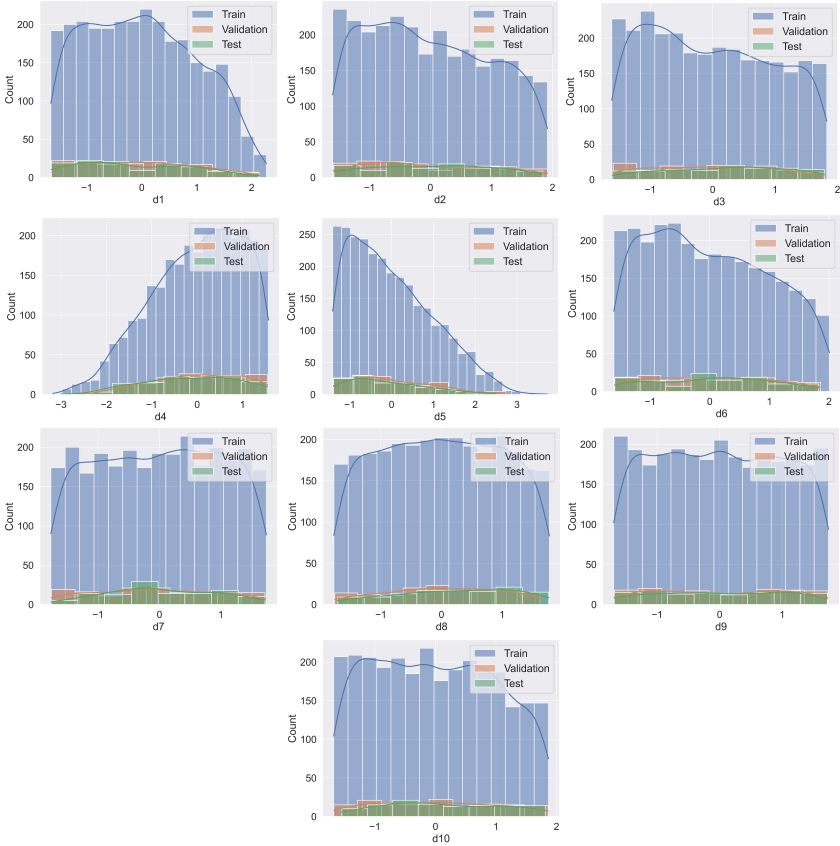


Figure 8.6: Distribution visualization of input design parameters.

inhomogeneously distributed. One possible explanation is that around 30% of the sampling points obtained through the LHS are geometrically infeasible, and their removal leads to an unbalanced distribution in the input space.

8.3.3 Training results of 6.3

Below are the supplementary model prediction performance results for Chapter 6.3. Fig. 8.7 illustrates the performance of DE on the test set. It can be observed that the DE demonstrates strong performance in predicting torque ripple, with minimal variation in prediction uncertainty. In addition, it exhibits high prediction accuracy when forecasting maximum power and material costs. Fig. 8.8 illustrates the performance of MC Dropout on the test set. MC Dropout also shows lower uncertainty when predicting torque ripple. However, it demonstrates relatively large prediction errors in maximum torque and maximum power. Fig. 8.9 illustrates the performance of VI on the test set. VI displays high uncertainty in predicting torque ripple. Moreover, it exhibits relatively larger errors in forecasting both maximum torque and costs.

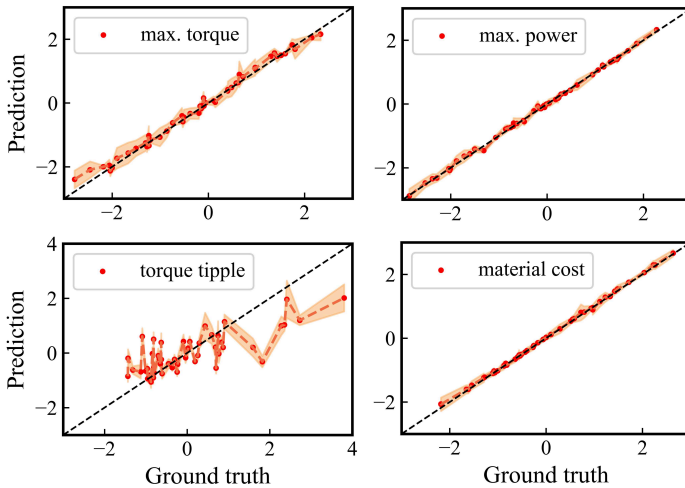


Figure 8.7: KPI predictions with uncertainty intervals of DE over test set.

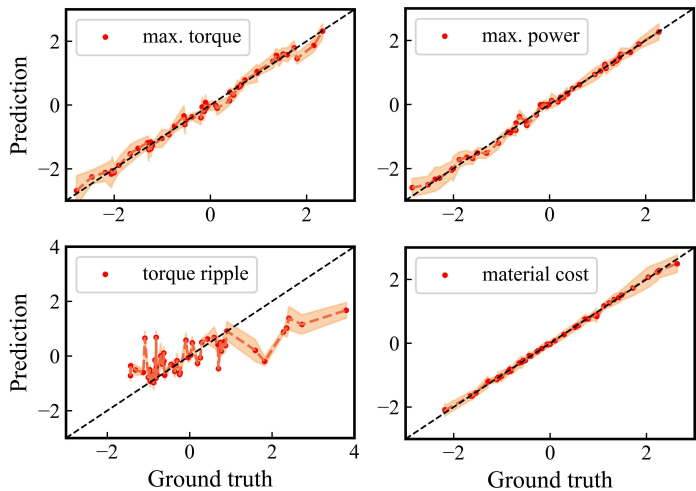


Figure 8.8: KPI predictions with uncertainty intervals of MC Dropout over test set.

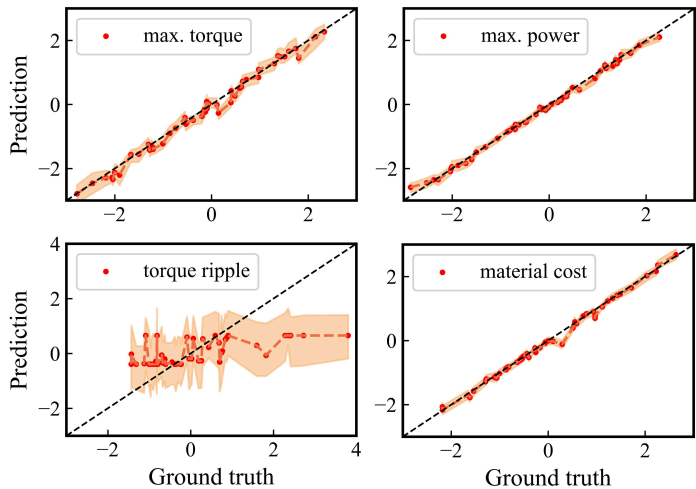


Figure 8.9: KPI predictions with uncertainty intervals of VI over test set.

List of figures

1.1	Conventional design optimization workflow for electric machines.	2
1.2	Comparison between simulation-based and machine learning-based optimization.	4
1.3	Outline of this dissertation.	7
2.1	Cross-sections of different electric machines. (a) Asynchronous motor. (b) Permanent magnet synchronous motor. (c) Switched reluctance motor.	10
2.2	Discretization of a 2D field area by a grid of triangles and rectangles.	12
2.3	Analogy between magnetic vector potential and magnetic field intensity.	15
2.4	General FEM simulation process for electric machines [62]. . .	17
2.5	Different phases of FEM of a PMSM example.	18
2.6	Illustration of first-order and second-order elements.	18
2.7	Definition of the n -th triangle element.	19
2.8	Loss diagram of a PMSM.	26
2.9	An example Pareto front for a 2D-objective optimization with min-min.	29
2.10	Comparison of different optimization models and categorization of generally used optimization methods	30
2.11	Gradient descents with different learning rates.	31
2.12	Illustration of the genetic algorithm.	35
2.13	Illustration of the process of velocity update.	37
3.1	The relation among AI, ML, and DL.	40
3.2	Illustration of SVM principle.	43
3.3	Illustration of an example GPR model.	46

3.4	An example architecture of an artificial neural network.	48
3.5	Summary table of commonly used loss functions with their typical applications.	53
3.6	Visualization of a CNN architecture with different layers and computation process.	55
3.7	Demonstration of stride and padding.	56
3.8	General design optimization workflow for electric machines using ML metamodels.	59
3.9	Difference between (a) parameter vector- and (b) image-based inputs.	60
3.10	Structure of ML paradigms used for optimization of electric machines.	62
3.11	Proposed self-training framework based on semi-supervised learning.	63
3.12	Basic workflow of GAN.	64
3.13	Architecture of the actor-critic RL algorithm.	65
3.14	Workflow of VAE-based optimization.	68
4.1	Different rotor topologies of PMSMs.	73
4.2	Comparison between deterministic and probabilistic metamodels.	73
4.3	Proposed data-driven design optimization workflow for electric motors.	75
4.4	Flowchart of the in-house FEA-based simulation toolchain.	77
4.5	Reference PMSM geometry and design parameters.	78
4.6	Correlation between design parameters and KPIs for dataset III.	80
4.7	Example rotor images of subdatasets.	81
4.8	Proposed workflow for aggregating pre-trained NNs.	84
4.9	Illustration of five-fold cross validation.	85
4.10	Evaluation of prediction performance based on learning rate of transferred neurons under 10 sets of random seeds.	88
4.11	Prediction performance of MNN III based on dataset III.	90
4.12	Prediction performance comparison among MNN III, MNN III without TL, and MLP.	91
4.13	Pareto fronts for maximum torque and maximum power based on two different optimization algorithms NSGA-II and NSGA-III.	92
5.1	Left: each weight is a deterministic value. Right: each weight is associated with a probability distribution.	100

5.2	The overall framework of the proposed method for electric machine optimization.	102
5.3	Geometry of the reference PMSM.	103
5.4	Sample distribution visualization of output parameters.	105
5.5	Prediction performance of BNN.	110
5.6	Prediction performance of GPR.	111
5.7	Frequency statistics of predictive uncertainty for each KPI using GPR and associated prediction error.	112
5.8	KPI prediction performance comparison between probabilistic and deterministic metamodels.	113
5.9	Pareto front results regarding two strategies.	116
5.10	Frequency statistics and prediction accuracy for each KPI based on strategy II optimization.	120
6.1	Workflow of data-efficient learning method.	127
6.2	General workflow of self-training methods.	128
6.3	Self-training framework based on semi-supervised learning for electric machine design.	129
6.4	Difference between Mahalanobis distance and Euclidean distance.	131
6.5	Reference PMSM geometry with partial design parameters.	132
6.6	Comparison of a standard NN and a NN with Dropout layer.	134
6.7	Illustration of MC Dropout architecture with Dropout layers.	135
6.8	MC Dropout neural network architecture with two residual blocks.	137
6.9	Prediction performance of KPI predictor based on initial dataset and expanded dataset after first iteration. (a) Initial dataset. (b) Expanded dataset.	141
6.10	ROC curves of geometry classifier.	142
6.11	Confusion matrix based on initial dataset and expanded datasets with threshold 0.5.	143
6.12	Flowchart of active learning process.	145
6.13	Illustration of DE with multiple inference processes.	146
6.14	Illustration of BNN with probability distributed weights.	147
6.15	Illustration of MCMC's workflow adapted from [212].	149
6.16	Geometry of the reference PMSM.	150
6.17	Correlation map of varied input and output parameters.	150
6.18	KPI prediction performance of MLP over test set.	154
6.19	KPI prediction performance with uncertainty intervals of GPR over test set.	154

6.20	KPI prediction performance with uncertainty intervals of MCMC over test set.	155
6.21	The overall framework based on active learning for electric machine optimization.	157
6.22	Comparison of computation time and KPI prediction errors for different numbers of key data.	159
6.23	Predicted Pareto fronts using Strategy I compared with FEA verification.	161
6.24	Predicted Pareto fronts using Strategy II compared with FEA verification.	162
6.25	Predicted Pareto fronts using Strategy III compared with FEA verification.	163
6.26	Predicted Pareto fronts using Strategy IV compared with FEA verification.	164
6.27	The chosen key data of each iteration regarding Strategies I and II.	165
6.28	Workflow of the optimization strategy II.	169
6.29	Comparison of different Pareto fronts of each iteration based on dataset 100.	171
6.30	Comparison of final Pareto fronts based on five datasets.	171
7.1	Design process for electric machines	178
8.1	User interface of the toolchain.	182
8.2	Correlation between design parameters and KPIs for dataset I.	184
8.3	Correlation between design parameters and KPIs for dataset II.	184
8.4	Prediction performance based on dataset I.	185
8.5	Prediction performance based on dataset II.	185
8.6	Distribution visualization of input design parameters.	186
8.7	KPI predictions with uncertainty intervals of DE over test set.	187
8.8	KPI predictions with uncertainty intervals of MC Dropout over test set.	188
8.9	KPI predictions with uncertainty intervals of VI over test set.	188

List of tables

3.1	Studies on data-driven models for electric machines.	70
4.1	Details of reference machine parameters.	79
4.2	Hyperparameter details of MNNs.	87
4.3	Evaluation of all datasets.	89
4.4	FEA verification of the optimal designs from Pareto fronts. . . .	93
4.5	Optimized design parameter values of selected design A and design B.	94
4.6	FEA verification of five group sampling from the results of NSGA-II using SMAPE.	94
5.1	Details of reference machine parameters.	104
5.2	Details of BNN hyperparameters.	106
5.3	Details of GPR Hyperparameters.	108
5.4	Comparison of machine designs obtained by different optimization strategies.	117
5.5	Global sensitivity results for the design solutions under different constraints.	122
6.1	Details of reference machine parameters.	133
6.2	Hyperparameter details of the NN with MC Dropout	138
6.3	Confusion matrix	138
6.4	Evaluation summary of KPI predictor based on different datasets	140
6.5	Details of reference machine template parameters	151
6.6	Details of hyperparameters for each metamodel in form of o_1 o_2 o_3 o_4	152
6.7	KPIs evaluation of six metamodels in form of o_1 o_2 o_3 o_4 . .	156

6.8	SMAPE between FEA verification and optimal solutions in form of $o_1 \mid o_2$	165
6.9	Time balance of different optimization strategies	167
6.10	Training results based on five initial datasets	169
6.11	SMAPE between predicted optimal designs and FEA verification after each iteration	170
6.12	Time cost based on five different initial datasets	172
8.1	General settings of the simulation toolchain	183

Bibliography

Own Publications

- [O1] N. Chen and M. Doppelbauer, “Review of machine learning-based electromagnetic design optimization for electrical machines,” *International Conference on Electrical Machines and Systems*, pp. 3728–3734, 2025.
- [O2] N. Chen and M. Doppelbauer, “Transfer learning-based modular neural network for multi-objective optimization of interior permanent magnet synchronous motors,” *International Conference on Electrical Machines*, pp. 1–7, 2024.
- [O3] N. Chen, C. Digel, and M. Doppelbauer, “Uncertainty quantification-based multi-objective optimization design of electrical machines using probabilistic metamodels,” *IEEE Transactions on Energy Conversion*, vol. 40, no. 2, pp. 860–872, 2024.
- [O4] N. Chen, C. Digel, and M. Doppelbauer, “A semi-supervised method for key performance indicator prediction of electrical machines under a self-training framework,” *International Conference on Electrical Machines*, pp. 1–7, 2024.
- [O5] N. Chen, C. Digel, M. Brodatzki, and M. Doppelbauer, “Active learning-based multiobjective optimization of electric machines using probabilistic metamodeling,” *IEEE Transactions on Energy Conversion*, vol. 40, no. 2, pp. 860–872, 2024.
- [O6] N. Chen, C. Digel, Y. Wang, and M. Doppelbauer, “Multiobjective optimization of electrical machines using probabilistic surrogate modeling with limited data,” *IEEE International Electric Machines & Drives Conference*, pp. 415–420, 2025.

- [O7] Y. Wang, T. Yang, H. Huang, T. Zou, N. Chen, and C. Gerada, “Artificial intelligence enhanced scaling design database for electrical machine inverse design,” *Energy Conversion Congress and Expo Europe (ECCE Europe)*, pp. 1–6, 2025.
- [O8] Y. Wang, T. Yang, H. Huang, T. Zou, J. Li, N. Chen, and Z. Zhang, “Data driven automatic electrical machine preliminary design with artificial intelligence expert guidance,” *arXiv: 2411.11221*), 2024.

References

- [1] V. Santos, J. Eras, and M. Ulloa, "Evaluation of the energy saving potential in electric motors applying a load-based voltage control method," *Energy*, vol. 303, p. 132 012, 2024.
- [2] A. Mayr et al., "Electric motor production 4.0 – application potentials of industry 4.0 technologies in the manufacturing of electric motors," *2018 8th International Electric Drives Production Conference (EDPC)*, pp. 1–13, 2018.
- [3] U. Das et al., "Development of automatic cnc machine with versatile applications in art, design, and engineering," *Array*, vol. 24, p. 100 369, 2024.
- [4] M. Hu, W. Yu, J. Lei, Z. Wu, W. Hua, and Y. Hu, "Sensorless control of a high-speed pmsm with rapid acceleration for air compressors using a high-order extended state observer," *2021 IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 4781–4787, 2021.
- [5] C. Candelo, J. Riba, A. Espinosa, and P. Blanch, "Customized pmsm design and optimization methodology for water pumping applications," *IEEE Transactions on Energy Conversion*, vol. 37, no. 1, pp. 454–465, 2021.
- [6] Z. Liu, Z. Zhang, W. Lin, X. Yu, C. Buccella, and C. Cecati, "Disturbance rejection-based motion control of linear motors via integral sliding mode observer," *IEEE Transactions on Industrial Electronics*, vol. 72, no. 3, pp. 3061–3071, 2025.
- [7] G. Pellegrino, A. Vagati, P. Guglielmi, and B. Boazzo, "Performance comparison between surface-mounted and interior pm motor drives for electric vehicle application," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 803–811, 2012.
- [8] X. Liu, H. Chen, J. Zhao, and A. Belahcen, "Research on the performances and parameters of interior pmsm used for electric vehicles," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 6, pp. 3533–3545, 2016.
- [9] E. Agamloh, A. Von Jouanne, and A. Yokochi, "An overview of electric machine trends in modern electric vehicles," *Machines*, vol. 8, no. 2, p. 20, 2020.

- [10] D. Bucherl, R. Nuscheler, W. Meyer, and H. Herzog, "Comparison of electrical machine types in hybrid drive trains: Induction machine vs. permanent magnet synchronous machine," *2008 18th International Conference on Electrical Machines*, pp. 1–6, 2008.
- [11] C. Jeong, Y. Kim, and J. Hur, "Optimized design of pmsm with hybrid-type permanent magnet for improving performance and reliability," *IEEE Transactions on Industry Applications*, vol. 55, no. 5, pp. 4692–4701, 2019.
- [12] B. Poudel, E. Amiri, P. Rastgoufard, and B. Mirafzal, "Toward less rare-earth permanent magnet in electric machines: A review," *IEEE Transactions on Magnetics*, vol. 57, no. 9, pp. 1–19, 2021.
- [13] M. Cheng, X. Zhang, M. Dhimish, W. Qiu, and S. Niu, "A review of data-driven surrogate models for design optimization of electric motors," *IEEE Transactions on Transportation Electrification*, vol. 10, no. 4, pp. 8413–8431, 2024.
- [14] W. Qiu, X. Zhao, A. Tyrrell, S. Perinpanayagam, S. Niu, and G. Wen, "Application of artificial intelligence-based technique in electric motors: A review," *IEEE Transactions on Power Electronics*, vol. 39, no. 10, pp. 13 543–13 568, 2024.
- [15] L. Liu et al., "Review of surrogate model assisted multi-objective design optimization of electrical machines: New opportunities and challenges," *Renewable and Sustainable Energy Reviews*, vol. 215, p. 115 609, 2025.
- [16] R. Mukhamediev, A. Symagulov, Y. Kuchin, K. Yakunin, and M. Yelis, "From classical machine learning to deep neural networks: A simplified scientometric review," *Applied Sciences*, vol. 11, no. 12, 2021.
- [17] R. Zhong et al., "Accurate model of switched reluctance motor based on indirect measurement method and least square support vector machine," *IET Electric Power Applications*, vol. 10, no. 9, pp. 916–922, 2016.
- [18] J. Song et al., "An efficient multiobjective design optimization method for a pmslm based on an extreme learning machine," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 2, pp. 1001–1011, 2019.
- [19] H. Dhulipati et al., "Advanced design optimization technique for torque profile improvement in six-phase pmsm using supervised machine learning for direct-drive ev," *IEEE Transactions on Energy Conversion*, vol. 34, no. 4, pp. 2041–2051, 2019.

-
- [20] Y. Shimizu, S. Morimoto, M. Sanada, and Y. Inoue, "Using machine learning to reduce design time for permanent magnet volume minimization in ipmsms for automotive applications," *IEEJ Journal of Industry Applications*, vol. 10, no. 5, pp. 554–563, 2021.
 - [21] G. Li, R. Li, H. Hou, G. Zhang, and Z. Li, "A data-driven motor optimization method based on support vector regression—multi-objective, multivariate, and with a limited sample size," *Electronics*, vol. 13, no. 12, p. 2231, 2024.
 - [22] J. Zhang, J. Liu, and L. Gei, "Multi-objective optimization of switched reluctance motor based on gaussian process regression model," *2024 IEEE 10th International Power Electronics and Motion Control Conference*, pp. 4020–4025, 2024.
 - [23] H. Kurtović and I. Hahn, "Neural network meta-modeling and optimization of flux switching machines," *2019 IEEE International Electric Machines & Drives Conference (IEMDC)*, pp. 629–636, 2019.
 - [24] M. Tahkola, J. Keränen, D. Sedov, M. Far, and J. Kortelainen, "Surrogate modeling of electrical machine torque using artificial neural networks," *IEEE Access*, vol. 8, pp. 220 027–220 045, 2020.
 - [25] A. Khan and D. Lowther, "Physics informed neural networks for electromagnetic analysis," *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 2022.
 - [26] V. Parekh, D. Flore, and S. Schöps, "Variational autoencoder-based metamodeling for multi-objective topology optimization of electrical machines," *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 2022.
 - [27] V. Parekh, D. Flore, and S. Schöps, "Deep learning-based meta-modeling for multi-objective technology optimization of electrical machines," *IEEE Access*, vol. 11, pp. 93 420–93 430, 2023.
 - [28] B. Poudel and E. Amiri, "Deep learning based design methodology for electric machines: Data acquisition, training and optimization," *IEEE Access*, vol. 11, pp. 18 281–18 290, 2023.
 - [29] J. Gu, W. Hua, W. Yu, Z. Zhang, and H. Zhang, "Surrogate model-based multiobjective optimization of high-speed pm synchronous machine: Construction and comparison," *IEEE Transactions on Transportation Electrification*, vol. 9, no. 1, pp. 678–688, 2023.

- [30] V. Parekh, D. Flore, and S. Schöps, “Performance analysis of electrical machines using a hybrid data- and physics-driven model,” *IEEE Transactions on Energy Conversion*, vol. 38, no. 1, pp. 530–539, 2023.
- [31] A. Pop, Z. Cai, and J. Gyselinck, “Machine-learning aided multiobjective optimization of electric machines—geometric-feasibility and enhanced regression models,” *IEEE Journal of Emerging and Selected Topics in Industrial Electronics*, vol. 4, no. 3, pp. 844–854, 2023.
- [32] S. Son, H. Lee, D. Jeong, K. Oh, and K. Sun, “A novel physics-informed neural network for modeling electromagnetism of a permanent magnet synchronous motor,” *Advanced Engineering Informatics*, vol. 57, p. 102 035, 2023.
- [33] S. Doi, H. Sasaki, and H. Igarashi, “Multi-objective topology optimization of rotating machines using deep learning,” *IEEE Transactions on Magnetics*, vol. 55, no. 6, pp. 1–5, 2019.
- [34] A. Khan, V. Ghorbanian, and D. Lowther, “Deep learning for magnetic field estimation,” *IEEE Transactions on Magnetics*, vol. 55, no. 6, pp. 1–4, 2019.
- [35] H. Sasaki and H. Igarashi, “Topology optimization accelerated by deep learning,” *IEEE Transactions on Magnetics*, vol. 55, no. 6, pp. 1–5, 2019.
- [36] J. Asanuma, S. Doi, and H. Igarashi, “Transfer learning through deep learning: Application to topology optimization of electric motor,” *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 2020.
- [37] S. Barmada, N. Fontana, L. Sani, D. Thomopoulos, and M. Tucci, “Deep learning and reduced models for fast optimization in electromagnetics,” *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 2020.
- [38] N. Gabdullin, S. Madanzadeh, and A. Vilkin, “Towards end-to-end deep learning performance analysis of electric motors,” *Actuators*, vol. 10, no. 2, p. 28, 2021.
- [39] H. Sasaki, Y. Hidaka, and H. Igarashi, “Explainable deep neural network for design of electric motors,” *IEEE Transactions on Magnetics*, vol. 57, no. 6, pp. 1–4, 2021.
- [40] C. Lee and W. Ha, “Optimal design of ipm rotor shape using generative adversarial networks,” *2021 24th International Conference on Electrical Machines and Systems (ICEMS)*, pp. 2440–2444, 2021.

-
- [41] T. Aoyagi, Y. Otomo, H. Igarashi, H. Sasaki, Y. Hidaka, and H. Arita, "Prediction of current-dependent motor torque characteristics using deep learning for topology optimization," *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 2022.
 - [42] Y. Shimizu, S. Morimoto, M. Sanada, and Y. Inoue, "Automatic design system with generative adversarial network and convolutional neural network for optimization design of interior permanent magnet synchronous motor," *IEEE Transactions on Energy Conversion*, vol. 38, no. 1, pp. 724–734, 2023.
 - [43] H. Sato and H. Igarashi, "Fast topology optimization for pm motors using variational autoencoder and neural networks with dropout," *IEEE Transactions on Magnetics*, vol. 59, no. 5, pp. 1–4, 2023.
 - [44] Y. Shimizu, "Automatic design system with generative adversarial network and vision transformer for efficiency optimization of interior permanent magnet synchronous motor," *IEEE Transactions on Industrial Electronics*, vol. 71, no. 11, pp. 14 600–14 609, 2024.
 - [45] M. Kishi, S. Wakao, N. Murata, H. Makino, K. Takeuchi, and M. Matsushita, "Multi-objective topology optimization of synchronous reluctance motors using autoencoder-estimated flux barrier shapes," *IEEE Journal of Industry Applications*, vol. 14, no. 1, pp. 12–19, 2025.
 - [46] A. Khan, M. Mohammadi, V. Ghorbanian, and D. Lowther, "Efficiency map prediction of motor drives using deep learning," *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 2020.
 - [47] H. Sasaki, "Topology optimization for motor using multitask convolutional neural network under multiple current conditions," *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 2022.
 - [48] V. Parekh, D. Flore, and S. Schöps, "Deep learning-based prediction of key performance indicators for electrical machines," *IEEE Access*, vol. 9, pp. 21 786–21 797, 2021.
 - [49] Y. Gong, A. Gneiting, C. Zhao, and N. Parspour, "Comparative study of different surrogate models in the multi-physics optimization of synchronous reluctance machine," *2023 26th International Conference on Electrical Machines and Systems (ICEMS)*, pp. 1284–1290, 2023.
 - [50] T. Sato and M. Fujita, "A data-driven automatic design method for electric machines based on reinforcement learning and evolutionary optimization," *IEEE Access*, vol. 9, pp. 71 284–71 294, 2021.

- [51] A. Khan, C. Midha, and D. Lowther, “Reinforcement learning for topology optimization of a synchronous reluctance motor,” *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 2022.
- [52] C. Wang, T. Dong, L. Chen, G. Zhu, and Y. Chen, “Multi-objective optimization approach for permanent magnet machine via improved soft actor–critic based on deep reinforcement learning,” *Expert Systems with Applications*, vol. 264, p. 125 834, 2025.
- [53] J. Yang, K. Zhou, Y. Li, and Z. Liu, “Generalized out-of-distribution detection: A survey,” *International Journal of Computer Vision*, vol. 132, no. 12, pp. 5635–5662, 2024.
- [54] A. Adadi, “A survey on data-efficient algorithms in big data era,” *Journal of Big Data*, vol. 8, no. 1, pp. 1–54, 2021.
- [55] J. Pyrhönen, T. Jokinen, and V. Hrabovcova, *Design of rotating electrical machines*. Wiley, 2008.
- [56] B. Bilgin et al., “Modeling and analysis of electric motors: State-of-the-art review,” *IEEE Transactions on Transportation Electrification*, vol. 5, no. 3, pp. 602–617, 2019.
- [57] D. Poljak and C. Brebbia, *Boundary element methods for electrical engineers*. WIT press, 2005, vol. 4.
- [58] B. Malik, “Electric field calculations by numerical techniques,” 2009.
- [59] K. Hameyer and R. Belmans, *Numerical modelling and design of electrical machines and devices*. WIT press, 1999, vol. 1.
- [60] V. Parekh, *Deep learning based design and optimization of electrical machines*. Technical University of Darmstadt, 2025.
- [61] S. J. Salom, *Finite element analysis of electrical machines*. Kluwer Academic Publishers, 1995.
- [62] R. Bargallo, *Finite elements for electrical engineering*. Universitat Politècnica de Catalunya, 2006.
- [63] S. Chakraverty, N. Mahato, P. Karunakar, and T. Rao, “Weighted residual methods,” in *Advanced Numerical and Semi-Analytical Methods for Differential Equations*, Wiley, 2019, pp. 31–43.
- [64] L. Mu, J. Wang, X. Ye, and S. Zhang, “A weak galerkin finite element method for the maxwell equations,” *Journal of Scientific Computing*, vol. 65, no. 1, pp. 363–386, 2015.

-
- [65] L. Sabat and C. Kundu, "History of finite element method: A review," *Recent Developments in Sustainable Infrastructure: Select Proceedings of ICRDSI 2019*, pp. 395–404, 2020.
- [66] M. Popescu, "Prediction of the electromagnetic torque in synchronous machines through maxwell stress harmonic filter (hft) method," *Electrical Engineering*, vol. 89, pp. 117–125, 2006.
- [67] X. Chen, J. Hu, K. Chen, and Z. Peng, "Modeling of electromagnetic torque considering saturation and magnetic field harmonics in permanent magnet synchronous motor for hev," *Simulation Modelling Practice and Theory*, vol. 66, pp. 212–225, 2016.
- [68] R. H. Park, "Two-reaction theory of synchronous machines generalized method of analysis-part i," *Transactions of the American Institute of Electrical Engineers*, vol. 48, no. 3, pp. 716–727, 1929.
- [69] M. S. Rafaq, W. Midgley, and T. Steffen, "A review of the state of the art of torque ripple minimization techniques for permanent magnet synchronous motors," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 1019–1031, 2024.
- [70] S. I. Suriano-Sánchez, M. Ponce-Silva, and V. H. Olivares-Peregrino, "A review of torque ripple reduction design," *Eng*, vol. 4, no. 3, pp. 646–661, 2022.
- [71] W. Li, J. Xue, X. Fan, and L. Zhu, "Loss analysis of permanent magnet synchronous motor system based on strategy-circuit-field co-simulation and an accurate iron loss calculation method," *IEEE Access*, vol. 12, pp. 168 339–168 348, 2024.
- [72] H. B. Dwight, "Skin effect and proximity effect in tubular conductors," *Transactions of the American Institute of Electrical Engineers*, vol. XLI, pp. 189–198, 1922.
- [73] A. Al-Timimy et al., "Investigation of ac copper and iron losses in high-speed high-power density pmsm," *2018 XIII International Conference on Electrical Machines (ICEM)*, pp. 263–269, 2018.
- [74] G. Liu, M. Liu, Y. Zhang, H. Wang, and C. Gerada, "High-speed permanent magnet synchronous motor iron loss calculation method considering multiphysics factors," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 7, pp. 5360–5368, 2020.

- [75] Y. Huang, J. Dong, J. Zhu, and Y. Guo, “Core loss modeling for permanent magnet motor based on flux variation locus and finite element method,” *IEEE Transactions on Magnetics*, vol. 48, no. 2, pp. 1023–1026, 2012.
- [76] S. Park, E. Lee, J. Park, S. Hwang, and M. Lim, “Prediction of mechanical loss for high-power-density pmsm considering eddy current loss of pms and conductors,” *IEEE Transactions on Magnetics*, vol. 57, no. 2, pp. 1–5, 2021.
- [77] J. Klötzl, M. Pyc, and D. Gerling, “Permanent magnet loss reduction in pm-machines using analytical and fem calculation,” *SPEEDAM 2010*, pp. 98–100, 2010.
- [78] E. B. Agamloh, “A comparison of direct and indirect measurement of induction motor efficiency,” *IEEE International Electric Machines and Drives Conference*, pp. 36–42, 2009.
- [79] A. Lotov and K. Miettinen, *Multiobjective optimization: Interactive and evolutionary approaches*. Springer, 2008.
- [80] A. Cavagnino, G. Bramerdorfer, and J. A. Tapia, “A tutorial on bayesian optimization,” *arXiv: 1807.02811*, 2018.
- [81] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, no. 2, pp. 281–30, 2012.
- [82] D. Maclaurin, D. Duvenaud, and R. Adams, “Gradient-based hyperparameter optimization through reversible learning,” *Proceedings of the 39th International Conference on Machine Learning, PMLR*, pp. 2113–2122, 2015.
- [83] G. Bramerdorfer, J. A. Tapia, J. J. Pyrhönen, and A. Cavagnino, “Modern electrical machine design optimization: Techniques, trends, and best practices,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 10, pp. 7672–7684, 2018.
- [84] , “Optimization of electric machine designs–part i,” *IEEE Transactions on Industrial Electronics*, vol. 64, no. 12, pp. 9716–9720, 2017.
- [85] A. Cavagnino, G. Bramerdorfer, and J. A. Tapia, “Optimization of electric machine designs–part ii,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 2, pp. 1700–1703, 2017.

-
- [86] Z. P. Xia, Z. Q. Zhu, and D. Howe, "Analytical magnetic field analysis of halbach magnetized permanent-magnet machines," *IEEE Transaction on Magnetics*, vol. 40, no. 4, pp. 1864–1872, 2004.
- [87] P.-D. Pfister and Y. Perriard, "Very-high-speed slotless permanent-magnet motors: Analytical modeling, optimization, design, and torque measurement methods," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 1, pp. 296–303, 2010.
- [88] K. I. Laskaris and A. G. Kladas, "Permanent-magnet shape optimization effects on synchronous motor performance," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 9, pp. 3776–3783, 2010.
- [89] K. Yamazaki, and H. Ishigami, "Rotor-shape optimization of interior-permanent-magnet motors to reduce harmonic iron losses," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 1, pp. 61–69, 2010.
- [90] Y.-M. You, "Optimal design of pmsm based on automated finite element analysis and metamodeling, energies," *Energies*, vol. 12, no. 24, p. 4673, 2019.
- [91] W. Zhao, X. Wang, C. Gerada, H. Zhang, C. Liu, and Y. Wang, "Multi-physics and multi-objective optimization of a high speed pmsm for high performance applications," *IEEE Transaction on Magnetics*, vol. 54, no. 11, pp. 1–5, 2018.
- [92] L. Jolly, M. A. Jabbar, and L. Qinghua, "Design optimization of permanent magnet motors using response surface methodology and genetic algorithms," *IEEE Transaction on Magnetics*, vol. 41, no. 10, pp. 3928–3930, 2005.
- [93] J. Si, S. Zhao, H. Feng, R. Cao, and Y. Hu, "Multi-objective optimization of surface-mounted and interior permanent magnet synchronous motor based on taguchi method and response surface method," *Chinese Journal of Electrical Engineering*, vol. 4, no. 1, pp. 67–73, 2018.
- [94] S. Orlova, J. Auzins, V. Pugachov, A. Rassolkin, and T. Vaimann, "Response surface method for optimization of synchronous reluctance motor rotor," *Machines*, vol. 10, no. 10, p. 897, 2022.
- [95] C. Zhang, Z. Chen, Q. Mei, and J. Duan, "Application of particle swarm optimization combined with response surface methodology to transverse flux permanent magnet motor optimization," *IEEE Transaction on Magnetics*, vol. 53, no. 12, pp. 1–7, 2017.

- [96] S.-Y. Im et al., “Kriging surrogate model-based design of an ultra-high-speed surface-mounted permanent-magnet synchronous motor considering stator iron loss and rotor eddy current loss,” *IEEE Transaction on Magnetism*, vol. 58, no. 2, pp. 1–5, 2022.
- [97] M. Li, F. Gabriel, M. Alkadri, and D. Lowther, “Kriging surrogate model-based design of an ultra-high-speed surface-mounted permanent-magnet synchronous motor considering stator iron loss and rotor eddy current loss,” *IEEE Transaction on Magnetism*, vol. 52, no. 3, pp. 1–4, 2016.
- [98] Y. Wang et al., “Kriging-assisted multiobjective optimization of embedded magnetic encoder in pm synchronous machines,” *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–12, 2023.
- [99] H. Li, L. Cul, Z. Ma, and B. Li, “Multi-objective optimization of the halbach array permanent magnet spherical motor based on support vector machine,” *Energies*, vol. 13, no. 21, p. 5704, 2020.
- [100] X. Sun, J. Wu, G. Lei, Y. Cai, X. Chen, and Y. Guo, “Torque modeling of a segmented-rotor srm using maximum-correntropy-criterion-based lssvr fo torque calculation of evs,” *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 9, no. 3, pp. 2674–2684, 2021.
- [101] J. Wu, X. Sun, and J. Zhu, “Accurate torque modeling with pso-based recursive robust lssvr for a segmented-rotor switched reluctance motor,” *CES Transactions on Electrical Machines and Systems*, vol. 4, no. 2, pp. 96–104, 2020.
- [102] B. K. Bose, “Neural network applications in power electronics and motor drives-an introduction and perspective,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 14–33, 2007.
- [103] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv: 1609.04747*, 2016.
- [104] S. Jelassi and Y. Li, “An overview of gradient descent optimization algorithms,” *Proceedings of the 39th International Conference on Machine Learning, PMLR*, pp. 9965–10 040, 2022.
- [105] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *Proc. ICLR*, pp. 1–15, 2015.
- [106] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” *arXiv: 1904.09237*, 2019.

-
- [107] K. F. Man, K. S. Tang, and S. Kwong, “Genetic algorithms: Concepts and applications in engineering design,” *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, 1996.
 - [108] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: Past, present, and future,” *Multimedia Tools Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.
 - [109] J. Kennedy and R. Eberhart, “Particle swarm optimization,” *International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.
 - [110] T. M. Shami et al., “Particle swarm optimization: A comprehensive survey,” *IEEE Access*, vol. 10, pp. 10 031–10 061, 2022.
 - [111] F. Gillon and P. Brochet, “Screening and response surface method applied to the numerical optimization of electromagnetic devices,” *IEEE Transactions on Magnetics*, vol. 36, no. 4, pp. 1163–1167, 2002.
 - [112] G. Lei, J. Zhu, Y. Guo, C. Liu, and B. Ma, “A review of design optimization methods for electrical machines,” *Energies*, vol. 10, no. 12, p. 1962, 2017.
 - [113] N. K. Chauhan and K. Singh, “A review on conventional machine learning vs deep learning,” *IEEE International Conference on Computing, Power and Communication Technologies*, pp. 347–352, 2018.
 - [114] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
 - [115] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.
 - [116] R. Timofeev, “Classification and regression trees (cart) theory and applications,” *Humboldt University, Berlin*, vol. 54, p. 48, 2004.
 - [117] J. Wu, Y. Li, and Y. Ma, “Comparison of xgboost and the neural network model on the class-balanced datasets,” *IEEE International Conference on Frontiers Technology of Information and Computer*, pp. 457–461, 2021.
 - [118] C. Williams and C. Rasmussen, “Gaussian processes for regression,” *Advances in Neural Information Processing Systems*, pp. 514–520, 1995.
 - [119] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, “When gaussian process meets big data: A review of scalable gps,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

- [120] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, “A comprehensive survey on support vector machine classification: Applications, challenges and trends,” *Neurocomputing*, vol. 408, pp. 189–215, 2020.
- [121] P. E. Gill and E. Wong, “Methods for convex and general quadratic programming,” *Mathematical programming computation*, vol. 7, pp. 71–112, 2015.
- [122] J. Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*. 1998.
- [123] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
- [124] Y. Pan et al., “Evaluation of gaussian process regression kernel functions for improving groundwater prediction,” *Journal of Hydrology*, vol. 603, p. 126 960, 2021.
- [125] J. Wang, “An intuitive tutorial to gaussian process regression,” *Computing in Science & Engineering*, vol. 25, no. 4, pp. 4–11, 2023.
- [126] R. Wang, *Marginal and conditional distributions of multivariate normal distribution*. Available: <https://pages.hmc.edu/ruye/e161/lectures/gaussianprocess/node7.html>, 2006.
- [127] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, p. 386, 1958.
- [128] I. Goodfellow, Y. Bengio, and A. Courvill, *Deep learning*. MIT Press, 2016, vol. 1.
- [129] L. Alzubaidi, et al., “Review of deep learning: Concepts, cnn architectures, challenges, applications, future direction,” *Psychological Review*, vol. 8, no. 1, p. 53, 2021.
- [130] I. D. Mienye, T. G. Swart, and G. Obaido, “Recurrent neural networks: A comprehensive review of architectures, variants, and applications,” *Information*, vol. 15, no. 9, p. 517, 2024.
- [131] T. Lin, et al., “A survey of transformers,” *AI Open*, vol. 3, pp. 111–132, 2022.
- [132] J. Gui, et al., “A review on generative adversarial networks: Algorithms, theory, and applications,” *IEEE transactions on knowledge and data engineering*, vol. 35, no. 4, pp. 3313–3332, 2021.

-
- [133] K. Berahmand, et al., “Autoencoders and their applications in machine learning: A survey,” *Artificial Intelligence Review*, vol. 57, no. 2, p. 28, 2024.
 - [134] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, “A comprehensive survey of loss functions in machine learning,” *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, 2022.
 - [135] R. Rojas, “The backpropagation algorithm,” *Neural Networks*, Springer, 1996.
 - [136] 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.
 - [137] A. Fred Agarap, “Deep learning using rectified linear units (relu),” *arXiv: 1803.08375*, 2018.
 - [138] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *The International Conference on Learning Representations*, pp. 1–14, 2016.
 - [139] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” *Competition and Cooperation in Neural Nets*, pp. 267–285, 1982.
 - [140] Y. LeCun, et al., “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
 - [141] H. Gholamalinezhad and H. Khosravi, “Pooling methods in deep neural networks, a review,” *arXiv: 2009.07485*, 2020.
 - [142] M. A. El Mrabet, K. El Makkaoui, and A. Faize, “Supervised machine learning: A survey,” *2021 International Conference on Advanced Communication Technologies and Networking (CommNet)*, pp. 1–10, 2021.
 - [143] X. Yang, Z. Song, I. King, and Z. Xu, “A survey on deep semi-supervised learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 9, pp. 8934–8954, 2023.
 - [144] M.-R. Amini, V. Feofanov, L. Pauletto, L. Hadjadj, A. Devijver, and Y. Maximov, “Self-training: A survey,” *Neurocomputing*, vol. 616, pp. 128 904–8954, 2025.
 - [145] I. Goodfellow, et al., “Generative adversarial nets,” *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

- [146] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [147] Z. Ding, Y. Huang, H. Yuan, and H. Dong, “Introduction to reinforcement learning,” *Deep Reinforcement Learning: Fundamentals, Research and Applications*, Springer, pp. 47–123, 2020.
- [148] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *International Conference on Machine Learning*, vol. 0, no. 0, pp. 1861–1870, 2018.
- [149] B. Wu, S. Luo, and C. S. Suh, “A comprehensive review of propagation models in complex networks: From deterministic to deep learning approaches,” *arXiv: 2410.02118*, 2024.
- [150] A. Jain, et al., “Overview and importance of data quality for machine learning tasks,” *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3561–3562, 2020.
- [151] K. R. Davey, “Latin hypercube sampling and pattern search in magnetic field optimization problems,” *IEEE Transactions on Magnetics*, vol. 44, no. 6, pp. 974–977, 2008.
- [152] J. C. F. De Winter, S. D. Gosling, and J. Potter, “Pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data,” *Psychological Methods*, vol. 21, no. 3, p. 273, 2016.
- [153] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” *International Conference on Artificial Neural Networks*, pp. 270–279, 2018.
- [154] J. Yosinski, J. Clune, Y. Bengio and H. Lipson, “How transferable are features in deep neural networks?” *Advances in Neural Information Processing Systems*, 2014.
- [155] J. Qiao, X. Meng, W. Li, and B. M. Wilamowski, “A novel modular rbf neural network based on a brain-like partition method,” *Neural Computing and Applications*, vol. 32, pp. 899–911, 2020.
- [156] W. Wu, and P. Yun, “Adaptive modular convolutional neural network for image recognition,” *Sensors*, vol. 22, no. 15, p. 5488, 2022.

- [157] K. Cabello-Solorzano, et al., “Adaptive modular convolutional neural network for image recognition,” *International Conference on Soft Computing Models in Industrial and Environmental Applications*, pp. 344–353, 2023.
- [158] D. Barrar, “Cross validation,” *Encyclopedia of Bioinformatics and Computational Biology*, vol. 1, pp. 542–545, 2018.
- [159] L. Prechelt, “Early stopping—but when?” *Neural Networks: Tricks of the Trade*. Springer, pp. 53–67, 2012.
- [160] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, “Convolutional neural networks for medical image analysis: Full training or fine tuning?” *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.
- [161] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv: 1607.06450*, 2016.
- [162] K. Deb and H. Jain, “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints,” *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.
- [163] J. Blank and K. Deb, “Pymoo: Multi-objective optimization in python,” *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.
- [164] Z. Shen et al, “Towards out-of-distribution generalization: A survey,” *arXiv: 2108.13624*, 2021.
- [165] X. Wu, P. Wagner, and M. F. Huber, “Quantification of uncertainties in neural networks,” *Germany: Springer International Publishing*, pp. 276–287, 2023.
- [166] R. Mbuva, *Bayesian neural networks for short term wind power forecasting*. Institution of Computational Science and Technology, KTH Royal Institute of Technology, 2017.
- [167] D. Zhou et al., “Using bayesian deep learning for electric vehicle charging station load forecasting,” *Energies*, vol. 15, no. 17, p. 6195, 2022.
- [168] G. Tziolis et al., “Short-term electric net load forecasting for solarintegrated distribution systems based on bayesian neural networks and statistical post-processing,” *Energy*, vol. 271, no. 127018, 2023.
- [169] E. Goan and C. Fookes, “Bayesian neural networks: An introduction and survey,” *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair*, Springer, pp. 45–87, 2020.

- [170] L. Mosser, S. Purves, and E. Z. Naeini, “Deep bayesian neural networks for fault identification and uncertainty quantification,” *1st EAGE Digitalization Conference and Exhibition*, pp. 1–5, 2020.
- [171] N. Hauzenberger, F. Huber, K. Klieber, and M. Marcellino, “Enhanced bayesian neural networks for macroeconomics and finance,” *arXiv: 2211.04752*, 2021.
- [172] X. Chen, Y. Zhao, and C. Liu, “Medical image segmentation using scalable functional variational bayesian neural networks with gaussian processes,” *Neurocomputing*, vol. 500, pp. 58–72, 2022.
- [173] H. Sheng, J. Xiao, Y. Cheng, Q. Ni, and S. Wang, “Short-term solar power forecasting based on weighted gaussian process regression,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 1, pp. 300–308, 2018.
- [174] D. Jung, Q. Ahmed, and G. Rizzoni, “Design space exploration for powertrain electrification using gaussian processes,” *American Control Conference*, pp. 846–851, 2018.
- [175] G. Choi, G.-H. Jang, M. Choi, J. Kang, Y. G. Kang, and S. Kim, “Optimal design of a surface permanent magnet machine for electric power steering systems in electric vehicle applications using a gaussian process-based approach,” *Actuators*, vol. 13, no. 1, p. 13, 2023.
- [176] J. Chen et al., “Gaussian process kernel transfer enabled method for electric machines intelligent faults detection with limited samples,” *IEEE Transactions on Energy Conversion*, vol. 36, no. 4, pp. 3481–3490, 2021.
- [177] G. O. Sahinoglu, M. Pajovic, Z. Sahinoglu, Y. Wang, P. V. Orlik, and T. Wada, “Battery state-of-charge estimation based on regular/recurrent gaussian process regressions,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4311–4321, 2018.
- [178] Y. Zhang and X. Xu, “Fe-based superconducting transition temperature modeling through gaussian process regression,” *Journal of Low Temperature Physics*, vol. 202, no. 1, pp. 205–218, 2021.
- [179] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural networks,” *International Conference on Machine Learning*, pp. 1613–1622, 2015.
- [180] A. Graves, “Practical variational inference for neural networks,” *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

-
- [181] 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.
 - [182] Y. Gal, “Uncertainty in deep learning,” *Ph.D. dissertation, University Cambridge*, 2016.
 - [183] M. D. Hoffman, D. M. Blei, C. Wang, and J. W. Paisley, “Stochastic variational inference,” *Journal of Machine Learning Research*, vol. 14, pp. 1303–1347, 2012.
 - [184] F. Pedregosa et al., “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
 - [185] W. McKinney, “Pandas: A foundational python library for data analysis and statistics,” *Python for High Performance and Scientific Computing*, vol. 14, no. 9, pp. 1–9, 2011.
 - [186] E. Bingham et al., “Pyro: Deep universal probabilistic programming,” *Journal of Machine Learning Research*, vol. 20, no. 28, pp. 1–6, 2019.
 - [187] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.
 - [188] T. Duan, A. Anand, D. Y. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler, “Ngboost: Natural gradient boosting for probabilistic prediction,” *International Conference on Machine Learning*, pp. 2690–2700, 2020.
 - [189] H. Quan, D. Srinivasan, and A. Khosravi, “Uncertainty handling using neural network-based prediction intervals for electrical load forecasting,” *Energy*, vol. 73, pp. 916–925, 2014.
 - [190] M. Hewidy and O. Salem, “Integrating experimental modeling techniques with the pareto search algorithm for multiobjective optimization in the weld of inconel 718,” *The International Journal of Advanced Manufacturing Technology*, vol. 129, pp. 299–319, 2023.
 - [191] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente, “Direct multisearch for multiobjective optimization,” *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 1109–1140, 2011.

- [192] P. Offermann, H. Mac, T. T. Nguyen, S. Clénet, H. De Gersem, and K. Hameyer, “Uncertainty quantification and sensitivity analysis in electrical machines with stochastically varying machine parameters,” *IEEE Transactions on Magnetics*, vol. 51, no. 3, pp. 1–4, 2015.
- [193] B. Ma, J. Zheng, J. Zhu, J. Wu, G. Lei, and Y. Guo, “Robust design optimization of electrical machines considering hybrid random and interval uncertainties,” *IEEE Transactions on Energy Conversion*, vol. 35, no. 4, pp. 1815–1824, 2020.
- [194] G. Bramerdorfer and A.-C. Zavoianu, “Surrogate-based multi-objective optimization of electrical machine designs facilitating tolerance analysis,” *IEEE Transactions on Magnetics*, vol. 53, no. 8, pp. 1–4, 2017.
- [195] N. Xu, X. Sun, and C. H. Lee, “Hierarchical multiobjective optimization of spoke-type axial-flux pmsms using transfer learning and improved grey wolf algorithm,” *IEEE Transactions on Industrial Electronics*, 2026.
- [196] P. Cunningham, M. Cord, and S. J. Delany, “Supervised learning,” *Machine Learning Techniques for Multimedia*, pp. 21–49, 2008.
- [197] J. E. Van Engelen and H. H. Holger, “A survey on semi-supervised learning,” *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.
- [198] J. E. Van Engelen and H. H. Holger, “Instant-teaching: An end-to-end semi-supervised object detection framework,” *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4081–4090, 2021.
- [199] J. E. Van Engelen and H. H. Holger, “Semi-supervised bearing fault diagnosis and classification using variational autoencoder-based deep generative models,” *IEEE Sensors Journal*, vol. 21, no. 5, pp. 6476–6486, 2020.
- [200] N. Al-Azzam and I. Shatnawi, “Comparing supervised and semisupervised machine learning models on diagnosing breast cancer,” *Annals of Medicine and Surgery*, vol. 62, pp. 53–64, 2021.
- [201] B. Zhou, et al., “Short-term prediction of wind power and its ramp events based on semi-supervised generative adversarial network,” *International Journal of Electrical Power and Energy Systems*, p. 106411, 2021.
- [202] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” *International Conference on Machine Learning*, 2016.

-
- [203] M. T. Law, et al., “Closed-form training of mahalanobis distance for supervised clustering,” *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3909–3917, 2016.
- [204] A. Damianou and N.D. Lawrence, “Deep gaussian processes,” *Artificial Intelligence and Statistics*, pp. 207–215, 2013.
- [205] N. Srivastava, et al., “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [206] K. He, et al., “Deep residual learning for image recognition,” *IEEE Conference on Computer vision and Pattern Recognition*, pp. 770–778, 2016.
- [207] V. Bewick, L. Cheek, and J. Ball, “Statistics review 13: Receiver operating characteristic curves,” *Critical Care*, vol. 8, no. 6, pp. 1–5, 2004.
- [208] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
- [209] A. Tharwat and W. Schenck, “A survey on active learning: State-of-the-art, practical challenges and research directions,” *Mathematics*, vol. 11, no. 4, p. 820, 2023.
- [210] 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.
- [211] M. Abdar, et al., “A review of uncertainty quantification in deep learning: Techniques, applications and challenges,” *Information Fusion*, vol. 76, pp. 243–297, 2021.
- [212] J. Lee, W. Sung, and J.-H. Choi, “Metamodel for efficient estimation of capacity-fade uncertainty in li-ion batteries for electric vehicles,” *Energies*, vol. 8, no. 6, pp. 5538–5554, 2015.
- [213] S. Chib and E. Greenberg, “Understanding the metropolis-hastings algorithm,” *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.
- [214] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 741, no. 6, pp. 721–741, 1984.
- [215] M. D. Hoffman, A. Gelman et al., “The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

- [216] R. Chandra and J. Simmons, “Bayesian neural networks via mcmc: A python-based tutorial,” *IEEE Access*, vol. 12, pp. 70 519–70 549, 2024.
- [217] S. Minaee, et al., “Large language models: A survey,” *arXiv:2402.06196*, 2024.
- [218] J. Zhou et al., “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020.