DECOMPOSABLE NEURO SYMBOLIC REGRESSION WITH UNCERTAINTY AWARENESS

by

Giorgio L. Morales Luna

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

August 2025

ACKNOWLEDGEMENTS

This research was supported by the Data Intensive Farm Management project (USDA-NIFA-AFRI 2016-68004-24769 and USDA-NRCS NR213A7500013G021). Computational efforts were performed on the Tempest HPC System, operated by University Information Technology Research Cyberinfrastructure at MSU.

I would like to express my deepest gratitude to Dr. John Sheppard, my advisor and committee chair, for his guidance and mentorship throughout this intense and crucial chapter of my life. I am also grateful to my fellow members of the Numerical Intelligent Systems Laboratory at Montana State University for their valuable comments and suggestions. I extend my sincere thanks to the members of my thesis committee, Dr. Joseph Shaw, Dr. Sean Yaw, and Dr. Matthew Revelle, for their valuable feedback, time, and contributions to the development of this dissertation.

Finally, I am immensely thankful to my beloved wife, Flor, for her support, patience, and inspiration throughout the dissertation process, and for standing by me through the ever-changing and challenging years of my Ph.D. journey. I am also profoundly thankful to my parents, Jorge and Sandra, for their unconditional love and constant presence in my life; they have always been my foundation and greatest source of strength.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF TABLES – CONTINUED

Table Page

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

LIST OF FIGURES – CONTINUED

LIST OF FIGURES – CONTINUED

LIST OF ALGORITHMS

ABSTRACT

One of the fundamental goals of science is to discover laws that provide causal explanations for the observable world. Such discoveries may stem from distilling experimental data into analytical equations that allow interpretation of their underlying natural laws. This process is known as equation learning or symbolic regression (SR). However, most SR methods prioritize minimizing prediction error over identifying the governing equations, often producing overly complex or inaccurate expressions. Notably, they struggle to identify the functional form that explains the relationship between each variable and the system's response. To address this challenge, this dissertation presents a decomposable SR method that generates interpretable multivariate expressions by leveraging transformer models, genetic algorithms (GAs), and genetic programming (GP).

In particular, our interpretable SR method distills a trained "opaque" regression model[1] into mathematical expressions that serve as explanations of its computed function. It employs a Multi-Set Transformer model to generate multiple univariate symbolic skeletons that characterize how each variable influences the opaque model's response. The performance of the generated skeletons is evaluated using a GA-based approach to select a subset of high-quality candidates before incrementally merging them via a GP-based cascade procedure that preserves their original skeleton structure. The final multivariate skeletons undergo coefficient optimization via a GA. We evaluated our method on problems with controlled and varying degrees of noise, demonstrating lower or comparable interpolation and extrapolation errors compared to two GP-based and two neural SR methods. Unlike these methods, our approach consistently learned expressions that matched the original mathematical structure.

Complementing this effort, we explore the role of uncertainty quantification in enhancing symbolic model reliability. We investigate the use of prediction interval-generation neural networks to model total and potential epistemic uncertainty, and introduce an adaptive sampling strategy designed to minimize it. By integrating an uncertainty-aware sampling process guided by Gaussian process surrogates, we aim to reduce uncertainty not only in model predictions but also in the symbolic expressions extracted from them. This broader perspective highlights the importance of uncertainty awareness in SR, especially when symbolic models are intended for decision-making under limited or costly experimentation, such as in precision agriculture and other scientific domains.

[1] We prefer this term over "black-box": www.acm.org/diversity-inclusion/words-matter

CHAPTER ONE

INTRODUCTION

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them

- Sir William Bragg

Deep learning-based systems have gained significant attention and success in various domains due to their ability to learn complex non-linear functions. However, these systems are often described as "opaque models"[1] in the literature, emphasizing the high complexity of the functions they learn and their many required parameters. This aspect poses challenges in terms of interpretability and traceability from a human standpoint [105]. The term "opaque models" signifies that despite their accuracy in making predictions, these model's inner workings and decision-making processes remain obscure and difficult to comprehend. This lack of transparency limits their usefulness in applications that require a thorough understanding of the underlying processes that govern the studied phenomena.

Such understanding holds significant importance within the realm of physical sciences [22, 103]. Scientists employ a systematic approach that involves observing, refining, and testing models to understand and describe the behavior of phenomena and instantiations of the physical world. As such, one of the objectives is to obtain explanatory and causal models that shed light on the underlying mechanisms at play [143, 157]. The ultimate goal is to uncover and formalize general equations, laws, and parameterizations that govern these phenomena [22, 27]. Traditionally, scientific progress has relied on human expertise,

[1] We prefer this term over "black-box": www.acm.org/diversity-inclusion/words-matter

intuition, and theoretical frameworks to derive these fundamental principles [89]. However, the advent of algorithmic approaches presents a paradigm shift in scientific inquiry. These data-driven algorithms can identify patterns, relationships, and regularities that might elude human perception alone and are capable of extracting valuable insights and models directly from empirical observations [148], potentially leading to accelerated scientific progress and novel insights in the physical sciences and other disciplines.

1.1 Motivation

Motivation for this dissertation stems from three key areas: symbolic regression, uncertainty quantification, and precision agriculture, which are discussed below.

1.1.1 Symbolic Regression

Symbolic regression (SR) emerges as a powerful technique within the context of data-driven approaches for scientific inquiry. It offers a promising avenue for the automated discovery of explanatory and causal models from observed data, and an alternative to the use of opaque models. SR, also known as equation learning, aims to identify mathematical equations or symbolic expressions that capture the underlying relationships and dynamics of the studied phenomena [24]. One of the main advantages of the expressions learned by an SR model is that they can be easily analyzed by humans [49]. They allow for the identification of cause-effect relationships between the inputs and outputs of a system, which is one of the aims of the thriving area of explainable artificial intelligence (XAI) [105]. These techniques not only capture the behavior of empirical data through analytical equations, but also offer several practical advantages. They facilitate the incorporation of prior or expert knowledge during the learning process by specifying preferred operators or constraints [11]. Additionally, they reduce computational complexity during inference and often demonstrate stronger extrapolation capabilities compared to opaque models [113].

XAI aims to develop intelligent systems that can articulate the rationale behind their actions or recommendations in ways that are understandable to humans [105]. Explanations are essential for ensuring transparency [42], enhancing the reliability and credibility of opaque models [18], and, in certain contexts, are even regarded as a human right [192]. XAI generally encompasses two key approaches: explainability and interpretability.

Explainability refers to uncovering the internal logic and mechanisms of a machine learning model. For example, layer-wise relevance propagation (LRP) [10] decomposes a deep neural network's output into relevance scores for individual input features (e.g., image pixels), using the model's weights and activations to backtrack the influence of inputs through the network layers. *Interpretability*, in contrast, aims to help humans identify cause-effect relationships between the system's inputs and outputs. SR can serve both of these XAI goals. From an interpretability perspective, SR seeks to uncover mathematical relationships that reveal how input variables influence a system's response. From an explainability standpoint, SR can be used to approximate the internal function computed by an opaque model using transparent, human-readable expressions, thereby revealing its decision logic.

Symbolic regression methods have been successfully applied to real-world problems such as dynamical system modeling [161], astrophysics [35, 37], and fluid mechanics [154]. These methods have achieved performance metrics comparable to state-of-the-art opaque models, showcasing their potential in accurately capturing the complexities of observed data [139]. However, most existing SR approaches focus primarily on minimizing prediction error rather than extracting the true governing equations of a system [13].

This focus on predictive accuracy often results in generated equations that may exhibit high complexity, effectively approximating the observed data but failing to correspond to the underlying equations [24]. This tendency toward complexity over parsimony can hinder generalization, as the model may become excessively tailored to the training data, thereby limiting its ability to adapt to new, unseen scenarios.

SR is an NP-hard problem whose complexity grows with the number of observations, operators, and variables involved [176, 183]. As such, brute-force approaches become infeasible. Many SR methods are largely based on population-based algorithms, especially genetic programming (GP) [139]. Nevertheless, a notable drawback of GP-based SR methods is that they suffer from slow computation. The inherent complexity of the search space, coupled with the expensive iterative calls to numerical optimization routines after each generation, contributes to the computational inefficiency of GP-based methods [139]. Furthermore, these methods do not consider past experiences, as they require learning each problem from scratch. As such, the obtained models do not benefit from additional data or insights from different equations, hindering their capacity for improvement and limiting their generalization capabilities [15, 82].

In response to these limitations, recent advancements have seen the emergence of deep learning-based SR methods as a promising alternative. Some of these methods utilize pre-trained transformer neural network models that generate symbolic expressions through a single forward pass and, possibly, a single call to a numerical optimization routine [13, 15, 82, 179]. Notably, deep learning-based SR methods offer a substantial time speedup compared to GP-based approaches, overcoming the computational inefficiencies associated with the latter. Despite the time advantage, there remains a gap in terms of prediction accuracy between the two paradigms [82]. In addition, their end-to-end approach, which processes all variables simultaneously, often fails to capture the functional form between each variable and the system's response correctly [125].

1.1.2 Uncertainty Quantification

Motivated by the challenges discussed above, we recognize that symbolic regression benefits from robust uncertainty quantification techniques to enhance the accuracy and reliability of model discovery. For instance, uncertainty can be mitigated through suitable experimentation and informed data acquisition strategies. In general, effective uncertainty management is essential for enhancing trust in AI-powered systems and ensuring their practical adoption. This need is further emphasized by numerous reports indicating that current deep learning (DL) techniques often produce unstable predictions, which can arise unpredictably rather than solely in worst-case scenarios [32]. What is more, in many cases, the reliability of model predictions is as important as the predictions themselves, particularly in high-stakes environments where incorrect decisions may have significant consequences. As a result, DL models are frequently regarded as unreliable in applications where uncertainty is inherent in the data or the underlying system, including weather forecasting [197], electronic manufacturing [158], and precision agriculture [116]. In this context, reliability refers to a model's ability to perform consistently across real-world settings [173].

One of the limitations of conventional prediction models is that they only provide deterministic point estimates without any additional indication of their approximate accuracy [54]. Reliability and accuracy of the generated point predictions are affected by factors such as the sparsity of training data or target variables affected by probabilistic events [84]. One way to improve the reliability and credibility of such complex models is to quantify the uncertainty in the predictions they generate [167]. This uncertainty can be quantified using prediction intervals (PIs). PIs offer a bounded region, defined by an upper and a lower bound, within which a prediction is expected to lie with a given probability [85]. PIs not only add transparency to the model's output but also allow users to assess the degree of trust they can place in individual predictions.

We focus on the use of PIs for neural networks (NNs), as they are the opaque models employed throughout this dissertation. Obtaining PIs for NN predictions is particularly challenging since NNs lack an explicit mathematical formulation for uncertainty estimation. As a result, designing methods that enable NNs to produce high-quality PIs (i.e., intervals that are both sufficiently narrow and capture most of the probability density) is an open and non-trivial problem. Key challenges include designing suitable loss functions and architectural modifications that enable meaningful interval learning, and addressing the trade-off between prediction interval width and probability coverage.

One of the motivations for accurately quantifying uncertainty is that, by doing so, we may develop experimentation techniques that aim to reduce it. This is important given that, in various scientific and engineering fields, the development of accurate predictive models frequently relies on experimentation. Conducting these experiments can be costly and time-consuming, making it important to adopt strategies that extract the most valuable information from each experiment. Therefore, we focus on adaptive sampling (AS) techniques, which offer a promising solution by selecting samples intelligently that contribute most to improving model accuracy and reducing uncertainty [39].

The central idea is to guide data acquisition toward regions of the input space where predictions are less reliable, allowing the model to refine its understanding where it is most uncertain. This uncertainty-driven sampling is particularly relevant when dealing with epistemic uncertainty, the type of uncertainty arising from limited knowledge or insufficient data. Thus, in this dissertation, we examine how epistemic uncertainty, once quantified via prediction intervals, can be strategically reduced using AS. In addition, we aim to explore how AS can be used not only to reduce the uncertainty of predictions but also to improve the reliability of the symbolic expressions learned through SR.

1.1.3 Precision Agriculture

Another source of motivation arises from the field of Precision Agriculture (PA). PA is a management technique that leverages various Information Technologies (ITs) to gather spatial and temporal information from fields. This information aids in making informed management decisions, facilitating the improvement of crop and livestock management practices while promoting sustainability [60]. Accurate and reliable predictions are critical in PA, especially when advising decisions that directly affect profitability and environmental sustainability. For instance, selecting the optimal nitrogen fertilizer rate involves assessing how different rates affect crop yield, economic returns, and environmental impact [69].

Nevertheless, in real-world agricultural systems, such predictions are inherently uncertain due to variability in weather, soil conditions, and management practices. Thus, beyond accurate yield estimation, it is crucial to quantify the uncertainty associated with these predictions to assess the risk and confidence of each recommendation. Providing such uncertainty estimates helps farmers make informed, risk-aware decisions and avoid over- or under-application of inputs. To this end, we adapt our proposed techniques for prediction PI generation and adaptive sampling to address real-world problems in agriculture.

Moreover, our research focus on the analysis of nitrogen fertilizer-yield response (N-response) curves. An N-response curve is defined as a curve that exhibits the various values taken by the estimated crop yield to all admissible values of the N fertilizer rate (N-rate) [123]. These curves are largely used by agronomists to determine the economic optimum nitrogen rate (EONR), which represents the nitrogen rate at which crop yield increase is not large enough to pay for additional fertilizer [118]. The exploration of N-response curves is therefore of great importance for optimizing nitrogen fertilizer usage, improving agricultural efficiency, and maximizing economic returns for farmers.

Traditionally, the estimation of the EONR is achieved by assuming pre-selected parametric yield response functions and fitting them to observed crop yield data [20, 80, 188]. In addition, the issue of the variability in the functional form of the N-response curves within each field has received limited attention in previous studies [69]. The assumption that the N-response curves of all sites within a field correspond to a single functional form with the same parameters implies that the field is homogeneous and behaves similarly at every location. Nevertheless, recent works indicate that the functional form of the N-response curves exhibits variability across each field due to different terrain characteristics and soil composition [69, 130]. Based on this, we proposed in [123] an interpretable method that allows for the identification of the features with the greatest impact on the shape of N-response curves at each location of the field. This approach facilitates a site-specific understanding of the relevance of each feature (e.g., precipitation, terrain slope, terrain aspect) over the responsivity of the field to the fertilizer. However, it does not provide a thorough interpretation of the mathematical relationship between the input features and the field's responsivity to the N-rate. To the best of our knowledge, no method has been proposed that learns from data the mathematical functional forms that describe site-specific N-response curves, which at the same time are dependent on other features that may affect the yield response.

1.2 Research Questions

Based on the motivation discussed in the previous section, we formulate the core research questions of this dissertation.

1. Symbolic regression

    - How do we design an SR method that improves with experience and focuses on learning one or multiple underlying equations of a system?
    - How do we train a deep learning model to learn symbolic skeletons given multiple sets of input–response pairs, where all sets correspond to the same functional form but use different equation constants?
    - How do we merge univariate symbolic skeletons, where each describes the relationship between a single variable and the system's output, into a single multivariate equation that approximates the underlying equation of the system?

- How do end-to-end SR methods that predict the full multivariate expression directly compare to decomposable SR regression methods, which learn subexpressions as building blocks, in terms of their ability to learn underlying equations?

2. Uncertainty Quantification

   - How to produce high-quality PIs using NNs?
   - How to mitigate epistemic uncertainty using adaptive sampling techniques?
   - How does adaptive sampling support the discovery of mathematical expressions that are both accurate and certain?

3. Real-world application

   - How to quantify uncertainty in crop yield prediction?
   - How do SR methods learn parametric mathematical expressions that describe site-specific N-response curves?
   - How can AS techniques enhance experimental design for N rate selection, with the goal of reducing uncertainty and improving crop growth modeling?

1.3 Overview

This dissertation introduces a method called SeTGAP (**S**ymbolic R**e**gression using **T**ransformers, **G**enetic **A**lgorithms, and genetic **P**rogramming). Given a multivariate regression problem that can be expressed in terms of a mathematical equation, SeTGAP identifies univariate symbolic skeleton expressions for each explanatory variable, which are later merged to approximate the true underlying equation of the system. Note that a symbolic skeleton expression is an abstract representation of a mathematical expression that captures its structural form without identifying specific numerical values.

We hypothesize that SeTGAP generates functions that are more similar to the systems' underlying functions in comparison to other SR methods. This comparison is carried out by calculating the mean squared error (MSE) obtained by the learned functions using in-domain and out-of-domain data. Thus, a generated function closer to the system's underlying function is expected to yield lower MSE values when using in- and out-of-domain data.

Achieving high certainty in the mathematical expressions learned by SR methods is essential for enabling informed and trustworthy decision-making in scientific and engineering domains. However, the experimentation required to build such accurate models can be costly and time-consuming. Therefore, uncertainty quantification becomes a critical component of interpretable and actionable modeling. To address this, we introduce a method called **Dual A**ccuracy-**Q**uality-**D**riven (DualAQD) for generating high-quality prediction intervals using companion neural networks. One network focuses on minimizing the target error, while the other learns to generate reliable PIs. DualAQD allows not only to make accurate predictions but also to express a calibrated level of confidence around those predictions, which is particularly valuable when guiding decision-making or further data collection.

Building on this, we introduce an adaptive sampling approach aimed at reducing epistemic uncertainty in predictive models; that is, the component of total uncertainty that can be reduced by acquiring more information or improving the prediction model. Given that neural networks are the opaque models used throughout this dissertation, we propose a metric for estimating potential epistemic uncertainty by leveraging prediction interval-generation NNs. We hypothesize that our adaptive sampling approach, called ASPINN (**A**daptive **S**ampling with **P**rediction-**I**nterval **N**eural **N**etworks), reduces epistemic uncertainty more effectively compared to alternative methods. This comparison is carried out by estimating the mean epistemic uncertainty across the input domain after each iteration of the sampling process using synthetic problems that enable analytical quantification of uncertainty levels.

The proposed methods interact in an iterative learning cycle designed to improve both predictive certainty and functional interpretability, as illustrated in Fig. 1.1. Initially, an opaque predictive function $\hat{f}_{it}$ is trained using the available data $\mathcal{D}_{it} = \left(\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}\right)$ at iteration $it = 0$. To uncover the functional structure underlying $\hat{f}_{it}$, SeTGAP is employed to generate an interpretable expression $\tilde{f}_{it}$. To quantify the uncertainty associated with the opaque model predictions $\hat{f}_{it}(\mathbf{x})$, a companion prediction-interval model $\hat{f}_{\mathrm{PI},it}$, trained using DualAQD, generates calibrated prediction intervals $\hat{f}_{\mathrm{PI},it}(\mathbf{x})$. Building upon the generated PIs, ASPINN is used to identify regions of the input space where new samples would most effectively reduce epistemic uncertainty. New data points are then acquired, incorporated into the training set, and the process is repeated. It is hypothesized that, as the iterations progress and epistemic uncertainty is minimized, the symbolic expressions $\tilde{f}_{it}$ will converge toward more accurate and consistent mathematical models.

1.4 Contributions

This dissertation makes a number of contributions to the fields of symbolic regression and uncertainty quantification. Our specific contributions are summarized as follows:

- We present a decomposable SR method called SeTGAP that learns univariate skeleton subexpressions using a pre-trained transformer model. These subexpressions are then merged incrementally into multivariate expressions that approximate the system's underlying equation, employing GA- and GP-based techniques while preserving the originally identified skeleton structures.

- We introduce an SR problem called multi-set symbolic skeleton prediction (MSSP). It receives multiple sets of input–response pairs, where all sets share the same functional form but use different equation constants, and outputs a common skeleton expression.

- We present a novel transformer network model called "Multi-Set Transformer" to solve the MSSP problem. The model is pre-trained on a large corpus of synthetic

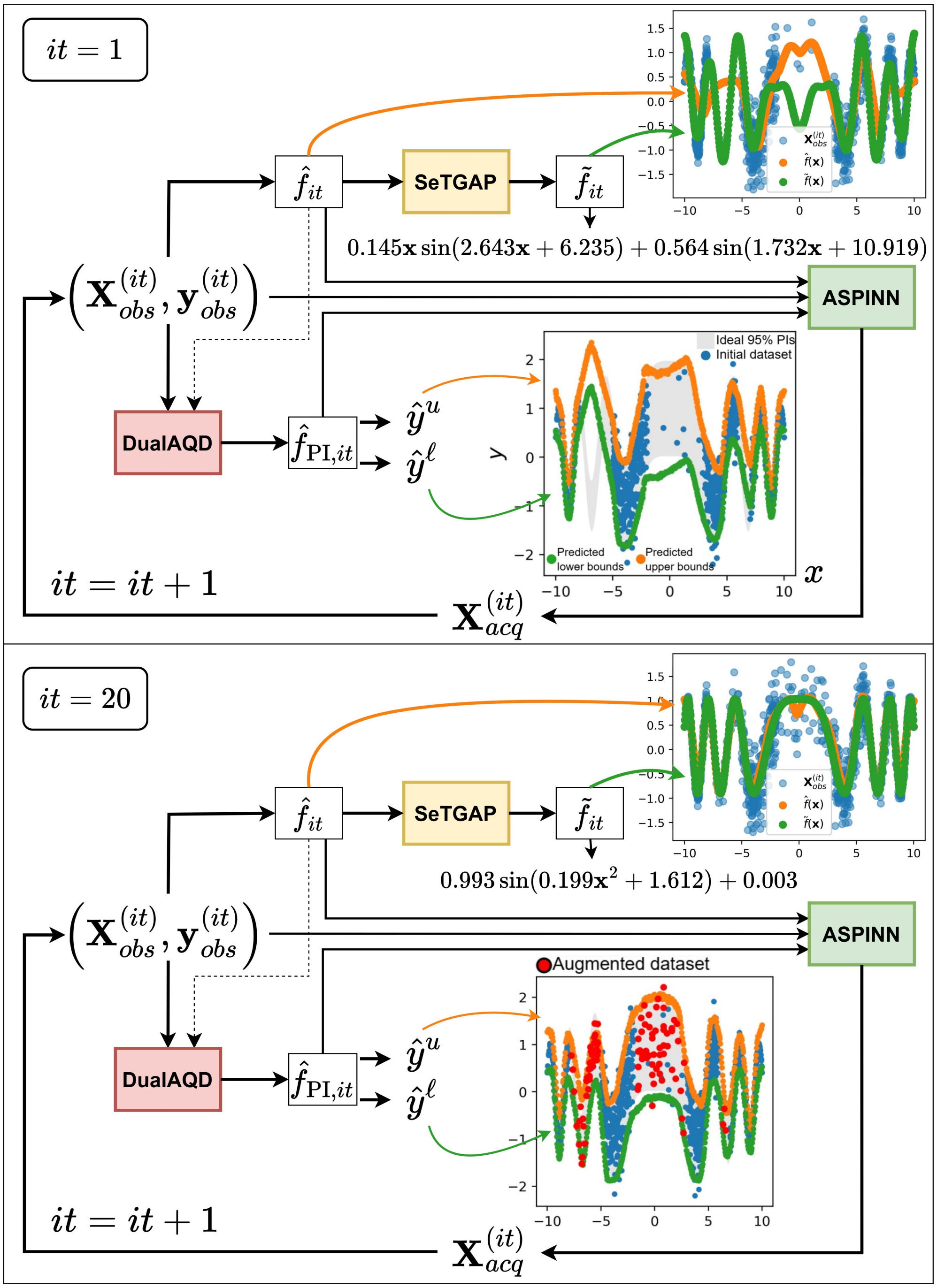


Figure 1.1: Overview of the proposed SR framework with uncertainty awareness.

symbolic skeleton expressions, from which training data is dynamically produced using a specialized data generation framework.

- We present a new method for assessing the accuracy of a given univariate skeleton expression in comparison to the underlying function.

- We present a loss function called DualAQD used to train a PI-generation NN. It is designed to solve a multi-objective optimization problem: minimizing the mean PI width while ensuring PI integrity using constraints that maximize the probability coverage implicitly.

- We introduce a PI-generation framework that uses two companion NNs: one that produces accurate target estimations, and another that generates high-quality PIs, avoiding the common trade-off between target estimation accuracy and quality of PIs.

- We present an AS method called ASPINN. At each iteration, it builds a Gaussian Process from calculated potential epistemic uncertainty levels. The GPR, a surrogate for the NN models, estimates potential epistemic uncertainty changes across the domain after sampling specific locations. An acquisition function then uses the GPR to select sampling locations, aiming to minimize global epistemic uncertainty.

- We introduce a novel metric based on NN-generated PIs to quantify potential levels of epistemic uncertainty across the input domain.

- We demonstrate that our PI-generation method, DualAQD, produces informative PIs that effectively capture spatial variations in uncertainty across agricultural fields.

- We apply SeTGAP during the AS process to assess how the learned mathematical expressions are affected under varying uncertainty levels.

- We apply SeTGAP in a crop production setting to enhance the understanding of how different regions of the field show different responsivity to the N fertilizer. Thus, we generate mathematical expressions that describe site-specific N-response curves.

- We demonstrate that our adaptive sampling approach, ASPINN, addresses the challenge of data acquisition in real agricultural environments effectively. Through simulations based on field-derived symbolic models, ASPINN converges faster to minimal epistemic uncertainty levels compared to competing methods.

The SeTGAP methodology has been described in the following article:

- Giorgio Morales and John W. Sheppard. Decomposable symbolic regression using Multi-Set Transformers and genetic programming *(submitted)*, 2025

Our univariate skeleton generation method using a novel Multi-Set Transformer has been published in the following article:

- Giorgio Morales and John W. Sheppard. Univariate skeleton prediction in multivariate systems using transformers. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 107–125, Vilnius, Lithuania, 2024

Our articles related to uncertainty quantification are listed below, with their intersection with SR to be explored in future work:

- Giorgio Morales and John W. Sheppard. Adaptive sampling to reduce epistemic uncertainty using prediction interval-generation neural networks. In *AAAI Conference on Artificial Intelligence*, volume 39, pages 19546–19553, 2025

- Giorgio Morales and John W. Sheppard. Dual accuracy-quality-driven neural network for prediction interval generation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(2):2843–2853, 2025

The list of published articles related to the PA application is as follows:

- Giorgio Morales and John W. Sheppard. Counterfactual analysis of neural networks used to create fertilizer management zones. In *International Joint Conference on Neural Networks*, Yokohama, Japan, June 2024

- Giorgio Morales and John W. Sheppard. Counterfactual explanations of neural Network-Generated response curves. In *International Joint Conference on Neural Networks*, Queensland, Australia, June 2023

- Giorgio Morales, John W. Sheppard, Paul Hegedus, and Bruce D. Maxwell. Improved yield prediction of winter wheat using a novel two-dimensional deep regression neural network trained via remote sensing. *Sensors*, 23(1):489, January 2023

- Giorgio Morales, John W. Sheppard, Amy Peerlinck, Paul Hegedus, and Bruce D. Maxwell. Generation of site-specific nitrogen response curves for winter wheat using deep learning. In *International Conference on Precision Agriculture*, 2022

- Giorgio Morales and John W. Sheppard. Two-dimensional deep regression for early yield prediction of winter wheat. In *SPIE Future Sensing Technologies 2021*, volume 11914, pages 49–63, November 2021

1.5 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we cover the necessary information to make the reader familiar with the various topics discussed in this work, including transformer neural networks, genetic algorithms, and genetic programming.

Chapter 3 introduces our Multi-Set Transformer network for solving the multi-set skeleton prediction problem. We provide details on the architecture of the model and present detailed insights into the equation generation and data sampling methodologies used to pre-train the model. In Chapter 4, we focus on SeTGAP, our novel multivariate SR method based on transformers, GAs, and GP. In particular, we provide details on the use of the pre-trained Multi-Set Transformer for univariate skeleton prediction, followed by an evolutionary-based technique to merge the generated skeletons into multivariate expression candidates. Furthermore, we describe the experimental results comparing the performance of SeTGAP with that of other GP-based and neural SR methods

Chapter 5 addresses uncertainty quantification and adaptive sampling techniques aimed at improving the reliability of prediction models across the input domain. It begins by introducing our dual-network approach, DualAQD, for generating high-quality PIs. Building on these PIs, the chapter presents ASPINN, an adaptive sampling strategy that leverages PI-based estimates of epistemic uncertainty to guide data acquisition. The chapter includes experiments on synthetic and benchmark datasets to evaluate the quality of the generated PIs, as well as controlled simulations to assess ASPINN's effectiveness in reducing uncertainty compared to baseline methods. The chapter concludes with an investigation of how varying uncertainty levels influence the symbolic expressions learned by SeTGAP.

Chapter 6 presents our real-world application in precision agriculture. We begin by defining key agricultural concepts relevant to our study, including the On-Farm Precision Experimentation framework, crop yield prediction, and fertilizer management zones. Here, we employ our PI-generation method, DualAQD, to construct uncertainty maps that highlight regions of the field with varying levels of predictive uncertainty. We then apply SeTGAP to learn site-specific nitrogen response curves, using a clustering-based approach to define fertilizer management zones. Leveraging the previously derived symbolic skeleton expressions, we generate a simulated field environment that enables the evaluation of different sampling strategies, including our adaptive sampling method, ASPINN. Finally, we briefly conclude and discuss future work in Chapter 7.

CHAPTER TWO

BACKGROUND

In this chapter, we introduce the main concepts and methods that form the foundation for the work presented in later chapters.

2.1 Transformer

Transformers are a type of deep learning architecture introduced by Vaswani *et al.* [181]. They revolutionized various natural language processing tasks [21, 90], and became the foundation for many subsequent advances in the field. Their success and effectiveness in handling sequential data have led to their adoption and impact in various other domains, such as computer vision [46], speech and audio processing [87], and drug discovery [66].

At its core, the transformer architecture is designed to process sequential data, such as sentences in natural language, by leveraging the concept of self-attention mechanisms. Traditional sequence-to-sequence models like recurrent neural networks (RNNs) [28] process input sequentially, which can lead to difficulties in handling long-range dependencies and can be computationally inefficient due to their sequential nature. Conversely, the transformer addresses these issues by employing a multi-head self-attention mechanism. This mechanism enables the model to weigh the importance of different positions in the input sequence when encoding each symbol or token in the sequence. This allows the model to focus on relevant parts of the input and capture dependencies between distant tokens effectively.

2.1.1 General Architecture

The original transformer model [181] consists of an encoder–decoder architecture, as shown in Figure 2.1. The encoder maps an input sequence of symbol representations $\mathbf{x} = \{x_1, \ldots, x_n\}$ (where n is the length of the sequence) to a sequence of d-dimensional latent

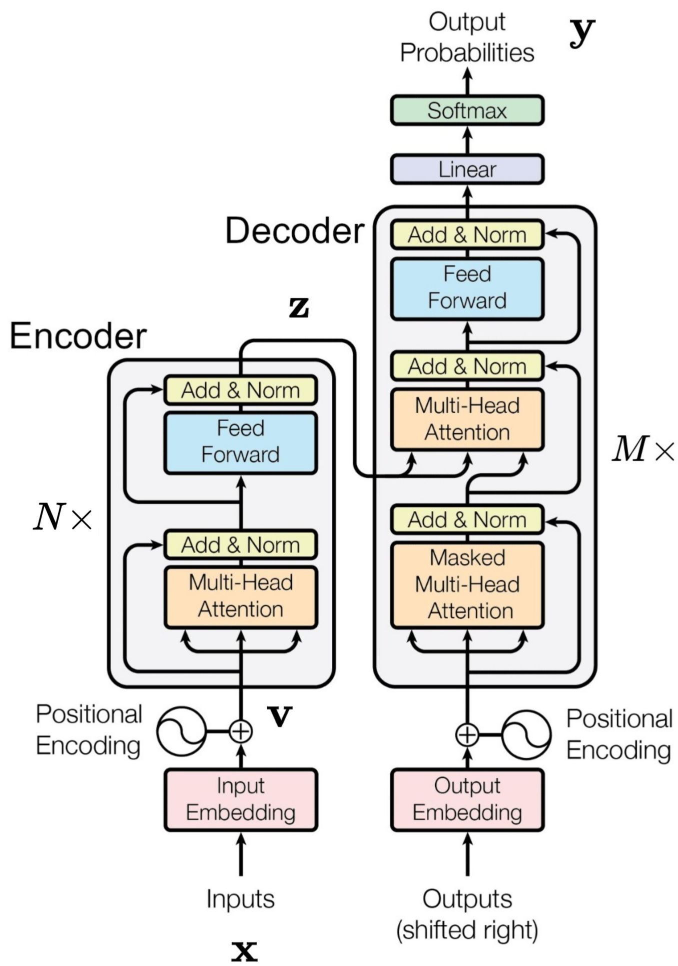


Figure 2.1: Transformer model architecture [181].

continuous representations $\mathbf{z} = \{z_1, \ldots, z_d\}$. Note that n is not fixed and it may vary for each input sequence. Then, $\mathbf{z}$, also known as the context vector, is processed by the decoder to generate the output sequence $\mathbf{y} = \{y_1, \ldots, y_m\}$ (i.e., the context vector length d and the output length m may differ from the input length n) one element at a time. The output generation process is auto-regressive [65], which means that the model generates the first token y_1 based on the input and then uses the generated token as input to predict the next token y_2, and so on until the entire output sequence is generated. Additional details about the encoder and the decoder, as well as other blocks shown in Figure 2.1 (e.g., input embedding and multi-head attention), are provided in the subsequent subsections.

2.1.2 Input Embedding and Positional Encoding

On the left side of Figure 2.1, the model inputs are affected by two processes before being fed into the encoder; i.e., input embedding and positional encoding. The former transforms each token x_i into a d_{in}-dimensional embedding vector using an embedding matrix $E \in \mathbb{R}^{d_{in} \times N_V}$, where N_V is the size of the vocabulary. The embedding for token x_i is denoted as $e_i \in \mathbb{R}^{d_{in}}$. Essentially, a vocabulary is a set of all unique words or tokens that appear in the dataset being processed. Its purpose is to assign a unique numerical identifier (index) to each token, allowing the transformer model to work with discrete representations of data. The embedding matrix E is learned during training in such a way that similar tokens are represented closer to each other in the embedding space. For example, in the context of natural language processing, the use of learned embeddings allows the model to discern and capture aspects of grammatical and syntactical structures within sentences [119].

On the other hand, positional embedding is required due to the fact that transformers lack inherent positional information, unlike traditional approaches such as RNNs. Thus, positional encoding vectors are added to the input embeddings e_i to distinguish tokens' positions in the original sequence. In the standard positional encoding, such as the one used by Vaswani *et al.* [181], the positional encoding vector for each token in the input sequence is determined based on a fixed mathematical function:

$$PE(pos, 2i) = \sin\left(\frac{pos}{L^{2i/d_{in}}}\right),$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{L^{2i/d_{in}}}\right),$$

where pos denotes the position of the token in the input sequence, i is the dimension index of the positional encoding vector, d_{in} is the dimensionality of the model input, and L is a scaling factor, commonly set to 10,000 in practice. We denote the vector we obtain after applying input embedding and positional encoding as $\mathbf{v} = \{v_1, \ldots, v_n\}$, where $v_i \in \mathbb{R}^{d_{in}}$.

Nevertheless, instead of using a fixed mathematical function to compute the positional encoding vectors, the encoding vectors could be learned as part of the training process. In other words, the model itself learns the positional encoding values that best suit the task and the data it is trained on. Learning positional encoding involves treating the positional encoding vectors as learnable parameters. During training, the model updates the positional encoding vectors along with other parameters using backpropagation and gradient descent techniques. By allowing the model to learn the positional encodings, it can adapt to the specific patterns and dependencies present in the data, potentially leading to improved performance on the task at hand [61].

2.1.3 Attention

The attention mechanism is the component responsible for capturing relationships between different positions in the input sequence. The key idea behind this mechanism is to compute attention weights that indicate the importance of each position with respect to a given query position. The attention mechanism is based on the use of queries, keys, and values. Queries are used to extract information from other tokens, keys determine the relevance of each token to the queries, and values provide the context-aware representations for each token. These components mirror the process of information retrieval in systems like search engines, where queries are used to retrieve relevant information (values) based on their similarity to the search terms (keys).

In particular, transformers use self-attention mechanisms; that is, the keys, values, and queries come from the same place [104]. This allows the model to capture dependencies and relationships between different positions within the same input sequence. Consider the input vector $\mathbf{v}$; that is, the resulting vector resulting after applying input embedding and positional encoding to the input sequence $\mathbf{x}$. To perform attention, the sequence $\mathbf{v}$ is transformed into three different representations: queries (Q), keys (K), and values (V). These transformations are done using learnable linear projections such that:

$$Q = \mathbf{v} \times W^Q,$$

$$K = \mathbf{v} \times W^K,$$

$$V = \mathbf{v} \times W^V,$$

where $W^Q \in \mathbb{R}^{d_{in} \times d_k}$, $W^K \in \mathbb{R}^{d_{in} \times d_k}$, $W^V \in \mathbb{R}^{d_{in} \times d_v}$ are learnable weight matrices. Notice that the linear projections obtained for the queries and keys have the same dimension d_k, while the dimension of the values, d_v, is not necessarily the same.

The attention scores are calculated by measuring the similarity between the queries and keys. This is carried out using a scaled dot-product, which consists of computing the dot products of the queries with all the keys, followed by a scaling operation to avoid overly large values. Then, the softmax function is applied along the rows to obtain normalized attention weights for each token in the input sequence. Note that the softmax activation function defined for each element of a vector $\mathbf{q} = \{q_0, ..., q_c\}$ is defined as:

$$\sigma(q_i) = \frac{e^{q_i}}{\sum_j e^{q_j}}$$

so that it produces a valid probability distribution over the vector elements. Finally, the scaled dot-product combines the attention scores with the values V to produce the final attention output:

$$\text{Attention}(Q, K, V) = \sigma\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Furthermore, instead of using a single self-attention function, Vaswani *et al.* [181], introduced the concept of multi-head self-attention. This is an extension of the standard attention mechanism that allows the model to attend to different semantic information from

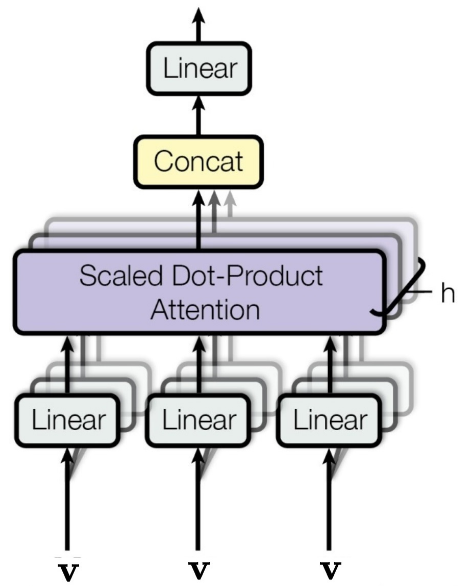


Figure 2.2: Multi-head self-attention [181].

different subspaces of the input data. It requires multiple sets of learnable query, key, and value linear projections, known as "heads," which operate in parallel. Let Q_i, K_i, and V_i denote the query, key, and value matrices obtained for the i-th head, and $W_i^Q \in \mathbb{R}^{d_{in} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{in} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{in} \times d_v}$ denote their corresponding linear projections. Thus, the output of the i-th head, head_i, is calculated as follows:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) = \text{Attention}(\mathbf{v} \times W_i^Q, \mathbf{v} \times W_i^K, \mathbf{v} \times W_i^V). \tag{2.1}$$

Finally, the outputs from all the attention heads are concatenated and linearly transformed to generate the final multi-head attention output, as depicted in Figure 2.2:

$$\text{MultiHead}(\mathbf{v}) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O, \tag{2.2}$$

where h is the total number of heads, and $W^O \in \mathbb{R}^{hd_v \times d_{in}}$ is a learnable projection matrix.

2.1.4 Encoder and Decoder Stacks

The encoder consists of a stack of N identical blocks, as illustrated in Figure 2.1. The encoder block starts with a multi-head self-attention layer, as described in Eq. 2.2. The output of the self-attention layer is added to the original input embeddings to form a residual connection [68], which helps mitigate the vanishing gradient problem during training. Layer normalization [9] is then applied to normalize the summed outputs.

The next component in the encoder block is a position-wise feedforward neural network. It processes each position produced after layer normalization independently through two linear transformations separated by a non-linear activation function such as the Rectified Linear Unit (ReLU), which is defined by $\texttt{ReLU}(\mathtt{q}) = \max(\mathtt{0}, \mathtt{q})$. Thus, the function of a position-wise feedforward network given the element q that belongs to the input vector $\mathbf{q}$ can be written as:

$$\mathrm{FFN}(q) = \max\left(0, q\, W_1 + b_1\right) W_2 + b_2,$$

where W_1 and W_2 are learnable weight matrices, and b_1 and b_2 are learnable bias vectors associated with the first and second linear transformations, respectively. Similar to the previous step, the output of the feedforward network is added to the previous output and normalized using layer normalization.

Similarly, the decoder block is composed of M identical blocks. While each encoder block contains two main layers (i.e., multi-head self-attention and a position-wise feedforward network), each decoder block includes a third layer called "encoder–decoder attention" that allows us to attend to the representations generated by the encoder stack. This layer emulates the common encoder–decoder attention mechanisms found in sequence-to-sequence models [194]. The decoder block starts with a masked multi-head self-attention layer. Unlike the encoder, the decoder's self-attention mechanism is "masked" to prevent the attention from attending to future positions in the output sequence during training. The masking ensures that the model can only attend to tokens that have already been generated in the output sequence up to the current position, maintaining the autoregressive property of the generation process we mentioned in Sec. 2.1.1. The masked multi-head self-attention layer is followed by the encoder–decoder attention layer, which, in turn, is followed by the position-wise feedforward neural network. Similar to the case of the encoder block, each layer in the decoder block is affected by a residual connection and layer normalization.

2.2 Genetic Algorithms

Genetic algorithms (GAs) are a class of optimization algorithms inspired by the concept of "survival of the fittest" [71]. They are particularly useful for solving complex optimization problems where traditional methods may be less effective. For instance, when derivatives are unavailable and the fitness landscape suffers from ill-conditioned parts, which are regions where small changes in the input variables lead to significant changes in the objective function, making it more challenging to find an optimal solution. In this section, we delve into the fundamental principles of genetic algorithms, including the key components and detailed explanations of each step in the algorithmic framework.

Genetic algorithms operate on a population of potential solutions represented as individuals or chromosomes. Each chromosome encodes a candidate solution to the problem at hand. The algorithm evolves this population over successive generations, applying genetic operators such as selection, crossover, and mutation to create offspring that can replace all or part of the population and improve the overall performance iteratively. This process is depicted in Figure 2.3.

2.2.1 Representation and Initialization

The choice of representation for individuals in a GA is a critical decision that influences the algorithm's performance [101]. The representation should be designed to effectively

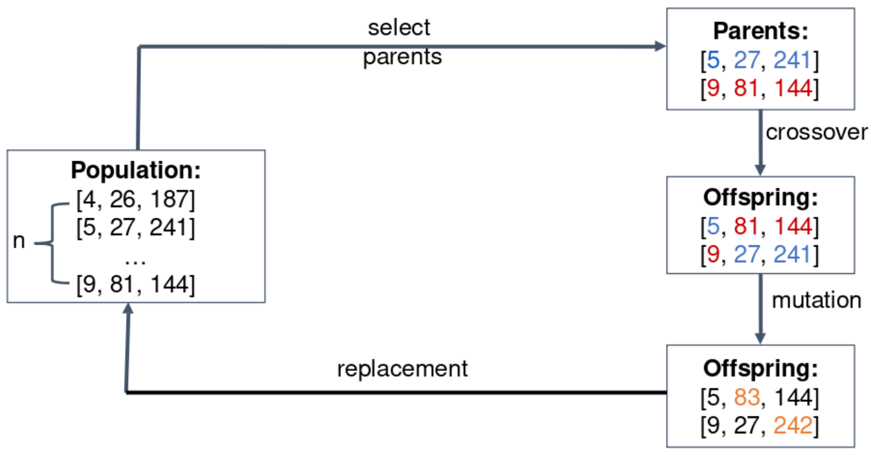


Figure 2.3: High-level flowchart of the genetic algorithm using one-point crossover and randomized mutation [144].

capture the problem's structure and characteristics. Common representations include:

- Binary Strings: Used for problems with binary decision variables, where each bit in the string represents a variable's value (0 or 1).
- Real-Valued Vectors: Suitable for continuous optimization problems, where each element in the vector corresponds to a variable's value.
- Permutations: Ideal for problems involving sequences, such as the traveling salesperson problem [52, 138], where the order of elements matters.
- Tree Structures: Applied to problems that can be represented as hierarchical structures, such as the mathematical expression trees in symbolic regression [23, 191] (see Sec. 2.3).

Furthermore, the process of initializing the population sets the stage for the GA's exploration. An appropriate initialization method should strike a balance between diversity and quality of the individuals of the population. Common approaches include random initialization, where individuals are generated with random values, and heuristic-based initialization, where prior knowledge of the problem is used to create initial solutions [52].

2.2.2 Fitness Evaluation

The fitness function quantifies how well an individual solution performs concerning the problem's objectives. The fitness function encapsulates domain-specific knowledge and defines the optimization problem. In practice, designing an appropriate fitness function is often the most challenging aspect of applying GAs to real-world problems, as it should accurately reflect the problem's goals and constraints.

Many strategies prioritize minimizing the number of fitness function evaluations. The computational cost of a GA is typically assessed by the number of fitness function calls needed to achieve an optimal or sufficiently accurate solution [94]. This focus on minimizing evaluations is crucial when calls are resource-intensive, such as generating complex construction elements or executing time-consuming simulation models to assess GA-generated parameters [152]. Thus, efficient use of fitness function evaluations is essential in optimizing computational processes.

2.2.3 Selection

In order to facilitate the convergence towards optimal solutions, it may be beneficial to choose the top-performing offspring solutions to become parents in the next generations. This process is known as selection and relies on the fitness values within the population. Common selection strategies include:

- Roulette wheel selection: Individuals are selected with probabilities proportional to their fitness scores.
- Tournament selection: Randomly chosen subsets of individuals to compete, and the winners become parents.
- Rank-based selection: Individuals are ranked by their fitness, and selection is based probabilistically on their rank.

These strategies seek a balance between favoring high-fitness individuals (exploitation) and exploring the diversity of the population.

2.2.4 Crossover

Crossover, also known as recombination, simulates genetic recombination observed in biological organisms [172]. It is a fundamental genetic operator in GAs. Different crossover operators define how genetic material from two or more parent individuals is combined to generate offspring. The most widely used crossover operators are the following:

- One-Point Crossover: A random crossover point is chosen, and genetic material is swapped between parents at that point. The intuition behind using such an operator is that both parents could present effective segments of solutions, and when combined, they might surpass the performance of the original parents. One-point crossover can be represented as:

$$\text{Offspring}_1 = \text{Parent}_1[: k] + \text{Parent}_2[k :]$$
$$\text{Offspring}_2 = \text{Parent}_2[: k] + \text{Parent}_1[k :],$$

  where k is a randomly chosen crossover point.

- Two-Point Crossover: Two random crossover points are selected, and the genetic material between these points is exchanged.

- Binomial/Multinomial Crossover: Genetic material is exchanged randomly, with each gene having an equal probability of being inherited from either parent.

The choice of crossover operator depends on the problem structure and the desired balance between exploration and exploitation.

2.2.5 Mutation

Mutation introduces small, random changes in the population's individuals. The strength of this disturbance is called mutation rate. It is a critical source of genetic diversity within the population and reduces premature convergence to suboptimal solutions. Mutation operators are specific to the chosen representation.

In binary strings, a mutation operation might flip a bit at a random position, while in real-valued vectors, it might add a small random value to a component. The bit flip mutation can be expressed as:

$$x_i[b] = \begin{cases} 1 - x_i[b], & \text{with probability } P_{\text{mutate}} \\ x_i[b], & \text{with probability } 1 - P_{\text{mutate}} \end{cases},$$

where $x_i[b]$ represents the b-th bit of the i-th individual of the population. In addition, P_{mutate} represents the mutation rate, which controls the likelihood of mutation occurring.

Mutation is vital for exploring the solution space and can be particularly useful when dealing with rugged fitness landscapes, which present multiple local minima, creating complex and challenging optimization environments where traditional optimization methods struggle. Note that mutation operators should avoid bias. That is, in unconstrained solution spaces without plateaus, the mutation operator should not introduce any directional bias, ensuring a neutral exploration [94]. However, in the context of constrained solution spaces, a certain degree of bias can be beneficial [93].

2.2.6 Replacement

Replacement determines how the next generation is formed based on parents and offspring. Various replacement strategies exist, each influencing the GA's exploration and exploitation balance. Common replacement strategies include [40]:

- Generational Replacement: The entire population is replaced by the offspring. It promotes exploration as the entire population is renewed.

- Steady-State Replacement: A subset of the population is replaced by the offspring, maintaining some continuity with the previous generation. It can be useful for preserving good solutions.

Some GAs incorporate elitism, where a fraction of the best-performing individuals from the current generation is preserved in the next generation to ensure that high-quality solutions are not lost.

2.2.7 Termination

The termination condition defines when the GA concludes its execution. Proper termination criteria are essential for controlling the algorithm's runtime and ensuring convergence. Some of the most common termination criteria are given below:

- Maximum Number of Generations: The algorithm stops after a predefined number of generations. Note that the cost of fitness function evaluations can impose limitations on the duration of the optimization process.

- Fitness Threshold: If a solution with a fitness exceeding a certain threshold is found, the algorithm terminates.

- Stagnation: The algorithm terminates if the fitness of the population does not significantly change over a certain number of generations.

- Time Limit: The algorithm stops after running for a specified duration.

2.3 Genetic Programming

Genetic Programming (GP) is an evolutionary algorithm that extends the principles of GAs to evolve computer programs or mathematical expressions to solve problems. GP was invented by John Koza in 1988 [91, 92], representing a pioneering step in the field of automated program synthesis. The problem of symbolic regression was introduced by Koza [92] as an application of GP where the objective is to discover mathematical expressions that best fit observed data automatically. In this context, the programs to be optimized are syntax trees consisting of functions and operations over input features and constants [191].

Below, we provide details of the fundamental components of Genetic Programming. Note that certain components, such as selection, replacement, and fitness, remain essentially the same as those explained in the previous section and are therefore not elaborated on here.

2.3.1 Representation and Initialization

In GP, solutions are commonly represented as hierarchical tree structures, where each tree corresponds to a computer program. The variables and constants in a program serve as the endpoints of the tree structure and are referred to as "terminals". On the other hand, internal nodes within the tree represent mathematical operations and are known as "non-terminals". Collectively, the permissible functions and terminals constitute what is termed the primitive set within a GP system. In addition, it is worth noting that there exist alternative representations in the form of linear GP, where solutions are expressed as linear sequences. In this alternative paradigm, variables, constants, and mathematical operations are arranged linearly, offering a different perspective on solution encoding. In this work, we represent tree expressions in "prefix" notation, which is a common choice in the GP literature [151]. For example, the mathematical expression $3x + \sqrt{x+2}$, given in the common "infix" notation, would be represented as $\{\texttt{add}, \texttt{mul}, \texttt{3}, \texttt{x}, \texttt{sqrt}, \texttt{add}, \texttt{x}, \texttt{2}\}$ in the prefix notation.

GP initiates the process with a population of randomly generated trees. The population size and the depth of these initial trees significantly influence the exploration of the solution space. The depth of a node is defined as the count of edges that must be traversed when starting from the tree's root node to reach that particular node; thus, a tree's depth corresponds to the depth of its deepest leaf node. The following are some of the most commonly used initialization techniques:

- Full Initialization: This method creates trees of a fixed depth, ensuring that all branches of the tree are fully expanded [92].

- Grow Initialization: This method adds nodes (i.e., terminals or non-terminals) to the tree recursively until a branch selects a terminal or reaches the specified maximum depth. This allows for the possibility of incomplete branches, which refers to portions of the tree that may not extend to the maximum depth, resulting in variable tree structures. This technique encourages the creation of diverse and variable tree structures, facilitating a more extensive exploration of the solution space [92].

- Ramped Half-and-Half Initialization: This technique combines both Full and Grow initialization methods. It generates individuals with depths ranging from minimal to a predefined maximum. Half of the population is created using the full method, and the other half uses the grow method. This approach provides a better balance between exploration and exploitation [92].

- Semantic Initialization: Semantic initialization focuses on generating individuals that are already functional or close to functional. It uses domain-specific knowledge to create initial individuals, enhancing the chances of generating high-quality solutions from the beginning [120].

2.3.2 Crossover

Genetic Programming (GP) distinguishes itself from other evolutionary algorithms primarily through its unique implementation of crossover and mutation operators. Subtree crossover is the most prevalent form in GP, where two parents are involved. Each parent's tree undergoes random independent selection of a crossover point, defined by a node of the expression tree. Subsequently, an offspring is created by substituting the subtree rooted at the crossover point in one parent with a copy of the corresponding subtree from the other parent, as shown in Figure 2.4. This procedure maintains the integrity of the original individuals, allowing them to contribute to multiple offspring programs if selected repeatedly. Crossover points are often chosen non-uniformly, favoring functions over leaves to counter the tendency of exchanging minimal genetic material [92].

Sub-tree crossover faces issues that constrain its effectiveness. For instance, it often results in offspring with vastly different behaviors compared to their parents. This is due to the exchanged sub-trees originating from different positions, and having varying sizes, shapes, and functionalities. Such significant changes in input context can lead to poor solutions [108]. Additionally, sub-tree crossover is related to code bloat; that is, the uncontrollable increase in the average tree size during evolution without substantial improvements in fitness [4]. Alternative techniques such as biased sampling and semantic crossover [177] have been proposed. Semantic crossover aims to maintain the semantics or functionality of the parent programs to a higher degree. It attempts to select subtrees that, when exchanged, retain similar functionality. This is achieved through various mechanisms, such as choosing crossover points accordingly. Biased sampling refers to a selective approach where certain parts of parent trees are more likely to be chosen for crossover, often based on their fitness or other heuristics, aiming to guide the genetic operators toward more promising regions of the search space. This strategy enhances the exploitation of high-performing building blocks during crossover, potentially improving the convergence speed and the quality

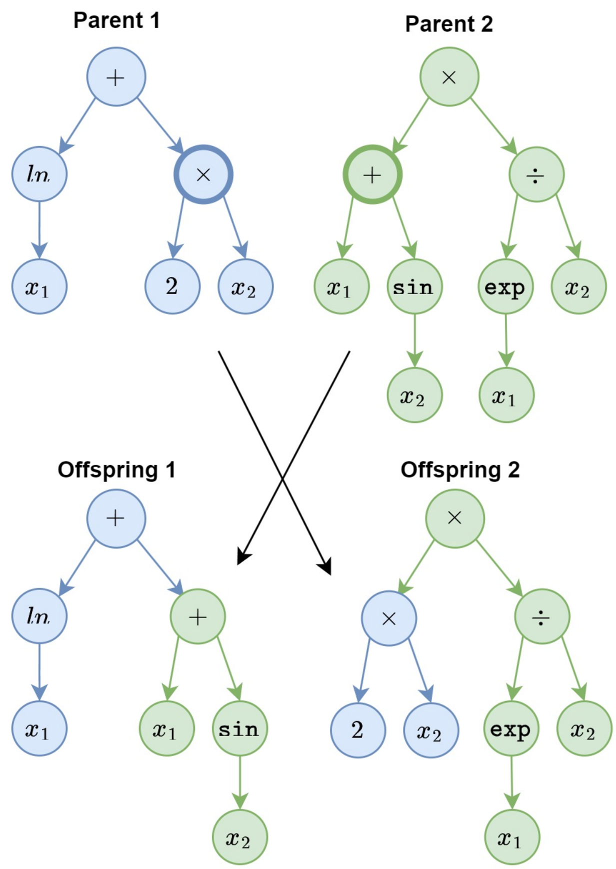


Figure 2.4: Sub-tree crossover example. Nodes with bold edges indicate the selected crossover points. Note that the crossover points need not occur at the same level in the two parents.

of solutions in the evolving population.

2.3.3 Mutation

Mutation is one of the fundamental genetic operators in GP, serving as a mechanism to introduce genetic diversity into the population of candidate programs or solutions. It helps maintaining the exploration capabilities of the evolutionary process. Unlike crossover, which combines genetic material from two parents, mutation acts unilaterally on a single program.

Mutation involves the random modification of a parent program by altering one or more components within it. These components can be individual nodes within the program's tree-like structure. The objective of mutation is to create small, often random changes to the program's structure or functionality while preserving some aspects of the original solution. This randomness allows GP to explore a broader space of potential solutions.

Several mutation operators have been proposed over the years, with some of the most common and successful techniques outlined below:

- Point Mutation: Point mutation randomly selects a single node within the program and alters it. This can involve replacing the node with another of the same type or a different type [92].
- Subtree Mutation: Subtree mutation selects a random subtree within the parent program and replaces it with a newly generated subtree. This can lead to more significant structural changes [6].
- Hoist Mutation: Hoist mutation focuses on functions within the program. It randomly selects a subtree representing a function and elevates it to a higher level in the program's hierarchy, simplifying the structure [88].
- Shrink Mutation: Shrink mutation reduces the program's size by replacing a subtree with a terminal node. This helps control program size and complexity [5].

2.4 Summary

This chapter has provided a comprehensive overview of the foundational concepts and methods crucial to understanding the subsequent chapters of this dissertation proposal. First, we introduced the transformer architecture, a groundbreaking deep learning model introduced by Vaswani *et al.* [181]. The transformer has not only revolutionized natural language processing tasks but has also found applications in diverse domains such as computer vision and symbolic regression. The chapter delved into the core components of the Transformer, including its general architecture, input embedding, positional encoding, and attention mechanisms.

Furthermore, the chapter introduced genetic algorithms and genetic programming as evolutionary optimization techniques, shedding light on their fundamental principles, representations, initialization strategies, crossover and mutation operators, and other key aspects. These insights lay the groundwork for the subsequent chapters, where we will apply and extend these concepts to address specific challenges in our research domain.

CHAPTER THREE

MULTI-SET SYMBOLIC SKELETON PREDICTION

The aim of this chapter is to introduce a new problem termed multi-set symbolic skeleton prediction (MSSP). This problem is derived from the symbolic skeleton prediction (SSP) problem that has been previously explored by existing research [13, 15, 149, 179]. To provide a foundation for our extension, we first establish the definition of the symbolic skeleton prediction problem. Subsequently, we will elaborate on the extensions made to define our multi-set symbolic skeleton prediction problem.

The symbolic skeleton prediction problem takes in a set of N_R input–response pairs $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_R}$ from a sensitive system. In this context, a sensitive system refers to a system whose behavior is responsive to variations in input conditions. Each input vector $\mathbf{x}_i$ is t-dimensional ($\mathbf{x}_i \in \mathbb{R}^t$). The objective is to produce a symbolic skeleton $\hat{\mathbf{e}}$ that describes the functional form of the system. Let f denote the underlying function of the system; i.e., $y_i = f(\mathbf{x}_i)$. $\kappa(\cdot)$ represents a skeleton function that replaces the numerical constants of a given symbolic expression by the placeholder c; e.g., $\kappa(3x^2 + e^{2x} - 4) = c_1\,x^2 + e^{c_2\,x} + c_3$. Thus, the objective is to return a symbolic skeleton $\hat{\mathbf{e}}$ such that $\kappa\,(f) \approx \hat{\mathbf{e}}$.

Some SR methods implement SSP as one of their main steps. In particular, the outline of these methods consists of using SSP to predict the symbolic skeleton that describes the data, and then using an optimization method, such as the Broyden–Fletcher–Goldfarb–Shannon (BFGS) algorithm [51], to estimate the numerical values of the constant placeholders of the generated skeleton. For example, Petersen *et al.* [149] proposed a reinforcement learning framework that uses RNNs to generate candidate skeletons, which are then fitted using the BFGS algorithm. The policy employed by this framework is designed to generate better-performing skeletons at each iteration. Other approaches are based on the use of large language models. Valipour *et al.* [179] presented a three-step SR method. The process involves acquiring an order-invariant embedding of the input dataset through a T-net [26], which is a type of network structure that uses max-pooling aggregation layers to provide order-invariance over its arbitrarily-sized input. Then, it generates a symbolic skeleton using a generative pre-trained transformer (GPT) language model [17], and subsequently optimizes constant values to complete the equation skeleton using BFGS. Similarly, Bigglio *et al.* [15] presented an SR method based on the use of transformer models. Nevertheless, unlike the approach of Valipour *et al.* of employing a separate order-invariant embedding component, they utilized a transformer encoder based on the Set Transformer [102], which inherently encodes the information from the input set in a permutation-invariant manner.

Chu *et al.* [30] pointed out that current SR methods based on large language models encounter challenges in scalability when dealing with multivariate equations. Therefore, they proposed to tackle the multivariate SR problem as a sequence of single-variable SR problems, which are combined in a bottom-up fashion. The process begins by selecting one control variable, generating a single-variable skeleton, and estimating the constant values using BFGS. Single-variable skeletons are generated using GP or a Monte Carlo tree search approach (MCTS) [171]. Then, variables are gradually added one by one while repeating the process until a symbolic expression involving all relevant variables is generated.

3.1 Problem Definition

Similar to the work presented by Chu *et al.* [30], in this dissertation, we tackle the SR problem by decomposing it into single-variable sub-problems. In order to do so, we deviate from the SSP paradigm employed by previous approaches. We explain the rationale behind this decision through the use of an example:

Consider a system whose response y depends on two stimuli $\mathbf{x} = [x_1, x_2]$ governed by the

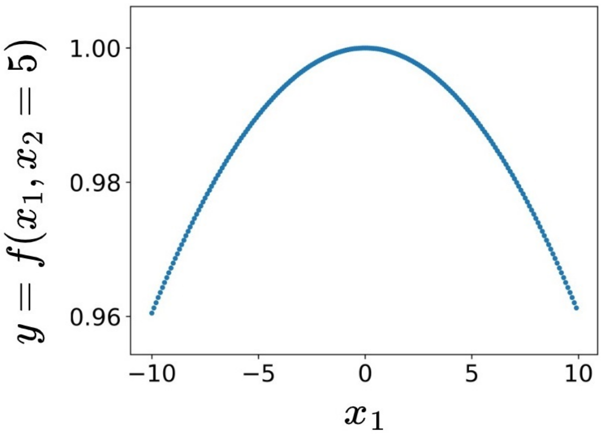


Figure 3.1: An example of a set $\mathbf{X}$ with 200 samples and a fixed value $x_2 = 5$, and the corresponding response $\mathbf{y}$.

underlying function $y = f(\mathbf{x}) = \sin\left(\frac{x_1}{10x_2} + \frac{\pi}{2}\right)^2$. For this example, suppose we have access to an oracle system, meaning it provides outputs for given inputs without explicitly revealing the function f. In practice, however, such direct access is rarely available, and we must infer the equation from an observed dataset instead. We analyze the relationship between x_1 and y by following the approach in [30], where unexamined variables are held fixed. To generate data, we sample a random set $\mathbf{X}$ of n points, allowing x_1 to vary while keeping x_2 fixed at a random value of 5. The corresponding response vector $\mathbf{y}$ is then obtained by querying the oracle system with $\mathbf{X}$. Figure 3.1 illustrates the generated dataset. Then, the set of input–response pairs $(\mathbf{X}, \mathbf{y})$ is fed into an SSP model to obtain a symbolic skeleton. In this example, we used the pre-trained SSP model proposed by [15], which produces the skeleton:

$$c_1\, x_1^2 + c_2.$$

Note that the obtained skeleton describes a quadratic function, which does not correspond to the functional form of the underlying function f, a sine squared function. This is expected considering that function f closely resembles a quadratic function within the selected region of the input domain (i.e., $x_1 \in [-10, 10]$) when the variable x_2 is fixed to the value 5. What is more, existing methods tend to select the least complex solution from a pool of equally performing candidate solutions, thus following Occam's razor [22]. In this context, the complexity of a skeleton could be related to the number of terms and operators used in it. As such, a skeleton of the form $c_1 x_1^2 + c_2$ would be preferred over a skeleton of the form $\sin(c_1 x^2 + c_2)^2$. Nevertheless, if the variable x_2 had been fixed to a different value (e.g., $x_2 = 0.1$), then the sine squared skeleton would have been correctly identified.

The fixed values of the remaining variables could project the function into a space where the functional form is not easily identifiable, which could be worsened due to the limited range of values that the analyzed variable can take. As a consequence, we argue that the SSP problem would benefit from the injection of additional context data. Specifically, when analyzing the variable x_v, we could employ multiple sets of input–response pairs, each of which is constructed using different fixed values for the remaining variables $\mathbf{x} \setminus \{x_v\}$. The key idea is to process the information from the multiple sets simultaneously to produce a symbolic skeleton that is common to all input sets. We refer to this new problem as *multi-set symbolic skeleton prediction* (MSSP).

More formally, we are given a data set $(\mathbf{X}, \mathbf{y})$ where $\mathbf{X} \in \mathbb{R}^{N_R \times t}$ and $\mathbf{y} \in \mathbb{R}^{N_R \times 1}$. Suppose we analyze the relationship between the v-th input variable, x_v (i.e., $v \in [1, \dots, t]$), and the response variable y. We construct a collection of N_S sets, denoted as $\mathbf{D} = \left\{\mathbf{D}^{(1)}, \dots, \mathbf{D}^{(N_S)}\right\}$. Each set $\mathbf{D}^{(s)}$ comprises n input–response pairs such that $\mathbf{D}^{(s)} = \left(\mathbf{X}_v^{(s)}, f\left(\mathbf{X}^{(s)}\right)\right) = \left(\mathbf{X}_v^{(s)}, \mathbf{y}^{(s)}\right)$, where $\mathbf{X}^{(s)} \in \mathbb{R}^{n \times t}$, $\mathbf{y}^{(s)} \in \mathbb{R}^{n}$, and $\mathbf{X}_v^{(s)}$ denotes the v-th column of $\mathbf{X}^{(s)}$ (i.e., the data corresponding to the x_v variable). $\mathbf{X}^{(s)}$ is constructed so that the variables in $\mathbf{x} \setminus \{x_v\}$ are assigned random values and held fixed for all samples.

$\mathbf{X}^{(s)}$ can be constructed by selecting n samples from $\mathbf{X}$ that satisfy the criterion that x_v is allowed to vary while the remaining variables in $\mathbf{x} \setminus \{x_v\}$ are held fixed. The corresponding response values form the set $\mathbf{y}^{(s)}$, which is directly obtained from the observed data. If

$\mathbf{X}$ is not large enough, $\mathbf{X}^{(s)}$ can be generated and its response values can be obtained as $\mathbf{y}^{(s)} = f(\mathbf{X}^{(s)})$ by querying the system as an oracle. However, using a system in this way is rarely possible. Instead, an opaque model $\hat{f}$ (e.g., a neural network) trained to approximate f (i.e., $\hat{f}(\mathbf{x}) \approx f(\mathbf{x})$) can be used to estimate the response values for $\mathbf{X}^{(s)}$. More details on this opaque model and its relevance to our SR problem will be provided in Chapter 4.

Since the variables in $\mathbf{x} \setminus \{x_v\}$ have been fixed to constant values to construct each set $\mathbf{D}^{(s)}$, the underlying function that explains the relationship between $\mathbf{X}_v^{(s)}$ and $\mathbf{y}^{(s)}$ could be expressed solely in terms of variable x_v. As such, the underlying function of the s-th set is denoted by $f^{(s)}(x_v)$. It is important to note that functions $f^{(1)}(x_v), \ldots, f^{(N_S)}(x_v)$ have been derived from the same function $f(\mathbf{x})$ and only differ in their coefficient values due to the selection of different values for the variables in $\mathbf{x} \setminus \{x_v\}$ for each constructed set. As a consequence, if we apply the skeleton function $\kappa(\cdot)$ to functions $f^{(1)}(x_v), \ldots, f^{(N_S)}(x_v)$, they would produce the same target symbolic skeleton $\mathbf{e}(x_v)$, in which the constant values have been replaced by placeholders; i.e., $\mathbf{e}(x_v) = \kappa\left(f^{(1)}(x_v)\right) = \kappa\left(f^{(2)}(x_v)\right) = \cdots = \kappa\left(f^{(N_S)}(x_v)\right)$.

Then, the collection $\mathbf{D}$ is fed as an input to the MSSP problem. The objective is to generate skeleton $\hat{\mathbf{e}}(x_v)$ that characterizes the functional form of all input sets, and approximates the target skeleton $\mathbf{e}(x_v)$; i.e., $\hat{\mathbf{e}}(x_v) \approx \mathbf{e}(x_v)$. For the sake of generality, we define the MSSP problem as follows:

Definition 1. *The Multi-Set Symbolic Skeleton Prediction (MSSP) problem takes an input consisting of a collection of N_S sets, denoted as $\mathbf{D} = \left\{\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(N_S)}\right\}$. Each set $\mathbf{D}^{(s)}$ comprises n input–response pairs such that $\mathbf{D}^{(s)} = \left(\mathbf{X}_v^{(s)}, \mathbf{y}^{(s)}\right)$, where $\mathbf{X}_v^{(s)} \in \mathbb{R}^n$ and $\mathbf{y}^{(s)} \in \mathbb{R}^n$. The underlying function of the s-th set is denoted by $f^{(s)}$; i.e., $\mathbf{y}^{(s)} = f^{(s)}\left(\mathbf{X}_v^{(s)}\right)$. The underlying functions of all input sets are assumed to share a common unknown symbolic skeleton, denoted as $\mathbf{e}$. Thus, the objective of the MSSP problem is to generate a symbolic skeleton $\hat{\mathbf{e}} \approx \mathbf{e}$ that characterizes the functional form of all input sets.*

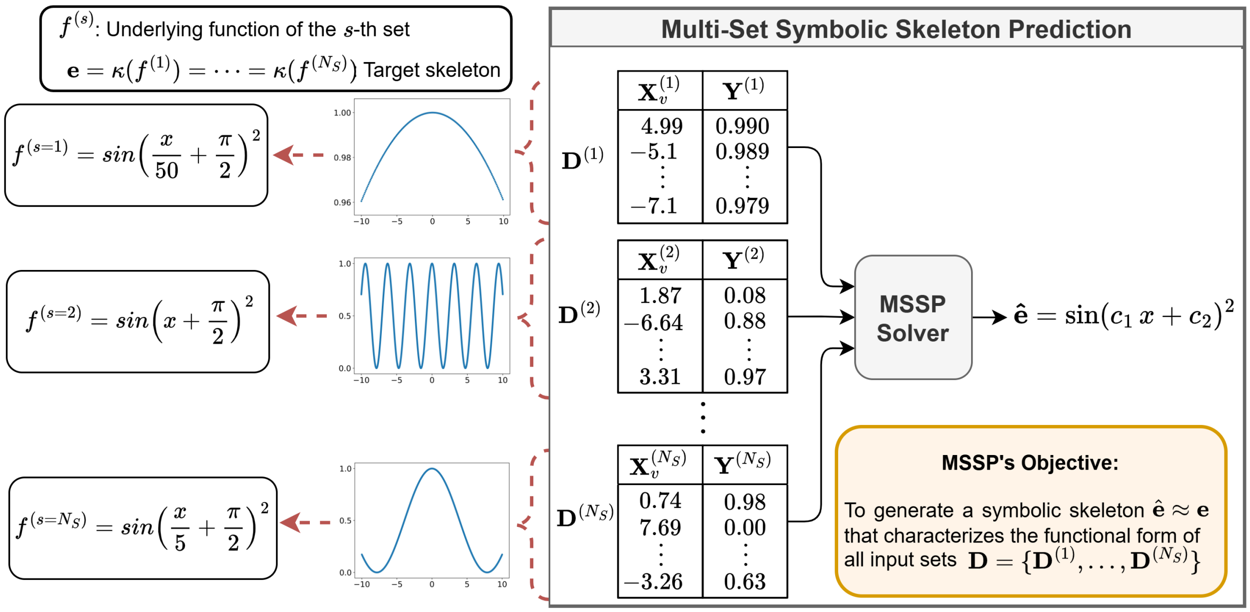


Figure 3.2: An example of an MSSP problem.

Figure 3.2 depicts an example of the type of problems we attempt to solve. In this example, each set $\mathbf{D}^{(s)}$ was generated using the expressions that are shown on the left side of the figure. Note, however, that the MSSP solver only has access to the sets of generated data $\mathbf{D}^{(s)}$ but not to the information regarding the expressions or processes that were used to generate such data. The proposed approach is an integral component of the SR framework proposed in Chapter 4. Within this framework, a series of single-variable skeletons will be generated and subsequently merged to produce a comprehensive multivariate expression. In this chapter, we will focus on developing a method to solve the MSSP problem.

3.2 Set Transformer

In this section, we discuss the Set Transformer [102] that we utilize as a basis to develop our Multi-Set Transformer for MSSP. In Section 2.1, we described the transformer model as a highly effective tool for processing sequential data. However, it faces challenges when dealing with set-structured data [102]. In various real-world applications, data are naturally represented as sets rather than sequences. For instance, in point cloud data, the points represent a set of spatial coordinates; in graph data, nodes and their connections form a set of elements. These types of data are applicable in multiple instance learning scenarios [132, 140, 165], where an input consists of a collection of instances, and the corresponding target is a label assigned to the entire set. Other statistical problems such as population statistic estimation and outlier detection can also be viewed as set-input problems [45, 196].

The transformer is designed to handle sequences with fixed orderings, where each element's position is crucial to understanding the data's meaning. In contrast, sets are collections of elements without any inherent order, and their permutations do not alter the underlying semantics. This inherent permutation invariance poses a significant obstacle for traditional sequence models when processing sets [45, 196]. In order to address the limitations of conventional approaches, Lee *et al.* [102] presented an attention-based neural network module called Set Transformer that is based on the transformer model described in Section 2.1. This method introduces modifications to the transformer architecture, enabling it to handle sets without assuming a fixed ordering of elements.

A model designed for set-input problems must meet two essential criteria to handle sets effectively: First, it should be capable of processing input sets of varying sizes. Second, it should exhibit permutation invariance. The latter means the output of the function represented by the model remains the same regardless of the order in which the elements of the input set are presented. More formally, Zaheer *et al.* [196] described this type of function as permutation invariant:

Definition 2. *Let $\mathscr{X}$ denote the space from which individual elements are drawn (e.g., if the elements are real-valued vectors of dimension d, then $\mathscr{X} = \mathbb{R}^d$). Consider a function $u : \mathcal{X} \to \mathcal{Y}$ with input set $\mathbf{x} = \{x_1, \ldots, x_n\}$, where $x_i \in \mathscr{X}$; i.e., the input domain is the power set $\mathcal{X} = 2^{\mathscr{X}}$. The function u operates on sets and must be permutation invariant to* *the order of elements in its input, meaning that for any permutation π it holds that:*

$$u(\{x_1, \ldots, x_n\}) = u(\{x_{\pi(1)}, \ldots, x_{\pi(n)}\}),$$

where $\pi \in S_n$ and S_n represents the set of all permutations of indices $\{1, \ldots, n\}$.

The Set Transformer consists of two main parts: an encoder ϕ and a decoder ψ. The process starts by encoding the set elements in an order-agnostic manner. As such, the encoding for each element should be the same regardless of its position in the set. Then, ψ aggregates the encoded features and produces the desired output. Let us consider an input set $\mathbf{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, where each element is d_{in}-dimensional ($\mathbf{S} \in \mathbb{R}^{n \times d_{in}}$). Therefore, the output T produced by the Set Transformer, whose computed function is denoted as g, is:

$$T = g(\mathbf{S}) = \psi\left(\phi\left(\{\mathbf{s}_1, \ldots, \mathbf{s}_n\}\right)\right). \tag{3.1}$$

Typically, the desired order-agnostic property is achieved by making ϕ to act on each element of a set independently (i.e., $g(\mathbf{S}) = \psi\left(\{\phi(\mathbf{s}_1), \ldots, \phi(\mathbf{s}_n)\}\right)$) [196]. Nevertheless, the Set Transformer uses self-attention mechanisms to encode the entire input set simultaneously so that it is capable of recognizing interactions among the set instances. In order to do so, the multi-head attention mechanism explained in Section 2.1.3 is adapted.

3.2.1 Set Attention Blocks

Let us consider two matrices $A, B \in \mathbb{R}^{n \times d_{in}}$ representing two sets whose elements are d_{in}-dimensional vectors. In Eq. 2.1, we presented the expression used to obtain the output of the i-th head of the multi-head self-attention mechanism in the transformer model. We modify this expression slightly, no longer considering a self-attention scenario; that is, the queries, keys, and values do not come from the source necessarily. Instead, we consider the case that the key and value matrices always come from the same source, which is represented by B, while the query matrix is represented by A. Thus, the output of the i-th head of the multi-head attention of the Set Transformer is given by:

$$\text{head}_i = \text{Attention}(A \times W_i^Q, B \times W_i^K, B \times W_i^V).$$

As such, based on Eq. 2.2, the output of the multi-head attention layer of the Set Transformer is expressed as follows:

$$\text{MultiHead}(A, B) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O.$$

Based on the modified multi-head attention layer, we define a multi-head attention block (MAB) that, similar to the transformer encoder block, includes residual connections and layer normalization:

$$\begin{aligned} \text{MAB}(A, B) &= \text{LayerNorm}(H + \text{FFN}(H)), \\ H &= \text{LayerNorm}(A + \text{MultiHead}(A, B)). \end{aligned}$$

Then, using the MAB, we define the set attention block (SAB) as follows:

$$\text{SAB}(A) = \text{MAB}(A, A), \tag{3.2}$$

applying self-attention between the elements of set A and produces a set of equal size. Note that the output of the previous operation computes pairwise interactions among the elements of A; therefore, a stack of multiple SAB operations would encode higher-order interactions.

One drawback of the use of SABs is that the attention mechanism is performed between two identical sets with n elements, which leads to a quadratic time complexity $\mathcal{O}(n^2)$. In order to alleviate this issue, Zaheer *et al.* [196] introduced the induced set attention block

(ISAB). Let $I \in \mathbb{R}^{m \times d_{in}}$ denote a matrix of "inducing points" consisting of m d_{in}-dimensional vectors (recall d_{in} represents the size of the input embeddings) whose values are learned during the training process. Here, m is a tunable hyperparameter and its value is chosen to be significantlly smaller than n ($m \ll n$). Thus, an ISAB with m inducing points is defined as:

$$\begin{aligned} \mathrm{ISAB}_m(A) &= \mathrm{MAB}(A, H) \in \mathbb{R}^{n \times d_{in}}, \\ H &= \mathrm{MAB}(I, A) \in \mathbb{R}^{m \times d_{in}}. \end{aligned} \tag{3.3}$$

This could be interpreted as if the input set A was projected into a lower-dimensional space and then reconstructed to produce outputs with the desired dimensionality. Since the attention mechanism in Eq. 3.3 is computed between a set of size m and a set of size n, its associated time complexity is $\mathcal{O}(mn)$. Furthermore, Lee *et al.* [102] proved that the SAB and ISAB blocks are universal approximators of permutation invariant functions.

3.2.2 Set Transformer Architecture

Having defined the MAB, SAB, and ISAB blocks in the previous section, we explain how they are used to build the architecture of the Set Transformer. As shown in Eq. 3.1, the Set Transformer starts by processing the input set $\mathbf{S}$ using the encoder ϕ. Recall that $\mathbf{S}$ is an input set of n d_{in}-dimensional elements. Here, d_{in} denotes the length of each element within the input set after undergoing pre-processing procedures such as input embedding (see Section 2.1.2). Then, $\phi : \mathcal{S}^{n \times d_{in}} \to \mathcal{Z}^{n \times d}$ maps the original input space to a latent space using a stack of SABs or ISABs. For instance:

$$Z = \phi(\mathbf{S}) = \mathrm{ISAB}_m(\mathrm{ISAB}_m(\mathrm{Embbeding}(\mathbf{S}))),$$

represents an encoder with $\ell = 2$ stacks. In general, ℓ is a hyperparameter and its value is decided depending on the complexity of the problem. In addition, Embedding($\cdot$) is the input embedding layer described in Section 2.1.2. Note that the Set Transformer, unlike the conventional transformer, does not require the use of a positional encoding layer given that the position of the elements in the input set $\mathbf{S}$ is not relevant for the final decision.

The next step is for the decoder ψ to aggregate the latent features Z into a set of k vectors, which are then processed by a feedforward layer to obtain the final outputs Y. Thus, $\psi : \mathcal{Z}^{n \times d} \rightarrow \mathcal{Y}^{k \times d_{out}}$ (k d_{out}-dimensional outputs) such that:

$$Y = \psi(Z) = \mathrm{FFN}(\mathrm{SAB}(\mathrm{PMA}_k(Z))), \tag{3.4}$$

where $\mathrm{PMA}_k(\cdot)$ is a pooling by multi-head attention (PMA) layer that aggregates the latent features by applying multi-head attention on a learnable set of k seed vectors $V \in \mathbb{R}^{k \times d}$. In particular, PMA_k is defined as:

$$\mathrm{PMA}_k(Z) = \mathrm{MAB}(V, \mathrm{FFN}(Z)).$$

From Eq. 3.4, note that a SAB block was used to process the outputs of the PMA layer in order to model the interactions among the k outputs.

It is important to clarify that the Set Transformer was designed to produce fixed-size outputs. This setting can be used for applications such as population statistic estimation (e.g., retrieving a unique value that presents the median of an input set), unique character counting (i.e., obtaining the number of unique characters in a set of images), or k-amortized clustering where the objective is to produce k pairs of output parameters of a mixture of Gaussians. In the following section, we explain the reasons why the direct use of the Set Transformer architecture is not feasible for the problem setting of interest in this dissertation.

3.3 Multi-Set Transformer

This section is dedicated to outlining the approach we suggest for solving the MSSP problem. Our method draws inspiration from the Set Transformer. Nevertheless, note that the Set Transformer was originally designed to serve a different purpose from the task we are currently undertaking. Thus, we propose a Multi-Set Transformer model, which presents modifications to address the limitations of the Set Transformer and adapt it to the specific requirements of our research.

The most evident limitation of the Set Transformer, in the context of the present problem, resides in its encoder structure, which is specifically designed to process a single input set, as shown in Eq. 3.1. This is important because the input of the MSSP problem is defined as a collection of N_S input sets ($N_S > 1$). Hence, the encoder of our Multi-Set Transformer is designed to process multiple input sets simultaneously. This aspect represents the main difference with respect to the SSP method proposed by Bigglio *et al.* [15], whose encoder only processes single input sets. Furthermore, the Set Transformer's output is k-dimensional; that is, its size is fixed depending on the problem. Conversely, the objective of the MSSP problem is to generate a symbolic skeleton string whose length is not known *a priori* and depends on each input collection. Therefore, the decoder of the Multi-Set Transformer is designed as a conditional-generative structure as it generates output sequences (i.e., the skeleton string) based on the encoded context.

3.3.1 Multi-Set Transformer Architecture

Recall from Definition 1 that each input set $\mathbf{D}^{(s)} = \left(\mathbf{X}_v^{(s)}, \mathbf{y}^{(s)}\right) = \left\{\left(x_{v,i}^{(s)}, y_i^{(s)}\right)\right\}_{i=1}^{n}$ (where $s \in [1, N_S]$) is defined as a set of n input–response pairs. The first step involves arranging $\mathbf{D}^{(s)}$ in a manner analogous to the input structure of the Set Transformer, which consisted of a matrix $\mathbf{S}$ where each row represented a d_{in}-dimensional element of the input

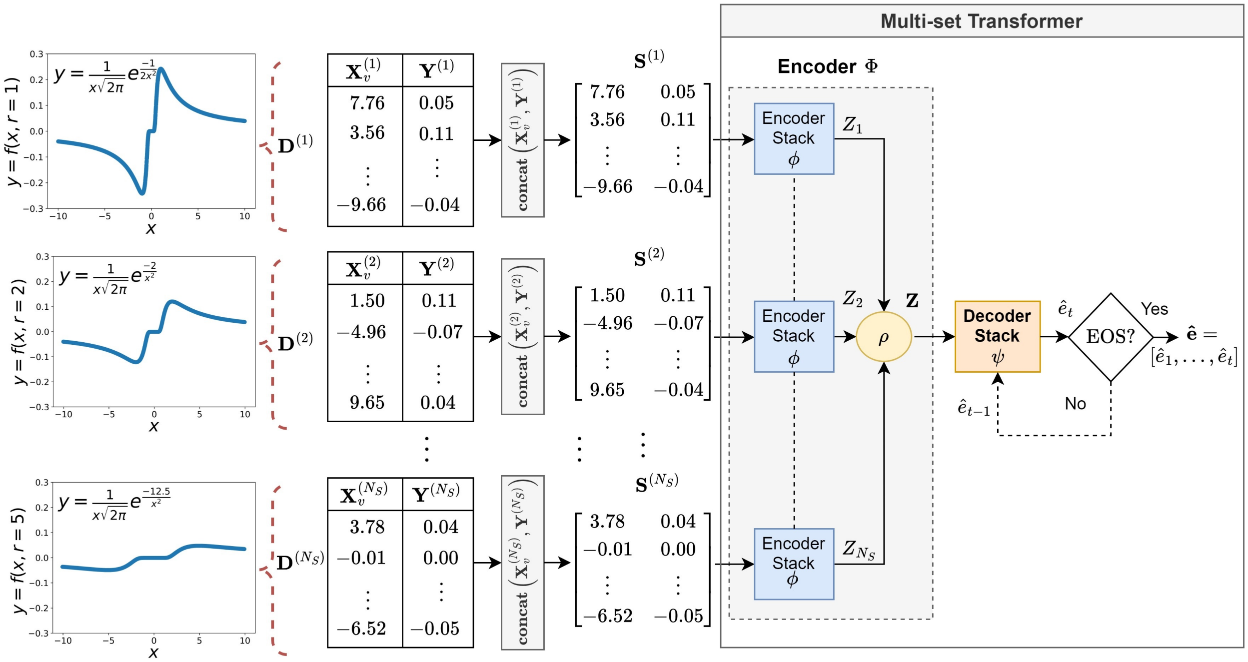


Figure 3.3: An example of a MSSP problem using the Multi-Set Transformer.

set (Eq. 3.1). Let us denote the s-th input of our proposed Multi-Set Transformer as $\mathbf{S}^{(s)} \in \mathbb{R}^{n \times d_{in}}$ such that its i-th row, $\mathbf{s}_i^{(s)}$, consists of the concatenation of the input value $x_{v,i}^{(s)}$ and its corresponding output $y_i^{(s)}$; i.e., $\mathbf{s}_i^{(s)} = \left[x_{v,i}^{(s)}, y_i^{(s)}\right]$. Hence, $\mathbf{S}^{(s)}$ is defined as a matrix with $d_{in} = 2$ columns. This process is depicted in Figure 3.3, as well as the main components of the Multi-Set Transformer architecture. Note that all input sets shown in this example were generated using the equation $y = \frac{1}{x\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{r}{x})^2}$ and a different r value was used for each set $\mathbf{D}^{(s)}$. Thus, their common target symbolic skeleton can be expressed as $\frac{c}{x} e^{\frac{c}{x^2}}$.

Our Multi-Set Transformer comprises two primary components: an encoder and a decoder. The purpose of the encoder is to map the information of all input sets into a unique latent representation $\mathbf{Z}$. To do so, an encoder stack ϕ, similar to the one used in the Set Transformer (see Section 3.2.2), transforms each input set $\mathbf{S}^{(s)}$ into a latent representation $Z^{(s)} \in \mathbb{R}^d$ (where d is context vector length or the "embedding size") individually. Our encoder, denoted as Φ, comprises the use of the encoder stack ϕ to generate N_S individual encodings $Z^{(1)}, \ldots, Z^{(N_S)}$, which are then aggregated into a unique latent representation $\mathbf{Z}$:

$$\mathbf{Z} = \Phi\left(\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(N_S)}, \theta_e\right) = \rho\left(\phi\left(\mathbf{S}^{(1)}, \theta_e\right), \ldots, \phi\left(\mathbf{S}^{(N_S)}, \theta_e\right)\right) = \rho\left(Z^{(1)}, \ldots, Z^{(N_s)}, \theta_e\right), \tag{3.5}$$

where $\rho(\cdot)$ is a pooling function, and θ_e represents the trainable weights of the encoder stack. We define ϕ as a stack of ℓ ISAB blocks so that it encodes high-order interactions among the elements of an input set in a permutation-invariant way. Furthermore, unlike the Set Transformer's encoder, we include a PMA layer in ϕ to aggregate the features extracted by the ISAB blocks, whose dimensionality is $n \times d$, into a single d-dimensional latent vector. Finally, the function $\rho(\cdot)$ that is used to aggregate the latent representations $Z^{(s)}$ is implemented using an additional PMA layer.

On the other hand, the objective of the decoder, denoted as ψ, is to generate sequences conditioned on the representation $\mathbf{Z}$ generated by Φ. This objective is aligned with that of the standard transformer decoder (see Section 2.1.4); thus, the same architecture is used for our Multi-Set Transformer. Specifically, ψ consists of a stack of M identical blocks, each of which is composed of three main layers: a multi-head self-attention layer, an encoder–decoder attention layer, and a position-wise feedforward network.

Let $\hat{\mathbf{e}} = \{\hat{e}_1, \ldots, \hat{e}_{N_{out}}\}$ denote the output sequence produced by the Multi-Set Transformer, which represents the symbolic skeleton as a sequence of indexed tokens in prefix notation. For instance, the skeleton $\frac{c}{x}e^{\frac{c}{x^2}}$ would be expressed as the sequence of tokens $\{\texttt{mul}, \texttt{div}, \texttt{c}, \texttt{x}, \texttt{exp}, \texttt{div}, \texttt{c}, \texttt{square}, \texttt{x}\}$ in prefix notation. In addition, each token in this sequence is transformed into a numerical index according to a pre-defined vocabulary that contains all unique symbols that appear in the dataset being processed. The vocabulary used in this work is provided in Table 3.1. According to this, the previous sequence in prefix notation would be expressed as the following sequence of indices:

Table 3.1: Vocabulary used to pre-train the Muli-Set Transformer.

Token	**Meaning**	**Index**
`SOS`	Start of sentence	0
`EOS`	End of sentence	1
`c`	Constant placeholder	2
`x`	Variable	3
`abs`	Absolute value	4
`acos`	Arc cosine	5
`add`	Sum	6
`asin`	Arc sine	7
`atan`	Arc tangent	8
`cos`	Cosine	9
`cosh`	Hyperbolic cosine	10
`div`	Division	11
`exp`	Exponential	12
`log`	Logarithmic	13
`mul`	Multiplication	14
`pow`	Power	15
`sin`	Sine	16
`sinh`	Hyperbolic sine	17
`sqrt`	Square root	18
`tan`	Tangent	19
`tanh`	Hyperbolic tangent	20
`-3`	Integer number	21
`-2`	Integer number	22
`-1`	Integer number	23
`0`	Integer number	24
`1`	Integer number	25
`2`	Integer number	26
`3`	Integer number	27
`4`	Integer number	28
`5`	Integer number	29
`E`	Euler's number	30

$\{0, 14, 11, 2, 3, 12, 11, 2, 18, 3, 1\}$.

During inference, each element $\hat{e}_i$ ($i \in [1, N_{out}]$) is generated in an auto-regressive manner. Specifically, the decoder ψ produces a probability distribution over the elements of the vocabulary as follows:

$$\sigma\left(\psi\left(\mathbf{Z}, \theta_d | \hat{e}_1, \ldots, \hat{e}_{i-1}\right)\right) = P\left(\hat{e}_i | \hat{e}_1, \ldots, \hat{e}_{i-1}, \mathbf{Z}\right),$$

where θ_d represents the trainable weights of the decoder stack. This distribution is obtained by applying a softmax function σ to the decoder's output. The element $\hat{e}_i$ is thus selected from the obtained probability distribution by using one of the following strategies: sampling decoding or greedy decoding [63]. The former samples a token from the distribution to allow diversity in the generated sequence, while the latter selects the token with the highest probability for a deterministic output. Hence, the generation process can be written as:

$$\hat{e}_i = \text{sample}\left(\sigma\left(\psi\left(\mathbf{Z}, \theta_d | \hat{e}_1, \ldots, \hat{e}_{i-1}\right)\right)\right),$$

which is also depicted in Figure 3.3 using an auto-regressive loop between the output and the decoder stack. The decoder keeps generating new sequence elements until the "end-of-sentence" token (`EOS`) is produced ($\hat{e}_i = 1$, according to Table 3.1) or the maximum output length allowed, denoted as N_{max}, is reached ($N_{out} = N_{max}$).

3.3.2 Multi-Set Transformer Training

We aim to build a Multi-Set Transformer model that, once trained, generates a symbolic skeleton $\hat{\mathbf{e}}$ when given an input set collection $\mathbf{D}$. The estimated skeleton $\hat{\mathbf{e}}$ aims to resemble its corresponding target skeleton $\mathbf{e}$ closely. Hence, a trained Multi-Set Transformer model can be viewed as a problem-independent MSSP solver in the sense that it does not need to be trained on data specific to each set collection $\mathbf{D}$ to retrieve an estimated skeleton.

To do so, we train our Multi-Set Transformer on a dataset of synthetically generated MSSP problems. Let $\mathbf{D}^b = \{\mathbf{D_1}, \ldots, \mathbf{D}_B\}$ denote a training batch with B samples, each of which represents a collection of N_S input sets; i.e., $\mathbf{D}_j = \left\{\mathbf{D}_j^{(1)}, \ldots, \mathbf{D}_j^{(N_S)}\right\}$, where $j \in [1, \ldots, B]$. In addition, $\mathbf{E}^b = \{\mathbf{e}_1, \ldots, \mathbf{e}_B\}$ is the corresponding set of target skeletons, each of which represents a sequence of variable length; i.e., $\mathbf{e}_j = \left\{e_{j,1} \ldots, e_{j,N_j}\right\}$ and $N_j = |\mathbf{e}_j|$. The function computed by the model is denoted as $g(\cdot)$, and Θ denotes its weights. Note that $\Theta = [\theta_e, \theta_d]$ encompasses the weights of the encoder and the decoder stacks. Given an input set collection $\mathbf{D}_j$, $g(\mathbf{D}_j, \Theta)$ computes the estimated skeleton $\hat{\mathbf{e}}_j$ with length $N_{out,j}$. Thus, this model is trained to generate accurate estimated skeletons so that $\hat{\mathbf{e}}_j \approx \mathbf{e}_j$.

In the previous section, we explained how the model generates sequences autonomously by using its own predictions as conditional inputs during inference. Conversely, during training, the model aims to minimize the discrepancy between its predictions and the true target skeletons using a technique called "teacher forcing" [190]. This strategy consists of providing the model with past elements of the target skeleton sequence as inputs for generating subsequent tokens. More specifically, the normalized probability distribution produced by the decoder ψ for the i-th element of the predicted skeleton sequence of the j-th sample would be expressed as:

$$\sigma\left(\psi\left(\mathbf{Z}_j, \theta_d | e_{j,1}, \ldots, e_{j,i-1}\right)\right) = P\left(\hat{e}_{j,i} | e_{j,1}, \ldots, e_{j,i-1}, \mathbf{Z}_j\right).$$

This process is depicted in Figure 3.4, which differs from the generation process shown in Fig. 3.3 in that the autoregressive loop does not feed the previously generated tokens into the decoder. Instead, it feeds the previous $i-1$ tokens of the target skeleton $\mathbf{e}_j$.

To generate accurate skeletons $\hat{\mathbf{e}}_j$ that are close to their corresponding targets $\mathbf{e}_j$, we minimize the discrepancy between them, which is computed using the cross-entropy loss function. It is important to note that the length of the symbolic skeleton sequence $\hat{\mathbf{e}}_j$, obtained after using the teacher forcing strategy, may differ from the length of $\mathbf{e}_j$. This divergence in sequence lengths poses a challenge in calculating a loss function that enables element-wise comparison, as is the case of the cross-entropy function. Nevertheless, note that while teacher forcing does not guarantee that $\hat{\mathbf{e}}_j$ and $\mathbf{e}_j$ have the same length, it can encourage length similarity indirectly. This is due to the fact that the model learns to associate the length of the target skeleton with its own generation process during training.

In cases where $\hat{\mathbf{e}}_j$ and $\mathbf{e}_j$ differ in length, one common approach is to use padding and

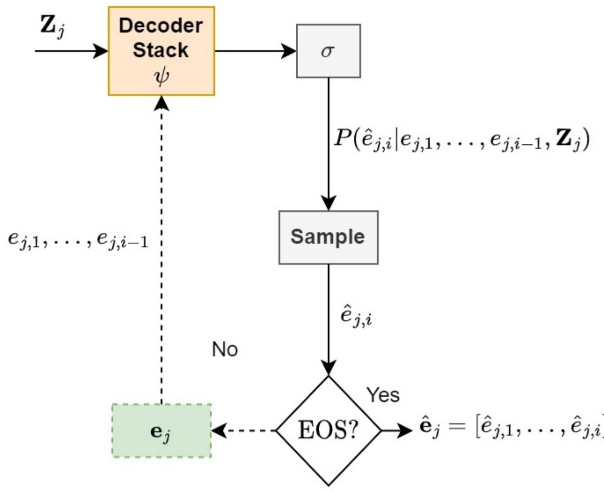


Figure 3.4: Modification of the Multi-Set Transformer output generation during training.

masking to ensure that the loss is only calculated for valid tokens. Hence, we add padding tokens to the shorter sequence, so both sequences have the same length $T_j = \max(N_j, N_{out,j})$. We choose a padding token value of 0, representing the "start-of-sentence" token (`SOS`) as per Table 3.1. This value lacks meaningful content and serves the purpose of padding within the sequence. The next step is to create a binary mask $\boldsymbol{\omega}_j = \{\omega_{j,1}, \ldots, \omega_{j,T_j}\}$ with values 0 for padding positions and 1 for non-padding positions, such that:

$$\omega_{j,i} = \begin{cases} 0, & \text{if } e_{j,i} = 0 \text{ and } \hat{e}_{j,i} = 0 \\ 1, & \text{otherwise} \end{cases}.$$

Thus, our optimization objective is defined as the cross-entropy loss between the padded target and predicted skeleton sequences, which is calculated using their corresponding probability distributions over the set of possible tokens:

$$\mathcal{L} = -\frac{1}{B}\sum_{j=1}^{B}\sum_{i=1}^{T_j} \omega_{j,i}\, P(e_{j,i}) \log P\left(\hat{e}_{j,i}|e_{j,1},\ldots,e_{j,i-1},\mathbf{Z}_j\right). \tag{3.6}$$

Hence, our optimization problem can be expressed as

$$\Theta = \underset{\Theta}{\operatorname{argmin}}\ \mathcal{L}.$$

3.3.3 Dataset Generation

Here, we explain how to generate the synthetic equations and corresponding data used to train our Multi-Set Transformer. We build upon the generation method proposed by Biggio *et al.* [15], which, in turn, was based on that by Lample and Charton [98]. We start by generating expression trees whose number of non-leaf nodes (i.e., nodes that represent mathematical operators) is sampled between 3 and 7. Each operator is assigned a probability score of being sampled (see Table 3.2). These values were chosen arbitrarily with the main objective of generating sound and not-too-complex mathematical expressions. For instance, the probability score assigned to the basic operators `add`, `mul`, and `div` is 10, which is higher than other complex operators, such as `tanh`, whose score is 2. By doing so, we reduce the chance of generating expressions that only contain complex operations.

In addition, we forbid certain combinations of operators during the generation process to avoid numerical inconsistencies or redundancy. For example, we avoid embedding the operator `log` within the operator `exp`, or vice versa, since such composition could lead to direct simplification (e.g., $\texttt{log}\,(\texttt{exp}(x)) = x$). We also avoid some combinations of operators that would generate extremely large values (e.g., $\texttt{exp}\,(\texttt{exp}(x))$ and $\texttt{sinh}\,(\texttt{sinh}(x))$). Table 3.3 shows the two types of forbidden operators. Type 1 represents all possible combinations of operators `exp` and `log`, including the cases where the operators are repeated (i.e., $\texttt{exp}\,(\texttt{exp}(\cdot))$ and $\texttt{log}\,(\texttt{log}(\cdot))$). Similarly, Type 2 represents all possible combinations of operators `exp`, $\texttt{pow}(\cdot, 3)$, $\texttt{pow}(\cdot, 4)$, $\texttt{pow}(\cdot, 5)$, `sinh`, `cosh`, and `tanh`, including repetitions. It is worth mentioning that Biggio *et al.* used 13 operators for generating random expressions and did not forbid specific combinations of operations. Conversely, our approach incorporates 19

Table 3.2: Un-normalized sampling probabilities of the unary and binary operators.

Operator	**Arity**	**Probability**
`abs`	1	2
`acos`	1	2
`add`	2	10
`asin`	1	2
`atan`	1	2
`cos`	1	4
`cosh`	1	3
`div`	2	10
`exp`	1	4
`log`	1	4
`mul`	2	10
`pow`	2	4
`sin`	1	4
`sinh`	1	3
`sqrt`	1	4
`tan`	1	4
`tanh`	1	2

operators, thereby introducing an augmented level of complexity.

Each generated tree is traversed to derive a mathematical expression in prefix notation. We employ a pre-order traversal scheme, wherein the root node is visited before traversing its left and right subtrees recursively. The expression is then transformed from prefix to infix notation, which is simplified using SymPy[1], a symbolic manipulation library. The simplified expression is transformed again into an expression tree. Then, each non-numerical node of the tree is multiplied by a unique placeholder and then added to another unique placeholder. An exception applies to the `exp`, `sinh`, `cosh`, and `tanh` operators, whose arguments are not affected by additive placeholders. This is because adding a constant to their arguments would lead to direct simplification (e.g., $c_1 e^{c_2\, x + c_3} = (c_1 e^{c_2}) e^{c_3\, x}$). Figure 3.5 illustrates the random expression generation process. Note that Biggio *et al.* also proposed to include unique placeholders in the generated expressions; however, they included only one placeholder at a time (additive or multiplicative). In contrast, by including multiple placeholders in a single

[1] https://www.sympy.org/

Table 3.3: Types of forbidden combinations of operators.

Combination	Operators
Type 1	exp
	log
Type 2	`exp`
	$\texttt{pow}(\cdot, 3)$
	$\texttt{pow}(\cdot, 4)$
	$\texttt{pow}(\cdot, 5)$
	`sinh`
	`cosh`
	`tanh`

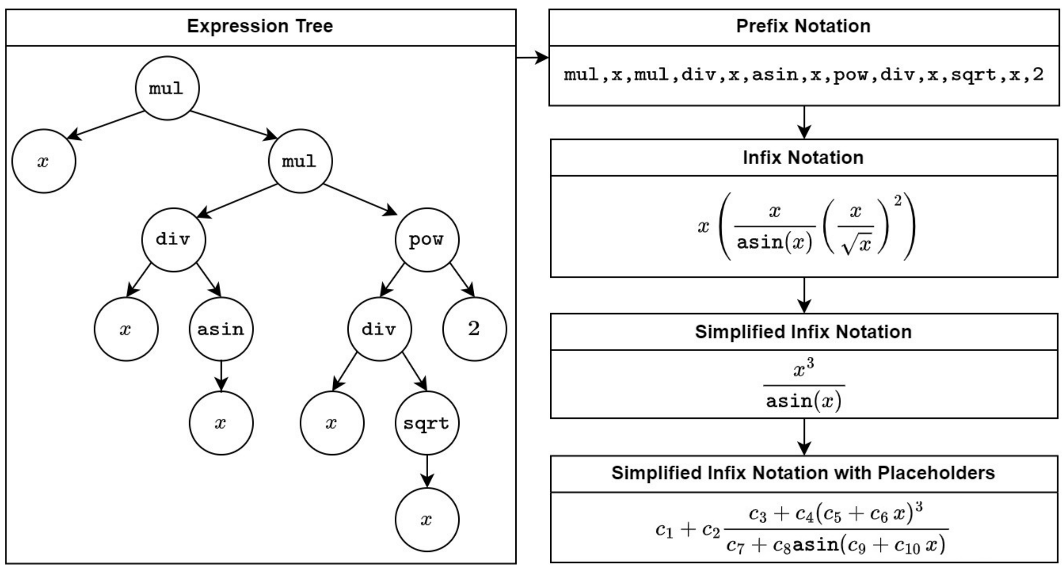


Figure 3.5: Example of a randomly generated expression.

expression, our approach generates more general expressions.

Each generated expression is stored in prefix notation. To prevent overly complex expressions, we impose a restriction on their maximum element count to ensure it remains below 20, as suggested by Lample and Charton [98] and Biggio *et al.* [15]. Our training dataset encompasses 1 million expressions while our validation set consists of 100,000 expressions. They can be viewed as sets of pre-generated expressions, with their constant values being randomly sampled each time they are accessed during the training process.

Algorithm 3.1 Multi-Set Transformer Training

Input: Pre-generated expressions $\mathbf{Q}$; initialized model g; number of input sets N_S; number of samples per input set n; batch size B
Output: Trained model g

function TRAINMODEL($\mathbf{Q}$, g, N_S, n, B)
  for each $t \in$ range(1, maxEpochs) **do**
    Batches $\leftarrow$ `getBatches`(N_T, B)
    for each batch $\in$ Batches **do**
      $\mathbf{E}^B, \hat{\mathbf{E}}^B \leftarrow [\,], [\,]$
      for each $j \in$ batch **do**
        $\mathbf{D}_j, \mathbf{e}_j =$ `generateSets`$(\mathbf{Q}[j], N_S, n)$
        $\hat{\mathbf{e}}_j =$ `forward`$(g, \mathbf{D}_j, \mathbf{e}_j)$
        $\mathbf{E}^B$`.append`$(\mathbf{e}_j)$
        $\hat{\mathbf{E}}^B$`.append`$(\hat{\mathbf{e}}_j)$
      $L \leftarrow \mathcal{L}(\mathbf{E}^B, \hat{\mathbf{E}}^B)$
      `update`(g, L)
  return g

Algorithm 3.1 shows the basic training routine of the Multi-Set Transformer. This algorithm takes as inputs the set of pre-generated expressions, denoted as $\mathbf{Q}$, the initialized model g, the number of input sets per data collection N_S, the number of samples per input set n, and the size of the mini-batches. Here, the function `getBatches` shuffles a list comprising indices ranging from 1 to $|\mathbf{Q}|$ (e.g., $|\mathbf{Q}|= 10^6$), and returns it in batches of B elements. Then, the function `generateSets`$(\mathbf{Q}[j], N_S, n)$ takes the j-th expression in $\mathbf{Q}$ and generates the skeleton $\mathbf{e}_j$ and corresponding data collection $\mathbf{D}_j$. This generation procedure is explained in detail in Algorithm 3.2. The estimated skeletons $\hat{\mathbf{e}}_j$ are obtained using the function `forward`$(g, \mathbf{D}_j, \mathbf{e}_j)$, which processes the input $\mathbf{D}_j$ and target $\mathbf{e}_j$ through the network g using the training configuration explained in Figure 3.4. Function $\mathcal{L}(\mathbf{E}^B, \hat{\mathbf{E}}^B)$ represents the loss function (Eq. 3.6) while `update`(g, L) encompasses the conventional backpropagation and gradient descent processes used to update the weights of model g.

As previously mentioned, Algorithm 3.2 obtains the actual skeletons $\mathbf{e}_j$ and corresponding data $\mathbf{D}_j$ for training. It takes as input a pre-generated expression **ex** ($\mathbf{Q}[j]$

Algorithm 3.2 Multi-Set Transformer Data Generation

Input: Pre-generated expression **ex**; number of input sets N_S; number of samples per input set n
Output: Collection **D** of N_S sets; common skeleton **e**

function GENERATESETS($\mathbf{ex}, N_S, n$)
    $\mathbf{c}, n_c \leftarrow$ `getConstants`($\mathbf{ex}$)
    $n_f \leftarrow$ `randInt`($2, n_c$)
    $\mathbf{ex} \leftarrow$ `selectConstants`($\mathbf{ex}, \mathbf{c}, n_f$)
    $\mathbf{D} \leftarrow [\,]$
    $s \leftarrow 1$
    while $s \leq N_S$ **do**
        $f^{(s)} =$ `sampleConstants`($\mathbf{ex}$)
        $\mathbf{X}_v^{(s)} \leftarrow$ `sampleSupport`(n)
        $\mathbf{X}_v^{(s)}, f^{(s)}, \mathrm{Xsing} =$ `avoidNaNs`($\mathbf{X}_v^{(s)}, f^{(s)}$)
        $\mathrm{temp} = \kappa(f^{(s)})$
        if $s = 1$ **then**
            $\mathbf{e} = \mathrm{temp}$
        else
            while $\mathrm{temp} \neq \mathbf{e}$ **do**   ▷ Verify that all sets correspond to the same skeleton
                `continue`
        $\mathbf{y}^{(s)} = f^{(s)}\left(\mathbf{X}_v^{(s)}\right)$
        $\mathbf{D}$.`append`$\left((\mathbf{X}_v^{(s)}, \mathbf{y}^{(s)})\right)$
        $s \leftarrow s + 1$
    return $\mathbf{D}, \mathbf{e}$

in Algorithm 3.1) with labeled additive and multiplicative placeholders. The function `getConstants`($\mathbf{ex}$) returns $\mathbf{c}$, the list of constant placeholders in **ex**, and $n_c = |\mathbf{c}|$. The number of constants we use from **ex**, n_f, is decided randomly ($2 \leq n_f \leq n_c$). Then, the function `selectConstants`($\mathbf{ex}, \mathbf{c}, n_f$) retrieves a new expression with n_f constant placeholders randomly selected. Note that the constant placeholders that were not selected are replaced by the values 0, in the case of additive placeholders, and 1, in the case of multiplicative placeholders. For instance, consider $\mathbf{ex} = c_1 + c_2 \tan(c_3 + c_4 x)$, where $n_c = 4$. Suppose we randomly select $n_f = 3$ constants; then, we could obtain the following expression: $\mathbf{ex} = c_1 + \tan(c_4 x)$; i.e., $c_2 = 1$ and $c_3 = 0$. Hence, expressions with slightly different skeletons are generated at each epoch despite corresponding to the same pre-generated expressions,

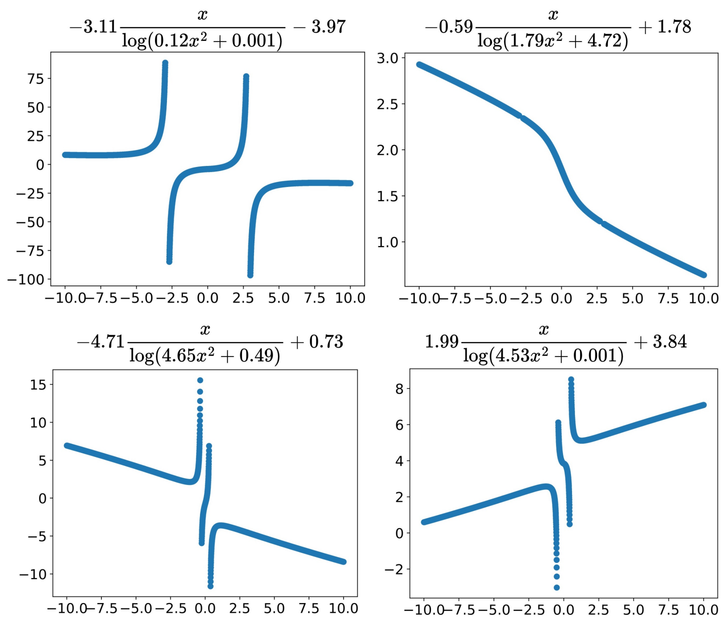


Figure 3.6: Example of four input–response pairs sets generated from $\mathbf{e}(x) = \frac{c_1\, x}{\log(c_2 x^2 + c_3)} + c_4$.

thus adding more diversity to the training data. It is worth pointing out that the approach by Biggio *et al.* consists of sampling skeleton expressions with up to three constants. We argue that this entails an important limitation on the type of skeletons that can be generated. For instance, the skeleton $\frac{c\, e^{cx}}{\sin(cx)+c}$ could not be handled by their approach.

Once the expression **ex** has been defined, we generate the underlying functions $f^{(s)}$ corresponding to each of the N_S sets. This is achieved by using the function `sampleConstants`(**ex**), which samples the values of each constant in **ex** from a uniform distribution $\mathcal{U}(-10, 10)$ independently. For each set, n input points $\mathbf{X}_v^{(s)}$ are sampled via `sampleSupport`(n), also from $\mathcal{U}(-10, 10)$. Although the range $[-10, 10]$ is arbitrary, it is used consistently across all sets and expressions to ensure uniformity.

In addition, the function `avoidNaNs`($\mathbf{X}_v^{(s)}, f^{(s)}$) may modify some of the coefficients in $f^{(s)}$ or sample additional $\mathbf{X}_v^{(s)}$ values to avoid numerical inconsistencies. It also returns the variable `Xsing` that specifies the x positions at which the function $f^{(s)}(x)$ would output undefined values. For example, if $f^{(1)} = \sqrt{2x}$ and the minimum value in $\mathbf{X}^{(1)}$ is -10, `avoidNaNs` modifies $f^{(1)}$ so that there are no negative numbers inside the square root operation: $f^{(1)}(x) = \sqrt{2x+20}$. Operations such as division and `tan` exhibit singularities at specific input values (e.g., division by zero), which, if encountered within $\mathbf{X}_v^{(s)}$, are replaced by values that prevent undefined results or extremely large values. Figure 3.6 shows an example of four sets of input–response pairs generated from the symbolic skeleton $c_1\frac{x}{\log(c_2x^2+c_3)}c_4$. Here, the division and `log` operations may produce undefined values for certain input and coefficient values; however, `avoidNaNs` prevents this from happening, obviating the need for protected operators. More details about the `avoidNaNs` function are provided in the following section. Note that Biggio *et al.* suggested dropping out the input–response pairs that lead to undefined values. A major drawback of this approach is that it is possible to generate functions whose outputs are predominantly undefined; e.g., $\log(x-8)$ is undefined for $x \in [-10, 8]$. Therefore, removing input–response pairs with undefined values would lead to creating unrepresentative input sets with varying lengths.

Notice that after using the `avoidNaNs` function, the skeleton $f^{(s)}$ may have been altered. Thus, we include in Line 15 a while loop that prevents the algorithm from continuing until it generates a function with the same skeleton $\mathbf{e}$ as that of the previous function $f^{(s-1)}$. Each function $f^{(s)}$ is evaluated on the input values $\mathbf{X}_v^{(s)}$ to obtain the corresponding responses $\mathbf{y}^{(s)}$. As such, the algorithm returns the common skeleton $\mathbf{e}$ alongside a collection of N_S sets $\left\{\left(\mathbf{X}^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mathbf{X}^{(N_S)}, \mathbf{y}^{(N_S)}\right)\right\}$, each consisting of n input–response pairs.

3.3.4 Avoiding Invalid Operations

In this section, we present a detailed description of the function $\texttt{avoidNaNs}(\mathbf{x}, f)$ introduced in Algorithm 3.2. This function modifies certain coefficients within the function f or generates supplementary support values within the vector $\mathbf{x}$ to avoid numerical inconsistencies. We classify the operators that may generate undefined values as follows:

- Single-bounded operators: These operators have bounded numerical arguments due to the mathematical constraints imposed on their domains. We consider the following single-bounded operators:
  - Logarithm (`log`): It cannot process inputs lower than or equal to 0:

$$\text{Domain}\left(\texttt{log}(x)\right) = \{x \in \mathbb{R} \mid x > 0\}.$$

  - Square root (`sqrt`): Bounded on its left side to avoid generating complex numbers:

$$\text{Domain}\left(\texttt{sqrt}(x)\right) = \{x \in \mathbb{R} \mid x \geq 0\}.$$

  - Exponential (`exp`): Bounded on its right side to avoid extremely large values. A maximum input of 7 was selected manually to keep the output scale comparable to other operators and their combinations.

$$\text{Domain}\left(\texttt{exp}(x)\right) = \{x \in \mathbb{R} \mid x \leq 7\}.$$

- Double-bounded operators: Unlike the single-bounded operators, the numerical arguments of these operators are bounded on their left and right sides:
  - Arcsine (`asin`): It takes a value between -1 and 1 as its input and returns the angle whose sine is equal to that value:

$$\text{Domain}\left(\texttt{asin}(x)\right) = \{x \in \mathbb{R} \mid -1 \leq x \leq 1\}.$$

  - Arccosine (`acos`): Like arcsine, it also takes a value between -1 and 1 as its input:

$$\text{Domain}\left(\texttt{acos}(x)\right) = \{x \in \mathbb{R} \mid -1 \leq x \leq 1\}.$$

- Operators with singularities: Operators like the tangent ($\texttt{tan}$) or division ($\texttt{div}$) can exhibit singularities at specific input values. For instance, the tangent function becomes undefined when its input equals an odd multiple of $\pi/2$ (e.g., $\texttt{tan}(-\pi/2)$, $\texttt{tan}(\pi/2)$), resulting in an asymptotic behavior where the function approaches infinity.

Algorithm 3.3 shows the method proposed to address the NaN ("not a number") values arising from the aforementioned functions, which we refer to as "special functions". Here, $f.\texttt{args}$ returns a vector containing the numerical arguments of function f. Consider the function $f(x) = \texttt{asin}(\texttt{log}(x)) + x^2 + 3$; in this case, $f.\texttt{args} = [\texttt{asin}(\texttt{log}(x)), x^2, 3]$. If f is considered as an expression tree, $f.\texttt{func}$ returns the name of the operator located at the top of the tree. In the previous example, $f.\texttt{func} = \texttt{add}$. The algorithm analyzes each argument of f separately (Line 4). A given argument of f, denoted as $\texttt{arg}$, can be considered as a sub-expression tree. Hence, the function $\texttt{containSpecialF}(\text{arg})$ (Line 5) traverses the sub-expression tree and returns a true value if any special function is found within.

If the current sub-expression tree contains a special function, we check if the top operator of the subtree is special using function $\texttt{isSpecialF}(\text{arg}.\texttt{func})$ (Line 7). If not, we move down to a deeper level of the subtree using recursion (Line 8). Otherwise, the arguments of the sub-expression $\texttt{arg}$ may need to be modified to avoid undefined values. Before doing so, in Line 11, we verify if there is another special function contained inside the current subtree, in which case we explore a deeper level of the subtree using recursion (Line 12). Note that we apply the $\texttt{containSpecialF}(\text{arg}.\texttt{args}[0])$ function given that $\texttt{arg}$ is guaranteed to represent a special function and, as such, it is a unary function with a single argument. Consider the example introduced in the above paragraph. When analyzing the first argument, $\text{arg} = \texttt{asin}(\texttt{log}(x))$, we verify that the top operator of the current subtree is special (i.e., $\text{arg}.\texttt{func} = \texttt{asin}$). Thus, we may need to modify its inner argument, $\texttt{log}(x)$, to produce valid input values between -1 and 1. Notice that the inner argument $\texttt{log}(x)$ constitutes a sub-expression tree that contains a special function, $\texttt{log}$, whose corresponding inner argument, x, needs to be modified to produce valid input values greater than 0.

The function represented by the inner argument of the current function `arg` is denoted as `innArg` (Line 13). The function `arg` is then evaluated on the input values $\mathbf{x}$ and the obtained values are stored in the vector `vals` (Line 14). The function $\texttt{containNaNs}(\mathrm{vals})$ returns a true value in the event that one or more undefined values are found within the vector `vals`. This criterion also encompasses instances where the absolute values of numbers are exceedingly large (e.g., $> 10^5$). If an undefined value is found, a modification of the inner argument or the input vector $\mathbf{x}$ is needed. To do so, we first evaluate the function represented by `innArg` on the input values $\mathbf{x}$ and store the outcomes in the variable innArg (Line 16). In the case that the function `arg` is single-bounded, the domain of function `innArg` is modified accordingly using $\texttt{modifySBounded}(\mathrm{arg}, \mathrm{vals}, \mathrm{innArg}, \mathrm{innVals})$ (Line 18). Likewise, if `arg` is double-bounded, the domain of `innArg` is modified accordingly using $\texttt{modifyDBounded}(\mathrm{arg}, \mathrm{innArg}, \mathrm{innVals})$ (Line 20). In the case that `arg` represents a function with singularities, a new input vector $\mathbf{x}$ with resampled values is obtained using the function $\texttt{avoidSingularities}(\mathbf{x}, \mathrm{arg}, \mathrm{Xsing})$ (Line 22), where `Xsing` is a variable that will store all positions at which the function produces undefined values and is initialized as an empty list. The inner argument of function `arg` is then replaced with the modified function `innArg` (Line 24). Finally, the arguments of the original function f are replaced with the modified arguments stored in the list `newArgs` (Line 26).

Algorithm 3.4 shows the implementation of the function `modifySBounded`. Here, the method by which the inner argument of the function `arg` is altered depends on whether it is bounded on its left or right side. The function $\texttt{getSBound}(\mathrm{arg})$ returns the bound type and the corresponding bound value. The bound type is equal to “min” if `arg` is bounded on its left side, and “max” otherwise. In the first case, we count the number of NaN values present in `vals` using the function $\texttt{countNaNs}(\mathrm{vals})$. If the count of NaN values exceeds the count of non-NaN values, both `innArg` and innVals are substituted with their respective negations

Algorithm 3.3 Avoiding Invalid Operations

Input: Support points **x**; function f; initial singularity set `Xsing`; samples per input set n
Output: Updated **x**, f, and `Xsing`

```
1: function avoidNaNs(x, f, Xsing = [ ])
2:     args ← f.args
3:     newArgs ← [ ]
4:     for each arg ∈ args do
5:         if containSpecialF(arg) then
6:             // If the operator at the top of the current sub-expression tree is special
7:             if !isSpecialF(arg.func) then
8:                 x, arg, Xsing ← avoidNaNs(x, arg, Xsing)
9:             else
10:                // Check if there's another special function inside the current function
11:                if containSpecialF(arg.args[0]) then
12:                    x, arg, Xsing ← avoidNaNs(x, arg, Xsing)
13:                innArg ← arg.args[0]
14:                vals ← arg(x)
15:                if containNaNs(vals) then
16:                    innVals ← innArg(x)
17:                    if isSingleBounded(arg) then
18:                        innArg ← modifySBounded(arg, vals, innArg, innVals)
19:                    else if isDoubleBounded(arg) then
20:                        innArg ← modifyDBounded(arg, innArg, innVals)
21:                    else                                      ▷ Operations with singularities
22:                        x, Xsing ← avoidSingularities(length(x), arg, Xsing)
23:                arg.args[0] ← innArg                                  ▷ Update function
24:        newArgs.append(arg)
25:    f.args ← newArgs
26:    return x, f, Xsing
```

(Line 6). This approach helps in cases where all of the inner argument values are less than the given bound of `arg`. By negating `innArg` and, consequently, innVals, all inner argument values become greater than the bound, ensuring that they fall within the valid input domain of `arg`. If after this operation there are values in innVals that are still lower than the bound, we add a horizontal offset as shown in Line 9. For example, if $\text{arg}(x) = \texttt{sqrt}(x)$ and $x \in [-10, 10]$, the modified inner argument is given by $\text{innArg} = x + 10$; therefore, the

Algorithm 3.4 Modifying Single-Bounded Operations

Input: Function `arg` and its evaluated values `vals`; inner argument `innArg` and its evaluated values `innVals`
Output: Updated inner argument `innArg`

```
1: function modifySBounded(arg, vals, innArg, innVals)
2:     boundType, bound ← getSBound(arg)
3:     if boundType = “min” then
4:         NaNs ← countNaNs(vals)
5:         if NaNs > length(vals) − NaNs then
6:             innArgs, innVals ← −innArgs, −innVals
7:         if any (innVals < bound) then
8:             minVal ← minimum(innVals)
9:             innVals ← innVals + (bound − minVal)
10:    else
11:        innArg ← innArg/maximum(|innVals|) ∗ bound
12:    return innArg
```

Algorithm 3.5 Modifying Double-Bounded Operations

Input: Function `arg`; inner argument `innArg` and its evaluated values `innVals`
Output: Updated inner argument `innArg`

```
function modifyDBounded(arg, innArg, innVals)
    minBound, maxBound ← getDBound(arg)
    innArg ← (innArg − minimum(innVals)) /(maximum(innVals) − minimum(innVals))
    innArg ← innArg ∗ (maxBound − minBound) + minBound
    return innArg
```

modified function $\arg(x) = \texttt{sqrt}(x + 10)$ produces defined numbers for all values of x. On the other hand, if `arg` is bounded on its left side, we simply scale its inner argument as shown in Line 11 so that it does not produce values greater than the maximum bound.

Furthermore, Algorithm 3.5 presents the implementation details of the function `modifyDBounded`, which is utilized for modifying double-bounded functions, Here, the inner argument `innArg` is scaled in such a way that its minimum and maximum values are equal to the minimum and maximum bounds of the input domain of function `arg`.

Finally, Algorithm 3.6 details our approach for preventing the generation of NaN values

Algorithm 3.6 Handling Operations with Singularities

Input: Support's desired length, `len`; function `arg`; initial singularity set `Xsing`
Output: Updated inner argument `innArg`

```
1: function avoidSingularities(len, arg, Xsing)
2:     arg2 ← 1/arg
3:     xs ← linspace(−10, 10, 1000)                 ▷ Declare initial potential x values
4:     ys ← arg2(xs)
5:     sing ← whereIsZero(xs, ys)                   ▷ Find x values where arg2(xs) ≈ 0
6:     Xsing.append(sing)
7:     Xsing ← sort(Xsing)
8:     pairs ← validIntervals(−10, 10, Xsing)
9:     // Generate points within the list of valid intervals
10:    newX ← []
11:    totalL ← calcLength(pairs)        ▷ Calculate the total length across all valid intervals
12:    for each (start, end) in pairs do  ▷ Generate equidistant points within the current interval
13:        Npoints ← len(end − start)/totalL
14:        points ← []
15:        step ← (end − start)/(Npoints − 1)
16:        for each j ∈ [1, Npoints] do
17:            points.append(start + (j − 1) ∗ step)
18:        newX.append(points)
19:    return newX, Xsing
```

when the top operator of the sub-expression tree `arg` exhibits singularities at specific input values. The function `avoidSingularities`(len, arg, Xsing) receives as arguments the desired length of the support set, `len`; the currently analyzed sub-expression, `arg`; and a list of input values `Xsing` that produced undefined values in previously evaluated sub-expressions and that we should avoid. In this work, we consider that the operators that exhibit singularities are those that imply a division by zero at one or more positions, such as division or tangent. Hence, in Line 2, we take the inverse of `arg` to find the x values at which $\text{arg2}(x) = \frac{1}{\text{arg}(x)} \approx 0$, which is equivalent to finding the values at which $\text{arg}(x)$ is undefined; i.e., $\text{arg}(x) \approx \infty$. We generate the vector xs using the function `linspace`$(-10, 10, 1000)$, which returns 1000 evenly spaced numbers between -10 and 10. As explained in Section 3.3.3, this range was chosen

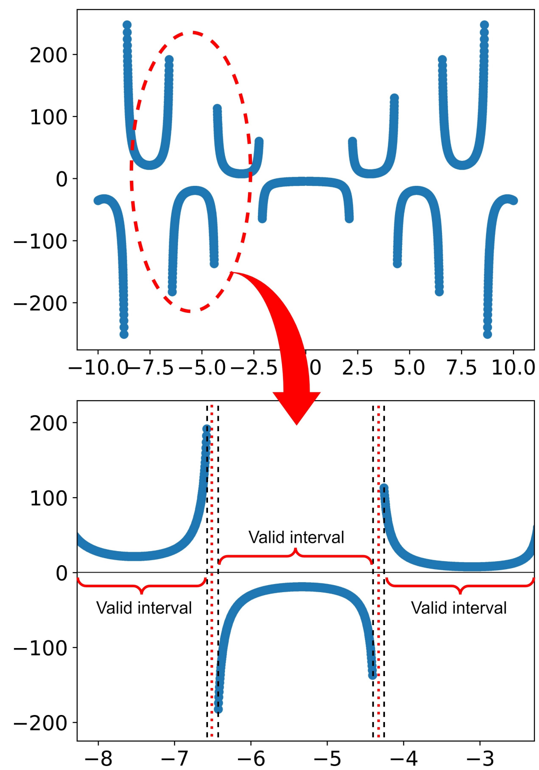


Figure 3.7: A generation example using function $f(x) = \frac{-3.12x}{\mathtt{sin}(1.45x)} - 2.2$. (Top) Generated data on the entire domain $[-10, 10]$. (Bottom) Detailed view of how singularities are avoided.

to ensure consistency across all generated expressions. Then, xs is evaluated on `arg2` so that $ys = \text{arg2}(x)$. These vectors are used to find the values of xs at which ys is close to zero. Specifically, the function `whereIsZero`(xs, ys) returns a vector `sing` containing the values of xs at which ys crosses the horizontal axis, indicating it is close to a singularity point. The vector `sing` is then added to `Xsing`, which contains singularity points of previously analyzed sub-expressions that we should avoid when analyzing the current sub-expression.

Furthermore, the function `validIntervals`$(-10, 10, \text{Xsing})$ returns a list of pairs that indicate intervals of values between -10 and 10 that do not produce NaN values. For example, if two singular values were encountered at $x = 2$ and $x = 5$, the function `validIntervals`$(-10, 10, [2, 5])$ would return the list $[(-10, 1.95), (2.05, 4.95), (5.05, 10)]$

indicating that the intervals $(-10, 1.95)$, $(2.05, 4.95)$ and $(5.05, 10)$ are free of NaN values. Note that we used a threshold of 0.05 to avoid getting too close to the singular points. Once we find the valid input intervals for the current sub-expression, we populate the vector of input values, `newX`. To do this, we generate equidistant points within each interval so that the total number of generated points is *len* and the number of points generated within each interval is proportional to the length of the interval (Lines 10–19). Fig. 3.7 depicts an example of how data are generated using the underlying function $f(x) = \frac{-3.12x}{\texttt{sin}(1.45x)} - 2.2$. The figure at the bottom shows in detail that the `avoidSingularities` function finds two singular points within the zoomed-in range, whose positions are highlighted by the red dotted line. From this, the valid intervals are calculated avoiding getting too close to the singular points. The limits of the valid intervals are represented by the black dashed lines.

3.4 Experimental Results

A training dataset[2] consisting of one million pre-generated expressions ($|\mathbf{Q}| = 10^6$) has been created to train the Multi-Set Transformer. These expressions allow up to one nested operation and contain a maximum of five unary operators. We also generated an independent validation set consisting of 10^5 expressions. For the model architecture, due to the memory limit of the available graphic processing units (GPU)[3] and the high computational expense associated with training a single model, we used a one-factor-at-a-time approach to choose the following hyperparameters:

- Number of input sets: $N_S = 10$.
- Number of input–response pairs in each input set: $n = 3000$.
- Optimizer: Adadelta [198] with an initial learning rate of 0.0001.

[2] The code and datasets are available at https://github.com/NISL-MSU/MultiSetSR
[3] Four NVIDIA H100 GPUs at the Tempest Research Cluster: https://montana.edu/uit/rci/tempest/

Table 3.4: Comparison of the number of trainable parameters of the different transformer-based SR methods.

SR Transformer Model	**# Parameters**
NeSymReS [15]	26,395,708
E2E [82]	93,451,508
Multi-Set Transformer (ours)	23,094,304

- Batch size: $B = 16$.
- Architecture:
  - Number of ISAB blocks in the encoder: $\ell = 3$.
  - Number of decoder blocks: $M = 5$.
  - Embedding size: $d = 512$.
  - Number of heads: $h = 8$.

Table 3.4 compares the number of trainable parameters across different transformer-based SR methods. Although E2E and NeSymRes address SSP problems (i.e., single-set inputs), their complexity exceeds that of the Multi-Set Transformer, which simultaneously processes multiple sets. Since E2E and NeSymRes solve a different problem type, their model outputs are not directly comparable to those of the Multi-Set Transformer. In Chapter 4, we analyze the SR frameworks in which these models were introduced and compare the complete mathematical expressions they generate.

Figure 3.8 depicts the training and validation curves obtained after training the Multi-Set Transformer during 33 epochs on dataset $\mathbf{Q}$. No signs of overfitting were observed, suggesting that the model complexity is appropriate for solving the MSSP problem. Although the curve trend indicates a potential early convergence stage, training was halted due to computational constraints, requiring approximately two weeks for 33 epochs.

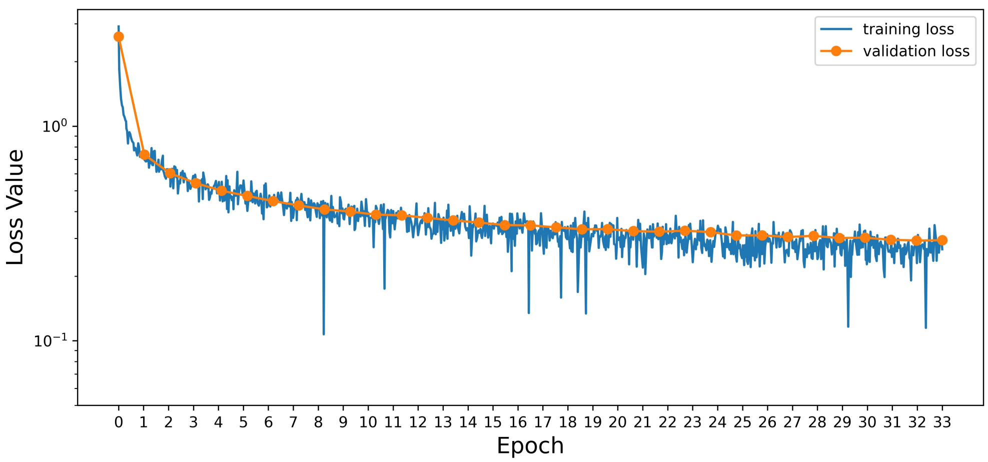


Figure 3.8: Learning curves obtained from training the Multi-Set Transformer.

To demonstrate the performance of the Multi-Set Transformer, we randomly select 100 skeletons from the validation set. For each skeleton, we generate $N_S = 10$ input–response sets using Algorithm 3.2 and feed them into the trained model. Table 3.5 compares the predicted skeletons with their target counterparts, highlighting incorrect predictions. Some predicted skeletons are not highlighted despite differing from the target skeletons. For example, in case 53, the model predicts $\hat{\mathbf{e}} = c_1 + 1/\sin(c_2 + x_1)$, while the target skeleton is $\mathbf{e} = c_1 + 1/\cos(c_2 + x_1)$. However, these expressions are mathematically equivalent since the subexpression $\sin(c + x_1)$ can be rewritten as $\cos((c + 3\pi/2) + x_1) = \cos(c' + x_1)$ by applying a phase shift. A similar equivalence holds for cases 75 and 83.

It is worth noting that the results in Table 3.5 correspond to a single sequence produced by the Multi-Set Transformer. In Chapter 4, we describe how to generate multiple distinct skeleton candidates using a diverse beam search strategy [182]. Additionally, the N_S input sets are sampled multiple times to enhance variability in the resulting skeletons. By generating multiple and diverse skeleton candidates, we increase the likelihood of recovering one or multiple skeletons with the correct mathematical structure.

3.5 Summary

In this chapter, we introduced a new problem termed Multi-Set Symbolic Skeleton Prediction. MSSP is an extension of the Symbolic Skeleton Prediction problem previously explored in existing research. The MSSP problem involves analyzing multiple sets of input-response pairs simultaneously. Assuming that all input sets have been generated from functions that share the same functional form, the objective is to generate a univariate symbolic skeleton that characterizes the underlying functional form of all input sets.

We tackle the MSSP problem using a novel transformer architecture called Multi-Set Transformer, which comprises an encoder and a decoder. The encoder maps information from multiple input sets into a shared latent representation, while the decoder generates symbolic skeletons as sequences conditioned on this representation. The chapter provides a detailed overview of the model architecture, including its internal components, mechanisms, and training process. Additionally, we described the construction of our large synthetic dataset and the process of generating training data from it. Experimental results demonstrated that the proposed method effectively recovers skeletons that approximate the target skeletons shared across all input–response sets. This capability makes the Multi-Set Transformer well-suited for tackling multivariate symbolic regression in a decomposable manner; that is, by analyzing one variable at a time, as discussed in the next chapter.

Table 3.5: Comparison of target and estimated skeletons on 100 validation skeletons.

#	$\mathbf{e}(x)$	$\hat{\mathbf{e}}(x)$	#	$\mathbf{e}(x)$	$\hat{\mathbf{e}}(x)$
1	$c_1\sqrt{c_2+\lvert x_1\rvert}+c_3$	$c_1\sqrt{c_2+\lvert x_1\rvert}+c_3$	51	$c_1+(c_2+x_1)^4$	$c_1+(c_2+x_1)^4$
2	$c_1+\tan(c_2x_1)/x_1$	$c_1+\tan(c_2x_1)/x_1$	52	$c_1+(c_2+x_1)^3$	$c_1+(c_2+x_1)^3$
3	$c_1+\log(c_2x_1+c_3)$	$c_1+\log(c_2x_1+c_3)$	53	$c_1x_1^3+c+1/x_1$	$c_1x_1^3+c_2$
4	$c_1x_1+c_2\sqrt{c_3+x_1}+c_4$	$c_1x_1+c_2\sqrt{c_3+x_1}+c_4$	54	$c_1+\cosh(c_2x_1+c_3)$	$c_1+\cosh(c_2x_1+c_3)$
5	$c_1\exp(x_1)+c_2$	$c_1\exp(x_1)+c_2$	55	$c_1+1/(c_2+x_1^2)$	$c_1+1/(c_2+x_1^2)$
6	$c_1\tan(c_2+x_1)+c_3$	$c_1\tan(c_2+x_1)+c_3$	56	$c_1x_1^3+c_2$	$c_1x_1^3+c_2$
7	$c_1+\tanh(c_2\lvert c_3+x_1\rvert)$	$c_1+\tanh(c_2+\lvert c_3+x_1\rvert)$	57	$c_1+1/\sin(c_2+x_1)$	$c_1+1/\cos(c_2+x_1)$
8	$c_1\tanh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	$c_1\tanh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	58	$c_1(c_2+x_1)^2+c_3$	$c_1(c_2+x_1)^2+c_3$
9	$c_1+c_2/\lvert c_3+x_1\rvert$	$c_1+c_2/\lvert c_3+x_1\rvert$	59	$c_1+\sin(c_2x_1)^2$	$c_1+\sin(c_2x_1)^2$
10	$c_1\cosh(c_2x_1+c_3)+c_4$	$c_1\cosh(c_2x_1+c_3)+c_4$	60	$c_1+x_1+\tan(c_2+x_1)$	$c_1+x_1+\tan(c_2+x_1)$
11	$c_1\sqrt{c_2+x_1}+c_3$	$c_1\sqrt{c_2+x_1}+c_3$	61	$c_1+c_2\cos(c_3x_1+c_4)/x_1$	$c_1+c_2\cos(c_3x_1+c_4)/x_1$
12	$c_1+\cosh(c_2x_1+c_3)$	$c_1+\cosh(c_2x_1+c_3)$	62	$c_1+\tanh(c_2\lvert x_1\rvert+c_3)$	$c_1+\tanh(c_2\lvert x_1\rvert+c_3)$
13	$c_1+c_2/\tanh(c_3x_1)$	$c_1+c_2/\tanh(c_3x_1)$	63	$c_1+c_2/(c_3x_1+c_4)^2$	$c_1+c_2/(c_3x_1+c_4)^2$
14	$c_1\exp(c_2\sin(c_3x_1))+c_4$	$c_1^{c_2\sin(c_3x_1)}+c_4$	64	$c_1\sin(c_2x_1^4+c_3)+c_4$	$c_1\sin(c_2x_1^4+c_3)+c_4$
15	$c_1(c_2+x_1)^3+c_3$	$c_1(c_2+x_1)^3+c_3$	65	$c_1\exp(c_2\sin(c_3x_1+c_4))+c_5$	$c_1\exp(c_2\sin(c_3x_1+c_4))+c_5$
16	$c_1\exp(c_2\sin(c_3x_1))+c_4$	$c_1\exp(c_2\sin(c_3+x_1))+c_4$	66	$c_1(c_2+x_1)^3+c_3$	$c_1(c_2+x_1)^3+c_3$
17	$c_1(c_2+x_1)^2+c_3$	$c_1(c_2+x_1)^2+c_3$	67	$c_1(c_2\log(c_3x_1+c_4)$ $+c_5)^2+c_6$	$c_1(c_2\log(c_3x_1+c_4)$ $+c_5)^2+c_6$
18	$c_1+(c_2+x_1)^4$	$c_1+(c_2+x_1)^4$	68	$c_1\exp(x_1)+c_2$	$c_1\exp(x_1)+c_2$
19	$c_1\tanh(c_2\lvert x_1\rvert+c)_3+c_4$	$c_1\tanh(c_2\lvert x_1\rvert+c)_3+c_4$	69	$c_1+\log(c_2x_1+c_3)$	$c_1+\log(c_2x_1+c_3)$
20	$c_1\sqrt{c_2+x_1}+c_3$	$c_1\sqrt{c_2+x_1}+c_3$	70	$c_1x_1^4+c_2$	$c_1x_1^4+c_2$
21	$c_1\exp(c_2x_1)+c_3$	$c_1\exp(c_2x_1)+c_3$	71	$c_1\cosh(c_2\sqrt{c_3x_1+c_4})+c_5$	$c_1\cosh(c_2\sqrt{c_3x_1+c_4})+c_5$
22	$c_1\sinh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	$c_1\sinh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	72	$c_1+\log(c_2x_1+c_3)^4$	$c_1+\log(c_2x_1+c_3)^4$
23	$c_1+c_2/(c_3\sqrt{c_4+x_1}+c_5)$	$c_1+c_2/(c_3\sqrt{c_4+x_1}+c_5)$	73	$c_1+c_2\log(c_3x_1+c)/x_1$	$c_1+c_2\log(c_3x_1+c)/x_1$
24	$c_1\cosh(c_2\lvert c_3+x_1\rvert+c_4)+c_5$	$c_1\cosh(c_2\lvert c_3+x_1\rvert+c_4)+c_5$	74	$c_1x_1+c_2\tan(c_3x_1)+c_4$	$c_1x_1+c_2\tan(c_3x_1)+c_4$
25	$c_1\exp(c_2\lvert c_3+x_1\rvert)+c$	$c_1\exp(c_2\lvert c_3+x_1\rvert)+c$	75	$c_1\cos(c_2x_1+c_3)+c_4$	$c_1\sin(c_2x_1+c_3)+c_4$
26	$c_1\tan(x_1)+$ $c_2\tanh(c_3x_1+c_4)+c_5$	$c_1\tan(x_1)\tanh(c_3x_1$ $+c_4)+c_5$	76	$c_1+x_1\exp(c_2x_1)$	$c_1+\exp(c_2x_1)$
27	$c_1(c_2+x_1)^2+c_3$	$c_1(c_2+x_1)^2+c_3$	77	$c_1\sin(c_2\sqrt{c_3x_1+c_4})+c_5$	$c_1\cos(c_2\sqrt{c_3x_1+c_4})+c_5$
28	$c_1+c_2/(c_3(c_4+x_1)^2+c_5)$	$c_1+c_2/(c_3(c_4+x_1)^2+c_5)$	78	$c_1\tan(c_2\sqrt{c_3x_1+c_4}+c_5)+c_6$	$c_1\tan(c_2\sqrt{c_3x_1+c_4}+c_5)+c_6$
29	$c_1+1/(c_2+\tanh(c_3x_1))$	$c_1+\tanh(c_2x_1)$	79	c_1+ $c_2/(c_3+\log(c_4x_1+c_5))$	$c_1(c_2+\log(c_3x_1+c_4))+c_5$
30	$c_1\sqrt{c_2+c_3/(c_4x_1+c_5)}+c$	$c_1\sqrt{c_2+c_3/(c_4x_1+c_5)}+c$	80	$c_1x_1+c_2+x_1^2$	$c_1x_1+c_2+x_1^2$
31	$c_1\tan(c_2x_1+c_3)+c_4$	$c_1\tan(c_2x_1+c_3)+c_4$	81	$c_1+1/(c_2+\sin(c_3x_1))$	$c_1+1/(c_2+\sin(c_3x_1))$
32	$c_1\cos(c_2x_1)+c_3$	$c_1\cos(c_2x_1)+c_3$	82	$c_1\cos(c_2x_1)+c_3$	$c_1\cos(c_2x_1)+c_3$
33	$c_1x_1^2+c_2$	$c_1x_1^2+c_2$	83	$c_1\cos(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	$c_1\cos(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$
34	$c_1+c_2/(c_3+x_1^3)$	$c_1+c_2/(c_3+x_1^3)$	84	$c_1+c_2/(c_3+x_1^4)$	$c_1+c_2/(c_3+x_1^4)$
35	$c_1\sinh(c_3x_1)+c_4$	$c_1\sinh(c_3x_1)+c_4$	85	$c_1+x_1\log(c_2x_1^2)$	$c_1+x_1\log(c_2+x_1^2)$
36	$c_1\sinh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	$c_1\cosh(c_2\sqrt{c_3x_1+c_4}$ $+c_5)+c_6$	86	$c_1x_1^4+c_2$	$c_1x_1^4+c_2$
37	$c_1+\exp(c_2\sin(x_1))$	$c_1+\exp(c_2\sin(x_1))$	87	$c_1\exp(\sin(x_1))+c_2$	$c_1\exp(\sin(x_1))+c_2$
38	$c_1\cosh(c_2\lvert c_3+x_1\rvert+c_4)+c_5$	$c_1\cosh(c_2\lvert c_3+x_1\rvert+c_4)+c_5$	88	$c_1+\cos(c_2\sinh(c_3x_1+c_4))$	$c_1+\cos(c_2\sinh(c_3x_1+c_4))$
39	$c_1x_1^4+c_2$	$c_1x_1^4+c_2$	89	$c_1+c_2/(c_3(c_4+x_1)^3+c_5)$	$c_1+c_2/(c_3(c_4+x_1)^3+c_5)$
40	$c_1+\sin(c_2+x_1)^4$	$c_1+\sin(c_2+x_1)^4$	90	$c_1\sin(c_1+c_2/(c_3+x_1))+c_4$	$c_1\sin(c_1+c_2/(c_3+x_1))+c_4$
41	$c_1x_1^3+c_2$	$c_1x_1^3+c_2$	91	$c_1+c_2\cos(x_1)/x_1$	$c_1+c_2\cos(x_1)/x_1$
42	$c_1\tanh(c_3x_1+c_4)+c_5$	$c_1\tanh(c_3x_1+c_4)+c_5$	92	$c_1+c_2\sinh(c_3x_1)/x_1$	$c_1+c_2\sinh(c_3x_1)/x_1$
43	$c_1(c_2+x_1)$ $\cosh(c_3x_1+c_4)+c_5$	$c_1(c_2+x_1)$ $\cosh(c_3x_1+c_4)+c_5$	93	$c_1+x_1\lvert c_2+x_1\rvert$	$c_1+x_1\lvert c_2+x_1\rvert$
44	c_1+ $c_2/(c_3+\log(c_4x_1+c_5))$	c_1+ $c_2/(c_3+\log(c_4x_1+c_5))$	94	$c_1x_1+c_2\sqrt{c_3+x_1}+c_4$	$c_1x_1+c_2\sqrt{c_3+x_1}+c_4$
45	$c_1\log(c_2x_1+c_3)+c_4$	$c_1\log(c_2x_1+c_3)+c_4$	95	$c_1+\cos(c_2\sinh(c_3x_1+c_4))$	$c_1+\cos(c_2\sinh(c_3x_1+c_4))$
46	$c_1x_1\sinh(c_2x_1)+c_3$	$c_1x_1^2\sinh(c_2x_1)+c_3$	96	$c_1(c_2+x_1)\tanh(c_3x_1+c_4)+c_5$	$c_1(c_2+x_1)\tanh(c_3x_1+c_4)+c_5$
47	$c_1+c_2\sin(c_3x_1)/x_1$	$c_1+c_2\sin(c_3x_1)/x_1$	97	$c_1+c_2/(c_3x_1+c_4)^3$	$c_1+c_2/(c_3x_1+c_4)^3$
48	$c_1+1/\cos(c_2+x_1)$	$c_1+1/\cos(c_2+x_1)$	98	$c_1x_1+c_2+\cos(c_3+\lvert c_4+x_1\rvert)$	$c_1x_1+c_2+\cos(c_3+\lvert c_4+x_1\rvert)$
49	$c_1+c_2/\sqrt{c_3x_1+c_4}$	$c_1+c_2/\sqrt{c_3x_1+c_4}$	99	$c_1\cos(x_1)^2+c_2$	$c_1\cos(x_1)^2+c_2$
50	$c_1\tan(c_2(c_3+x_1)^2$ $+c_4)+c_5$	$c_1\tan(c_2(c_3+x_1)^2$ $+c_4)+c_5$	100	$c_1+\exp(c_2\sqrt{c_3x_1+c_4})$	$c_1+\exp(c_2\sqrt{c_3x_1+c_4})$

CHAPTER FOUR

DEEP EVOLUTIONARY SYMBOLIC REGRESSION

In this chapter, we introduce a novel SR method we termed **S**ymbolic R**e**gression using **T**ransformers, genetic **A**lgorithms, and genetic **P**rogramming (SeTGAP). SeTGAP addresses multivariate SR problems by first extracting univariate symbolic skeletons using a Multi-Set Transformer, then merging them into a single multivariate structure using evolutionary techniques, and finally fitting their coefficients to approximate the true underlying equation of the observed data. We begin with an overview of related work, outline the overall workflow of SeTGAP, and present experimental results comparing its performance with other SR methods.

4.1 Background

Symbolic regression involves the process of acquiring a model representing data in the form of a mathematical expression. However, the general SR problem remains unsolved and becomes increasingly complex as the number of observations, operators, and variables involved increases [176]. Moreover, SR has been proven to be NP-hard [183], further emphasizing the computational challenges associated with solving it. As such, brute-force approaches become infeasible, even for medium-sized datasets.

SR is commonly tackled using GP-based methods [95]. They evolve a population of tree-like individuals using operations like selection, crossover, and mutation to improve their fitness over multiple generations. Each individual, or program, represents a symbolic expression that maps the inputs to the output, and its fitness function determines how well it fits the data set being modeled. Variations of this approach attempt to design improved operators and fitness functions to reduce the complexity of the search [67, 111, 139, 161].

Two significant challenges of GP for SR are code growth, also known as "code bloat", and the huge search space [4]. Bloat refers to the uncontrollable increase in the average tree size during evolution without substantial improvements in fitness. This is problematic as it not only leads to a computationally expensive evolution of large programs but also hinders the generalization ability of the solutions [41, 150]. Note, however, that the primary goal of this dissertation is not to achieve perfect generalization, but to interpret the functional form learned by an opaque model. Improved generalization may follow naturally from identifying a simpler mathematical model, which is inherently less prone to overfitting. On the other hand, the huge search space in GP is attributed to the variability in program lengths permitted during the evolutionary process. Consequently, this allows for the generation of multiple solution trees representing mathematically equivalent functions [44]. In other words, there exist many redundant solutions with the same phenotype but different genotypes. However, GPs tend to generate a greater number of large solution trees compared to smaller ones, according to the nature of program search space theory [99]. The high complexity of large solutions makes them hard to interpret and entails poor generalization performance.

In most GP-based symbolic regression methods, the fitness of a program is determined by its overall output and not by the intermediate outputs of its subexpressions. This implies that a program's subexpressions are only optimized indirectly. Arnaldo *et al.* [8] pointed out that the indirect subexpression optimization approach may result in suboptimal fitness optimization because it does not focus on evolving and learning suitable building blocks. Therefore, they proposed to optimize the generated programs before the selection step by optimizing the fitness contributions of its subexpressions. Specifically, the subexpressions of a program are identified and then combined using multiple linear regression (MLR) considering the target variable of the model as the response variable. In other words, MLR is used to place optimal coefficients in front of each subexpression before combining them and evaluating the program's output, which is then used to calculate its fitness value.

GP has also been combined with deep learning techniques for SR. For example, Mundhenk *et al.* [131] proposed a neural-guided GP that uses a recurrent neural network (RNN) to generate the initial population of a GP component. After the GP is executed, its top samples are used to retrain the RNN, which gradually learns to generate better initial populations. It is also possible that RNNs may be used to solve the SR problem directly. For instance, Petersen *et al.* [149] presented a gradient-based approach that utilizes RNNs in a reinforcement learning framework. Here, an RNN emits a distribution of mathematical expressions, which are then sampled and evaluated based on their fitness to the dataset. The fitness serves as the reward signal to train the RNN using a risk-seeking policy gradient algorithm, progressively adjusting the likelihood of expressions in accordance with their rewards to prioritize better-performing expressions. In addition, conventional feedforward NNs can be used to discover some properties of the data. Udrescu and Tegmark [176] proposed a recursive algorithm called AI Feynmann that uses NNs to identify properties such as symmetry, separability, and compositionality. This method uses other techniques such as dimensional analysis, polynomial fitting, and brute force algorithms that exploit the properties discovered by the NNs to define sub-problems that are easier to solve.

Several methods have proposed training NNs and gradually pruning irrelevant parts of them until a simple equation can be distilled from the network weights [113, 159, 175, 189] Martius and Lampert [113] proposed an NN architecture called EQL (which stands for "equation learning") that represents multiple symbolic expressions within its architecture. The training objective is to reduce the prediction error and gradually prune irrelevant parts of the network using regularization techniques until a simple equation can be extracted from the network. One of its limitations is that singularities in some operators or their gradients (e.g. division and logarithm) lead to unstable optimization. Sahoo *et al.* [159] built on the work of Martius and Lampert [113] and proposed the use of regularized operators that required additional parameters that avoid large gradients. These parameters are continuously reduced using a hard-coded schedule. Conversely, Werner *et al.* [189] introduced a learnable parameter that replaced the need to define a problem-dependent schedule. In addition, they proposed a modified version of the EQL architecture, called iEQL (which stands for "informed equation learning"), that uses skip connections and allows incorporating expert knowledge to better guide the search. Domain knowledge is incorporated by assigning complexity factors to each operator. Hence, operators with less preference are given higher complexity factors. The complexity of a candidate equation generated by iEQL is subsequently computed as a weighted sum, where the weights correspond to the complexity factors assigned to each operator. This evaluation takes into account both the number of operators in the equation and their associated complexity factors. Thus, equations with lower complexity are prioritized in the selection process. Experimental results on two real-world applications indicated that iEQL did not outperform the results obtained by GP-based techniques, although the equations it learned were notably more interpretable.

One of the main limitations of most SR approaches is that they do not leverage past experiences and, as such, each problem is learned from scratch. The inability to incorporate insights from different equations or domains restricts their ability to adapt and learn from diverse sources of information, hindering their capacity for improvement over time. Transformer-based methods have been proposed recently as an alternative. Biggio *et al.* [15] introduced the use of pre-trained transformer models for symbolic regression. A large dataset of multivariate equations is generated to pre-train a transformer neural network whose architecture is based on a set transformer [102]. This transformer acts as a general-purpose model that predicts the symbolic skeleton from a corresponding set of input–output pairs. The constants of the generated skeleton are then fitted using a non-convex optimizer, such as the BFGS algorithm [51]. Empirical results suggested that the obtained models improve over time with more data and compute time.

Additionally, Kamienny *et al.* [82] pointed out that the loss function minimized by the BFGS algorithm can be highly non-nonconvex, and thus the correct constants of the skeletons are not guaranteed to be found. Therefore, they avoided performing skeleton prediction as an intermediary step and proposed a transformer neural network that estimates the full mathematical expression directly. As an optional step, the constants learned by the transformer can be refined by feeding them to a non-convex optimizer. Experimental results have been shown to perform better than those obtained by previous NN-based methods and reduced the accuracy gap with respect to those obtained by state-of-the-art population-based methods. In addition, Bertschinger *et al.* [13] proposed an SR neuro-evolution approach that trains a population of transformer models using two objective functions: prediction error and symbolic loss. Since previous methods have shown scalability issues when dealing with high-dimensional equations with many variables, Chu *et al.* [30] proposed a method that decomposes multi-variable symbolic regression into a sequence of single-variable SR problems, combined in a bottom-up manner. The four-step process involves learning a data generator using NNs from observed data, generating variable-specific samples with controlled input variables, applying single-variable symbolic regression to estimate mathematical expressions, and iteratively adding variables until completion.

4.2 Proposed Method

We consider a system with response $y \in \mathbb{R}$ and t explanatory variables $\mathbf{x} = \{x_1, \ldots, x_t\}$ ($\mathbf{x} \in \mathbb{R}^t$). We assume that its underlying function $f(\mathbf{x}) = f(x_1, \ldots, x_t)$ can be constructed using a finite number of unary (e.g., `sin` and `log`) and binary (e.g., $+, -, \times, \div$) operations. The response is expressed as $y = f(\mathbf{x}) + \varepsilon_a$, where ε_a is a random variable representing the error term due to irreducible aleatoric uncertainty [75, 136]. Below, we define the SR problem formally and describe our SeTGAP methodology for solving it.

4.2.1 Problem Definition

Let us define our SR problem formally:

Definition 3. *Given a dataset* $(\mathbf{X}, \mathbf{y})$*, with inputs* $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_R}\} \subseteq \mathbb{R}^t$ *and corresponding target responses* $\mathbf{y} = \{y_1, y_2, \ldots, y_{N_R}\} \subseteq \mathbb{R}$*, the symbolic regression problem seeks to discover a function* $\tilde{f} : \mathbb{R}^t \to \mathbb{R}$ *that approximates the unknown underlying function* $f(\mathbf{x})$*. The function* $\tilde{f}$ *is represented as a composition of a finite number of unary and binary operators, such that* $\tilde{f}(\mathbf{x}) \approx f(\mathbf{x})$ *while capturing its functional structure and behavior.*

Thus, $\tilde{f}(\mathbf{x})$ can be expressed as $\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + \varepsilon_a + \varepsilon_e$, where ε_e is the error due to epistemic uncertainty, which is attributable to a lack of knowledge about f and can be reduced by acquiring additional information and improving the predictive model. In this context, solving the SR problem implies minimizing ε_e by selecting a suitable representation for $\tilde{f}(\mathbf{x})$ that aligns with function $f(\mathbf{x})$, and identifying an optimal set of parameters $\boldsymbol{\theta}_{\tilde{f}}$ for such representation. While accurate uncertainty quantification at a given location $\mathbf{x}$ is an important aspect, it is not the focus of this chapter. This issue is addressed in Chapter 5, where we introduce a prediction interval generation method and an adaptive sampling technique aimed at reducing epistemic uncertainty through strategic data collection.

4.2.2 Opaque Model Training

In this dissertation, we consider the case where the underlying function f is approximated by an opaque model, such as a neural network, trained on observed data. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_R}\}$ be a data set with N_R samples, where each sample is denoted as $\mathbf{x}_j = \{x_{j,1}, \ldots, x_{j,t}\}$, and $\mathbf{y} = \{y_1, \ldots, y_{N_R}\}$ is the set of corresponding target observations. A regression model, whose function is denoted as $\hat{f}(\cdot; \boldsymbol{\theta}_{NN})$ (where $\boldsymbol{\theta}_{NN}$ represents the weights of the network), is constructed to capture the association between $\mathbf{X}$ and $\mathbf{y}$. A target estimate for an input $\mathbf{x}_j$ is computed as $\hat{y} = \hat{f}(\mathbf{x}, \theta_{NN})$ or, simply, $\hat{y} = \hat{f}(\mathbf{x})$.

Here, we consider the simple case in which the parameters $\boldsymbol{\theta}_{NN}$ are obtained by minimizing the mean squared error of the predictions; that is, $\boldsymbol{\theta}^*_{NN} = \underset{\boldsymbol{\theta}_{NN}}{\text{argmin}} \frac{1}{N_R}\sum_{j=1}^{N_R}(\hat{y}_j - y_j)^2$. While alternative loss functions may enhance predictive performance, our focus is not on improving the opaque model itself, but on extracting interpretations from a model that has already been trained. We use neural networks to generate the function $\hat{f}$ due to their ease of training and high accuracy; however, other opaque regression models, such as random forests or gradient boosting machines, could also be applied.

4.2.3 Univariate Symbolic Skeleton Prediction

We deviate from SR approaches that prioritize the minimization of the prediction error of the learned functions. We argue that a correctly identified functional form $\tilde{f}$ inherently leads to a low estimation error and emphasizes interpretability and faithfulness to the underlying system's governing principles. Existing works on Symbolic Skeleton Prediction (SSP) [13, 15, 149, 179] attempt to address this issue by generating multivariate symbolic skeletons that describe the behavior of f.

Given a mathematical expression, its *symbolic skeleton* is an expression that replaces the numerical constants with placeholders. For example, if $f(\mathbf{x}) = 5\log x_1\,(\sin(x_2^2)+1) - 4$, its skeleton is expressed as $\mathbf{e}(\mathbf{x}) = \kappa(f(\mathbf{x})) = c_1 \log x_1\ (\sin(x_2^2) + c_2) + c_3$, where $\kappa(\cdot)$ represents the skeleton function and c_i are placeholders. In this work, we show that SSP methods struggle to identify the correct functional form of all variables in a system. Thus, we introduce a method for generating univariate skeletons for each variable in a multivariate system by framing the task as a sequence of multi-set symbolic prediction problems. While the MSSP was addressed in Chapter 3, here we leverage it to decompose the multivariate system into variable-specific representations. Each univariate skeleton captures how the corresponding variable contributes to or interacts with the system's overall response. Following the previous example, its univariate skeletons are: $\mathbf{e}(x_1) = \kappa(f(\mathbf{x}); x_1) = c_4(\log x_1) + c_5$, and $\mathbf{e}(x_2) =$

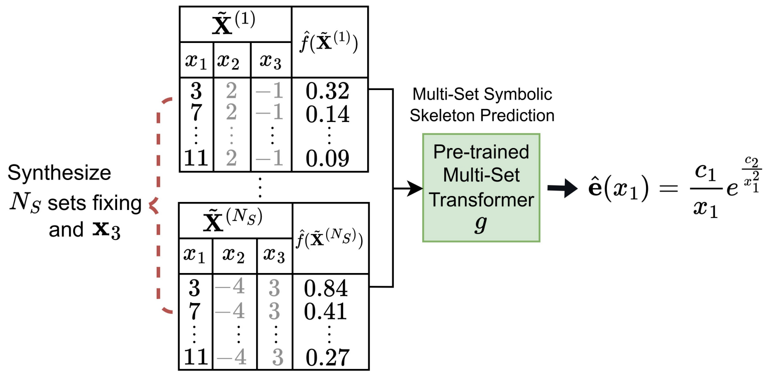


Figure 4.1: Multi-set symbolic skeleton prediction example. The analyzed variable is x_1. The remaining variables, x_2 and x_3 are held constant in each input set.

$\kappa(f(\mathbf{x}); x_2) = c_6 \sin(x_2^2) + c_7$. Here, the skeleton function $\kappa(\cdot; x_v)$ considers the remaining variables $\mathbf{x} \setminus x_v$ irrelevant when describing the functional form between x_v ($v \in [1, \dots, t]$) and the system's response. In this case, the placeholders c_i may represent numeric constants or functions of other variables.

We analyze each variable x_v separately. To do this, N_s artificial sets of input–response pairs $\{(\tilde{\mathbf{X}}^{(1)}, \tilde{\mathbf{y}}^{(1)}), \dots, (\tilde{\mathbf{X}}^{(N_s)}, \tilde{\mathbf{y}}^{(N_s)})\}$ are generated. To isolate the influence of variable x_v, the set $\tilde{\mathbf{X}}^{(s)}$ ($s \in [1, \dots, N_s]$) is constructed such that the variable x_v (i.e., the v-th column $\tilde{\mathbf{X}}_v^{(s)}$) is allowed to vary while the other variables are fixed to random values. Specifically, the s-th synthetic set is denoted as $\tilde{\mathbf{X}}^{(s)} = \{\tilde{\mathbf{x}}_1^{(s)}, \dots, \tilde{\mathbf{x}}_n^{(s)}\}$ and it consists of n samples. The value of the v-th dimension of the j-th sample is obtained by sampling from the distribution $\mathcal{U}(x_v^{min}, x_v^{max})$ whose lower and upper bounds, x_v^{min} and x_v^{max}, respectively, are calculated from the observed data. The values assigned to the remaining dimensions are sampled independently using similar uniform distributions; however, the same value is shared across all samples (i.e., $\tilde{\mathbf{x}}_{1,k}^{(s)} = \tilde{\mathbf{x}}_{2,k}^{(s)} = \cdots = \tilde{\mathbf{x}}_{n,k}^{(s)}$, $\forall k \in [1, t]$ and $k \neq i$). Thus, $\tilde{\mathbf{y}}^{(s)}$ denotes the estimated response of inputs $\tilde{\mathbf{X}}^{(s)}$ using the trained model $\hat{f}$ as $\tilde{\mathbf{y}}^{(s)} = \hat{f}(\tilde{\mathbf{X}}^{(s)})$.

Fixing the columns corresponding to the variables $\mathbf{x} \setminus x_v$ ensures that $\tilde{\mathbf{y}}^{(s)}$ depends only on the column $\tilde{\mathbf{X}}_v^{(s)}$. However, this process may project the function into a space where its functional form becomes less recognizable. To address this, the influence of x_v on the system's response is analyzed using N_s sets of input–response pairs, each reflecting a different effect of the variables $\mathbf{x} \setminus \{x_v\}$. As such, each set $(\tilde{\mathbf{X}}^{(s)}, \tilde{\mathbf{y}}^{(s)})$ is generated independently by fixing $\mathbf{x} \setminus \{x_v\}$ at different values. The relationship between each $\tilde{\mathbf{X}}_v^{(s)}$ and $\tilde{\mathbf{y}}^{(s)}$ can be described by a univariate function $f_v^{(s)}$. Note that functions $f_v^{(1)}, \ldots, f_v^{(N_S)}$ have been derived from the same function $f(\mathbf{x})$ and should share the same symbolic skeleton $\mathbf{e}(x_v)$, which is unknown. Note that predicting a skeleton $\hat{\mathbf{e}}(x_v)$ that describes the shared function form of input $\tilde{\mathbf{D}}_v = \{\tilde{\mathbf{D}}_v^{(1)}, \ldots, \tilde{\mathbf{D}}_v^{(N_s)}\}$ (i.e., $\hat{\mathbf{e}}(x_v) \approx \mathbf{e}(x_v)$), s.t. $\tilde{\mathbf{D}}_v^{(s)} = (\tilde{\mathbf{X}}_v^{(s)}, \tilde{\mathbf{y}}^{(s)})$, illustrated in Fig. 4.1, represents a multi-set symbolic skeleton prediction problem. The MSSP problem was tackled in Chapter 3 by designing a Multi-Set Transformer, whose function and parameters are denoted by $g(\cdot)$ and Θ, respectively. It was trained on a dataset of synthetically generated MSSP problems to produce accurate estimated skeletons. Given an input collection $\tilde{\mathbf{D}}_v$, the estimated skeleton obtained for variable x_v is computed as $\tilde{\mathbf{e}}(x_v) = g(\tilde{\mathbf{D}}_v, \Theta)$.

Since our method derives univariate skeletons based on multiple sets of input–estimated response pairs, it is important to use a prediction model that learns a function $\hat{f}$ that is as close as possible to f so that it accurately estimates how the real system would respond to the synthetic inputs in $\tilde{\mathbf{X}}^{(s)}$. As a consequence, our analysis can be regarded as an interpretability method that generates univariate symbolic skeletons as interpretations of the function approximated by the regression model.

Furthermore, we generate up to n_{cand} distinct candidate skeletons rather than a single one, as described in Algorithm 4.1. We generate a $\tilde{\mathbf{D}}_v$ collection and feed it into g to obtain n_B skeletons using a diverse beam search (DBS) strategy [182] to promote variability among the n_B generated skeletons. Beam search is a heuristic search algorithm that explores a fixed number of the most likely sequences at each decoding step. DBS extends this by partitioning the beams into groups and promoting diversity across them through dissimilarity penalties. This process is repeated n_{cand} times, yielding a total of $n_{\text{cand}} n_B$ skeletons. Each repetition generates a new $\tilde{\mathbf{D}}_v$ collection using different combinations of fixed values of $\mathbf{x} \setminus \{x_v\}$, increasing input diversity and potentially leading to different skeletons. Then, we discard identical or mathematically equivalent skeletons. For example, $c_1 \sin(c_2 x_v + c_3)$ is equivalent to $c_4 \cos(c_5 x_v + c_6)$ if $c_1 = c_4$, $c_2 = c_5$, and $c_3 = c_6 - \pi/2$.

If the generated skeleton list, $\texttt{genSks}_v$, exceeds n_{cand} elements, we evaluate their performance and select the top n_{cand} candidates. To do this, an additional collection $\tilde{\mathbf{D}}_v$ is generated. Since a skeleton expression for variable $\mathbf{x}_v$ is expected to describe the functional form of all sets in $\tilde{\mathbf{D}}_v$, we choose a random set $\tilde{\mathbf{D}}_v^{(\text{test})} = (\tilde{\mathbf{X}}_v^{(\text{test})}, \tilde{\mathbf{y}}^{(\text{test})}) \subset \tilde{\mathbf{D}}_v$ and use it to fit the coefficients of each skeleton $\hat{\mathbf{e}}_k(x_v)$, where $k \in \{1, \ldots, |\texttt{genSks}_v|\}$. The coefficient fitting problem is described as follows. Let $f_{est}(x_v) = \texttt{setConstants}(\hat{\mathbf{e}}_k(x_v), \mathbf{c})$ be the function obtained when replacing the n_c coefficients in $\hat{\mathbf{e}}_k(x_v)$ with the numerical values in a given set $\mathbf{c} \in \mathbb{R}^{n_c}$. Then, the objective is to find an optimal set $\mathbf{c}^*$ that maximizes the Pearson correlation between $f_{est}(\tilde{\mathbf{X}}_v^{(\text{test})})$ and the estimated response $\tilde{\mathbf{y}}^{(s)}$:

$$\mathbf{c}^* = \underset{c}{\operatorname{argmax}} \ \operatorname{corr}(f_{est}(\tilde{\mathbf{X}}_v^{(\text{test})}), \tilde{\mathbf{y}}^{(\text{test})}),$$

Note that in generating $\tilde{\mathbf{D}}_v$, we fixed the values of $\mathbf{x} \setminus x_v$, so we assume that all coefficients in $\mathbf{c}$ are numerical values. These learned coefficients are then discarded, as they serve only to evaluate the fit of a univariate skeleton to the data. This problem is solved using a genetic algorithm (GA) [71]. The individuals of our GA are arrays of n_c elements that represent potential $\mathbf{c}$ sets. Then the optimization process is carried out by function $\texttt{fitCoefficients}(\hat{\mathbf{e}}_k(x_v), \tilde{\mathbf{D}}_v^{(\text{test})})$. This optimization process is repeated for all system variables to derive their univariate skeleton expressions with respect to the system's response.

4.2.4 Merging Univariate Symbolic Skeletons

Having identified a set of univariate skeleton candidates for each variable, the next step is to merge the identified univariate skeleton candidates to produce multivariate skeleton

Algorithm 4.1 Univariate Skeleton Generation

Input: Index v of current variable; samples per input set n; number of input sets N_S; opaque model $\hat{f}$; Multi-Set Transformer g; number of skeleton candidates n_{cand}; beam size n_B

Output: Generated list of candidate skeletons for the v-th variable $\texttt{genSks}_v$; corresponding correlation values $\texttt{corrVals}_v$

function GENERATEUNIVSKS($v, n, N_s, \hat{f}, g, n_{\text{cand}}, n_B$)
  $\text{genSks}_v \leftarrow [\,]$
  for each $i \in (1, n_{\text{cand}})$ **do**
    $\tilde{\mathbf{D}}_v \leftarrow \texttt{generateCollection}(v, n, N_s, \hat{f})$
    $\text{genSks}_v.\texttt{append}(g(\tilde{\mathbf{D}}_v, \Theta; n_B))$
  $\text{genSks}_v \leftarrow \text{removeDuplicates}(\text{genSks}_v)$ ▷ $\text{genSks}_v = \{\hat{\mathbf{e}}_1(x_v), \ldots, \hat{\mathbf{e}}_{|\text{genSks}_v|}(x_v)\}$
  $\tilde{\mathbf{D}}_v^{(\text{test})} \leftarrow \texttt{generateCollection}(v, n, N_s, \hat{f})$
  $\text{corrVals}_v \leftarrow \text{zeros}(|\text{genSks}_v|)$
  for each $k \in (1, n_{\text{cand}})$ **do**
    $\text{corrVals}_v[k] \leftarrow \texttt{fitCoefficients}(\hat{\mathbf{e}}_k(x_v), \tilde{\mathbf{D}}_v^{(\text{test})})$
  $\text{genSks}_v \leftarrow \texttt{sortSkeletons}(\text{genSks}_v, \text{corrVals}_v)$
  if $|\text{genSks}_v| > n_{\text{cand}}$ **then**
    $\text{genSks}_v, \text{corrVals}_v \leftarrow \text{genSks}_v[1 : n_{\text{cand}}], \text{corrVals}_v[1 : n_{\text{cand}}]$
  return $\text{genSks}_v, \text{corrVals}_v$

expressions. This process is carried out incrementally in a cascade fashion until a final expression incorporating all variables is formed.

4.2.4.1 Merging Skeleton Expressions Given two skeleton expressions, multiple mathematically valid ways to combine them may exist. Here, we explore how to generate such combinations. We start with the following proposition:

Proposition 1. *Let $f(\mathbf{x})$ be a scalar-valued function defined using a finite composition of real-valued unary and binary operators applied to scalar sub-expressions. Then, $f(\mathbf{x})$ can always be expressed as: $f(\mathbf{x}) = c_0 + \sum_i c_{i,j} \prod_j \nu_{i,j}(T_{i,j}(\mathbf{x}))$, where $c_0, c_{i,j} \in \mathbb{R}$, $\nu_{i,j}$ is a unary operator (including the identity function $I(f(\mathbf{x})) = f(\mathbf{x})$), and $T_{i,j}(\mathbf{x})$ is a sub-expression. Moreover, each $T_{i,j}(\mathbf{x})$ can be recursively decomposed in the same structure as f, continuing until the decomposition reduces to variables or constants.*

Proof. We prove the proposition by structural induction on the composition of operations that define $f(\mathbf{x})$. We begin with the base cases. If $f(\mathbf{x})$ is a constant, the required form is satisfied trivially by setting $c_0 = c$ with no additional terms. If $f(\mathbf{x}) = x_i$ is a single-variable function, the structure is preserved by setting $c_0 = 0$, $c_{1,1} = 1$, and $T_{1,1}(\mathbf{x}) = x_i$.

For our inductive hypothesis, assume the decomposition holds for functions $h(\mathbf{x})$ and $u(\mathbf{x})$, composed of unary or binary operations. That is, each can be written in the form $h(\mathbf{x}) = c_0' + \sum_i c_{i,j}' \prod_j \nu_{i,j}'(T_{i,j}'(\mathbf{x}))$ and $u(\mathbf{x}) = c_0'' + \sum_i c_{i,j}'' \prod_j \nu_{i,j}''(T_{i,j}''(\mathbf{x}))$, with terms $T_{i,j}'$ and $T_{i,j}''$ themselves recursively decomposable in the same way.

To complete the inductive step, we consider the result of applying unary or binary operations to such functions. For a unary operation $f(\mathbf{x}) = \nu(h(\mathbf{x}))$, the expression satisfies the required structure by treating the composition as a single term ($i = 1$, $j = 1$) where $T_{1,1}(\mathbf{x}) = h(\mathbf{x})$, $\nu_{1,1} = \nu$, $c_0 = 0$, and $c_{1,1} = 1$. Since $h(\mathbf{x})$ itself satisfies the recursive form by the inductive hypothesis, $f(\mathbf{x})$ does as well.

Now consider a binary operation $f(\mathbf{x}) = h(\mathbf{x}) \circ u(\mathbf{x})$, where $\circ$ is a binary operator. For $\circ = +$, substituting the expressions and grouping constants and summation terms yields:

$$f(\mathbf{x}) = (c_0' + c_0'') + \sum_i c_{i,j}' \prod_j \nu_{i,j}'(T_{i,j}'(\mathbf{x})) + \sum_i c_{i,j}'' \prod_j \nu_{i,j}''(T_{i,j}''(\mathbf{x})).$$

This matches the desired structure $f(\mathbf{x}) = c_0 + \sum_i c_i \prod_j \nu_{i,j}(T_{i,j}(\mathbf{x}))$, where $c_0 = c_0' + c_0''$, and the summation terms come directly from the sub-expressions of $h(\mathbf{x})$ and $u(\mathbf{x})$.

For $\circ = \cdot$, the product expansion gives:

$$\begin{aligned} f(\mathbf{x}) = c_0'c_0'' &+ c_0' \sum_i c_{i,j}'' \prod_j \nu_{i,j}''(T_{i,j}''(\mathbf{x})) + c_0'' \sum_i c_{i,j}' \prod_j \nu_{i,j}'(T_{i,j}'(\mathbf{x})) \\ &+ \sum_{i,j} c_{i,j}' c_{i,j}'' \prod_j \nu_{i,j}'(T_{i,j}'(\mathbf{x})) \prod_j \nu_{i,j}''(T_{i,j}''(\mathbf{x})). \end{aligned}$$

Each term fits into the structure $f(\mathbf{x}) = c_0 + \sum_i c_i \prod_j \nu_{i,j}(T_{i,j}(\mathbf{x}))$: the constant term can be expressed as $c_0 = c_0' \cdot c_0''$; the second and third terms are sums over products of unary operator applications, consistent with the required form; and the fourth term comprises products of sub-expressions from $h(\mathbf{x})$ and $u(\mathbf{x})$, which can be grouped as new terms $\prod_j \nu_{i,j}(Ti, j(\mathbf{x}))$.

For other scalar-valued binary operations that are algebraically reducible, such as subtraction and division, we use the identities $h(\mathbf{x}) - u(\mathbf{x}) = h(\mathbf{x}) + (-u(\mathbf{x}))$ and $h(\mathbf{x})/u(\mathbf{x}) = h(\mathbf{x}) \cdot u(\mathbf{x})^{-1}$. Negation and inversion are unary operations, and since the unary case has been established, the result follows. Binary operations that involve non-scalar interactions (e.g., convolution or vector operations) are beyond the scope of this structural decomposition.

Thus, any function $f(\mathbf{x})$, defined by finite compositions of unary and binary operations, can always be expressed in the required form. Since the base case holds and the inductive step is proven, the proposition is established by structural induction. $\square$

Let $\mathbf{e}_1(\mathbf{x}_S)$ and $\mathbf{e}_2(x_q)$ be two candidate skeletons to be merged. Here, S is an index set, $S \subset \{1, \ldots, t\}$, specifying the variables that $\mathbf{e}_1$ depends on; i.e., $\mathbf{x}_S = \{x_r | r \in S\}$. In contrast, x_q is a variable distinct from those in $\mathbf{x}_S$ ($q \notin S$). Following from Proposition 1, the skeletons can be expressed using the same mathematical structure as $\mathbf{e}_1(\mathbf{x}_S) = c_0 + \sum_i c_{i,j} \prod_j \nu_{i,j}(T_{i,j}(\mathbf{x}_S))$ and $\mathbf{e}_2(x_q) = c_0' + \sum_i c_{i',j'}' \prod_{j'} \nu_{i',j'}'(T_{i',j'}'(x_q))$. Below, we explain how the subtrees of both expressions can be merged recursively.

The key idea is that a constant placeholder in $\mathbf{e}_1(\mathbf{x}_S)$ may be replaced by a subtree of $\mathbf{e}_2(x_q)$, and vice versa. Given two skeletons, $\mathbf{e}_1(\mathbf{x}_S) = c_1 T_1(\mathbf{x}_S)$ and $\mathbf{e}_2(x_q) = c_2 T_2(x_q)$, a straightforward way to merge them is by recognizing that part of c_1 may be a function of x_q, while part of c_2 may be a function of $\mathbf{x}_S$. Thus, their combination, denoted by the operation $\bowtie$, is given by $\mathbf{e}_3(\mathbf{x}_S \cup x_q) = \mathbf{e}_1(\mathbf{x}_S) \bowtie \mathbf{e}_2(x_q) = c_3(c_4 + T_1(\mathbf{x}_S))(c_5 + T_2(x_q))$. Expanding this expression conforms to the expected functional form of a subtree stated in Proposition 1. Note that the skeletons with respect to the corresponding initial variable sets remain unchanged; that is, $\kappa(\mathbf{e}_1(\mathbf{x}_S); \mathbf{x}_S) = \kappa(\mathbf{e}_3(\mathbf{x}_S \cup x_q); \mathbf{x}_S) = c_1 T_1(\mathbf{x}_S)$ and $\kappa(\mathbf{e}_2(x_q); x_q) = \kappa(\mathbf{e}_3(\mathbf{x}_S \cup x_q); x_q) = c_q T_2(x_q)$. Applying the same principle to a more general case in which $\mathbf{e}_1(\mathbf{x}_S) = c_1 + c_2 \prod_j T_{1,j}(\mathbf{x}_S)$ and $\mathbf{e}_2(x_q) = c_3 + c_4 \prod_{j'} T_{2,j'}(x_q)$, we obtain $\mathbf{e}_3(\mathbf{x}_S \cup x_q) = \mathbf{e}_1(\mathbf{x}_S) \bowtie \mathbf{e}_2(x_q) = c_5 + c_6 \prod_j (c_{7,j} + T_{1,j}(\mathbf{x}_S)) \prod_{j'} (c_{8,j'} + T_{2,j'}(\mathbf{x}_S))$. Nevertheless, more combinations are possible if the candidate skeletons share functions with compatible mathematical structures. For example, if $\mathbf{e}_1(x_1, x_2) = c_1 \sin(c_2\, x_1 x_2 + c_3)$ and $\mathbf{e}_2(x_3) = c_4 \sin(c_5\, x_3 + c_6)$, we could obtain $\mathbf{e}_3(x_1, x_2, x_3) = \mathbf{e}_1(x_1, x_2) \bowtie \mathbf{e}_2(x_3) = c_7(c_8 + \sin(c_9\, x_1 x_2 + c_{11}))(c_{11} + \sin(c_{12}\, x_3 + c_{13}))$, as shown before, but also $\mathbf{e}_3(x_1, x_2, x_3) = c_{13} \sin(c_{14}\, x_1 x_2 + c_{15} x_3)$ and $\mathbf{e}_3(x_1, x_2, x_3) = c_{16} \sin(c_{17}\, x_1 x_2 x_3 + c_{18})$, which yield to the same skeletons with respect to the initial variable sets.

Algorithm 4.2 shows our recursive merging procedure, `merge`$(\mathrm{ex}_1, \mathrm{ex}_2)$, which takes as inputs two skeleton expressions $\texttt{ex}_1$ and $\texttt{ex}_2$. The lists of subtrees of both expressions are retrieved by function `getSubtreesLists`$(\texttt{ex}_1, \texttt{ex}_2)$, returning the one with fewer subtrees first. We shuffle both lists to introduce variability in the process. If `exShort` lacks a constant term, we add one to conform to the structure in Proposition 1 and ensure that it is the last element. If both expressions represent sums, the algorithm iterates over the elements of `exShort`, attempting to merge each with compatible subtrees from `exLong`; i.e., subtrees that share the same unary or binary operator. The compatibility check is performed by `findComp`, and a subset of matching subtrees is randomly selected for merging, as depicted in Fig 4.2. If only one compatible subtree is selected, it is merged recursively with `exShort`$[i]$. However, if multiple subtrees are selected, the only way to maintain the original skeleton structure is to treat their sum as a single, indivisible entity, effectively a constant term with respect to x_q, which is then multiplied by `exShort`$[i]$ (Line 16). Then, the constant term that appears at the end of `exShort` absorbs any remaining terms from `exLong`.

If neither expression is a sum or a product but they share the same function, the function remains unchanged, and their inner arguments are merged recursively. For products, if `exShort` consists only of symbols, the expressions are combined as explained previously (Line 23). Otherwise, it iterates through the terms, merging them based on compatibility,

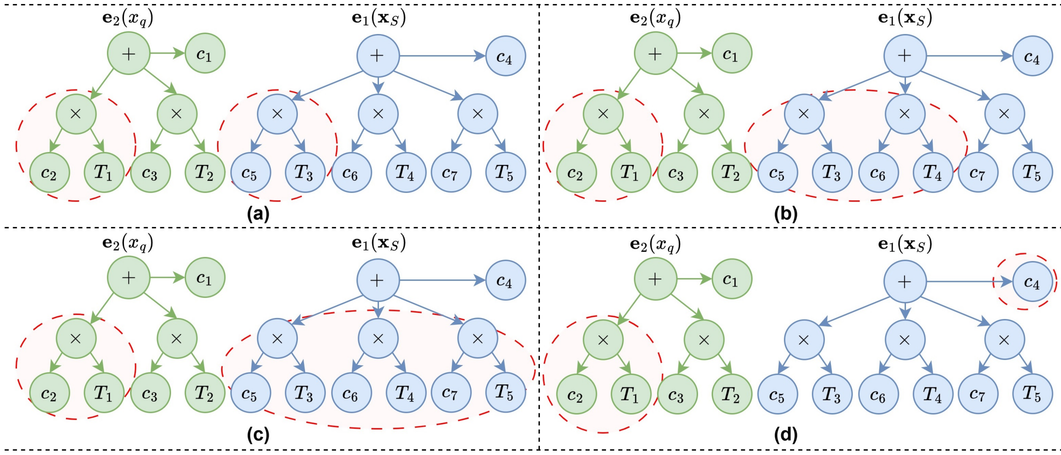


Figure 4.2: Example of a selected subtree of $\mathbf{e}_2(x_q)$ within a sum merging with one or more subtrees from $\mathbf{e}_1(\mathbf{x}_S)$, illustrating four out of the nine possible cases.

as was done for the sum case. However, unlike in the sum case, each subtree `exShort`$[i]$ can merge with only one subtree from `exLong` at a time, with a 0.5 probability of merging to introduce variability. Finally, the function returns the merged expression, which integrates elements from both inputs while preserving their initial structures.

4.2.4.2 Selecting Combined Skeleton Expressions Algorithm 4.2 generates a single combination of skeletons. Here, we extend this process to construct a population of candidate skeletons and select the top-performing one. Thus, we employ an evolutionary strategy to assess the performance of each skeleton combination.

The population of skeletons, `candSks`, is constructed by repeatedly applying Algorithm 4.2. Since the merging process involves randomized subtree selection and probabilistic merging, each application of the algorithm may yield a different skeleton combination. The process continues until a population size $P_{\max}$ is reached or a patience criterion is met. As such, if no new valid skeleton is found after a certain number of attempts, the generation process halts, assuming that all viable combinations have been explored.

We aim to identify the most promising combination in `candSks` by evaluating how well

Algorithm 4.2 Recursive Skeleton Merging

Input: Skeleton expressions to be merged $\texttt{ex}_1$ and $\texttt{ex}_2$
Output: Random merged skeleton expression `mergedEx`

```
function MERGE(ex_1, ex_2)
    exShort, exLong ← getSubtreesLists(ex_1, ex_2)                ▷ Return lists of subtrees in sum
    exShort.shuffle(), exLong.shuffle()
    if isSum(ex_1) and isSum(ex_2) then
        for each i ∈ (1, |exShort|) do
            if i = |exShort| and |exLong| > 0 then
                exShort[i] ← exShort[i] + ∑_j exLong[j]
            else
                args ← findComp(exLong, exShort[i])        ▷ Find exLong args compatible w/exShort[i]
                selectedArgs ← sample(args, randInt(0, |args|))
                if |selectedArgs| = 0 then continue
                exLong.remove(selectedArgs)
                if |selectedArgs| = 1 then
                    exShort[i] ← merge(exShort[i], selectedArgs[0])
                else
                    exShort[i] ← exShort[i] ∑_j selectedArgs[j]
        mergedEx ← ∑_i exShort[i]
    else
        if !isMult(ex_1) then                              ▷ If compatible, merge inner arguments
            mergedEx ← ex_1.func(merge(ex_1.args, ex_2.args))
        else
            if isAllSymbols(exShort) then
                mergedEx ← c_1 ∏_i (c_{2,i} + exShort[i]) ∏_j (c_{3,j} + exLong[j])
            else
                mergedEx ← null
                for each i ∈ (1, |exShort|) do
                    if i = |exShort| and |exLong| > 0 then
                        mergedEx ← exShort[i] ∏_{i'=0}^{i-1} (c_{1,i'} + exShort[i']) ∏_j (c_{2,j} + exLong[j])
                    else
                        args ← findComp(exLong, exShort[i])
                        if |args| = 0 then continue
                        selectedArg ← choice(args)
                        if random(0, 1) < 0.5 then continue
                        exLong.remove(selectedArg)
                        exShort[i] ← merge(exShort[i], selectedArg)
                if mergedEx is null then
                    mergedEx ← ∏_i exShort[i]
    return mergedEx
```

each skeleton can fit the test data. Algorithm 4.3 provides a high-level overview of this process. We generate test data $\tilde{\mathbf{D}}^{(\text{test})} = (\tilde{\mathbf{X}}^{(\text{test})}, \tilde{\mathbf{y}}^{(\text{test})})$. Unlike in Section 4.2.3, here the columns in $S' = \{S \cup q\}$ (i.e., the indices of variables present in the combined skeletons)

Algorithm 4.3 Skeleton Combination with Genetic Programming

Input: Population of skeletons `candSks`; population size per skeleton `rep`; maximum number of generations `maxG`; test data $\tilde{\mathbf{D}}^{(\text{test})}$

Output: Selected skeleton and corresponding fitness value

function SELECTCOMBINATION$(\text{candSks}, \text{rep}, \text{maxG}, \tilde{\mathbf{D}}_{S'}^{(\text{test})})$
  Pop $\leftarrow$ [ ]
  for each $s \in (1, |\text{candSks}|)$ **do**
    exps $\leftarrow$ [ ]
    for $i = 1$ to rep **do** exps.`append`(`assignValues`(candSks[s]))
    Pop.`append`(exps)
  for each gen $\in (1, \text{maxG})$ **do**
    fitnesses, bestFitnesperSk $\leftarrow$ `evalCorr`(Pop, $\tilde{\mathbf{D}}_{S'}^{(\text{test})}$)
    Pop $\leftarrow$ `evolveSks`(Pop, fitnesses)
  return candSks[`argmax`(bestFitnesperSk)], `max`(bestFitnesperSk)

are allowed to vary, while all other variables are fixed to random values. The set $\tilde{\mathbf{D}}_{S'}^{(\text{test})} = (\tilde{\mathbf{X}}_{S'}^{(\text{test})}, \tilde{\mathbf{y}}^{(\text{test})})$ is then used for evaluation. Each skeleton in `candSks` is replicated `rep` times, with randomly assigned coefficient values, to form an initial pool of expressions `Pop`.

The population then undergoes an evolution process for `maxG` generations. In each iteration, the fitness of every expression in `Pop` is evaluated using `evalCorr`, which assesses the Pearson correlation value between $\tilde{\mathbf{y}}^{(\text{test})}$ and the output produced by the expression when evaluated on $\tilde{\mathbf{X}}^{(\text{test})}$. The evolutionary step `evolveSks` updates the population by selecting high-performing candidates and introducing variations through crossover and mutation. Importantly, while coefficient values are subject to modification, the structure of each skeleton remains unchanged. This is ensured by restricting crossover operations to expressions derived from the same skeleton. Finally, the skeleton that corresponds to the highest fitness across all evolved instances is selected as the best candidate.

This approach bears similarities to GP, with the distinction that the search for viable symbolic skeletons is decoupled from the evolutionary optimization process. In particular, rather than evolving the expressions dynamically, we first enumerate all admissible skeleton combinations via Algorithm 4.2 and then apply a constrained form of genetic programming, where the evolution process occurs with the structure of all mathematical expressions fixed to the precomputed combined skeletons.

4.2.4.3 Cascade Merging The construction of multivariate skeletons follows an incremental merging process in which univariate skeleton candidates are combined progressively. For each variable x_v Algorithm 4.1 generates up to n_{cand} candidate skeletons $\texttt{genSks}_v$ along with their corresponding correlation values $\texttt{corrVals}_v$. To determine the merging order, the variables are ranked according to the maximum correlation value obtained across their generated skeletons. Variables with higher correlation scores are prioritized in the merging order, as their generated skeletons exhibit stronger relationships with the data. This reduces the risk of propagating structural uncertainty from variables with less reliable skeletons.

Let S denote the indices of the variables whose skeletons have already been merged. Initially, S contains only the index of the variable that corresponds to the skeleton with the highest correlation value. At each iteration, a new variable x_q is selected according to the ranking, and its univariate skeletons are merged with those corresponding to S. Specifically, Algorithms 4.2 and 4.3 are applied to combine each skeleton generated for $\mathbf{x}_S$ with each skeleton generated for x_q, producing at most n_{cand}^2 skeleton combinations. Since Algorithm 4.3 returns both the selected merged skeleton and its fitness, we apply a greedy selection strategy, retaining only the n_{cand} highest-performing skeletons at each step. This process repeats until skeletons incorporating all t variables have been generated.

We argue that our cascade approach ensures a structured integration of variables into the final multivariate skeleton. Directly constructing a multivariate skeleton from all univariate candidates would prioritize minimizing overall prediction error, potentially obscuring individual contributions and leading to overfitting. Instead, incorporating one variable at a time allows for a more interpretable learning process, where each newly added variable's effect is evaluated in the context of the previously analyzed ones.

4.2.4.4 Underlying Function Estimation Here, we utilize the N_e multivariate skeletons $\hat{\mathbf{e}_1}(\mathbf{x}), \ldots, \hat{\mathbf{e}_{N_e}}(\mathbf{x})$ produced by our cascade merging procedure, where $N_e \leq n_{\text{cand}}$. The goal is to construct functions $\tilde{f}_i(\mathbf{x})$ that approximate the underlying function $f(\mathbf{x})$ based on the corresponding skeleton $\hat{\mathbf{e}}_i(\mathbf{x})$, $\forall i \in (1, \ldots, N_e)$. This constitutes a coefficient fitting problem similar to the one presented in Section 4.2.3. Unlike the previous sections, this optimization task minimizes the mean squared error (MSE) of the response calculated by evaluating $\tilde{f}_i(\mathbf{x})$ on the original dataset $(\mathbf{X}, \mathbf{y})$. This is possible because $\hat{\mathbf{e}}(\mathbf{x})$ contains all system variables and there is no need to generate a synthetic set of estimated points using $\hat{f}$.

The new coefficient fitting problem consists of finding an optimal set of coefficient values that minimizes the prediction MSE

$$\mathbf{c}^* = \arg\min_{c} \frac{1}{|\mathbf{X}|} \sum_{(\mathbf{x}_j, y_j) \in (\mathbf{X}, \mathbf{y})} \left(\texttt{setConstants}(\hat{\mathbf{e}}_i(\mathbf{x}_j), \mathbf{c}) - y_j\right)^2,$$

such that $\tilde{f}_i(\mathbf{x}) = \texttt{setConstants}(\hat{\mathbf{e}}_i(\mathbf{x}), \mathbf{c}^*)$. This optimization is carried out using the GA-based function $\texttt{fitCoefficients}_{\text{MSE}}(\hat{\mathbf{e}}_i(\mathbf{x}_j), (\mathbf{X}, \mathbf{y}))$ introduced in Section 4.2.3 but modified to minimize MSE between the observed response and that generated by the learned expression. Note that, while the current formulation focuses solely on minimizing the MSE, future work will explore regularized optimization strategies that also account for model complexity, encouraging parsimonious representations and improving generalization.

4.2.5 Skeleton Performance Evaluation

The performance of SR methods is usually assessed by the mean squared error achieved by the learned expressions on benchmark data sets called fitness cases. Nevertheless, this evaluation approach does not provide insights into how effectively the learned expressions align with the underlying functional form of the system. To the best of our knowledge, no previous work has addressed this evaluation problem. Recall that SeTGAP produces as intermediate outputs a series of univariate skeletons with non-numerical coefficients that describe the functional form between each variable and the system's response. In this section, we are interested in evaluating the suitability of the functional form of the estimated skeletons and comparing them to those generated by other SR techniques.

We present a method to test the similarity between the underlying skeleton corresponding to the variable x_v, represented as $\mathbf{e}(x_v) = \kappa(f(\mathbf{x}), x_v)$, and the estimated skeleton $\hat{\mathbf{e}}(x_v)$. We assign random numerical values to the coefficients of skeleton $\mathbf{e}(x_v)$ using the `sampleConstants` routine (see Algorithm 3.2) in order to obtain a function $f_{target}(x_v)$. Let $f_{est}(x_v) = \texttt{setConstants}(\hat{\mathbf{e}}(x_v), \mathbf{c})$ denote the function obtained when replacing the n_c coefficients in $\hat{\mathbf{e}}(x_v)$ with the numerical values in a given set $\mathbf{c} \in \mathbb{R}^{n_c}$. If the functional form of $\hat{\mathbf{e}}(x_v)$ is mathematically equivalent to that of $\mathbf{e}(x_v)$, then there exists a set of values $\mathbf{c}$ so that the difference between $f_{target}(x_v)$ and $f_{est}(x_v)$ is zero for all values of x_v. The problem of finding the optimal set $\mathbf{c}^*$ is expressed as:

$$\mathbf{c}^* = \frac{1}{|\mathbf{X}_v^{test}|} \underset{c}{\operatorname{argmin}} \sum_{x_v \in \mathbf{X}_v^{test}} |f_{target}(x_v) - f_{est}(x_v)|^2,$$

where $\mathbf{X}_v^{test}$ is a test set of N_{test} elements whose elements are drawn from a distribution $\mathcal{U}(2\,x_v^{min}, 2\,x_v^{max})$. Notice that the domain of $\mathbf{X}_v^{test}$ is larger than that used for training (i.e., $[x_v^{min}, x_v^{max}]$). The reasoning behind this is that if the estimated skeleton matches the underlying functional form of the system, it should do so regardless of the variable domain. Similar to Section 4.2.4.4, the optimization is carried out using the GA-based function $\texttt{fitCoefficients}_{\text{MSE}}(\hat{\mathbf{e}}(x_v), (\mathbf{X}_v^{test}, f_{est}(\mathbf{X}_v^{test})))$.

If $\hat{\mathbf{e}}(x_v)$ and $\mathbf{e}(x_v)$ are not equivalent, the error

$$r = \sum_{x_v \in \mathbf{X}_v^{test}} |f_{target}(x_v) - \texttt{setConstants}(\hat{\mathbf{e}}(x_v), \mathbf{c}^*)|^2$$

is greater than 0. We use r as a performance metric that indicates the closeness between $\hat{\mathbf{e}}(x_v)$ and $\mathbf{e}(x_v)$ given the sampled values of the constants of $\mathbf{e}(x_v)$. Note, however, that if $\hat{\mathbf{e}}(x_v)$ and $\mathbf{e}(x_v)$ are similar, the error r should be low regardless of the sampled values of the constants of $\mathbf{e}(x_v)$. For example, consider that $\mathbf{e}(x_v) = \sin(c_1\, x_v + c_2) + c_3$ and $\hat{\mathbf{e}}(x_v) = c_3 \cos(c_4\, x_v + c_5) + c_6$. Suppose that we sample the function $f_{target}(x_v) = \sin(3x_v + 2) + 6$ based on $\mathbf{e}(x_v)$; thus, we find the numerical values of the constant placeholders of $\hat{\mathbf{e}}(x_v)$ such that $f_{est}(x_v) = \cos(-3\, x_v - 0.429) + 6$ is mathematically equivalent to $f_{est}(x_v)$. If we sampled another function $f_{target}(x_v) = \sin(-4x_v - 1) - 0.5$, it is possible to find an equivalent function $f_{est}(x_v) = \cos(4\, x_v + 2.57) - 0.5$. For the sake of generality, we repeat this process 30 times; that is, we sample ten different $f_{target}(x_v)$ functions and solve 30 optimization problems. Finally, we report the mean and the standard deviation of the 30 resulting error metrics.

4.3 Experimental Results

In this section, we evaluate the performance of the intermediate univariate skeletons produced by SeTGAP as well as their multivariate expressions generated. We compare these results against other symbolic regression methods to assess the effectiveness of our approach.

4.3.1 Synthetic Datasets

We assessed SeTGAP's performance using 13 synthetic SR problems generated by equations inspired by previous works and equations proposed in this work, as reported in Table 4.1. Note that previous works used narrow domain ranges for all variables (e.g., $[-1, 1]$) while we used extended ranges (e.g., $[-5, 5]$ and $[-10, 10]$) to increase the difficulty of the

Table 4.1: Equations used for experiments

Eq.	**Underlying equation**	**Reference**	**Domain range**
E1	$(3.0375x_1x_2 + 5.5\sin(9/4(x_1 - 2/3)(x_2 - 2/3)))/5$	[79]	$[-5, 5]^2$
E2	$5.5 + (1 - x_1/4)^2 + \sqrt{x_2 + 10}\sin(x_3/5)$	—	$[-10, 10]^2$
E3	$(1.5e^{1.5x_1} + 5\cos(3x_2))/10$	[79]	$[-5, 5]^2$
E4	$((1 - x_1)^2 + (1 - x_3)^2 + 100(x_2 - x_1^2)^2 + 100(x_4 - x_3^2)^2)/10000$	Rosenbrock-4D	$[-5, 5]^4$
E5	$\sin(x_1 + x_2x_3) + \exp(1.2x_4)$	—	$x_1 \in [-10, 10]$, $x_2 \in [-5, 5]$, $x_3 \in [-5, 5]$, $x_4 \in [-3, 3]$
E6	$\tanh(x_1/2) + \lvert x_2\rvert\cos(x_3^2/5)$	—	$[-10, 10]^3$
E7	$(1 - x_2^2)/(\sin(2\pi\, x_1) + 1.5)$	[189]	$[-5, 5]^2$
E8	$x_1^4/(x_1^4 + 1) + x_2^4/(x_2^4 + 1)$	[174]	$[-5, 5]^2$
E9	$\log(2x_2 + 1) - \log(4x_1^2 + 1)$	[174]	$[0, 5]^2$
E10	$\sin(x_1\, e^{x_2})$	[14]	$x_1 \in [-2, 2], x_2 \in [-4, 4]$
E11	$x_1\, \log(x_2^4)$	[14]	$[-5, 5]^2$
E12	$1 + x_1\, \sin(1/x_2)$	[14]	$[-10, 10]^2$
E13	$\sqrt{x_1}\, \log(x_2^2)$	[14]	$x_1 \in [0, 20], x_2 \in [-5, 5]$

problems. We adapted the benchmark equations proposed by Bertschinger *et al.* [14] (i.e., E10–E13) to a multivariate setting.

In all cases, the training datasets consisted of 10,000 points where each variable was sampled using a uniform distribution. For the opaque models, $\hat{f}$, we trained feed-forward NNs with varying depths: three hidden layers for problem E2; five hidden layers for problems E1, E4, E5, and E7; and four hidden layers for the other cases. Each layer consisted of 500 nodes with ReLU activation (where $\texttt{ReLU}(q) = \max(0, q)$)). We used 90% of the samples for training and 10% for validation.

4.3.2 Univariate Skeleton Prediction Performance

We used the pre-trained Multi-Set Transformer g described in Section 3.4. As such, when generating the synthetic sets of points $\{\tilde{\mathbf{X}}^{(1)}, \ldots, \tilde{\mathbf{X}}^{(N_S)}\}$ used for univariate skeleton prediction, we used the same hyperparameters used for training g; that is, $N_S = 10$ input sets, each consisting of $n = 3000$ data points.

Fig. 4.3 depicts an example of the skeleton prediction process on problem E2 using the pre-trained Multi-Set Transformer g of SeTGAP. Here, $N_S = 10$ synthetic datasets $\tilde{\mathbf{X}}^{(s)} \in \mathbb{R}^{5000\times 3}$, where $s \in [1, 10]$, have been generated. In this example, we aim to obtain the skeleton for variable x_2, $\hat{\mathbf{e}}(x_2)$; thus, only x_2 is allowed to vary within each set $\tilde{\mathbf{X}}^{(s)}$ while the others, x_1 and x_2, are held constant. The estimated response corresponding to each set $\tilde{\mathbf{X}}^{(s)}$ was obtained using the feedforward NN $\hat{f}$. As a reference, we also plotted the real response that would have been obtained using the underlying function f. Recall f is unknown during the skeleton prediction process. Notice that the outputs of f and $\hat{f}$ are closely aligned, resulting in similar curve shapes across all cases. The only exception is $\tilde{\mathbf{X}}^{(6)}$, the highlighted plot, which shows a clear deviation. In this case, the behavior of the estimated response differs from the real response in a small region of the domain due to prediction uncertainties. Despite this behavior, the pre-trained Multi-Set Transformer g produced the skeleton $\hat{\mathbf{e}}(x_2) = c_1\sqrt{c_2\, x_2 + c_3} + c_4$, which is mathematically equivalent to the target skeleton $\mathbf{e}(x_2) = c_1\sqrt{x_2 + c_2} + c_3$.

We assessed skeleton performance using the method described in Section 4.2.5. The optimization process was configured with a population size of 500 and terminated when the objective function change remained below 10^{-6} for 30 consecutive generations. We employed tournament selection, binomial crossover, and generational replacement, as this configuration consistently yielded effective optimization results across all experiments in this study. The GA-based optimization processes in the following section follow the same setup.

We compared the skeletons produced by SeTGAP's Multi-Set Transformer to the ones

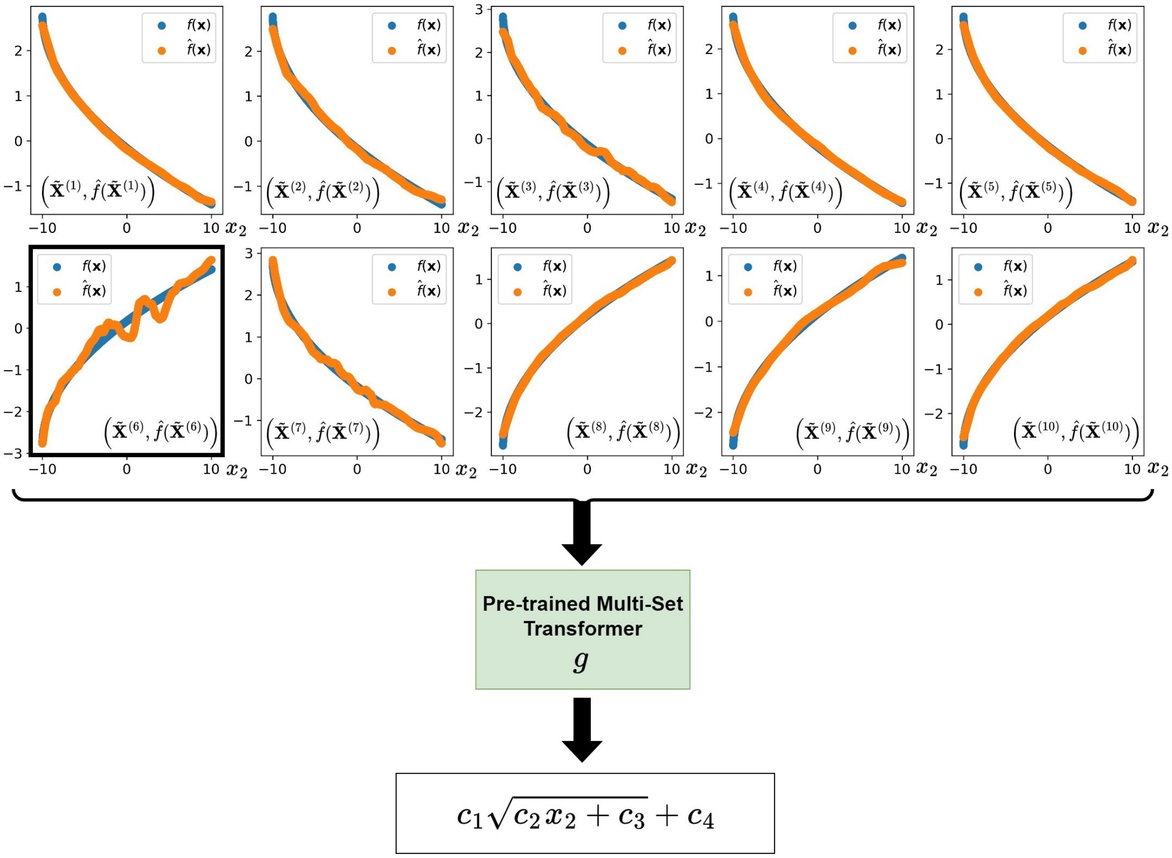


Figure 4.3: Skeleton prediction example for variable x_2 on problem E2.

extracted from the expressions generated by four other methods: two GP-based methods (PySR [34] and TaylorGP [67]) and two neural SR methods (NeSymReS [15] and E2E [82]). The comparison is limited to neural SR methods with publicly available models, as training large transformer-based architectures is computationally prohibitive. NeSymReS could not be executed on E4 and E5 because its model was trained using expressions limited to three variables. NeSymReS, E2E, and the Multi-Set Transformer were trained using vocabularies with the same unary and binary operators: $+, \times, /$, `abs`, `acos`, `asin`, `atan`, `cos`, `cosh`, `exp`, `log`, `pow2`, `pow3`, `pow4`, `pow5`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`. Thus, PySR and TaylorGP were executed using the same set of operators. Our experimentation with the GP-based methods involved a maximum of 10,000 iterations, though convergence was consistently achieved in

Table 4.2: Comparison of skeleton prediction results for problem E2

Method	x_1	x_2	x_3
PySR	$c_1 + \lvert c_2 + \lvert c_3 + x_1\rvert\rvert$	c_1	$c_1 + c_2\,x_3$
TaylorGP	$c_1 + c_2\,x_1$	$c_1 + c_2\,x_2$	$c_1 + c_2\,x_3$
NeSymReS	$c_1 + c_2\,x_1$	$c_1 + \exp(\exp(c_2\,x_2))$	$c_1 + c_2\,x_3$
E2E	$c_1 + c_2\,x_1 + c_3(c_4 + c_5\,x_1)^2$	$c_1 + c_2\,(c_3 + c_4\,x_2)$	$c_1 + c_2\,x_3 + c_3(c_4 + c_5\cos(c_6 + c_7\,x_3))$
SeTGAP	$c_1 + c_2(c_3 + c_4\,x_1)^2$	$c_1\sqrt{c_2\,x_2 + c_3} + c_4$	$c_1 + c_2\sin(c_3\,x_3 + c_4)$
Target $\mathbf{e}(x)$	$c_1 + (c_2 + c_3\,x_1)^2$	$c_1\sqrt{x_2 + c_2} + c_3$	$c_1 + c_2\sin(c_3\,x_3)$

fewer iterations across all cases. Population sizes of 100, 200, 500, and 1000 were tested, with no discernible advantage observed beyond a size of 500.

The compared methods produce multivariate expressions, from which the skeleton variable corresponding to variable x_v is obtained using the skeleton function $\kappa(\cdot, x_v)$. Table 4.2 shows the target and estimated skeletons corresponding to each variable for problem E2. The skeletons obtained for the other problems are presented in Tables A.1 and A.2 of Appendix A. The size of the test sets is set to $N_{\text{test}} = 3000$. Using $N_{\text{test}} > 3000$ did not vary the obtained results. Table 4.3 reports the rounded mean and the standard deviation of the error metrics obtained after 30 repetitions of the proposed evaluation. The bold entries indicate the method that achieved the lowest mean error r and that its difference with respect to the values obtained by the other methods is statistically significant according to Tukey's honestly significant difference test performed at the 0.05 significance level.

4.3.3 Underlying Function Estimation Performance

In this section, we assess SeTGAP's performance by evaluating the learned mathematical expressions on both in-domain and out-of-domain data, comparing the resulting prediction errors with those of other SR methods.

SeTGAP's hyperparameters include the beam size n_B for beam search and the number

Table 4.3: Skeleton evaluation performance comparison

Eq.	**Var.**	**PySR**	**TaylorGP**	**NeSymReS**	**E2E**	**SeTGAP**
E1	x_1	1.4 ± 0.8	1.4 ± 0.8	0.9 ± 0.7	$\mathbf{0.2 \pm 0.4}$	$\mathbf{0.01 \pm 0.02}$
	x_2	1.5 ± 0.9	1.5 ± 0.9	1.3 ± 0.8	1.5 ± 0.9	$\mathbf{0 \pm 0}$
E2	x_1	303.5 ± 167.3	310.0 ± 170.1	310.0 ± 170.1	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	5.4 ± 5.0	4.2 ± 5.3	4.6 ± 5.0	4.2 ± 5.3	$\mathbf{0.02 \pm 0.03}$
	x_3	1.7 ± 1.0	1.7 ± 1.0	1.7 ± 1.0	$\mathbf{0.01 \pm 0.02}$	$\mathbf{0 \pm 0}$
E3	x_1	$2 \times 10^{12} \pm 5 \times 10^{12}$	939.4 ± 1419.9	1.9 ± 1.2	$\mathbf{0.8 \pm 1.8}$	$\mathbf{0.8 \pm 1.8}$
	x_2	1.3 ± 1.0	1.3 ± 1.0	0.8 ± 0.8	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E4	x_1	4576.2 ± 2695.7	4581.5 ± 2697.4	—	$\mathbf{2.3 \pm 3.6}$	$\mathbf{1.1 \pm 0.7}$
	x_2	79.6 ± 41.3	80.2 ± 40.8	—	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_3	3995.5 ± 2815.6	4304.6 ± 2843.7	—	$\mathbf{2.0 \pm 4.0}$	$\mathbf{1.0 \pm 0.9}$
	x_4	74.5 ± 48.0	75.5 ± 47.0	—	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E5	x_1	$\mathbf{0 \pm 0}$	0.6 ± 0.05	—	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	1.5 ± 1.0	—	1.5 ± 1.0	$\mathbf{0 \pm 0}$
	x_3	$\mathbf{0 \pm 0}$	0.6 ± 0.05	—	0.6 ± 0.05	$\mathbf{0 \pm 0}$
	x_4	$\mathbf{0 \pm 0}$	487.4 ± 461.9	—	$\mathbf{0.6 \pm 0.8}$	$\mathbf{0.6 \pm 0.8}$
E6	x_1	0.8 ± 0.08	0.8 ± 0.08	0.3 ± 0.01	0.04 ± 0	$\mathbf{0 \pm 0}$
	x_2	16.8 ± 12.2	16.8 ± 12.2	13.8 ± 11.1	1.3 ± 0.9	$\mathbf{0 \pm 0}$
	x_3	2.9 ± 1.3	1.9 ± 0.7	1.6 ± 0.9	1.6 ± 0.9	$\mathbf{0 \pm 0}$
E7	x_1	29.5 ± 1.0	1.8 ± 2.0	1.8 ± 2.0	1.1 ± 1.3	$\mathbf{0 \pm 0}$
	x_2	63.6 ± 43.8	1.6 ± 1.0	42.4 ± 24.7	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E8	x_1	0.02 ± 0.01	0.04 ± 0.01	0.8 ± 1.2	0.02 ± 0.02	$\mathbf{0 \pm 0}$
	x_2	0.02 ± 0.01	0.04 ± 0.01	0.9 ± 1.3	0.02 ± 0.01	$\mathbf{0 \pm 0}$
E9	x_1	271.3 ± 446.8	239.8 ± 428.2	375.1 ± 485.5	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	0.2 ± 0.09	2.7 ± 1.7	0.05 ± 0.01	$\mathbf{0 \pm 0}$
E10	x_1	$\mathbf{0 \pm 0}$	0.6 ± 0.2	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	0.4 ± 0.06	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E11	x_1	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E12	x_1	21.8 ± 13.1	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	2.4 ± 1.5	2.5 ± 1.6	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$
E13	x_1	$\mathbf{0 \pm 0}$	0.8 ± 0.5	0.7 ± 0.6	0.7 ± 0.8	$\mathbf{0 \pm 0}$
	x_2	$\mathbf{0 \pm 0}$	3.8 ± 3.5	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$	$\mathbf{0 \pm 0}$

of univariate skeleton candidates n_{cand} (see Algorithm 4.1). We set $n_B = 3$ and $n_{\text{cand}} = 4$, as higher values did not yield more distinct skeletons across all tested problems. The GAs in Secs. 4.2.3 and 4.2.4 share the same configuration but differ in loss functions: the former maximizes Pearson correlation, while the latter minimizes MSE. Algorithm 4.3 uses the number of instance expressions per candidate skeleton combination `rep` and the maximum number of generations `maxG`, with values set to `rep` = 150 and `maxG` = 300. In addition, the initial population `candSks` used in Algorithm 4.3 was generated with a maximum size $P_{max} = 5000$, though none of the cases reached this limit. This setup was chosen for its consistent effective optimization results across all evaluated problems.

For each problem, SeTGAP generates up to n_{cand} multivariate expressions, but for brevity, we report only the one with the lowest MSE. These results are compared against expressions obtained from the same SR methods considered in Section 4.3.2. Table 4.4 presents the learned expressions, with shaded cells indicating that the learned expression's functional form matches that of the underlying function. To ensure a fair comparison, the evaluation was repeated nine additional times, each with a newly generated dataset using a different random seed. The expressions obtained by all compared methods across all problems and iterations are reported in Tables A.4–A.12 of Appendix A.

We evaluated the extrapolation capability of the learned expressions by testing them on an extended domain range. The original domain range, referred to as the interpolation range, for a variable x_v is denoted as $[x_v^\ell, x_v^u]$, while its extrapolation range is defined as $[2x_v^\ell, x_v^\ell[\, \cup \,]x_v^u, 2x_v^u]$. Each extrapolation set comprised 10,000 points sampled uniformly within this range. This evaluation was repeated for each of the 10 expressions learned by each method. Table 4.5 presents the rounded mean and standard deviation of the extrapolation MSE across these runs. Bold entries indicate the method that achieved the lowest mean error and for which the difference relative to the other methods is statistically significant, as determined by Tukey's honestly significant difference test at the 0.05 significance level.

Finally, we tested SeTGAP under noisy conditions. We considered a normal error term $\varepsilon_a = \mathcal{N}(0, \sigma_a \sigma_{\mathbf{y}})$ and four noise levels: $\sigma_a = \{0, 0.01, 0.03, 0.05\}$. Here, $\sigma_{\mathbf{y}}$ denotes the standard deviation of the response variable so that the noise is scaled relative to the dispersion of each problem. The obtained interpolation and extrapolation MSE are shown

Table 4.4: Comparison of predicted expressions with rounded numerical coefficients

Eq.	**PySR**	**TaylorGP**	**NeSymReS**	**E2E**	**SeTGAP**
E1	$0.61x_0x_1$	$0.64x_0x_1$	$0.59x_0x_1+$ $\cos(0.01(x_1-x_0-0.08)^2)$	$1.08(0.56x_0x_1-0.03x_0+0.02x_1-$ $\sin(0.01x_0^2+8.6x_0+0.45)-0.01)$	$0.61x_0x_1+$ $1.15\sin((2.24x_0-1.5)(x_1-0.68))$
E2	$0.41x_2+$ $\|\|x_0-3.51\|-1.95\|$ $+4.11$	$-0.5x_0+0.001x_1$ $+0.39x_2+8.62$	$-x_0+0.40x_2+$ $e^{e^{-0.001x_1}}+5.88$	$0.06x_0^2-0.51x_0-0.22x_1\cos(0.18x_2$ $+1.43)-0.01x_1+0.01x_2-$ $3.25\cos(0.18x_2+1.43)+6.56$	$0.06x_0^2-0.5x_0+(3.37\sqrt{0.1x_1+1}$ $-0.19)(\sin(0.2x_2)+0.01)+6.49$
E3	$0.34e^{x_0}\|\sinh(0.47x_0)\|$	$0.23x_0e^{x_0}$	$9.10e^{0.72x_0}\cos(0.15x_1)$	$0.14e^{1.52x_0}+$ $0.52\cos(3.45x_1+0.05)+0.11$	$0.15e^{1.5x_0}+0.5\sin(3x_1-4.71)$
E4	$0.21x_0^2-0.18x_1+$ $0.21x_2^2-0.18x_3-0.76$	$0.29x_0^2$	—	$0.001\|8.99(-0.88x_1+(x_0+0.01)^2$ $+0.62)^2+9.72(-x_3+$ $0.98(x_2+0.01)^2+0.01)^2\|+0.0023$	$0.01x_0^4-0.02x_0^2x_1-0.001x_0+0.01$ $x_1^2+0.01x_2^4+0.01x_3^2-$ $(0.02x_2^2+0.004)(x_3-0.11)-0.02$
E5	$e^{1.2x_3}+\sin(x_0+x_1x_2)$	$0.51e^{x_3}e^{\sin(0.87x_3)}$	—	$e^{1.2x_3}-0.91\cos((2.62x_0+0.15)$ $(24.66x_1+1.24))-0.05$	$0.999e^{1.2x_3}-\sin(x_0+x_1x_2+9.42)$
E6	$\tanh(e^{x_2})$	$-\frac{sin(0.34x_2^2)}{-\sqrt{\|x_2\|}+\sin(\sqrt{\|x_2\|})}$	$-0.39x_0+x_1\sin(\frac{x_0}{x_1}-0.001x_2)$	$0.01x_1(-7.5\cos(15.41x_1+0.21)-$ $0.18)+0.69\,\text{atan}(0.75x_0+0.05)+0.47$	$\cos(0.2x_2^2+0.05)\|x_1\|+\tanh(0.5x_0)$
E7	$(0.56-0.59x_0^2)/(\sinh(\sinh$ $(\tanh(e^{\sinh(\sin(6.28x_1))}))))$	$\sqrt{\|x_1\|}-x_1^2$	$\frac{0.12x_0+x_1^2}{\cos(3.1(-0.02x_1-1)^2)-0.31}$	$(-0.03x_1-0.03)(0.34x_1-0.35)$ $(41.59(1-0.5\sin(6.74x_0+0.23))^2+40)$	$\frac{4.53-4.54x_1^2}{4.54\sin(6.28x_0+6.28)+6.81)}$
E8	$(\tanh(\cosh(x_0)-1.04)+$ $\tanh(\cosh(x_1)-1.04))$	2	$\cos(\sin(1.69x_0)/(x_0x_1))+0.71$	$2.01-1.05e^{-0.06\|x_0 2.73-0.14\|\|0.59x_1+0.1\|}$	$2-\frac{19.76}{19.31x_0^4+0.12x_0^3+0.42x_0^2+19.72}-$ $\frac{5.33}{5.44x_1^4-0.09x_1^2+5.34}$
E9	$\log(\frac{x_1+0.5}{0.5+2x_0^2})$	$\log(\frac{0.79}{\|x_0\|})-$ $2.36e^{-x_1}$	$1.12\log(\|x_1/x_0\|)-1.37$	$2-0.60\log(13.36(0.004-x_0)^2$ $(1-0.13/(-0.06x_1-0.02))^2+0.8)$	$-\log(13.95x_0^2+3.48)+$ $\|\log(8.32x_1+4.18)\|-0.18$
E10	$\sin(x_0e^{x_1})$	$x_0e^{-\sqrt{\|x_1\|}}$	$\sin(x_0e^{x_1})$	$-0.98\sin((0.06-2.86e^{1.03x_1})$ $(0.32x_0+0.002))-0.007$	$\sin(x_0e^{0.999\,x_1})$
E11	$x_0\log(x_1^4)$	$4x_0\log(\|x_1\|)$	$x_0\log(x_1^4)$	$-0.74x_0(-5.62\log(0.07\|-6.94x_1+0.13\|$ $+0.01)-3.74)$	$1.998x_0\log(x_1^2)$
E12	$\sin(\frac{x_0}{x_1/0.12})\|x_0\|+0.99$	$(x_0+\sqrt{\|x_1\|}-$ $0.91)\sin(\frac{0.73}{x_1})$	$(x_0+x_1)\sin(1/x_1)$	$(0.79x_0-0.04)\sin(4.5/(3.4x_1+0.08)$	$x_0\sin(1/x_1)+1$
E13	$\sqrt{x_0}\log(x_1^2)$	$\sqrt{e^{\sqrt{x_0}}}\log(\|x_1\|)+$ $\log(\|x_1\|)+0.58$	$0.31x_0+3.19\log(x_1^2)-3.25$	$(-90.0+\frac{9}{0.12\|3.4x_1+0.12\|+0.04})$ $(0.09-0.1\log(0.17x_0+3.29))$	$2.0\sqrt{x_0}\log\|x_1\|$

in Table 4.6, where shaded cells indicate an incorrectly identified functional form. The learned expressions are provided in Table A.13 of Appendix A.

4.4 Discussion

In this section, we analyze the results presented above, discussing the effectiveness of SeTGAP in both univariate skeleton prediction and underlying function estimation. We examine the quality and reliability of the learned symbolic representations, highlighting the

Table 4.5: Extrapolation MSE Comparison

Eq.	PySR	TaylorGP	NeSymRes	E2E	SeTGAP
E1	3.673e-01 ± 4.015e-01	1.221 ± 1.310	3.372 ± 3.486	1.217 ± 1.262e-01	**1.182e-01 ± 8.567e-02**
E2	9.899e+01 ± 6.470e+01	2.385e+02 ± 7.563e+01	8.589e+07 ± 1.909e+08	6.885e+01 ± 6.822e+01	**8.391e-02 ± 7.624e-02**
E3	2.068e+03 ± 6.194e+03	3.151e+04 ± 3.960e+03	4.653e+04 ± 9.214e+03	1.448e+05 ± 3.675e+05	**1.028e+01 ± 1.789e+01**
E4	2.206e+05 ± 9.044e+04	6.621e+03 ± 1.057e+03	—	8.762e+02 ± 2.323e+03	**1.621 ± 1.887**
E5	**5.007e-02 ± 1.502e-01**	2.224e+04 ± 2.985e+04	—	2.054e+03 ± 2.805e+03	5.492 ± 1.205e+01
E6	1.826e+01 ± 3.446e+01	1.161e+02 ± 3.858	1.892e+02 ± 4.071e+01	1.302e+02 ± 2.514e+01	**3.832 ± 5.890**
E7	5.707e+02 ± 1.110e+03	1.042e+03 ± 6.623	1.621e+03 ± 8.193e+02	1.781e+03 ± 5.259e+02	**3.597e-02 ± 4.490e-02**
E8	**3.322e-04 ± 4.755e-04**	1.175 ± 2.123	1.209e-01 ± 1.908e-02	1.323e-01 ± 3.846e-01	**3.380e-08 ± 6.313e-08**
E9	7.794e+02 ± 2.334e+03	3.013e-01 ± 1.501e-01	1.032 ± 4.763e-02	4.279e-01 ± 1.737e-01	**2.721e-06 ± 4.089e-06**
E10	**7.266e-11 ± 1.539e-10**	1.686e-01 ± 2.124e-01	**0.000 ± 0.000**	3.630e-01 ± 4.760e-02	**3.717e-04 ± 5.617e-04**
E11	**1.942e-02 ± 5.825e-02**	3.076 ± 8.020	**0.000 ± 0.000**	9.880e+01 ± 1.048e+02	**9.358e-05 ± 2.807e-04**
E12	**8.048e-01 ± 2.413**	2.965 ± 1.574	2.574e-02 ± 0.000	5.312 ± 1.662	**1.109e-06 ± 2.219e-06**
E13	**2.722 ± 8.162**	9.434 ± 2.311e+01	6.055e+01 ± 1.563e+01	1.829e+02 ± 3.300e+02	**8.592e-07 ± 1.779e-06**

strengths and limitations of the proposed approach compared to existing SR methods. The discussion is structured into two subsections, corresponding to the key aspects evaluated in the results section: univariate skeleton prediction (Section 4.3.2) and underlying function estimation (Section 4.3.3).

4.4.1 Univariate Skeleton Predictions Results

Our method involves probing an opaque regression model, specifically a feedforward neural network, by formulating an MSSP problem for each system variable and solving it with a Multi-Set Transformer. This process produces univariate skeletons that describe the functional relationship between each variable and the system's response. After evaluation of our univariate skeleton prediction method across the tested problems, we observed that it generated skeletons that matched or were equivalent to the target skeleton for all variables across all problems. For instance, for problem E11, the target skeleton for variable x_2 is given by $\mathbf{e}(x_2) = c_1 \log(x_2^4)$, and our method produces the skeleton $\hat{\mathbf{e}}_{\text{SeTGAP}}(x_2) = c_1' + c_2' \log(c_3' x_2^2)$. These skeletons are equivalent if $c_1' = 0$, $c_2' = 2\, c_1$, and $c_3' = 1$. From the skeleton performance

Table 4.6: MSE comparison using SeTGAP with noisy data

Eq.	Interpolation ($\sigma_a = 0$)	Interpolation ($\sigma_a = 0.01$)	Interpolation ($\sigma_a = 0.03$)	Interpolation ($\sigma = 0.05$)	Extrapolation ($\sigma_a = 0$)	Extrapolation ($\sigma_a = 0.01$)	Extrapolation ($\sigma_a = 0.03$)	Extrapolation ($\sigma_a = 0.05$)
E1	7.159e-03	1.259e-02	1.236e-01	5.455e-01	1.012e-01	4.433e-01	1.645	3.845
E2	7.024e-03	4.459e-03	1.563e-02	4.754e-02	1.300e-01	5.232e-02	1.017e-01	3.504e-01
E3	1.063e-03	1.022e-03	7.219e-03	2.003e-02	2.961e-01	2.055e+02	7.467e+01	1.652e+02
E4	1.158e-04	2.969e-03	2.061e-01	2.228e-02	4.121e-02	1.022e+01	7.420e+01	3.105e+01
E5	7.211e-06	6.807e-03	5.288e-01	1.694e-01	3.069e-01	1.097e+01	8.402e+01	2.312e+02
E6	5.619e-03	1.147e-02	1.726e-02	4.819e-02	4.508e-02	4.651	2.081e-01	5.598e-01
E7	2.655e-06	1.076e-02	6.822e-02	1.903e-01	6.447e-05	2.517e-01	1.247	3.520
E8	1.742e-06	2.413e-05	2.103e-04	5.828e-04	1.817e-10	1.686e-07	7.539e-09	1.672e-07
E9	4.907e-08	2.160e-04	1.943e-03	5.408e-03	2.780e-07	3.063e-04	2.756e-03	7.923e-03
E10	2.282e-06	3.298e-05	2.950e-04	8.192e-04	1.857e-03	2.233e-04	4.793e-04	3.379e-03
E11	1.178e-04	1.838e-02	1.654e-01	4.595e-01	9.308e-04	1.453e-01	1.308	3.632
E12	8.029e-06	4.938e-04	4.436e-03	1.232e-02	7.273e-06	1.031e-03	9.257e-03	2.571e-02
E13	2.583e-07	4.200e-03	3.991e-02	1.047e-01	1.015e-06	9.599e-03	9.348e-02	2.389e-01

evaluation shown in Table 4.3, we verified that our method consistently attained lower or comparable error metrics compared to other SR methods. These results strongly support our hypothesis that our method would generate univariate skeletons that are more similar to those corresponding to the underlying equations in comparison to other SR methods.

Note that E2E produced the correct skeleton for at least one of the variables in most of the cases. Recall that E2E generated expressions that minimize the prediction error; thus, it did not prioritize identifying the correct functional form of the variables that do not contribute substantially to the overall error. In addition, in some cases, E2E generated skeletons that were equivalent to the target skeletons but larger than those produced by SeTGAP. For example, in problem E4, E2E generated the skeleton $\hat{\mathbf{e}}_{E2E}(x_1) = c_1 + c_2|c_3 + c_4\, x_1 + c_5\, x_1^2 + c_6\, x_1^3 + c_7\, x_1^4|$ for variable x_1, which is equivalent to the one produced by SeTGAP, $\hat{\mathbf{e}}_{\text{SeTGAP}}(x_1) = c_1 + c_2\, x_1 + c_3\, x_1^2 + c_4\, x_1^3 + c_5\, x_1^4$.

Another advantage over the other neural SR methods is that SeTGAP's Multi-

Set Transformer requires only 24.2 million parameters, while E2E requires 93.5 million. NeSymReS, in comparison, requires 26.4 million parameters (i.e., 2.2 million more than ours), and Table 4.3 shows that it failed to identify the correct functional form in most cases and is limited to solving problems with up to three variables.

It is worth pointing out that in problems E5, E6, E8, E9, and E13, some compared methods achieved low error metrics but are not comparable to the ones achieved by our method. For example, in problem E6, E2E generated the skeleton $\hat{\mathbf{e}}_{E2E}(x_1) = c_1 + c_2\,\mathrm{atan}(c_3 + c_4\,x_1)$ for variable x_1, which does not coincide with the functional form of the underlying skeleton $\mathbf{e}(x_1) = c_1 + \tanh(c_2\,x_1)$. However, E2E achieved low error metrics because, during the coefficient fitting process, the GA found appropriate values for the constant that multiplies the argument of the `atan` function, stretching or compressing the curve, making it resemble the shape of `tanh` and minimizing the error. Hence, the skeleton generated by E2E produced low error metrics and is considered to be similar to the target skeleton. Conversely, the functional form of the skeleton generated by SeTGAP coincided with that of the target skeleton exactly (i.e., $\hat{\mathbf{e}}_{\mathrm{SeTGAP}}(x_1) = c_1 + c_2\tanh(c_3\,x_1)$) and thus produced significantly lower error.

One potential limitation of our approach, as well as any neural SR method, lies in its ability to generate skeletons whose complexity is bounded inherently by the expressions produced during the pre-training phase of the Multi-Set Transformer. For instance, we would not be able to identify the skeleton $c_1 + c_2\,x_2^2/\sin(c_3\,e^{c_4\,x_2})$ as it requires eight operators, while our training set was limited to expressions with up to seven operators. However, it is feasible to overcome this limitation through transfer learning, so that the MST model can be trained on more complex tasks, potentially enabling the recognition of such complex skeletons.

4.4.2 Underlying Function Estimation Results

SeTGAP can be viewed as a *post-hoc* interpretability method, as it extracts mathematical expressions that align with the functional response learned by a given opaque regression model. Our decomposable approach learns and preserves functional relationships between input variables and the system's response, and increments them progressively, allowing for an interpretable evolution. This prevents the resulting expressions from focusing solely on error minimization, encouraging alignment with the true functional form instead.

From Table 4.4, we confirmed that SeTGAP successfully learned mathematical expressions equivalent to the underlying functions in Table 4.1 across all tested problems. In contrast, competing methods correctly identified the functions in no more than seven out of the 13 cases. Notably, some methods only captured the functional form of the most influential variables; i.e., those contributing most to the response value. For instance, E2E recovered correctly the term $0.06x_0^2 - 0.51x_0$ for x_0 in E2 but failed for the remaining variables. It is worth noting that E2E was the only method that produced expressions longer than reported, requiring simplification via a symbolic manipulation library.

Table 4.5 confirms that SeTGAP achieved lower or comparable extrapolation MSE values across all problems. These results suggest that other methods, which optimize purely for in-domain MSE, tend to overfit the training data and learn expressions that lack the structural flexibility needed for extrapolation. In contrast, SeTGAP's decompositional approach learns functional forms that better capture the underlying relationships, enabling superior generalization outside the training domain. For example, in problem E8, most SR methods effectively minimized prediction MSE. TaylorGP, prioritizing parsimony, evolved the expression $\tilde{f}(\mathbf{x}) = 2$, which effectively smooths out the data but does not provide a meaningful solution. As such, the simplest solution is not always the best, as small variations in the data may correspond to functional forms that may play an important role when generalizing to unseen data. Table 4.5 also shows cases where NeSymRes achieved zero extrapolation error across the 10 iterations for E10 and E11, while SeTGAP obtained low but nonzero MSE values. This occurs because competing methods learned expressions that matched the underlying functional form perfectly, avoiding the need for coefficient fitting. PySR, for example, identified $\sqrt{x_0}\log(x_1^2)$ for E13 in its first iteration, which contains no numerical coefficients. In contrast, SeTGAP produced $2.000137\sqrt{x_0}\log|x_1|$, where the fitted coefficient introduced minor prediction errors.

Since all previous experiments were conducted on noiseless data, and SeTGAP was the only method that consistently identified the correct functional form, we assessed its robustness under varying noise levels, as shown in Table 4.6. As expected, interpolation and extrapolation errors increased with higher noise levels.

Interpolation errors remained low across all cases, while extrapolation errors showed a few exceptions due to poor coefficient fitting or incorrect functional form identification. For example, E3 and E5 exhibited high extrapolation errors because their functions contain exponential terms. Even when the learned expressions matched the expected functional form, small coefficient errors in the exponential term led to significant deviations for larger values of x_1. A similar case was observed in E1, where extrapolation errors were high for $\sigma_a \geq 0.03$ despite correctly identifying the functional form. The ability to recover the correct functional form in the presence of noise can be attributed to the fact that the NN $\hat{f}$ used during inference to generate the multiple input sets $\tilde{\mathbf{D}}_v$ smooths the estimated response values, mitigating the impact of noise. In the remaining cases, we observed two outcomes. SeTGAP correctly identified the functional forms of individual variables, but noise hindered the detection of relationships between variables. Otherwise, incorrect but reasonable univariate skeletons were identified, leading to expressions with small errors; e.g., for E9 and $\sigma_a = 0.05$, SeTGAP produced $\tilde{f}(\mathbf{x}) = 5.965\sqrt{0.63\log(9.4x_1 + 6.4) + 1} - \log(11.26x_0^2 + 2.82) - 7.71$.

A limitation of our approach is the computational cost due to multiple intermediate optimization processes, making SeTGAP less efficient than end-to-end approaches like E2E.

However, in applications like scientific discovery, where the goal is to derive interpretable and reliable mathematical expressions rather than simply optimizing predictive accuracy, the additional computational effort is warranted.

4.5 Summary

In this chapter, a novel symbolic regression method named "Symbolic Regression using Transformers, Genetic Algorithms, and Genetic Programming" is introduced. SeTGAP is designed to address multivariate SR problems by probing trained opaque regression models to distill them into multivariate mathematical expressions that allow us to interpret the regression model's functional form. The chapter presents the workflow of the SeTGAP method, delves into the univariate skeleton prediction method, and describes how these skeletons are then merged, serving as fundamental building blocks for the final expressions.

In particular, we explored the role of the Multi-Set Transformer model in generating multiple univariate symbolic skeletons that characterize how each variable influences the system's response. A GA-based approach is employed to select the best skeleton candidates, which are then incrementally merged using a GP-based cascade procedure that preserves their original structure. The final multivariate skeletons undergo coefficient optimization via GA to refine prediction accuracy while maintaining the learned functional form.

Furthermore, we introduce the first performance evaluation method for assessing how well the functional form of the learned symbolic skeletons matches the system's underlying mathematical structure. Our experimental results demonstrate that, when compared to two GP-based SR methods and two neural SR methods, SeTGAP consistently produced univariate skeletons that are more similar to the target skeletons for all system variables across all tested problems. As a result of identifying univariate expressions accurately, they serve as reliable building blocks in SeTGAP's merging process. Thus, we obtained full multivariate expressions that matched the original mathematical structure in all cases, unlike the compared methods. These results highlight the effectiveness of SeTGAP in learning human-readable and structurally accurate symbolic representations of multivariate systems.

CHAPTER FIVE

UNCERTAINTY MANAGEMENT

In various scientific and engineering fields, the development of accurate predictive models frequently relies on effective uncertainty quantification and strategic experimentation. Uncertainty quantification is crucial in fostering confidence in AI-driven systems, especially those built on inherently opaque models, whose internal mechanisms are not directly interpretable. Without a clear understanding of how a model arrives at its predictions, users may be reluctant to rely on its outcomes, particularly in high-stakes applications [156], or in applications where uncertainty is inherent in the data or the underlying system, including weather forecasting [197], electronic manufacturing [158], and precision agriculture [116]. By rigorously quantifying the uncertainty associated with predictions, practitioners can better assess the reliability of model outputs, identify regions of low confidence, and make more informed decisions about when and how to trust the system.

Furthermore, experimental design aimed at reducing uncertainty offers another avenue for improving the reliability of AI-assisted decision-making. However, such experiments are often costly and time-consuming, highlighting the need to adopt strategies that extract the most valuable information from each experiment. One notable example is precision agriculture, where experimental results may require an entire growing season to manifest, and only a portion of the field is allocated for such trials [100]. This is exacerbated by the fact that data can often only be collected every other year, due to crop rotation. Thus, critical decisions, such as those aimed at maximizing profit or minimizing environmental impact, must be made under significant uncertainty. For instance, determining optimal fertilizer application rates relies on accurately estimating nitrogen–yield response curves [20, 123]. These curves describe expected crop yields at specific field locations as a function of varying fertilizer inputs. However, uncertainty across the input domain can distort the shape of these curves, compromising their estimation and leading to unreliable fertilizer recommendations.

This chapter focuses on uncertainty quantification and sampling techniques designed to reduce uncertainty in the prediction models across the entire input domain. As briefly introduced in Section 4.2, we distinguish between two main types of uncertainty: epistemic and aleatoric. The former represents the portion of total uncertainty that can be reduced by gathering more information or improving the prediction model. On the other hand, aleatoric uncertainty is the inherent and irreducible component of uncertainty due to the random nature of the data itself [75, 136]. The total uncertainty associated with a prediction (σ_y^2) encapsulates both the aleatoric (σ_a^2) and epistemic (σ_e^2) components; i.e., $\sigma_y^2 = \sigma_a^2 + \sigma_e^2$. It is important to clarify the distinction between these terms and the corresponding random error terms ε_a and ε_e, introduced in Section 4.2. For instance, while $\sigma_a^2(\mathbf{x})$ represents the aleatoric uncertainty at a given point, $\varepsilon_a(\mathbf{x})$ denotes a single realization of that uncertainty.

Prediction intervals (PIs) offer a comprehensive representation of this total uncertainty by estimating the upper and lower bounds within which a prediction is expected to fall with a given probability [85]. To address the challenge of producing high-quality PIs that are both sufficiently narrow and capture most of the probability density, we present a method for automatically learning PIs alongside conventional target predictions in regression-based neural networks. In particular, we train two companion neural networks: one that uses one output, the target estimate, and another that uses two outputs, the upper and lower bounds of the corresponding PI. The PI-generation network uses a novel loss function called "**Dual** **A**ccuracy-**Q**uality-**D**riven" (DualAQD) that takes into account the output of the target-estimation network and has two optimization objectives: minimizing the mean prediction interval width and ensuring the PI integrity using constraints that maximize the prediction interval probability coverage implicitly. Furthermore, we introduce a self-adaptive coefficient that balances both objectives within the loss function, which alleviates the task of fine-tuning.

Furthermore, we present a method to reduce epistemic uncertainty through adaptive sampling using PIs generated by neural networks. Our method, **A**daptive **S**ampling with **P**rediction-**I**nterval **N**eural **N**etworks (ASPINN), uses a dual NN architecture comprising a target-estimation network and a PI-generation network that produces high-quality PIs that reflect both aleatoric and epistemic uncertainties. Then, we introduce a novel metric based on NN-generated PIs to quantify potential levels of epistemic uncertainty. At each iteration, ASPINN builds a Gaussian Process (GPR)[1] from calculated potential epistemic uncertainty levels. The GPR, a surrogate for the NN models, estimates potential epistemic uncertainty changes across the domain after sampling specific locations. An acquisition function then uses the GPR to select sampling locations, aiming to minimize global epistemic uncertainty throughout the input domain. Finally, we use the symbolic regression technique introduced in Chapter 4, SeTGAP, to study how varying degrees of epistemic uncertainty observed through the AS process affect the learned mathematical expressions that describe the data.

5.1 Background

This section reviews related work in the areas of prediction interval learning and uncertainty minimization. We highlight key methodologies, examine their limitations, and explore how they relate to the specific challenges addressed in this chapter.

5.1.1 Prediction Interval Learning

One of the more common approaches to uncertainty quantification for regression tasks is via Bayesian approaches, such as those represented by Bayesian neural networks (BNNs), which model the NN parameters as distributions. As such, they have the advantage that they allow for a natural quantification of uncertainty. In particular, uncertainty is

[1]We use GPR to refer to Gaussian Process to avoid confusion with GP, which previously referred to Genetic Programming.

quantified by learning a posterior weight distribution [25, 133]. The inference process involves marginalization over the weights, which in general is intractable, and sampling processes such as Markov chain Monte Carlo (MCMC) can be computationally prohibitive. Thus, approximate solutions have been formulated using variational inference (VI) [16]. However, Wu *et al.* [193] argued that VI approaches are fragile since they require careful initialization and tuning. To overcome these issues, they proposed approximating moments in NNs to eliminate gradient variance. They also presented an empirical Bayes procedure for selecting prior variances automatically. Moreover, Izmailov *et al.* [76] discussed scaling BNNs to deep neural networks by constructing low-dimensional subspaces of the parameter space. By doing so, they were able to apply elliptical slice sampling and VI, which struggle in the full parameter space. In addition, Lut *et al.* [110] presented a Bayesian-learning-based sparse stochastic configuration network that replaces the Gaussian distribution with a Laplace one as the prior distribution for output weights.

Despite the aforementioned improvements in Bayesian approaches, they still suffer from various limitations. Namely, the high dimensionality of the parameter space of deep NNs, including complex models such as CNNs, makes the cost of characterizing uncertainty over the parameters prohibitive [195]. Attempts to scale BNNs to deep NNs are considerably more expensive computationally than VI-based methods and have been scaled up to low-complexity problems only, such as MNIST [47]. Conversely, non-Bayesian methods do not require the use of initial prior distributions and biases to train the models [85]. Recent works have demonstrated that non-Bayesian approaches provide better or competitive uncertainty estimates than their Bayesian counterparts [85, 96, 142]. In addition, they are scalable to complex problems and can handle millions of parameters.

MC-Dropout was proposed by Gal and Ghahramani [54] to quantify model uncertainty in NNs. They cast dropout training in deep NNs as approximate Bayesian inference in deep Gaussian processes. The method uses dropout repeatedly to select subsamples of active nodes in the network, turning a single network into an ensemble. Hence, model uncertainty is estimated by the sample variance of the ensemble predictions. MC-Dropout is not able to estimate PIs themselves, as it does not account for data noise variance. Therefore, Zhu and Laptev [202] proposed estimating PIs by quantifying the model uncertainty through MC-Dropout, coupled with estimating the data noise variance as the mean squared error (MSE) calculated over an independent held-out validation set.

Recently, several non-Bayesian approaches have been proposed for approximate uncertainty quantification. Such approaches use models whose outputs provide estimations of the predictive uncertainty directly. For instance, Schupbach *et al.* [162] proposed a method that estimates confidence intervals in NN ensembles based on the use of U-statistics. Other techniques estimate PIs by using ensembles of feedforward networks [86] or stochastic configuration networks [109] and bootstrapping. Lakshminarayanan *et al.* [96] presented an ensemble approach based on the Mean-Variance Estimation (MVE) method introduced by Nix and Weigend [137]. Here, each NN has two outputs: one that represents the mean (or target estimation) and the other that represents the variance of a normal distribution, which is used to quantify the data noise variance. Other approaches use models that generate PI bounds explicitly. Khrosavi *et al.* [85] proposed a Lower Upper Bound Estimation (LUBE) method that uses a NN and a loss function to minimize the PI width while maximizing the probability coverage using simulated annealing.

Similar approaches have attempted to optimize the LUBE loss function using methods such as genetic algorithms [201] and particle swarm optimization [56]. Pearce *et al.* [142] proposed a method called QD-Ens that consists of a quality-driven loss function similar to LUBE but that is compatible with gradient descent. Then Salem *et al.* [160] proposed QD+ which is based on QD-Ens, which uses exactly the same two penalty functions to reduce the PI width and maximize the probability coverage. They used three-output NNs and included a third penalty term that aims to decrease the mean squared error of the target predictions and a fourth penalty term to enforce the point predictions to lay inside the generated PIs. In our approach, we use only three penalty terms; the differences are explained in Section 5.2.6. Finally, both QD-Ens and QD+ used an ensemble approach to estimate the model uncertainty while we use a Monte Carlo approach on a single network.

5.1.2 Uncertainty Minimization

Adaptive sampling techniques offer a promising solution by selecting samples intelligently that contribute most to improving model accuracy and reducing uncertainty [39]. Several methods have been proposed to reduce uncertainty through iterative sampling. The majority of these methods have been developed within the framework of active learning (AL) [12, 135] or in contexts where the objective is to identify the location of local or global optima [70, 134]. Note that AS and AL fields do not completely overlap [39]. In AL, the objective is to select training data within a limited budget to maximize model performance. AL can be categorized into population-based AL, where the test input distribution is known, and pool-based AL, where a pool of unlabeled samples is provided. The problem configuration addressed in this chapter does not align with those categories as it is not limited to predefined data pools or known distributions. Instead, it samples from an open domain continuously, focusing on reducing epistemic uncertainty across the entire input space.

Our problem shares similarities with Bayesian Optimization (BO), where at each iteration, data points are sampled at locations expected to yield significant improvements in the objective function according to a specified acquisition function. BO methods build a probabilistic model of the objective function, often a GPR, to select the most promising points for evaluation [59]. Traditional BO methods explore the domain space sequentially; however, Gonzalez *et al.* [64] proposed a batch sampling strategy for BO that accounts for the interactions between different evaluations in the batch using a penalized acquisition function. Some BO strategies focus on maximizing information gain. For instance, Wang and Jegelka [187] introduced an acquisition function called max-value entropy search, which balances exploration of areas with higher uncertainty in the surrogate model and exploitation towards the believed optimum. In addition, Nguyen *et al.* [134] presented the predictive variance reduction search strategy, which reduces uncertainty at perceived optimal locations, leading to convergence when uncertainty at all perceived optimal locations is minimized.

In typical BO applications, the objective is to identify a single location that corresponds to the local or global optimum of an objective function ($\arg\max f(\mathbf{x})$). In contrast, the solution to our problem consists of an augmented dataset that yields minimum epistemic uncertainty across the entire input space. In the fertilizer rate optimization problem discussed at the beginning of the chapter, and revisited in Section 6.1.1, finding the rate that produces the higher estimated yield value does not necessarily coincide with the economic optimum nitrogen rate (EONR). The EONR is the N rate beyond which there is no actual profit for the farmers, and its calculation depends on the shape of the nitrogen-yield response curves [20]. Therefore, the epistemic uncertainty across all admissible N rates should be reduced to provide reliable EONR recommendations for future growing seasons.

Similarly, active learning is closely related to our problem. The primary distinction is that AL, given known input distributions (population-based AL) or a set of unlabeled points (pool-based AL), aims to select the minimum number of training examples to maximize model performance [39]. In contrast, our approach is agnostic of the input distribution and is not restricted to a fixed pool of training candidates. Furthermore, our focus on reducing uncertainty only considers model prediction improvement as a side-effect. What is more, it allows for repetitive sampling at a single location.

Despite the distinction above, some AL techniques can be adapted to our problem. In particular, we are interested in methods that decompose uncertainty into its aleatoric and epistemic components. A common approach is to use MC-Dropout [54] (see Section 5.2.5) to quantify epistemic uncertainty in NNs as the sample variance of the ensemble predictions.

Furthermore, Valdenegro-Toro and Mori [178] used a variance attenuation (VA) loss function to disentangle the epistemic and aleatoric components from the outputs of ensemble models. However, Zhang *et al.* [200] pointed out that VA-based methods overestimate aleatoric uncertainty. In response, they presented a denoising approach that involves incorporating a variance approximation module into a trained prediction model to identify the aleatoric uncertainty. Finally, Berry and Meger [12] proposed using an ensemble of normalizing flows (NFs), created using dropout masks, to estimate both aleatoric and epistemic uncertainty. To demonstrate their results, they suggested an AL framework that compares various uncertainty estimation methods. These methods are used to sample multiple-point candidates and select those with the highest epistemic uncertainty.

5.2 Prediction Interval-Generation Neural Networks

One of the limitations of conventional neural networks is that they only provide deterministic point estimates without any additional indication of their approximate accuracy [54]. Reliability and accuracy of the generated point predictions are affected by factors such as the sparsity of training data or target variables affected by probabilistic events [84]. Here, reliability is defined as the ability for a model to work consistently across real-world settings [173]. One way to improve the reliability and credibility of such complex models is to quantify the uncertainty in the predictions they generate [167]. This uncertainty can be quantified using PIs, which provide an estimate of the upper and the lower bounds within which a prediction will fall according to a certain probability [85]. Hence, the amount of uncertainty for each prediction is provided by the width of its corresponding PI.

Recently, some NN-based methods have been proposed to solve the PI generation problem [56, 85, 142, 160, 169, 201]. These methods train NNs using loss functions that aim to balance at least two of the following three objectives: minimizing mean PI width, maximizing PI coverage probability, and minimizing the mean error of the target predictions. Although

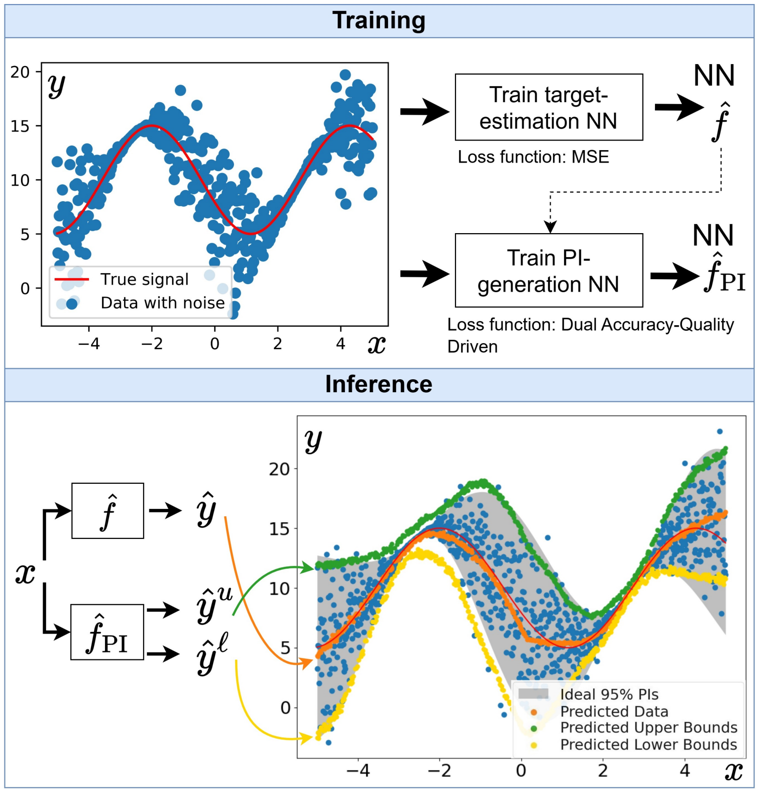


Figure 5.1: An example of our PI-generation method on a synthetic dataset [128].

the aforementioned works have achieved promising results, there exist some limitations that need to be addressed. For instance, they rely on the use of deep ensembles; however, training several models may become impractical when applied to complex systems and large datasets [57]. Furthermore, their performance is sensitive to the selection of multiple tunable hyperparameters whose values may differ substantially depending on the application. Thus, fine-tuning an ensemble of deep NNs becomes a computationally expensive task. Finally, methods that generate PI bounds and target estimations simultaneously have to deal with a trade-off between the quality of generated PIs and the accuracy of the target estimations.

Pearce *et al.* [142] coined the term *High-quality (HQ) principle*, which refers to the requirement that PIs be as narrow as possible while capturing some specified proportion of the predicted data points. Following this principle, we pose the PI generation problem for regression as a multi-objective optimization problem. This approach, which we discuss in this section, involves training two NNs: one that generates accurate target estimations and one that generates narrow PIs (see Fig. 5.1). The first NN is trained to minimize the mean squared error of the target estimations. We introduce a loss function for the second NN that, besides the generated PI bounds and the target, considers the output of the first NN as an additional input. It minimizes the mean prediction interval width and uses constraints to ensure the integrity of the generated PIs while implicitly maximizing the probability coverage. In addition, we present a method that updates the coefficient that balances the two optimization objectives of our loss function. Our method avoids generating unnecessarily wide PIs by using a technique that sorts the mini-batches at the beginning of each training epoch according to the width of the generated PIs.

5.2.1 Dual Accuracy-Quality-Driven Loss Function

Let $\mathbf{X}^b = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a training batch with N samples where each sample $\mathbf{x}_i \in \mathbb{R}^t$ consists of t covariates. Furthermore, let $\mathbf{y}^b = \{y_1, \ldots, y_N\}$ be a set of corresponding target observations where $y_i \in \mathbb{R}$. We construct a NN regression model that captures the association between $\mathbf{X}^b$ and $\mathbf{y}^b$. More specifically, $\hat{f}(\cdot)$ denotes the function computed by the NN, and $\boldsymbol{\theta}_f$ denotes its weights. Hence, given an input $\mathbf{x}_i$, $f(\mathbf{x}_i, \boldsymbol{\theta}_f)$ computes the target estimate $\hat{y}_i$. This network is trained to generate accurate estimates $\hat{y}_i$ with respect to y_i. We quantify this accuracy by calculating the mean squared error of the estimation $MSE_{est} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$. Thus, $\hat{f}$ is conventionally optimized as follows:

$$\boldsymbol{\theta}_{\hat{f}} = \underset{\boldsymbol{\theta}_{\hat{f}}}{\operatorname{argmin}}\ MSE_{est}.$$

Note that our focus is on learning prediction intervals, not on optimizing predictive performance. Therefore, to ensure a fair comparison across all experiments and methods, we consistently use simple MSE minimization, even though overall performance could potentially benefit from regularization techniques.

Once network $\hat{f}(\cdot)$ is trained, we use a separate NN whose goal is to generate prediction intervals for $\mathbf{y}^b$ given data $\mathbf{X}^b$. Let $\hat{f}_{\mathrm{PI}}(\cdot)$ denote the function computed by this PI-generation NN, and $\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}$ denotes its weights. Given an input $\mathbf{x}_i$, $\hat{f}_{\mathrm{PI}}(\mathbf{x}_i, \boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}})$ generates its corresponding upper and lower bounds, $\hat{y}_i^u$ and $\hat{y}_i^\ell$, such that $[\hat{y}_i^\ell,\, \hat{y}_i^u] = \hat{f}_{\mathrm{PI}}(\mathbf{x}_i, \boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}})$. Note that there is no assumption of $\hat{y}_i^\ell$ and $\hat{y}_i^u$ being symmetric with respect to the target estimate $\hat{y}_i$ produced by network $\hat{f}(\cdot)$. We describe its optimization procedure below.

We say that a training sample $\mathbf{x}_i \in \mathbf{X}^b$ is covered (i.e., we set $k_i = 1$) if both the predicted value $\hat{y}_i$ and the target observation y_i fall within the estimated PI:

$$k_i = \begin{cases} 1, & \text{if } \hat{y}_i^\ell < \hat{y}_i < \hat{y}_i^u \text{ and } \hat{y}_i^\ell < y_i < \hat{y}_i^u \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Then, using k_i, we define the prediction interval coverage probability ($PICP$) for $\mathbf{X}^b$ as the percent of covered training samples with respect to the batch size N: $PICP = \sum_{i=1}^{N} k_i / N$.

The HQ principle suggests that the width of the prediction intervals should be minimized as long as they capture the target observation value. Thus, Pearce *et al.* [142] considered the mean prediction interval width of captured points ($MPIW_{capt}$) as part of their loss function:

$$MPIW_{capt} = \frac{1}{\epsilon + \sum_i k_i} \sum_{i=1}^{N} (\hat{y}_i^u - \hat{y}_i^\ell)\, k_i, \tag{5.2}$$

where ϵ is a small number used to avoid dividing by zero. However, we argue that minimizing $MPIW_{capt}$ does not imply that the width of the PIs generated for the non-captured samples will not decrease along with the width of the PIs generated for the captured samples[2].

[2]We provide a toy example demonstrating this behavior in the following link https://github.com/NISL-MSU/PredictionIntervals/tree/master/src/PredictionIntervals/models/QD_toy_example.ipynb

Furthermore, consider the case where none of the samples are captured by the PIs, as likely happens at the beginning of the training. Then, the penalty is minimum (i.e., $MPIW_{capt} = 0$). Hence, the calculated gradients of the loss function forces the weights of the NN to remain in the state where $\forall i,\ k_i = 0$, which contradicts the goal of maximizing $PICP$.

Instead of minimizing $MPIW_{capt}$ directly, we let

$$PI_{pen} = \frac{1}{N}\sum_{i=1}^{N}(|\hat{y}_i^u - y_i|+|y_i - \hat{y}_i^\ell|), \tag{5.3}$$

which we minimize instead. This function quantifies the width of the PI as the sum of the distance between the upper bound and the target and the distance between the lower bound and the target. We argue that PI_{pen} is more suitable than $MPIW_{capt}$ given that it forces $\hat{y}_i^u$, y_i, and $\hat{y}_i^\ell$ to be closer together. For example, suppose that the following case is observed during the first training epoch: $y_i = 24$, $\hat{y}_i = 25$, $\hat{y}_i^u = 0.2$, and $\hat{y}_i^\ell = 0.1$. Then $MPIW_{capt} = 0$ given that the target is not covered by the PI, while $PI_{pen} = 47.7$. As a result, PI_{pen} will penalize this state while $MPIW_{capt}$ will not. Thus, we define our first optimization objective as:

$$\min_{\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}} \mathcal{L}_1 = \min_{\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}} PI_{pen}.$$

However, minimizing $\mathcal{L}_1$ is not enough to ensure the integrity of the PIs. Their integrity is given by the conditions that the upper bound must be greater than the target and the target estimate ($\hat{y}_i^u > y_i$ and $\hat{y}_i^u > \hat{y}_i$) and that the target and the target estimate, in turn, must be greater than the lower bound ($y_i > \hat{y}_i^\ell$ and $\hat{y}_i > \hat{y}_i^\ell$). Note that if the differences $(\hat{y}_i^u - y_i)$ and $(y_i - \hat{y}_i^\ell)$ are greater than the maximum estimation error within the training batch $\mathbf{X}^b$ (i.e., $(\hat{y}_i^u - y_i) > \max_i|\hat{y}_i - y_i|$ and $(\hat{y}_i^u - y_i) > \max_i|\hat{y}_i - y_i|$, $\forall i \in [1, \dots, N]$), it is implied that all samples are covered ($k_i = 1$, $\forall i \in [1, \dots, N]$).

Motivated by this, we include an additional penalty function to ensure PI integrity and maximize the number of covered samples within the batch simultaneously. Let us denote the mean differences between the PI bounds and the target estimates as $d_u = \sum_{i=1}^{N}(\hat{y}_i^u - y_i)/N$ and $d_\ell = \sum_{i=1}^{N}(y_i - \hat{y}_i^\ell)/N$. Let $\xi = \max_i|\hat{y}_i - y_i|$ denote the maximum distance between a target estimate and its corresponding target value within the batch ($\xi > 0$). From this, our penalty function is defined as follows:

$$P = e^{\xi - d_u} + e^{\xi - d_\ell}, \tag{5.4}$$

Here, if the PI integrity is not met (i.e., $d_u < 0$ or $d_\ell < 0$) then their exponent magnitude becomes larger than ξ, producing a large penalty value. Moreover, these terms encourage both d_u and d_ℓ not only to be positive but also to be greater than ξ. This implies that the distance between the target y_i and any of its bounds will be larger than the maximum error within the batch, ξ, thus the target y_i will lie within the PI. As a consequence, we define our second optimization objective as follows:

$$\min_{\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}} \mathcal{L}_2 = \min_{\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}} P.$$

Then our Dual Accuracy-Quality-Driven loss function is given by

$$Loss_{\mathrm{DualAQD}} = \mathcal{L}_1 + \lambda\,\mathcal{L}_2, \tag{5.5}$$

where λ is a self-adaptive coefficient that controls the relative importance of $\mathcal{L}_1$ and $\mathcal{L}_2$. Hence, our multi-objective optimization problem can be expressed as:

$$\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}} = \underset{\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}}{\operatorname{argmin}}\; Loss_{\mathrm{DualAQD}}.$$

For simplicity, we assume that $\hat{f}(\cdot)$ and $\hat{f}_{\mathrm{PI}}(\cdot)$ have L layers and the same network architecture except for the output layer. Network $\hat{f}(\cdot)$ is trained first. Then, weights $\boldsymbol{\theta}_{\hat{f}_{\mathrm{PI}}}$

are initialized using weights $\boldsymbol{\theta}_f$ except for those of the last layer: $\boldsymbol{\theta}^{(0)}_{\hat{f}_{\mathrm{PI}}}[1:L-1] = \boldsymbol{\theta}_{\hat{f}}[1:L-1]$. Note, that, in general, DualAQD can use different network architectures for $\hat{f}(\cdot)$ and $\hat{f}_{\mathrm{PI}}(\cdot)$. However, assuming both networks share the same architecture in the first $L-1$ layers enables the use of transfer learning, allowing us to initialize $\hat{f}$PI with the pretrained weights of $\hat{f}$ and thereby accelerate the training process.

5.2.2 Batch Sorting

Function $\mathcal{L}_2$ minimizes the term P (Eq. 5.4), forcing the distance between the target estimate and its PI bounds to be larger than the maximum absolute error within its corresponding batch. This term assumes there exists a similarity among the samples within a batch. However, consider the case depicted in Fig. 5.2 where we show four samples that have been split randomly into two batches. In Fig. 5.2a, the PIs of the second and third samples already cover their observed targets. According to $\mathcal{L}_2$, these samples will yield high penalties because the distances between their target estimates and their PI bounds are less than $\xi^{(1)}$ and $\xi^{(2)}$, respectively, forcing their widths to increase unnecessarily.

Thus, we introduced a method called "batch sorting", which consists of sorting the training samples with respect to their corresponding generated PI widths after each epoch. By doing so, the batches will process samples with similar widths, avoiding unnecessary widening. For example, in Fig. 5.2b, the penalty terms are low given that $d_u^{(1)}, d_\ell^{(1)} > \xi^{(1)}$ and $d_u^{(2)}, d_\ell^{(2)} > \xi^{(2)}$. Note that, during testing, the PI generated for a given sample is independent of other samples and, as such, batch sorting becomes unnecessary during inference.

5.2.3 Self-adaptive Coefficient

The coefficient λ of Eq. 5.5 balances the two optimization objectives $\mathcal{L}_1$ and $\mathcal{L}_2$. In this section, we propose that, instead of λ being a tunable hyperparameter with a fixed value throughout training, it should be adapted throughout the learning process automatically.

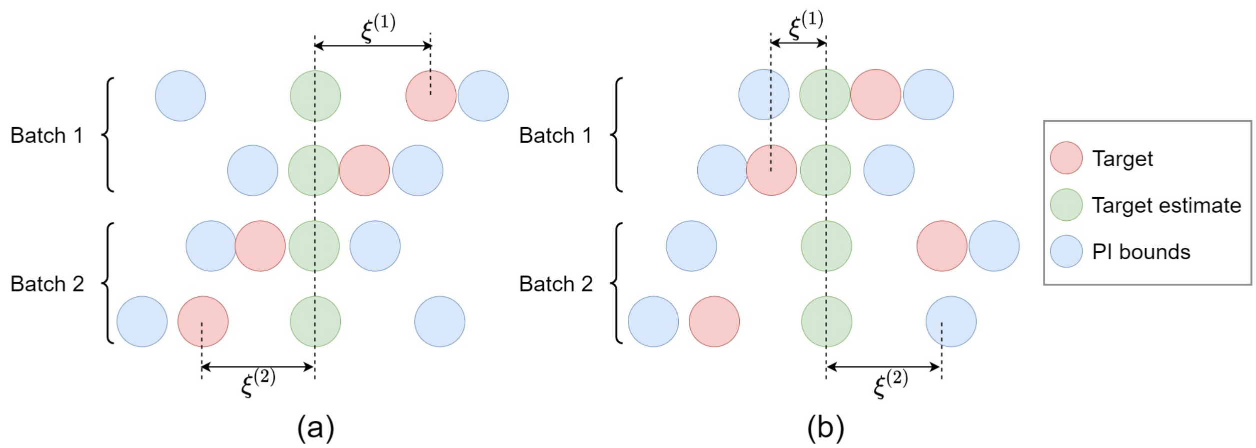


Figure 5.2: $\mathcal{L}_3$ penalty calculation, **(a)** without batch sorting; **(b)** with batch sorting [128].

Typically, the $PICP$ value improves as long as the $MPIW$ value increases; however, extremely wide PIs are not useful. We usually aim to obtain PIs with a nominal probability coverage no greater than $(1-\alpha)$. A common value for the significance level α is 0.05, in which case we say that we are 95% confident that the target value will fall within the PI.

Let $PICP_{train}^{(ep)}$ denote the $PICP$ value calculated on the training set $\mathbf{X}_{train}$ after the ep-th training epoch. If $PICP_{train}^{(ep)}$ is below the confidence target $(1-\alpha)$, more relative importance should be given to the objective $\mathcal{L}_2$ that enforces PI integrity (i.e., λ should increase). Likewise, if $PICP_{train}^{(ep)}$ is higher than $(1-\alpha)$, more relative importance should be given to the objective $\mathcal{L}_1$ that minimizes $MPIW$ (i.e., λ should decrease).

This intuition is formalized by defining the cost $\mathcal{C}$ that quantifies the distance from $PICP_{train}^{(ep)}$ to the confidence target $(1-\alpha)$: $\mathcal{C} = (1-\alpha) - PICP_{train}^{(ep)}$. Then, λ is increased or decreased proportionally to the cost function $\mathcal{C}$ after each training epoch as:

$$\lambda^{(ep)} = \lambda^{(ep-1)} + \eta \cdot \mathcal{C}, \tag{5.6}$$

where $\lambda^{(ep)}$ is the value of the coefficient λ at the ep-th iteration (we consider that $\lambda^{(0)} = 1$),

Algorithm 5.1 DualAQD method

Input: Training dataset $(\mathbf{X}_{train}, \mathbf{y}_{train})$; target prediction NN $\hat{f}$; PI-generation NN $\hat{f}_{\text{PI}}$; significance level α; adaptive coefficient η
Output: Trained PI-generation NN $\hat{f}_{\text{PI}}$

function TrainNNwithDualAQD$(\mathbf{X}_{train}, \mathbf{y}_{train}, \hat{f}, \hat{f}_{\text{PI}}, \alpha, \eta)$
    $\lambda \leftarrow 1$
    for each $t \in$ range(1, maxEpochs) **do**
        if $t > 1$ **then**
            Batches $\leftarrow$ `batchSorting`$(\mathbf{X}_{train}, \mathbf{y}_{train}, \text{widths})$
        else
            Batches $\leftarrow$ `shuffle`$(\mathbf{X}_{train}, \mathbf{y}_{train})$
        for each batch $\in$ Batches **do**
            $\mathbf{x}, y \leftarrow$ batch
            $\hat{y} \leftarrow \hat{f}(\mathbf{x})$
            $\hat{y}^u, \hat{y}^\ell \leftarrow \hat{f}_{\text{PI}}(\mathbf{x})$
            loss $\leftarrow$ `DualAQD`$(\lambda, y, \hat{y}, \hat{y}^u, \hat{y}^\ell)$
            `update`$(\hat{f}_{\text{PI}}, \text{loss})$
        $PICP^{(ep)}_{train}, \text{widths}^{(ep)} \leftarrow$ `metrics`$(\mathbf{X}_{train}, \mathbf{y}_{train})$
        $\mathcal{C} \leftarrow ((1-\alpha) - PICP^{(ep)}_{train})$ ▷ Update coefficient λ
        $\lambda = \lambda + \eta \cdot \mathcal{C}$
    return $\hat{f}_{\text{PI}}$

and η is a tunable scale factor (see Algorithm 5.1).

Note that Algorithm 5.1 takes as inputs the data $\mathbf{X}_{train}$ and corresponding targets $\mathbf{y}_{train}$ as well as the trained prediction network $\hat{f}$, the untrained network $\hat{f}_{\text{PI}}$, the significance level α, and the scale factor η. Function `batchSorting`$(\mathbf{X}_{train}, \mathbf{y}_{train}, \text{widths}^{(t-1)})$ returns a list of batches sorted according to the PI widths generated during the previous training epoch (see Section5.2.2). Function `DualAQD`$(\lambda, y, \hat{y}, \hat{y}^u, \hat{y}^\ell)$ represents the DualAQD loss function (Eq.5.5) while `update`$(\hat{f}_{\text{PI}}, loss)$ encompasses the conventional backpropagation and gradient descent processes used to update the weights of network g. Function `metrics`$(\mathbf{X}_{train}, \mathbf{y}_{train})$ passes $\mathbf{X}_{train}$ through g to generate the corresponding PIs and their widths, and to calculate compares the output to $\mathbf{y}_{train}$ to calculate the $PICP^{(ep)}_{train}$ value using $\mathbf{y}_{train}$.

5.2.4 Parameter and Hyperparameter Selection

The PI-generation NN is trained on the training set $\mathbf{X}_{train}$ during T epochs using $Loss_{\text{DualAQD}}$ as the loss function. After the *ep*-th training epoch, we calculate the performance metrics $z_{ep} = \{PICP_{val}^{(ep)}, MPIW_{val}^{(ep)}\}$ on the validation set $\mathbf{X}_{val}$, also referred to as a "calibration set" [7]. Thus, we consider that the set of optimal weights of the network, $\theta_{\hat{f}_{\text{PI}}}$, will be those that maximize performance on the validation set. The remaining question is what are the criteria to compare two solutions z_i and z_j.

Taking this criterion into account, we consider that a solution z_i dominates another solution z_j ($z_i \preceq z_j$) if the following conditions are met:

- $PICP_{val}^{(i)} > PICP_{val}^{(j)}$ and $PICP_{val}^{(i)} \leq (1-\alpha)$.
- $PICP_{val}^{(i)} == PICP_{val}^{(j)} < (1-\alpha)$ and $MPIW_{val}^{(i)} < MPIW_{val}^{(j)}$
- $PICP_{val}^{(i)} \geq (1-\alpha)$ and $MPIW_{val}^{(i)} < MPIW_{val}^{(j)}$

Hence, if $\alpha = 0.05$, we seek a solution whose $PICP_{val}$ value is at least 95%. After exceeding this value, z_i is said to dominate another solution z_j only if it produces narrower PIs.

A grid search is used to tune the hyperparameter η for training (Eq. 5.6). For each value, an NN is trained using 10-fold cross-validation, and the average performance metrics on the validation sets are calculated. Then, the hyperparameters are selected using the dominance criteria explained above.

5.2.5 PI Aggregation Using MC-Dropout

A model trained using $Loss_{\text{DualAQD}}$ generates PI estimates based on the training data; however, model uncertainty remains unaccounted for. Unlike previous work that used explicit NN ensembles [96, 142], we employ a Monte Carlo-based approach. Specifically, we use MC-Dropout [170], which consists of using dropout layers that ignore each neuron of the network according to some probability or dropout rate. Then, during each forward pass with active dropout layers, a slightly different network architecture is used and, as a result, a slightly different prediction is obtained. According to Gal and Ghahramani [54], this process can be interpreted as a Bayesian approximation of the Gaussian process.

In particular, this approach consists of performing M forward passes through the network with active dropout layers. Given an input $\mathbf{x}_i$, the estimates $\hat{y}_i^{(m)}$, $\hat{y}_i^{u(m)}$, and $\hat{y}_i^{\ell(m)}$ are obtained at the m-th iteration. Hence, the expected target estimate $\bar{y}_i$, the expected upper bound $\bar{y}_i^u$, and the expected lower bound $\bar{y}_i^\ell$ are calculated as the following averaged values: $\bar{y}_i = \frac{1}{M}\sum_{m=1}^{M} \hat{y}_i^{(m)}$, $\bar{y}_i^u = \frac{1}{M}\sum_{m=1}^{M} \hat{y}_i^{u(m)}$, $\bar{y}_i^\ell = \frac{1}{M}\sum_{m=1}^{M} \hat{y}_i^{\ell(m)}$.

5.2.6 Comparison to QD-Ens and QD+

Here we consider the differences between our method (DualAQD) and the two methods QD-Ens [142] and QD+ [160]. The loss functions used by QD-Ens and QD+ are as follows:

$$
\begin{aligned}
Loss_{QD} = & \, MPIW_{capt} + \\
& \delta \, \frac{N}{\alpha(1-\alpha)} \max(0, (1-\alpha) - PICP)^2. \\
Loss_{QD+} = & \, (1-\lambda_1)(1-\lambda_2) MPIW_{capt} + \\
& \lambda_1 (1-\lambda_2) \max(0, (1-\alpha) - PICP)^2 + \\
& \lambda_2 \, MSE_{est} + \\
& \frac{\xi}{N} \sum_{i=1}^{N} \left[\max(0, (\hat{y}_i^u - \hat{y}_i) + \max(0, (\hat{y}_i - \hat{y}_i^\ell)\right],
\end{aligned}
$$

where δ, λ_1, λ_2, and ξ are hyperparameters used by QD-Ens and QD+ to balance the learning objectives. The differences compared to our method are listed in order of importance from highest to lowest:

- QD-Ens and QD+ use objective functions that maximize $PICP$ directly aiming to a goal of $(1-\alpha)$ at the batch level. We maximize $PICP$ indirectly through $\mathcal{L}_2$, which encourages the model to produce PIs that cover as many training points as possible. This is achieved by producing PIs whose widths are larger than the maximum absolute error within each training batch. Then the optimal weights of the network are selected as those that produce a coverage probability on the validation set of at least $(1 - \alpha)$.

- $PICP$ is not directly differentiable as it involves counting the number of samples that lay within the predicted PIs; however, QD-Ens and QD+ force its differentiation by including a sigmoid operation and a softening factor (i.e., an additional hyperparameter). On the other hand, the loss functions of DualAQD are already differentiable.

- Our objective $\mathcal{L}_1$ minimizes PI_{pen}, which is a more suitable penalty than $MPIW_{capt}$.

- Our objective $\mathcal{L}_2$ maximizes $PICP$ and ensures PI integrity simultaneously. QD+ uses a truncated linear constraint and a separate function to maximize $PICP$.

- NN-based PI generation methods aim to balance three objectives: (1) accurate target prediction, (2) generation of narrow PIs, and (3) high coverage probability. QD-Ens uses a single coefficient δ within its loss function that balances objectives (2) and (3) and does not optimize objective (1) explicitly, while QD+ uses three coefficients λ_1, λ_2, and ξ to balance the three objectives. All of the coefficients are tunable hyperparameters. Our loss function, $Loss_{\mathrm{DualAQD}}$, uses a balancing coefficient whose value is not fixed but is adapted throughout the training process using a single hyperparameter (i.e., the scale factor η).

- Our approach uses two companion NNs $\hat{f}(\cdot)$ and $\hat{f}_{\mathrm{PI}}(\cdot)$ that optimize objective (1) and objectives (2) and (3), respectively, to avoid the trade-off between them. Conversely, the other approaches optimize a single NN architecture.

- We use MC-Dropout to account for model uncertainty. By doing so, we need to train only a single model instead of an ensemble of models, as in QD-Ens and QD+. Also, QD+ requires fitting a split normal density function [185] for each data point to aggregate the PIs produced by the ensemble, thus increasing the complexity of their learning process.

5.3 Adaptive Sampling with Prediction-Interval Neural Networks

Here, we examine a system defined by an input vector $\mathbf{x} \in \mathbb{R}^t$ and a scalar response $y \in \mathbb{R}$. The system's underlying function $f : \mathcal{X} \rightarrow \mathcal{Y}$ maps the input value space and the response value space such that $y = f(\mathbf{x}) + \varepsilon_a(\mathbf{x})$, where $\varepsilon_a(\mathbf{x})$ is a random variable representing the error term that is a function of the system's aleatoric uncertainty, $\sigma_a^2(\mathbf{x})$.

Let $\mathcal{D}_{it} = (\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)})$ represent the dataset at the it-th iteration with n_{it} observations, where $\mathbf{X}_{obs}^{(it)} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_{it}}\}$ and $\mathbf{y}_{obs}^{(it)} = \{y_1, \ldots, y_{n_{it}}\}$. A prediction model $\hat{f}_{it} : \mathcal{X} \rightarrow \mathcal{Y}$ with parameters $\boldsymbol{\theta}_f$ is trained by minimizing the mean squared error of the estimation:

$$\min_{\boldsymbol{\theta}_f} \frac{1}{n_{it}} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{it}} (\hat{f}_{it}(\mathbf{x}_i) - y_i)^2.$$

We aim to identify a batch $\mathbf{X}_{acq}^{(it)} = \{\mathbf{x}_{it,1}, \ldots, \mathbf{x}_{it,B}\}$ of B recommended sampling locations for the next iteration. These locations are chosen to minimize the epistemic uncertainty across the entire input space given a model $\hat{f}_{it}$ trained on $\mathcal{D}_{it}$. The epistemic uncertainty, $\sigma_e^2(\mathbf{x}_p)$, arises from the lack of knowledge about f and is due to the limitations of the prediction model trained on the observed dataset.

Preferences over potential sampling locations are encoded by an acquisition function $\alpha_{it}(\mathbf{x})$. Suppose $J(\mathcal{D}_{it})$ is a function that reflects the total potential epistemic uncertainty across the input domain. Then function $\alpha_{it}(\mathbf{x})$ is designed to reflect the expected decrease in epistemic uncertainty $\mathbb{E}[J(\mathcal{D}_{it}) - J(\mathcal{D}_{it} \cup (\mathbf{x}, y))]$ after making an observation at location $\mathbf{x}$. Fig. 5.3 depicts an instance of our problem. Here, x^* represents the selected sampling position at each iteration (i.e., $B = 1$). For the general case where $B > 1$, the decision on where to sample the k-th element of the batch, $\mathbf{x}_{it,k}$, depends on the estimated effect of the previous $k-1$ samples of the same batch. This requires the design of a batch sampling strategy, which will be explored later in this chapter.

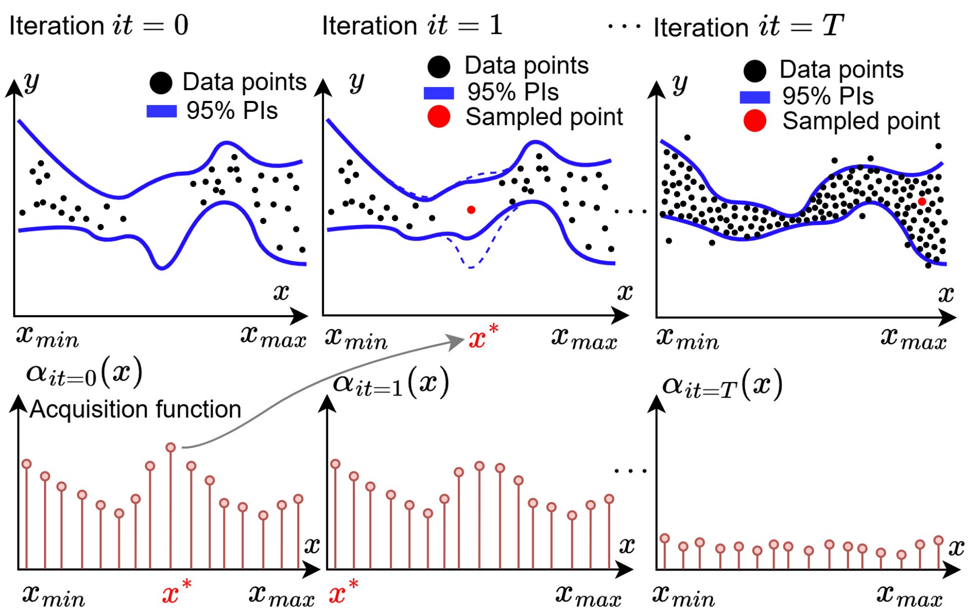


Figure 5.3: Epistemic uncertainty minimization through AS.

In the following, we describe the components of our ASPINN method. We lay out the steps to derive a metric that reflects the epistemic uncertainty associated with an input value based on PIs. The metric is then used to design an acquisition function that allows for the selection of a batch of sampling locations, which are expected to minimize the global epistemic uncertainty during the next AS iteration.

5.3.1 Prediction Interval Generation

We generate PIs for quantifying the total uncertainty associated with a given sample, thus accounting for both aleatoric and epistemic uncertainty. We employ an NN-based PI generation method called DualAQD [128], described in Section 5.2. This method uses two companion NNs: a target-estimation NN and a PI-generation NN, whose computed functions are here denoted as $\hat{f}_{it}(\cdot)$ and $\hat{f}_{\text{PI},\,it}(\cdot)$, respectively. Network $\hat{f}_{it}(\cdot)$ is trained on $\mathcal{D}_{it}$ to minimize the estimation error so that $\hat{y} = \hat{f}_{it}(\mathbf{x})$ and $\hat{y} \approx y$. Network $\hat{f}_{\text{PI},\,it}(\cdot)$ produces two outputs $[\hat{y}^{\ell},\, \hat{y}^{u}] = \hat{f}_{\mathrm{PI},\,it}(\mathbf{x})$, which correspond to the PI lower and upper bounds. Note that $\hat{f}_{\mathrm{PI},\,it}(\mathbf{x})$ makes no assumptions about the underlying uncertainty distribution.

Network $\hat{f}_{\mathrm{PI},\,it}(\cdot)$ is trained using the DualAQD loss function to produce high-quality PIs that are as narrow as possible while capturing some specified proportion of the predicted data points (e.g., 95%). However, the model should produce wider PIs for out-of-distribution (OOD) samples since these samples are not well-represented in the training set, leading to higher associated epistemic uncertainty. To address this, the bias weights of $\hat{f}_{\mathrm{PI},\,it}(\cdot)$ are initialized to generate wide PIs, similar to the approach proposed by Liu *et al.* [107]. The rationale is that these bias weights will decrease during training for in-distribution samples, resulting in narrower PIs, but will remain high for OOD samples, ensuring appropriately wider PIs to reflect the increased uncertainty.

5.3.2 Potential Epistemic Uncertainty

Let $\sigma_e^2(\mathbf{x}_p)$ represent the epistemic uncertainty at a certain location $\mathbf{x}_p \in \mathcal{X}$. The PI lower and upper bounds generated by the network $\hat{f}_{\mathrm{PI},\,it}(\cdot)$ at $\mathbf{x}_p$ are denoted as $\hat{y}^{\ell}_{it}(\mathbf{x}_p)$ and $\hat{y}^{u}_{it}(\mathbf{x}_p)$, respectively. We claim that using PIs alone does not provide sufficient information to determine $\sigma_e^2(\mathbf{x}_p)$. Consider $\mathbf{x}_p$ as an OOD sample. We may state that the total uncertainty associated with $\mathbf{x}_p$ is primarily due to epistemic uncertainty given the lack of knowledge of the prediction model about the system's behavior in this region of the input domain. Nevertheless, we cannot estimate the aleatoric uncertainty around $\mathbf{x}_p$ until we gather observations in such a domain region. Alternative methods can be used but they require making assumptions about the noise distribution [163], training an ensemble of models [12], or using additional trainable modules [200]. As a consequence, the total uncertainty conveyed by the generated interval $[\hat{y}^{\ell}_{it}(\mathbf{x}_p), \hat{y}^{u}_{it}(\mathbf{x}_p)]$ cannot be split effectively into its epistemic and aleatoric components without further information.

Instead of attempting to provide a metric that accurately estimates $\sigma_e^2(\mathbf{x}_p)$, we introduce a metric that reflects the potential levels of epistemic uncertainty. Let $\mathcal{N}(\mathbf{x}_p) = \{\mathbf{x} \in \mathbf{X}_{obs}^{(it)} \mid \|\mathbf{x} - \mathbf{x}_p\|_2 \leq \theta\}$ denote a neighborhood that considers all samples whose Euclidian distance to $\mathbf{x}_p$ is less than a hyperparameter threshold θ. The set of pairs $\mathcal{R}(\mathcal{N}(\mathbf{x}_p)) = \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in \mathcal{D}_{it}, \mathbf{x} \in \mathcal{N}(\mathbf{x}_p), \hat{y}^\ell(\mathbf{x}) \leq y \leq \hat{y}^u(\mathbf{x})\}$ is created using the samples in $\mathcal{N}(\mathbf{x}_p)$ whose response values fall within their corresponding PI. Thus, we present the metric $Q_{it}(\mathbf{x}_p)$:

$$Q_{it}(\mathbf{x}_p) = \begin{cases} \min\limits_{(\mathbf{x},y)\in\mathcal{R}(\mathcal{N}(\mathbf{x}_p))} (\hat{y}^u(\mathbf{x}) - y) + \min\limits_{(\mathbf{x},y)\in\mathcal{R}(\mathcal{N}(\mathbf{x}_p))} (y - \hat{y}^\ell(\mathbf{x})) & \text{if } \mathcal{N}(\mathbf{x}_p) \neq \emptyset \\ \hat{y}_{it}^u(\mathbf{x}_p) - \hat{y}_{it}^\ell(\mathbf{x}_p) & \text{if } \mathcal{N}(\mathbf{x}_p) = \emptyset \end{cases} \tag{5.7}$$

The local neighborhood of $\mathbf{x}_p$ may contain important contextual information that an analysis at a single location $\mathbf{x}_p$ cannot capture. For instance, Fig.5.4a illustrates an interval $\text{PI}(\mathbf{x}_p) = [\hat{y}_{it}^\ell(\mathbf{x}_p), \hat{y}_{it}^u(\mathbf{x}_p)]$ generated at a single location. Suppose $Q_{it}(\mathbf{x}_p)$ is calculated using $\text{PI}(\mathbf{x}_p)$ only (i.e., $\theta = 0$). Since a single point lies within the interval, $Q_{it}(\mathbf{x}_p)$ is equal to the PI width, indicating that the epistemic uncertainty at $\mathbf{x}_p$ can potentially be completely reduced. Fig.5.4b depicts a case in which the PI shown in Fig.5.4a is located in a region of the domain with low data density. As such, there exists an epistemic component that entails that the PI width could be reduced by acquiring more data in this region.

Conversely, Fig.5.4c shows a similar PI in a high data density context. Here, a reduction in $\text{PI}(\mathbf{x}_p)$ will also lead to a decrease in the PI widths of adjacent locations, provided that the uncertainty at $\mathbf{x}_p$ is not independent of its surroundings. However, model $\hat{f}_{\text{PI},\,it}(\cdot)$ is trained to produce narrow PIs while maintaining a nominal coverage (e.g., 95%). Thus, it will not reduce $\text{PI}(\mathbf{x}_p)$ if this reduction would result in several samples near the PI bounds being excluded from their intervals. Notice that if $\theta > 0$, then $Q_{it}(\mathbf{x}_p) \approx 0$, indicating minimal potential epistemic uncertainty around $\mathbf{x}_p$.

Algorithm 5.2 describes the function `PotEpistUnc` that calculates the potential

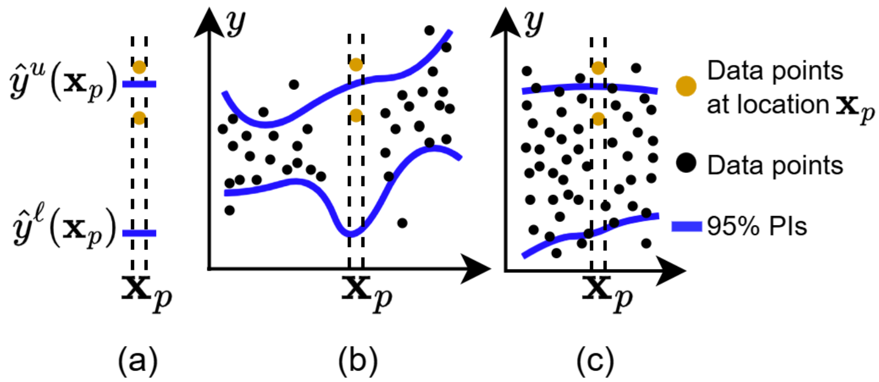


Figure 5.4: PIs generated at location $\mathbf{x}_p$. **(a)** Data points located at $\mathbf{x}_p$ only. **(b)** PI width is affected by epistemic uncertainty. **(b)** PI width is mainly due to aleatoric uncertainty.

epistemic uncertainty at each candidate position for sampling. It takes as inputs the dataset $\mathcal{D}_{it} = (\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)})$ available during the it-th iteration of the AS process, the PI-generation NN $\hat{f}_{\mathrm{PI},\,it}(\cdot)$ trained on $\mathcal{D}_{it}$, the set $\mathbf{X}_{test}$ of all candidate positions for sampling (i.e., the input space), and the neighbor distance threshold θ. For each candidate position $\mathbf{x}_p$, the algorithm constructs a neighborhood $\mathcal{N}(\mathbf{x}_p)$ consisting of all samples in $\mathbf{X}_{obs}^{(it)}$ within a radius of θ with respect to $\mathbf{x}_p$. If $\mathcal{N}(\mathbf{x}_p)$ is empty, the potential epistemic uncertainty is given by the PI width $\hat{y}_{it}^{u}(\mathbf{x}_p) - \hat{y}_{it}^{\ell}(\mathbf{x}_p)$. Otherwise, it builds the set of input–response pairs $\mathcal{R}(\mathcal{N}(\mathbf{x}_p))$ using the samples in $\mathcal{N}(\mathbf{x}_p)$ whose response values fall within their corresponding PI. From the data points in $\mathcal{R}(\mathcal{N}(\mathbf{x}_p))$, the potential epistemic uncertainty $Q_{it}(\mathbf{x}_p)$ is calculated as the sum of the minimum distance between the predicted upper bounds and the observed values, and the minimum distance between the observed values and the predicted lower bounds.

5.3.3 Batch Sampling

When multiple locations are sampled at each iteration, decisions for the entire batch are made based on the current model without observing any data from the batch until the next iteration. Hence, it is necessary to simulate the decisions that would be made under the equivalent sequential policy (i.e., when $B = 1$) [64]. In other words, the decision of selecting

Algorithm 5.2 ASPINN's potential epistemic uncertainty

Input: Dataset during the i-th iteration $\mathcal{D}_{it}$; PI-generation NN $\hat{f}_{\mathrm{PI},\,it}$; set of candidate positions for sampling $\mathbf{X}_{test}$; radius θ
Output: Potential epistemic uncertainty Q_{it}

function POTEPISTUNC($\mathcal{D}_{it}, \hat{f}_{\mathrm{PI},\,it}, \mathbf{X}_{test}, \theta$)
  $\left(\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}\right) \leftarrow \mathcal{D}_{it}$
  $Q_{it} \leftarrow \mathrm{zeros}(\mathrm{size}(\mathbf{X}_{test}))$
  for $\mathbf{x}_p \in \mathbf{X}_{test}$ **do**
    $\mathcal{N}(\mathbf{x}_p) \leftarrow \{\mathbf{x} \in \mathbf{X}_{obs}^{(it)} \mid \|\mathbf{x} - \mathbf{x}_p\|_2 \leq \theta\}$
    if $\mathcal{N}(\mathbf{x}_p) \neq \emptyset$ **then**
      $\mathcal{R}(\mathcal{N}(\mathbf{x}_p)) \leftarrow [\,]$ ▷ $\mathbf{x}_p$'s neighbors falling within the PIs
      $Y_{\mathrm{sub}}^{u}, Y_{\mathrm{sub}}^{\ell} \leftarrow [\,], [\,]$
      for $\mathbf{x} \in \mathcal{N}(\mathbf{x}_p)$ **do**
        $\hat{y}^{\ell}(\mathbf{x}), \hat{y}^{u}(\mathbf{x}) \leftarrow g_{it}(\mathbf{x})$
        if $\hat{y}^{\ell}(\mathbf{x}) \leq y \leq \hat{y}^{u}(\mathbf{x})$ **then** ▷ $(\mathbf{x}, y) \in \mathcal{D}_{it}$
          $\mathcal{R}(\mathcal{N}(\mathbf{x}_p)).\mathtt{append}((\mathbf{x}, y))$
          $Y_{\mathrm{sub}}^{u}.\mathtt{append}(\hat{y}^{u}(\mathbf{x}))$
          $Y_{\mathrm{sub}}^{\ell}.\mathtt{append}\big(\hat{y}^{\ell}(\mathbf{x})\big)$
      $(X_{\mathrm{sub}}, Y_{\mathrm{sub}}) \leftarrow \mathcal{R}(\mathcal{N}(\mathbf{x}_p))$
      $Q_{it}(\mathbf{x}_p) \leftarrow \min(Y_{\mathrm{sub}}^{u} - Y_{\mathrm{sub}}) + \min(Y_{\mathrm{sub}} - Y_{\mathrm{sub}}^{\ell})$
    else
      $\hat{y}^{\ell}(\mathbf{x}_p), \hat{y}^{u}(\mathbf{x}_p) \leftarrow g_{it}(\mathbf{x}_p)$
      $Q_{it}(\mathbf{x}_p) \leftarrow \hat{y}_{it}^{u}(\mathbf{x}_p) - \hat{y}_{it}^{\ell}(\mathbf{x}_p)$
  return Q_{it}

the k-th element of the it-th batch, $\mathbf{x}_{it,k}$, should incorporate the estimates of change in uncertainty after sampling at the previous locations $\mathbf{x}_{it,1:k-1} = \{\mathbf{x}_{it,1}, \ldots, \mathbf{x}_{it,k-1}\}$. In that case, following a greedy sampling strategy, we have:

$$\mathbf{x}_{it,k} = \underset{x_p \in X}{\mathrm{argmax}}\ \alpha_{it}(\mathbf{x}_p \,|\, \mathbf{x}_{it,1:k-1}). \tag{5.8}$$

We consider an acquisition function that estimates the reduction in the total potential epistemic uncertainty across the domain when making an observation at a given location $\mathbf{x}_p$:

$$\alpha_{it}(\mathbf{x}_p \,|\, \mathbf{x}_{it,1:k-1}) = J\left(\mathcal{D}_{it,k-1}\right) - J(\mathcal{D}_{it,k-1} \cup (\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p))).$$

$\mathcal{D}_{it,k-1}$ is the dataset $\mathcal{D}_{it}$ augmented with the first $k-1$ samples of the batch and their corresponding estimated response values. The potential epistemic uncertainty at $\mathbf{x}$ during the it-iteration after sampling the first k elements of the batch is denoted as $Q_{it,k}(\mathbf{x})$. Thus, the total potential epistemic uncertainty is calculated as $J(\mathcal{D}_{it,k}) = \sum_{\mathbf{x}\in\mathcal{X}} Q_{it,k}(\mathbf{x})$, where $J(\mathcal{D}_{it,0}) = J(\mathcal{D}_{it})$ and $Q_{it,0}(\mathbf{x}) = Q_{it}(\mathbf{x})$.

Thus, $J(\mathcal{D}_{it})$ is computed based on $Q_{it}(\mathbf{x})$, which is derived from the outputs produced by NNs $\hat{f}_{it}(\cdot)$ and $\hat{f}_{\mathrm{PI},\,it}(\cdot)$ (Eq. 5.7), trained on $\mathcal{D}_{it}$. To calculate $J(\mathcal{D}_{it,k-1} \cup (\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p)))$ in a similar manner, it is necessary to train both NNs on the augmented dataset $\mathcal{D}_{it,k-1} \cup (\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p))$. According to Eq. 5.8, this operation would need to be repeated $\forall \mathbf{x}_p \in \mathcal{X}$ and $\forall k \in [1, \ldots, B]$ and, as such, becomes impractical. Therefore, motivated by most BO-based approaches, we use a GPR as a surrogate model. The objective is to simulate, with low computational cost, how the potential epistemic uncertainty would be affected throughout the entire domain after observing a sample at a given position.

Let us define a GPR $p(\hat{f}_{it}) = \mathcal{GP}(\mu_{it}, \mathbf{K}_{it})$ that serves as a surrogate model for $\hat{f}_{it}(\cdot)$ and its associated epistemic uncertainty during the it-th iteration. This GPR is characterized by the mean function μ_{it} and the positive-definite covariance matrix $\mathbf{K}_{it}$. Functions μ_{it} and $\mathbf{K}_{it}$ are initialized based on the estimations generated by $\hat{f}_{it}(\cdot)$ and $\hat{f}_{\mathrm{PI},\,it}(\cdot)$, trained on $\mathcal{D}_{it}$.

For the mean function, we consider $\mu_{it}(\mathbf{x}) = \hat{f}_{it}(\mathbf{x})$. On the other hand, the diagonal elements of $\mathbf{K}_{it}$ reflect the uncertainty in the predictions $\hat{f}_{it}(\mathbf{x})$ due to epistemic uncertainty. Since this uncertainty varies across the domain, it represents heteroscedastic noise. Considering that the uncertainty at a given position may be correlated with nearby positions, $\mathbf{K}_{it}$ is structured as a matrix with non-zero off-diagonal elements. Thus, the scale of $\mathbf{K}_{it}$ depends on location and is calculated according to the potential epistemic uncertainty:

$$\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}') = \begin{cases} Q_{it}(\mathbf{x}), & \text{if } \mathbf{x} = \mathbf{x}' \\ \rho(\mathbf{x}, \mathbf{x}')\sqrt{Q_{it}(\mathbf{x}) Q_{it}(\mathbf{x}')}, & \text{otherwise,} \end{cases}$$

where $\rho(\mathbf{x}, \mathbf{x}')$ indicates the correlation between positions $\mathbf{x}$ and $\mathbf{x}'$. We use the radial basis function (RBF) such that $\rho(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2r^2}}$, where r is a tunable hyperparameter.

Given we want to assess the impact of observing a data point at a given position $\mathbf{x}_p$, we condition the GPR on the data point $(\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p))$, resulting in a GPR posterior $p(\hat{f}_{it}|(\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p)))$ whose covariance matrix is denoted as $\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}' \,|\, \mathbf{x}_p)$. In general, the covariance matrix when sampling the k-th element of the batch is denoted as $\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}' \,|\, \mathbf{x}_{it,1}, \ldots, \mathbf{x}_{it,k})$ and $Q_{it,k} = \mathrm{diag}\left(\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}' \,|\, \mathbf{x}_{it,1}, \ldots, \mathbf{x}_{it,k})\right)$.

Given $\mathbf{x}_p$, the covariance matrix is updated as follows:

$$\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}' \,|\, \mathbf{x}_p) = \mathbf{K}_{it}(\mathbf{x}, \mathbf{x}') - \mathbf{K}_{it}(\mathbf{x}, \mathbf{x}_p)\mathbf{K}_{it}(\mathbf{x}_p, \mathbf{x}_p)^{-1}\mathbf{K}_{it}(\mathbf{x}_p, \mathbf{x}').$$

Hence, the updated GPR variance at $\mathbf{x}_p$ collapses to zero after observing a data point at that position. Note that this would only happen when $Q_{it}(\mathbf{x}_p)$ reflects the level of epistemic uncertainty exclusively. In practice, this assumption may not hold. Nevertheless, it allows us to construct a heuristic that guides the search toward locations where new observations would potentially cause the greatest uncertainty reduction. The next sampling location is selected using Eq. 5.8 based on the total potential epistemic uncertainty after observing a data point at $\mathbf{x}_p$, which is given by:

$$J\left(\mathcal{D}_{it} \cup (\mathbf{x}_p, \hat{f}_{it}(\mathbf{x}_p))\right) = \sum \mathrm{diag}\left(\mathbf{K}_{it}(\mathbf{x}, \mathbf{x}' \,|\, \mathbf{x}_p)\right).$$

ASPINN's batch sampling strategy is shown in Algorithm 5.3. It takes as inputs the set $\mathbf{X}_{test}$ of all candidate positions for sampling, their corresponding potential epistemic uncertainty values Q_{it} during the it-th iteration of the AS process, the batch size B, and the kernel length r. In Lines 3–11, the algorithm initializes the covariance matrix $\mathbf{K}_{it}$ of a GPR surrogate model. The diagonal of $\mathbf{K}_{it}$ is set to be equal to Q_{it}. Since the uncertainty at a given position may be correlated with nearby positions, $\mathbf{K}_{it}$ is structured as a matrix with

Algorithm 5.3 ASPINN's batch sampling method

Input: Set of candidate positions for sampling $\mathbf{X}_{test}$; potential epistemic uncertainty Q_{it}; batch size B; kernel bandwidth r

Output: Recommended sampling locations $\mathbf{X}_{acq}^{(it)}$

1: **function** SAMPLE($\mathbf{X}_{test}, Q_{it}, B, r$)
2: $\quad n_Q \leftarrow \text{size}(Q_{it})$
3: $\quad \mathbf{K}_{it} \leftarrow \text{zeros}(n_Q, n_Q)$ ▷ Init GP's covariance matrix
4: $\quad$ **for** $i \in (0, n_Q)$ **do**
5: $\quad\quad$ **for** $j \in (i, n_Q)$ **do**
6: $\quad\quad\quad$ **if** $i = j$ **then**
7: $\quad\quad\quad\quad k \leftarrow Q_{it}(\mathbf{X}_{test}[i])$
8: $\quad\quad\quad$ **else**
9: $\quad\quad\quad\quad \rho \leftarrow \text{RBF}(\mathbf{X}_{test}[i], \mathbf{X}_{test}[j]; r)$ ▷ r: kernel size
10: $\quad\quad\quad\quad k \leftarrow \rho\sqrt{Q_{it}(\mathbf{X}_{test}[i])Q_{it}(\mathbf{X}_{test}[j])}$
11: $\quad\quad\quad \mathbf{K}_{it}(i, j) = \mathbf{K}_{it}(k, i) = k$
12: $\quad \mathbf{X}_{acq}^{(it)} \leftarrow [\,]$
13: $\quad$ **while** $\text{size}(\mathbf{X}_{acq}^{(it)}) < B$ **do** ▷ Batch sampling loop
14: $\quad\quad \Delta J_{\max} \leftarrow [\,]$
15: $\quad\quad$ **for** $\mathbf{x}_p \in \mathbf{X}_{test}$ **do**
16: $\quad\quad\quad \mathbf{K}'_{it} \leftarrow \mathbf{K}_{it}(\mathbf{x}, \mathbf{x}') - \mathbf{K}_{it}(\mathbf{x}, \mathbf{x}_p)\mathbf{K}_{it}(\mathbf{x}_p, \mathbf{x}_p)^{-1}$
17: $\quad\quad\quad\quad \mathbf{K}_{it}(\mathbf{x}_p, \mathbf{x}')$ ▷ $\forall \mathbf{K}_{it}(\mathbf{x}, \mathbf{x}') \in \mathbf{K}_{it}$
18: $\quad\quad\quad \Delta J \leftarrow \sum \text{diag}(\mathbf{K}) - \sum \text{diag}(\mathbf{K}')$
19: $\quad\quad\quad$ **if** $\Delta J > \Delta J_{\max}$ **then**
20: $\quad\quad\quad\quad \Delta J_{\max} \leftarrow \Delta J$
21: $\quad\quad\quad\quad \mathbf{x}_{t,k} \leftarrow \mathbf{x}_p$
22: $\quad\quad\quad\quad \mathbf{K}_{\text{best}} \leftarrow \mathbf{K}'_{it}$
23: $\quad\quad \mathbf{X}_{acq}^{(it)}$.`append`$(\mathbf{x}_{t,k})$
24: $\quad\quad \mathbf{K}_{it} \leftarrow \mathbf{K}'_{it}$
25: $\quad$ **return** $\mathbf{X}_{acq}^{(it)}$

non-zero off-diagonal elements. The off-diagonal elements combine the potential epistemic uncertainty values at different positions based on their correlation value, which is calculated using a radial basis function (RBF) with kernel bandwidth r.

Once $\mathbf{K}_{it}$ is initialized, we assess the potential uncertainty reduction when observing each candidate position $\mathbf{x}_p$. Specifically, we condition the GPR on a data point at $\mathbf{x}_p$ and update its covariance matrix as shown in Line 16. The total potential epistemic uncertainty across the domain, $J(\mathcal{D}_{it})$, is determined by summing the diagonal elements $\mathbf{K}_{it}$. The estimated reduction in the total potential epistemic uncertainty, when making an observation at $\mathbf{x}_p$, is thus calculated as the difference δJ between the sum of the diagonal elements of $\mathbf{K}_{it}$ before and after the observation (Line 18). Therefore, the k-th element of the it-th batch, $\mathbf{x}_{t,k}$, is selected as the position that yields the greatest δJ value.

5.4 Integrating Symbolic Regression into Adaptive Sampling

In this section, we incorporate the symbolic regression technique introduced in Chapter 4, SeTGAP, into the adaptive sampling process to investigate its effectiveness in data-scarce scenarios. We aim to evaluate how the learned mathematical expressions evolve as the sampling strategy reduces epistemic uncertainty in the input domain. By iteratively refining the dataset through uncertainty-aware sampling, we examine the impact of improved data coverage on the accuracy and stability of the discovered symbolic models.

Recall that, at each AS iteration, we have access to the dataset $\mathcal{D}_{it} = \left(\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}\right)$ and the corresponding trained model $\hat{f}_{it}$. Given that $\mathbf{X}_{test}$ defines a fixed input domain grid, the model prediction $\hat{\mathbf{y}}_{test}^{(it)} = \hat{f}_{it}(\mathbf{X}_{test})$ represents the estimated response across that domain. To derive a mathematical interpretation of the function computed by $\hat{f}_{it}$ in the form of a mathematical expression, we analyze the set $\hat{\mathcal{D}}_{it} = \left(\mathbf{X}_{test}, \hat{\mathbf{y}}_{test}^{(it)}\right)$, which pairs the input domain with the model's estimated response.

SeTGAP was originally designed to solve the multivariate SR problem in a decomposable way. When analyzing a specific variable, the approach generates multiple input–response pairs by fixing the remaining system variables to randomly sampled values. Each resulting set enables examining the relationship between the variable of interest and the system's response under varying contextual conditions. However, the experiments in this section focus on a one-dimensional setting. In such cases, a symbolic skeleton prediction method [13, 15, 149, 179] can be used to infer a univariate expression from a single input—response pair, capturing the relationship between $\mathbf{X}_{test}$ and $\hat{\mathbf{y}}_{test}^{(it)}$. Alternatively, we can replicate the set $\hat{\mathcal{D}}_{it}$ N_S times to form a collection of sets suitable for processing by our Multi-Set Transformer $g(\cdot)$.

Nevertheless, to leverage SeTGAP's multi-set symbolic skeleton prediction capabilities, we adopt a different strategy. Consider the curve shown in Fig. 5.5, generated from the function $f(x) = \frac{1}{\sin(x^2)+5}$. Rather than simply replicating $\hat{\mathcal{D}}_{it}$ N_S times (with $N_S = 4$ in this example), we construct a collection of distinct sets $\mathbf{D} = \left\{\mathbf{D}^{(1)}, \dots, \mathbf{D}^{(N_S)}\right\}$ by randomly sampling subsets from the original input domain. Since each set in $\mathbf{D}$ is derived from the same original curve, they all share a common symbolic skeleton. The task of the MSSP solver is then to recover a skeleton $\hat{\mathbf{e}}$ that approximates this shared structure.

This strategy introduces diversity and contextual variation into the SR problem, which helps mitigate the impact of localized uncertainty. For instance, certain regions may exhibit distortions due to limited observations, as it likely happens at the boundaries of the input domain, or increased noise, leading traditional approaches to overfit by generating unnecessarily complex functions that model these artifacts. In contrast, by sampling distinct subregions and seeking a shared symbolic structure across them, the MSSP framework emphasizes the recovery of the underlying functional form rather than the noise-driven anomalies of specific regions. This approach is most effective when the input domain is broad enough to exhibit meaningful variation across subregions. That is, in cases with limited domain coverage or low variability, alternative techniques may be more appropriate.

Algorithm 5.4 presents a variant of Algorithm 4.1, originally developed to extract univariate skeletons in multivariate settings. Function `generate1DExpr` takes as input the datasets $\mathcal{D}_{it}$ and $\hat{\mathcal{D}}_{it}$, the Multi-Set Transformer g, the number of input sets N_S, the number of points per set n, and two hyperparameters, n_B and n_{cand}, which control the number of skeleton candidates generated. On line 6, the function `getRandomRange` randomly selects a portion of the input domain, between 80% and 100%, to construct the i-th input set. Function `selectAndInterpolate` is then used to extract the subset of data points in $\hat{\mathcal{D}}_{it}$ that

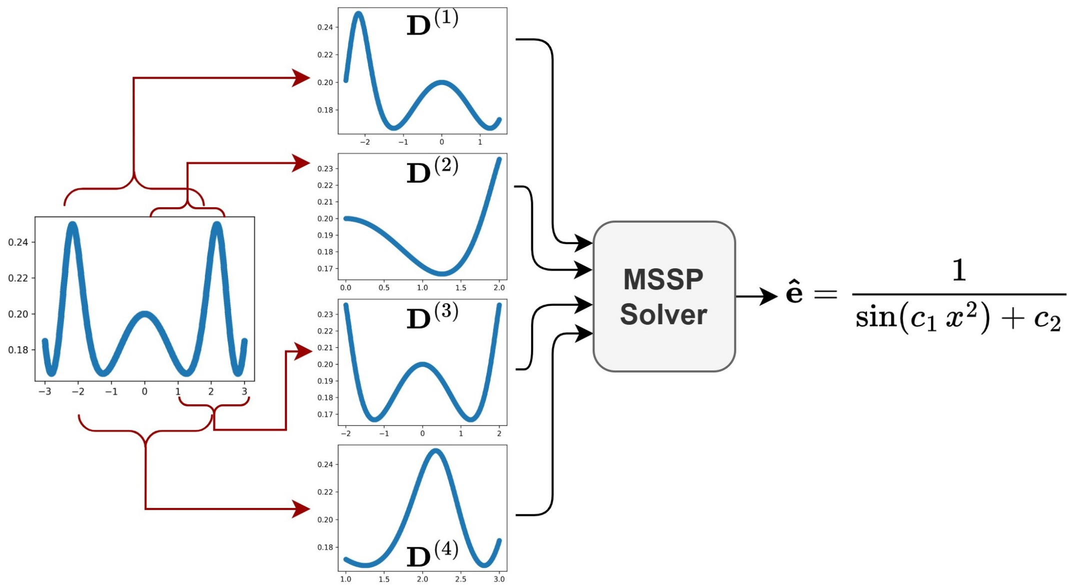


Figure 5.5: MSSP example for solving a 1-D problem.

lie within the selected input range. If the number of points in this region is insufficient, linear interpolation is applied to resample the data and produce exactly n points. The multiple generated sets allow us to construct the collection $\mathbf{D}$, which is processed by the model g to generate n_B outputs. This procedure is repeated n_{cand} times to obtain a diverse set of candidate skeletons. After generating n_{cand} sets of skeletons, the coefficients of each candidate are fitted to the observed data $\mathcal{D}_{it}$ using the GA-based optimizer `fitCoefficients`, which minimizes the MSE between the fitted expression and the target values. The final output is the fitted expression $\tilde{f}$ with the lowest error.

5.5 Experimental Results

This section presents experimental results organized into four main parts. First, we evaluate the performance of the proposed DualAQD method in generating high-quality prediction intervals. Second, we introduce the synthetic problems used for adaptive sampling, which serve as the basis for the experiments in the following sections. Third, we assess

Algorithm 5.4 MSSP applied to 1-D problems

Input: Dataset during the i-th iteration $\mathcal{D}_{it} = (\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)})$; input domain grid $\mathbf{X}_{test}$ and its estimated response $\hat{\mathbf{y}}_{test}^{(it)}$; Multi-Set Transformer g; number of input sets N_S; number of skeleton candidates n_{cand}; beam size n_B

Output: Estimated expression $\tilde{f}(\mathbf{x})$

function GENERATE1DEXPR($\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}, \mathbf{X}_{test}, \hat{\mathbf{y}}_{test}^{(it)}, g, N_s, n, n_{\text{cand}}, n_B$)
    genSks $\leftarrow$ [ ]
    for each $i \in (1, n_{\text{cand}})$ **do**
        $\mathbf{D} \leftarrow [\,]$
        for each $i \in (1, n_{\text{cand}})$ **do**
            range $\leftarrow$ `getRandomRange`($\mathbf{X}_{test}$)
            $\mathbf{X}^{(s)}, \mathbf{y}^{(s)} \leftarrow$ `selectAndInterpolate`($\mathbf{X}_{test}, \hat{\mathbf{y}}_{test}^{(it)}$, range, n)
            $\mathbf{D}$.`append` $\left((\mathbf{X}^{(s)}, \mathbf{y}^{(s)})\right)$
        genSks.`append`($g(\mathbf{D}, \Theta; n_B)$)
    genSks $\leftarrow$ removeDuplicates(genSks)
    MSEvals, genExps $\leftarrow$ zeros(|genSks|), zeros(|genSks|)
    for each $k \in (1, n_{\text{cand}})$ **do**
        MSEvals[k], genExps[k] $\leftarrow$ `fitCoefficients` $\left(\hat{\mathbf{e}}_k(\mathbf{x}), \left(\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}\right)\right)$ ▷ Minimize MSE
    $\tilde{f}(\mathbf{x}) \leftarrow$ `getTopExpr`(genExps, MSEvals)
    return $\tilde{f}(\mathbf{x})$

the effectiveness of our ASPINN method in adaptive sampling scenarios aimed at reducing epistemic uncertainty. Finally, we analyze how applying ASPINN and reducing uncertainty in the collected datasets influence the quality of the symbolic expressions learned by SeTGAP.

5.5.1 Prediction-Interval Learning

We present experiments organized into two parts. The first involves a synthetic dataset designed with high fluctuations and extreme uncertainty, enabling analytical computation of ideal PIs for direct comparison with those produced by the evaluated PI-generation methods. The second part includes experiments on eight open-access benchmark datasets.

5.5.1.1 Experiments with Synthetic Data To illustrate the behavior of DualAQD and compare it against alternative PI-generation methods, we designed a challenging synthetic dataset with heteroscedastic noise. Specifically, consider a system with response $y = f(x) + \varepsilon_a$, where the underlying function is $f(x) = 5\cos(x) + 10$, and ε_a denotes a heteroscedastic aleatoric noise term defined as $\varepsilon_a = (2\cos(1.2\,x) + 2)\,v$ with $v \sim \mathcal{N}(0,1)$. Using these parameters, a dataset of 1000 points was generated.

Knowing the probability distribution of the noise at each position x allows us to calculate the ideal 95% PIs ($\alpha = 0.05$), $[y^u, y^\ell]$, as follows:

$$y^u(x) = y(x) + 1.96\,\varepsilon_a, \text{ and } y^\ell(x) = y(x) - 1.96\,\varepsilon_a,$$

where 1.96 is the approximate value of the 95% confidence interval of the normal distribution. With this, a new metric, PI_δ, was defined by summing the absolute differences between the estimated bounds and the ideal 95% bounds for all the samples within a set $\mathbf{X}$:

$$PI_\delta = \frac{1}{|\mathbf{X}|} \sum_{x \in \mathbf{X}} \left(|y^u(x) - \hat{y}^u(x)| + |y^\ell(x) - \hat{y}^\ell(x)| \right). \tag{5.9}$$

We compared the performance of DualAQD using batch sorting and without using batch sorting (denoted as "DualAQD_noBS" in Table 5.1). All networks were trained using a fixed mini-batch size of 16 and the Adadelta optimizer. Table 5.1 gives the average performance for the metrics calculated on the validation sets, MSE_{val}, $MPIW_{val}$, $PICP_{val}$, and $PI_{\delta val}$, and corresponding standard deviations. The DualAQD PI generation methodology was also compared to three other NN-based methods: QD+ [160], QD-Ens [142], and a PI generation method based on MC-Dropout alone [202] (denoted MC-Dropout-PI). For the sake of consistency and fairness, the same configuration was used for all the networks trained in these experiments (i.e., network architecture, optimizer, and batch size). For the case of QD+, QD-Ens, and MC-Dropout-PI, it was found that batch sorting either helped to improve their performance or there was no significant change. Thus, for the sake of fairness

Table 5.1: PI metrics evaluated on the synthetic dataset using 5×2 cross-validation.

Method	MSE_{val}	$MPIW_{val}$	$PICP_{val}(\%)$	$PI_{\delta val}$
DualAQD	5.27 ± 0.27	7.30 ± 0.29	95.5 ± 0.48	1.52 ± 0.13
DualAQD_noBS	5.27 ± 0.27	9.16 ± 0.35	96.3 ± 0.77	3.08 ± 0.19
QD+	5.28 ± 0.29	8.56 ± 0.14	95.5 ± 0.31	3.12 ± 0.24
QD-Ens	5.31 ± 0.26	10.17 ± 0.79	94.0 ± 1.57	4.88 ± 0.17
MC-Dropout-PI	5.22 ± 0.30	9.31 ± 0.27	93.3 ± 0.63	5.04 ± 0.08

and consistency, batch sorting was used for all compared methods. In addition, we tested Dropout rates between 0.1 and 0.5. The obtained results did not indicate a statistically significant difference; thus, we used a Dropout rate of 0.1 for all networks and datasets.

Note that the only difference between the network architecture used by the four methods is that QD+ requires three outputs, QD-Ens requires two (i.e., the lower and upper bounds), and MC-Dropout-PI requires one. For DualAQD and MC-Dropout-PI, we used $F = 100$ forward passes with active dropout layers. For QD+ and QD-Ens, we used an ensemble of five networks and a grid search to choose the hyperparameter values. Fig. 5.6 shows the PIs generated by the four methods from the first validation set together with the ideal 95% PIs.

5.5.1.2 Benchmarking Experiments We experimented with eight open-access datasets from the UC Irvine Machine Learning Repository [43]:

- **Boston**: Predicts housing prices based on features such as crime rate, number of rooms, and property tax in Boston suburbs.
- **Concrete**: Estimates the compressive strength of concrete from ingredients like cement, water, and aggregate.
- **Energy**: Predicts the heating load requirements of buildings using architectural and environmental features.

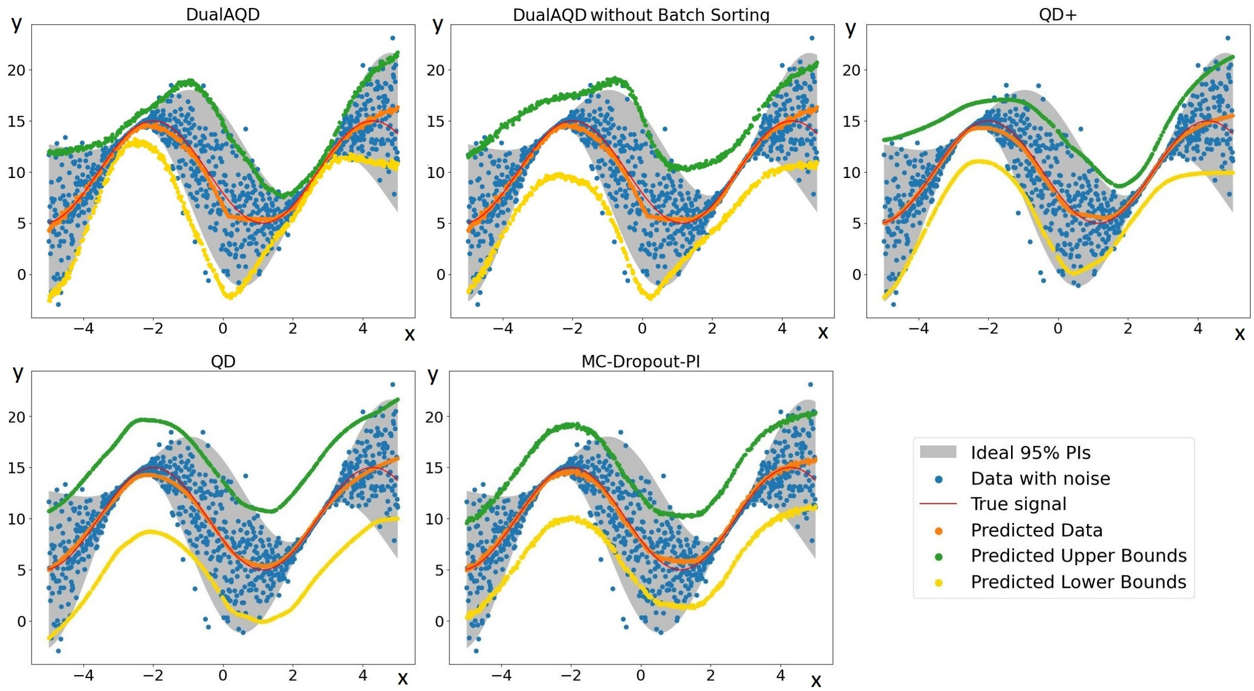


Figure 5.6: Performance of the PI generation methods on the synthetic dataset.

- **Kin8nm**: A synthetic dataset modeling the dynamics of an 8-link robot arm, with highly non-linear input-output relationships.
- **Power**: Predicts the electrical energy output of a combined-cycle power plant from ambient temperature, humidity, and pressure.
- **Protein**: Estimates the distance between amino acid residues in protein structures using physicochemical properties.
- **Yacht**: Predicts the hydrodynamic resistance (drag) of sailing yachts based on hull geometry and velocity.
- **Year**: Predicts the release year of songs from features extracted from raw music data.

For each dataset, we used a feed-forward neural network whose architecture was the same as that described in Section 5.5.1.1. We used 10-fold cross-validation to train and evaluate all networks. Table 5.2 gives the average performance for the metrics calculated on the validation sets, MSE_{val}, $MPIW_{val}$, and $PICP_{val}$, and corresponding standard deviations. We applied z-score normalization to each feature in the training set while the exact same scaling was applied to the features in the validation and test sets. Likewise, min-max normalization was applied to the response variable; however, Table 5.2 shows the results after re-scaling to the original scale. Similar to Section 5.5.1.1, all networks were trained using a fixed mini-batch size of 16, except for the Protein and Year datasets that used a mini-batch size of 512 due to their large size.

The bold entries in Table 5.2 indicate the method that achieved the lowest average $MPIW_{val}$ value and that its difference with respect to the values obtained by the other methods is statistically significant according to a paired t-test performed at the 0.05 significance level. The results obtained by DualAQD were significantly narrower than the compared methods while having similar MSE_{val} and $PICP_{val}$ of at least 95%. Furthermore, Fig. 5.7 depicts the distribution of the scores achieved by all the compared methods on all the datasets, where the line through the center of each box indicates the median F1 score, the edges of the boxes are the 25th and 75th percentiles, whiskers extend to the maximum and minimum points (not counting outliers), and outlier points are those past the end of the whiskers (i.e., those points greater than $1.5 \times IQR$ plus the third quartile or less than $1.5 \times IQR$ minus the first quartile, where IQR is the inter-quartile range).

Note that even though QD-Ens uses only one hyperparameter (see Section 5.2.6), it is more sensitive to small changes. For example, a hyperparameter value of $\delta = 0.021$ yielded poor PIs with $PICP_{val} < 40\%$ while a value of $\delta = 0.02105$ yielded too wide PIs with $PICP_{val} < 100\%$. For this reason, the hyperparameter δ of the QD-Ens approach was chosen manually while the scale factor η of DualAQD was chosen using a grid search with values $\{0.001, 0.005, 0.01, 0.05, 0.1\}$. Fig. 5.8 shows the difference between the learning curves obtained during one iteration of the cross-validation for the Power dataset using two different η values (i.e., $\eta = 0.01$ and $\eta = 0.1$). The dashed lines indicate the training epoch at which the optimal weights $\boldsymbol{\theta}_g$ were selected according to the dominance criteria explained

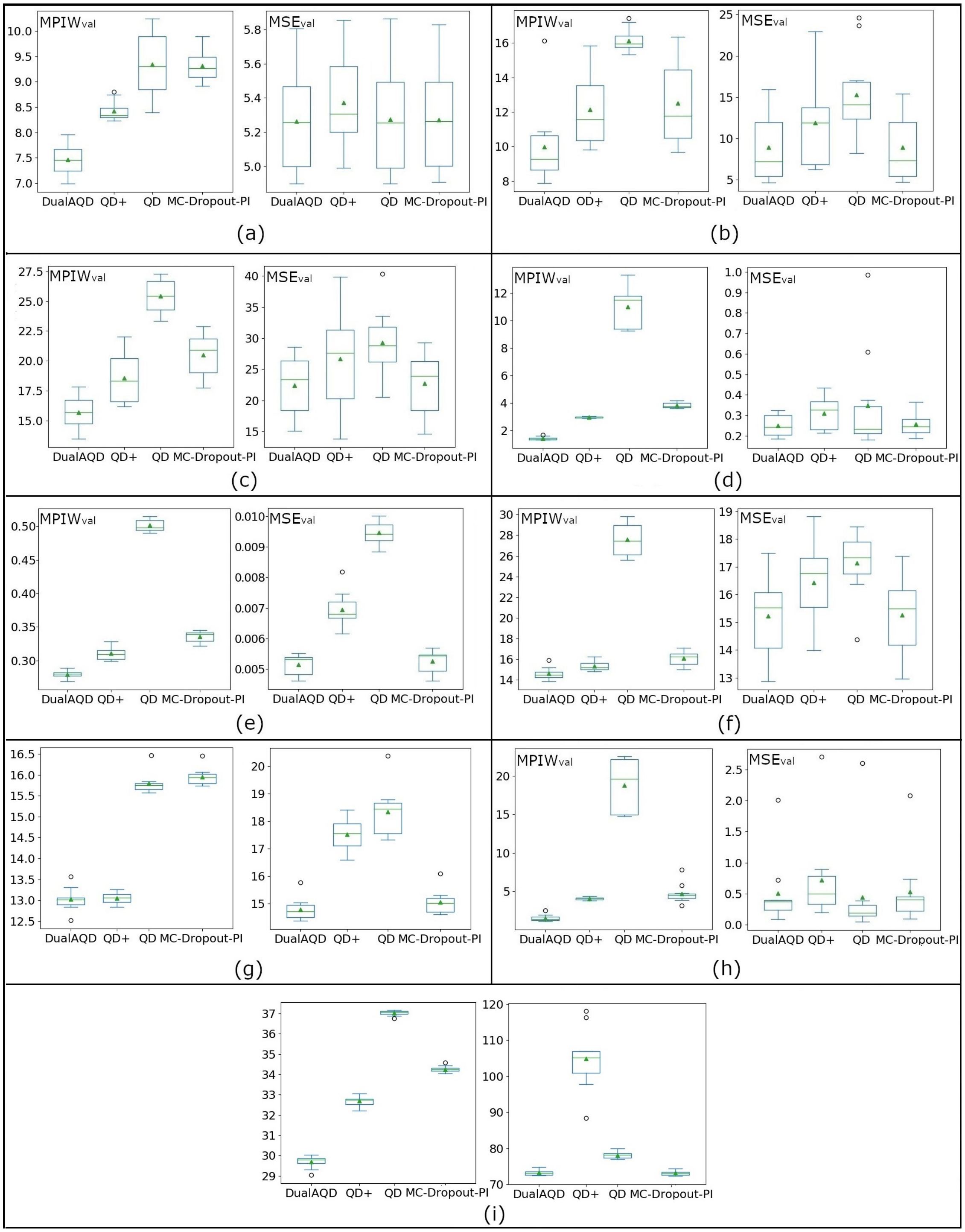


Figure 5.7: Box plots of the $MPIW_{val}$ and MSE_{val} scores of DualAQD, QD+, QD-Ens, and MC-Dropout-PI PI generation methods on the tested datasets: **(a)** Synthetic. **(b)** Boston. **(c)** Concrete. **(d)** Energy. **(e)** Kin8nm. **(f)** Power. **(g)** Protein. **(h)** Yacht. **(i)** Year.

Table 5.2: PI metrics evaluated on the benchmark datasets using 10-fold cross-validation.

Dataset	**Metric**	**DualAQD**	**QD+**	**QD-Ens**	**MC-Dropout-PI**
Boston	$MPIW_{val}$	**9.99±2.26**	12.14±2.05	16.13±0.67	12.52±2.28
	MSE_{val}	8.91±3.90	11.91±5.24	15.29±5.07	8.94±3.87
	$PICP_{val}(\%)$	95.0±1.6	95.6±1.9	97.2±1.3	96.0±0.9
Concrete	$MPIW_{val}$	**15.72±1.42**	18.57±2.06	25.42±1.30	20.52±1.74
	MSE_{val}	22.45±4.79	26.65±8.02	29.30±5.25	22.71±4.96
	$PICP_{val}(\%)$	95.2±0.5	95.2±1.3	97.9±1.6	95.7±1.2
Energy	$MPIW_{val}$	**1.41±0.12**	2.94±0.05	10.99±1.47	3.81±0.21
	MSE_{val}	0.25±0.05	0.31±0.08	0.35±0.25	0.26±0.05
	$PICP_{val}(\%)$	96.5±0.6	99.0±1.0	100.0±0.0	99.5±0.6
Kin8nm	$MPIW_{val}$	**0.280±0.01**	0.311±0.01	0.502±0.01	0.336±0.01
	MSE_{val}	0.005±0.00	0.007±0.00	0.009±0.00	0.005±0.00
	$PICP_{val}(\%)$	95.1±0.1	96.6±0.4	98.5±0.3	97.5±0.4
Power	$MPIW_{val}$	**14.60±0.35**	15.31±0.44	27.57±1.54	16.08±0.63
	MSE_{val}	15.23±1.34	16.43±1.34	17.14±1.11	15.26±1.31
	$PICP_{val}(\%)$	95.2±0.1	95.7±0.3	99.6±0.2	96.4±0.5
Protein	$MPIW_{val}$	**13.02±0.26**	**13.05±0.14**	15.79±0.24	15.95±0.20
	MSE_{val}	14.79±0.40	17.51±0.59	18.35±0.87	15.05±0.42
	$PICP_{val}(\%)$	95.0±0.1	95.4±0.4	95.1±0.5	94.8±0.1
Yacht	$MPIW_{val}$	**1.56±0.42**	4.10±0.17	10.99±1.47	4.74±1.20
	MSE_{val}	0.51±0.53	0.72±0.70	0.35±0.25	0.53±0.54
	$PICP_{val}(\%)$	97.1±0.9	98.4±2.2	100.0±0.0	100.0±0.0
Year	$MPIW_{val}$	**29.68±0.29**	32.68±0.25	37.03±0.13	34.25±0.16
	MSE_{val}	73.26±0.76	104.8±8.1	78.12±0.87	73.13±0.69
	$PICP_{val}(\%)$	95.1±0.1	95.4±0.9	37.03±0.1	93.82±0.0

in Section 5.2.4. On the other hand, the hyperparameters λ_1 and λ_2 of QD+ were chosen using a random search since it requires significantly higher training and execution time.

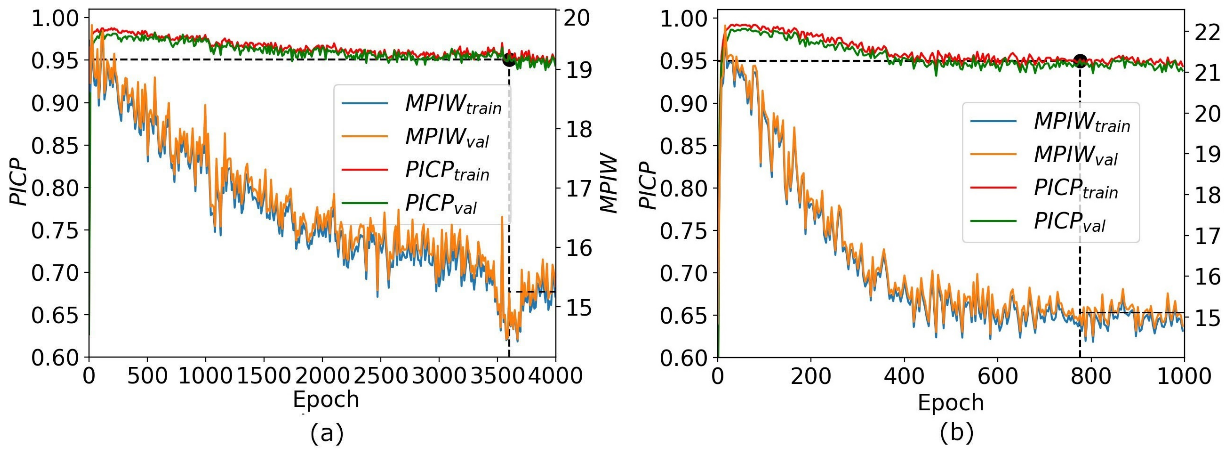


Figure 5.8: $MPIW$ and $PICP$ learning curves obtained for the Power dataset using DualAQD. **(a)** $\eta = 0.01$. **(b)** $\eta = 0.1$.

5.5.2 Syntehtic Datasets for Adaptive Sampling

We considered three synthetic 1-D problems for all AS problems considered in this section: `cos` [128], `hetero` [38], and `cosqr`. All three problems are affected by heteroscedastic noise, and their function equations are shown in Table 5.3. Unlike most AL and AS approaches, we do not initiate the experiments from empty datasets. For each case, we generated incomplete datasets as initial states, as shown in Fig. 5.9. The motivation for this is to produce areas with low data density, which entails high epistemic uncertainty. Thus, methods that estimate potential epistemic uncertainty more accurately and select sampling locations designed to reduce such uncertainty should require fewer AS iterations to approximate the ground-truth distribution of the problem. Below, we report the admissible search space for each problem:

- `cos`: $\mathcal{X} = \left\{-5 + \frac{10(i-1)}{99}, \mid i = 1, 2, \ldots, 100\right\}$
- `hetero`: $\mathcal{X} = \left\{-4.5 + \frac{9(i-1)}{299}, \mid i = 1, 2, \ldots, 300\right\}$
- `cosqr`: $\mathcal{X} = \left\{-10 + \frac{20(i-1)}{499}, \mid i = 1, 2, \ldots, 500\right\}$

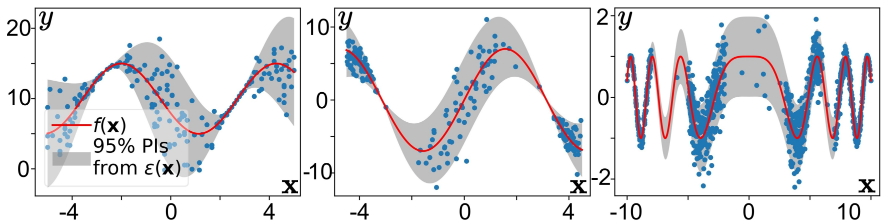


Figure 5.9: Initial `cos`, `hetero`, and `cosqr` datasets and the ideal 95% PIs calculated from $\varepsilon_a(\mathbf{x})$ across the domain.

Table 5.3: Functions and noise terms of the 1-D problems.

Name	**Function** $f(\mathbf{x})$	**Noise** $\varepsilon_a(\mathbf{x})$
`cos`	$10 + 5\cos(\mathbf{x} + 2)$	$\mathcal{N}(0, 2 + 2\cos(1.2\mathbf{x}))$
`hetero`	$7\sin(\mathbf{x})$	$\mathcal{N}(0, 3\cos(\mathbf{x}/2))$
`cosqr`	$10 + 5\cos(\frac{\mathbf{x}^2}{5})$	$\mathcal{N}(0, \frac{1}{2}(1 - \frac{\mathbf{x}^2}{100}))$

For the case of the `cos` problem, we generate the initial set of observations $\mathbf{X}_{obs}^{(it=0)}$ by uniformly sampling 200 elements from the discrete set $\mathcal{X}$. The initial datasets corresponding to the `hetero` problem are generated as recommended by Depeweg *et al.* [38]. In particular, a mixture of three Gaussian is created with means $\mu_1 = -4$, $\mu_2 = 0$, and $\mu_3 = 4$ and corresponding variances $\sigma_1 = \frac{2}{5}$, $\sigma_2 = 0.9$, and $\sigma_1 = \frac{2}{5}$. Each Gaussian component is equally weighted. We considered an initial dataset size of $|\mathbf{X}_{obs}^{(it=0)}| = 200$.

For the `cosqr` problem, the initial dataset is generated by first sampling 2,000 elements from the discrete set $\mathcal{X}$ uniformly and then applying a series of masks to select specific ranges of values. The process is as follows:

- Points in the intervals $[-10, -8)$, $[-5, -2)$, $[3, 6)$, and $[7, 10]$ are directly included.
- Additional points are selected from the intervals $[-8, -5)$, $[-2, 3)$, and $[6, 7)$ with specific sizes of 1, 10, and 3 elements, respectively.

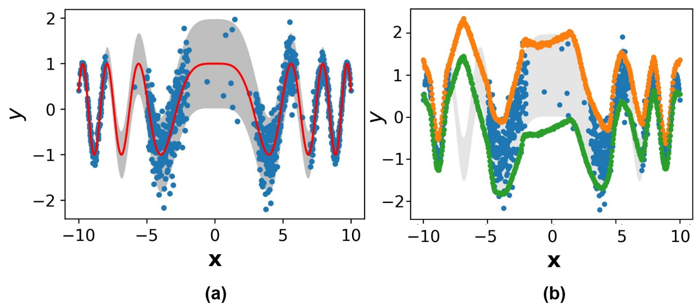


Figure 5.10: `cosqr` problem. **(a)** An initial generated dataset and the ideal 95% PIs calculated from $\varepsilon_a(\mathbf{x})$ across the domain. **(b)** Initial PIs estimated using DualAQD.

By doing so, we aim to generate a complex dataset with different areas with low data density, as depicted in Fig 5.10. Note that the low-density regions correspond to distinct behaviors in both the function $f(\mathbf{x})$ and the noise $\varepsilon_a(\mathbf{x})$. This approach ensures that the AS process remains focused on capturing meaningful variations in the data, rather than merely estimating data density for selecting future sample locations. For example, the intervals $[-8, -5)$ and $[6, 7)$ each contain only a single observed point. However, the former covers an entire oscillation of the function, whereas the latter spans a much smaller range. Consequently, when using a PI-generation neural network to analyze the unobserved areas, there is a greater discrepancy between the estimated and ideal PIs in the first case. All PI-generation NNs are trained using the DualAQD loss function (Section 5.2.1). Furthermore, the initial dataset size for problem `cosqr` varies according to the selected initialization seed. Specifically, the obtained sizes $|\mathbf{X}_{obs}^{(it=0)}|$ for the ten AS iterations are: 1,102, 1,106, 1,123, 1,078, 1,114, 1,163, 1,084, 1,159, 1,079, and 1,104, respectively.

5.5.3 Adaptive Sampling

We compared ASPINN to three methods adapted for AS: Normalizing flows ensembles (NF-Ensemble) [12], a standard GPR [58], and MC-Dropout [54]. For our experiments, we considered three synthetic 1-D regression problems. We used synthetic problems given that, in AS, we are required to sample at locations with high uncertainty that could not have been observed previously. By utilizing problems with known underlying target and noise functions, which are unknown to the AS methods, we can simulate and evaluate accurately the performance improvements resulting from the decisions made by each method in previous iterations. In addition to the results presented in this section, ASPINN was also applied to a real-world multidimensional regression task, which will be discussed in detail in Chapter 6.

For ASPINN, we trained feed-forward NNs with varying depths: two hidden layers with 100 units for problems `cos` and `hetero`; and three hidden layers with 500, 100, and 50 units, respectively, for `cosqr`. The networks $\hat{f}_{it}$ and $\hat{f}_{\mathrm{PI},\,it}$ share the same architecture except for the last layer, as $\hat{f}_{it}$ uses one output, while $\hat{f}_{\mathrm{PI},\,it}$ uses two outputs. Furthermore, ASPINN uses two hyperparameters: the neighbor distance threshold θ and the kernel length r. We performed a grid search with the values $\theta = [0.1, 0.15, 0.2, 0.25]$ and $r = [0.1, 0.15, 0.2, 0.25]$, and selected $\theta = 0.25$ and $r = 0.15$ for all experiments. DualAQD, the PI-generation method used by ASPINN, uses a hyperparameter η as a scale factor to adapt the coefficient that balances the two objectives of the DualAQD loss function (see Section 5.2.3). We chose a scale factor $\eta = 0.1$. Using other η values (i.e., $\{0.001, 0.005, 0.01, 0.05, 0.1\}$), we achieved similar results but with slower convergence rates.

For MC-Dropout, we used the same architecture as the target-estimation NN in ASPINN. For NF-Ensemble, we used flows with 200 hidden units for problems `cos` and `hetero` and 300 hidden units for problem `cosqr`. We employed ensembles consisting of five models trained during 30,000 epochs. For the standard GPR, we used the same RBF kernel used by ASPINN. We utilized an inference implementation based on black-box matrix-matrix multiplication [58] that uses 3000 training epochs.

Our objective is to reduce the epistemic uncertainty with as few AS iterations as possible. These experiments consider a fixed batch size $B = 5$. Based on Eq. 5.9, we define the performance metric $PI_{\delta}^{(it)}$ to quantify epistemic uncertainty relative to the ground truth at the *it*-th iteration:

$$PI_{\delta}^{(it)} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \left(|y^u(\mathbf{x}) - \hat{y}_{it}^u(\mathbf{x})| + |y^\ell(\mathbf{x}) - \hat{y}_{it}^\ell(\mathbf{x})| \right).$$

Here, $y^\ell(\mathbf{x})$ and $y^u(\mathbf{x})$ represent the ideal lower and upper PI bounds, respectively, calculated from the aleatoric noise function: $y^u(\mathbf{x}) = f(\mathbf{x}) + 1.96\,\varepsilon_a(\mathbf{x})$ and $y^\ell(\mathbf{x}) = f(\mathbf{x}) - 1.96\,\varepsilon_a(\mathbf{x})$. This metric is applicable to problems with normally distributed aleatoric noise, which is the case for the problems evaluated in this section. However, none of the tested methods make assumptions about the noise distribution. Note that if $PI_{\delta}^{(it)} = 0$, the estimated PIs match the ideal intervals, implying that the model's epistemic uncertainty has been minimized, and the total uncertainty is purely aleatoric. A non-zero $PI_{\delta}^{(it)}$ indicates a discrepancy between the estimated and ideal PIs, suggesting the presence of epistemic uncertainty. The greater the $PI_{\delta}^{(it)}$, the higher the epistemic uncertainty. To ensure fairness, $\hat{y}_{it}^\ell(\mathbf{x})$ and $\hat{y}_{it}^u(\mathbf{x})$ are generated by an independent NN, $\hat{f}_{\mathrm{PI},\,it}(\cdot)$, trained on the dataset $\mathcal{D}_{it}$ produced by each compared method at each iteration. Regardless of the uncertainty estimation model used by each method, we trained an additional PI-generation NN using the DualAQD loss to maintain a consistent uncertainty metric across all comparisons.

It is worth mentioning that other works have used different evaluation approaches. For instance, Berry and Meger [12] employed an approach where they sampled 50 random locations from the domain. For each location, they generated 1000 samples using the ground-truth distribution and 1000 samples using the distribution predicted by each method. They then calculated the Kullback-Leibler divergence between the ground truth and the model-generated distributions. However, we believe this approach does not provide a consistent basis for evaluation, as each method employs different mechanisms for estimating uncertainty.

For our experiments, the AS process was executed for each problem for 50 iterations. Although some methods may converge earlier, evaluating all methods over the same number

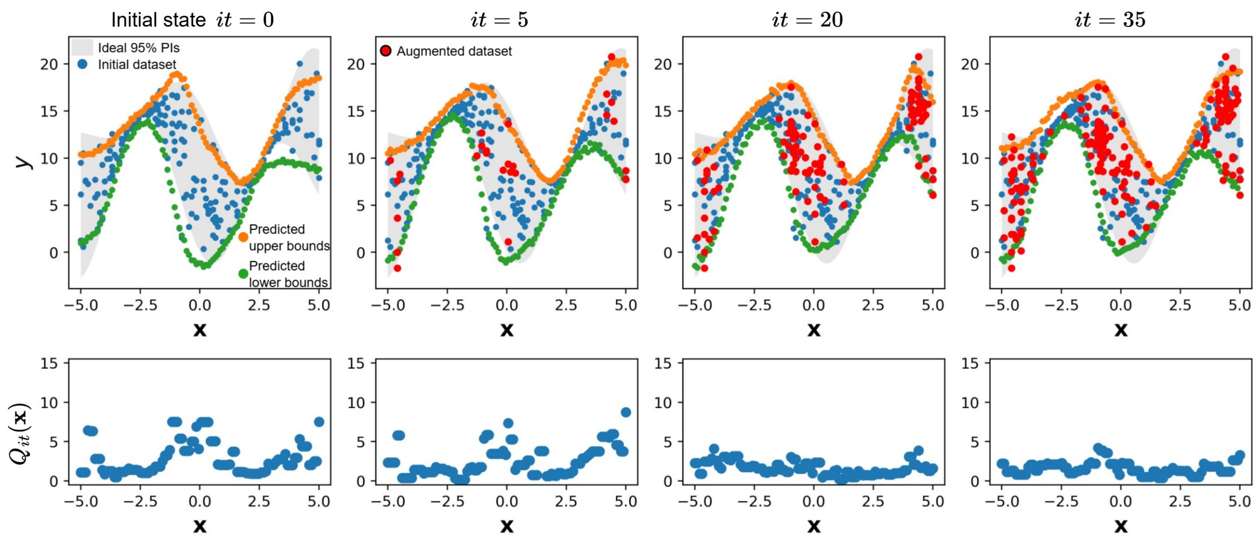


Figure 5.11: Adaptive sampling process using ASPINN on the `cos` problem.

of steps provides a standardized basis for assessing their sampling efficiency. This process is repeated 10 times, initializing the problems with a different seed each time. Figures 5.11, 5.12, and 5.13 illustrate the results obtained by ASPINN for problems `cos`, `hetero`, and `cosqr`, respectively. The figures show the problems' initial state along with the augmented datasets obtained during iterations $it = 5$, $it = 20$, and $it = 35$. They also display the calculated potential epistemic uncertainty $Q_{it}(\mathbf{x})$ for all values of the input domain. Figure 5.14 shows the evolution of the mean $PI_{\delta}^{(it)}$ value and its corresponding standard deviation, calculated across the values obtained from the 10 repetitions at each it.

In addition, we calculated the area under the uncertainty curve (AUUC) for each learning curve. For each problem, Table 5.4 gives the average AUUC for the four methods and corresponding standard deviations. The bold entries indicate the method that achieved the lowest average AUUC value and that its difference with respect to the values obtained by the other methods is statistically significant according to a paired t-test performed at the 0.05 significance level. Table 5.5 presents the p-values from the paired t-tests comparing ASPINN with the other methods. Here, the upward-pointing arrow ($\uparrow$) indicates that

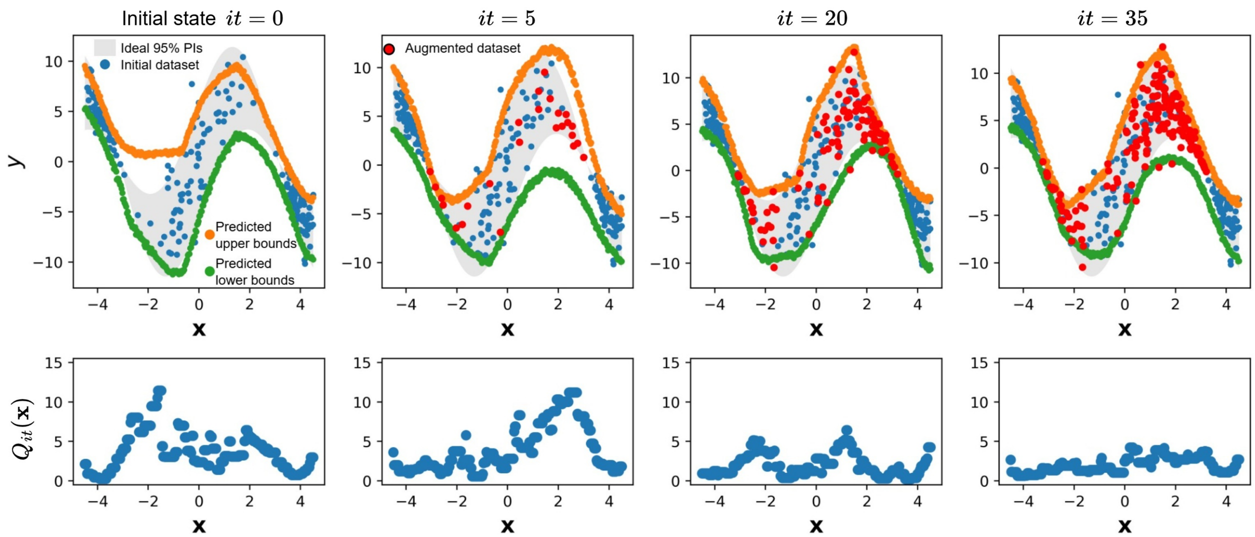


Figure 5.12: Adaptive sampling process using ASPINN on the `hetero` problem.

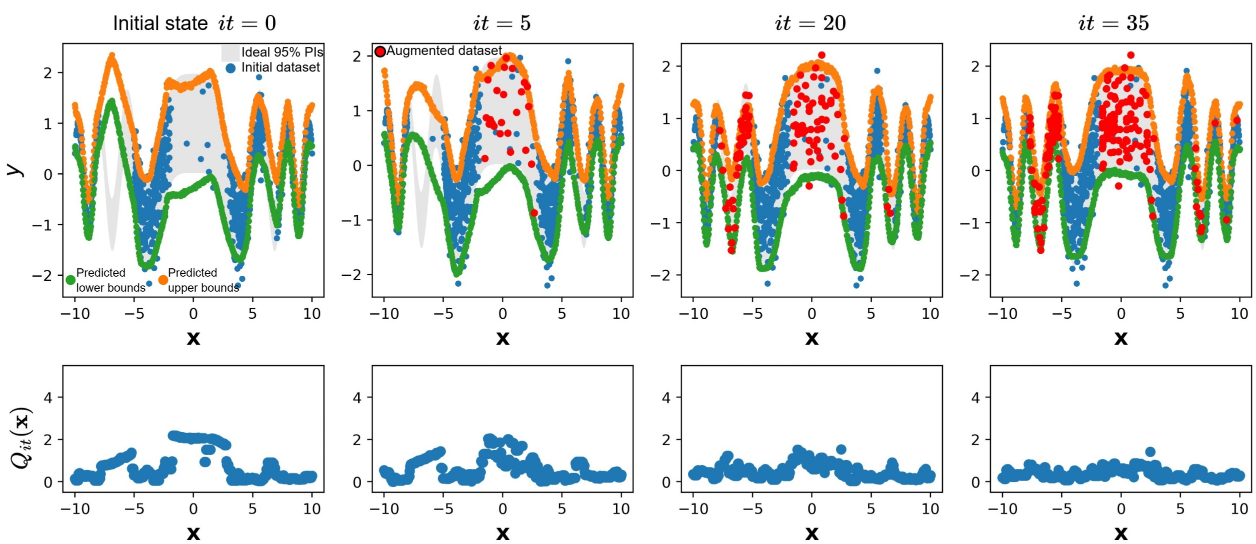


Figure 5.13: Adaptive sampling process using ASPINN on the `cosqr` problem.

ASPINN performed significantly better (i.e., p-value < 0.05).

5.5.4 Symbolic Regression and Adaptive Sampling

To evaluate the effectiveness of symbolic regression within our adaptive sampling framework, we analyze how the learned symbolic expressions evolve over AS iterations

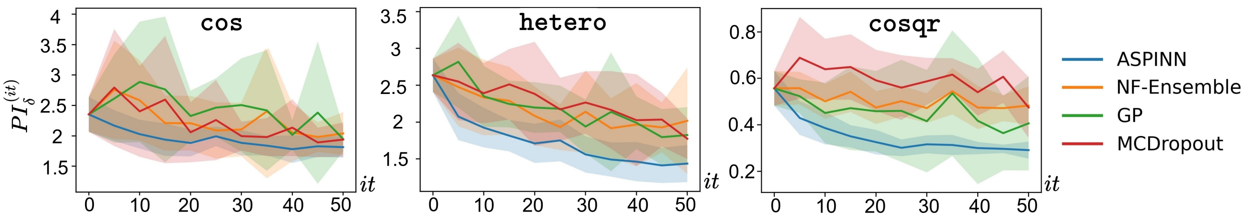


Figure 5.14: Evolution of the mean $PI_\delta^{(it)}$ and its standard deviation for the 1-D problems.

Table 5.4: AUUC comparison for the 1-D problems

Problem	**MCDropout**	**GP**	**NF-Ensemble**	**ASPINN**
`cos`	112.57±24.20	123.87±26.22	113.39±19.49	**97.26±7.87**
`hetero`	113.80±13.38	110.21±13.59	106.44±16.26	**85.95±9.11**
`cosqr`	30.39±3.56	23.12±5.55	25.60±2.67	**17.13±1.42**

Table 5.5: Statistical significance tests between ASPINN and the compared methods.

Compared Method	`cos`	`hetero`	`cosqr`
NF-Ensemble	4.4E-2 (↑)	6.4E-3 (↑)	5.3E-6 (↑)
GP	5.4E-3 (↑)	8.6E-4 (↑)	7.7E-3 (↑)
MC-Dropout	3.8E-2 (↑)	5.7E-4 (↑)	3.1E-6 (↑)

and assess whether they converge toward the underlying data-generating functions. These experiments aim to determine whether the integration of uncertainty-aware sampling and symbolic regression can lead to interpretable models that progressively approximate the true functional form, even in data-scarce regimes.

At each AS iteration, it, we applied SeTGAP to recover a symbolic expression $\tilde{f}_{it}$ that approximates and explains the predictive function of the trained NN $\hat{f}_{it}$. Specifically, we applied an adapted version of SeTGAP, as described in Algorithm 5.4. This approach operates by constructing a collection of subdomain datasets from $\mathcal{D}_{it} = \left(\mathbf{X}_{obs}^{(it)}, \mathbf{y}_{obs}^{(it)}\right)$, where each subdomain contains a random portion of the input domain. This allowed us to take full advantage of SeTGAP's multi-set skeleton prediction configuration. As such, SeTGAP was tasked with recovering a symbolic skeleton expression that generalizes across the subdomain variations. Using the identified skeleton, a full mathematical expression was then derived.

To ensure consistency with the Multi-Set Transformer's training regime (see Section 3.4), we used $N_S = 10$ input sets or subdomains per MSSP instance, each containing $n = 3000$ points. Across all experiments, the Multi-Set Transformer was configured with a beam size $n_B = 3$, generating $n_{\text{cand}} = 5$ skeleton candidates per instance. Increasing these values was found to offer no additional benefit in terms of discovering distinct top-performing candidates. Each skeleton was subsequently fitted to the observed dataset $\mathcal{D}_{it}$, and the final expression was selected based on the lowest mean squared error.

Table 5.6 summarizes the expressions recovered by SeTGAP at AS iterations $it \in \{1, 5, 10, \ldots, 50\}$ for each tested problem. At each iteration, the dataset $\mathbf{X}_{obs}^{(it)}$ is augmented using ASPINN, following the configuration detailed in Section 5.5.3. For each expression, we also report the predicted MSE on the entire dataset, computed using the learned expression $\tilde{f}_{it}$. Highlighted cells indicate that the identified expression matches the functional form of the underlying function f. These results demonstrate that as more informative samples are acquired through the adaptive sampling process, the learned symbolic models become increasingly aligned with the ground truth.

Finally, Figs. 5.15–5.17 compare the predicted curves obtained from the model $\hat{f}_{it}$ with those generated by the corresponding symbolic expressions $\tilde{f}_{it}$ at each iteration. Early iterations often yield inaccurate or overfitted expressions due to sparse data coverage, but as AS progresses, the symbolic models stabilize and converge toward compact expressions that match the true function in both form and performance.

5.6 Discussion

In this section, we analyze the results and performance of DualAQD for prediction-interval generation (Section 5.5.1), ASPINN for adaptive sampling (Section 5.5.3), and the

Table 5.6: Evolution of the identified expressions during the AS process

it	cos		hetero		cosqr	
	$\tilde{f}_{it}$	MSE	$\tilde{f}_{it}$	MSE	$\tilde{f}_{it}$	MSE
1	$5.153\sin(62.757\sqrt{1-0.032\mathbf{x}}+5.994)$ $+9.933$	1.502	$3.940\mathbf{x}\tanh(0.006\mathbf{x}-6.212)$ $+10.742\tanh(1.777\mathbf{x}-0.123)$	1.508	$0.145\mathbf{x}\sin(2.643\mathbf{x}+6.235)+$ $0.564\sin(1.732\mathbf{x}+10.919)$	0.325
5	$10.001-5.037\sin(1.002\mathbf{x}+12.989)$	1.585	$7.261\sin(0.998\mathbf{x}-6.327)-0.276$	1.358	$0.012-0.983\sin(0.197\mathbf{x}^2+4.911)$	0.222
10	$8.973\sqrt{1-0.971\sin(0.974\mathbf{x}+6.555)}$ $+2.123$	1.768	$7.203\sin(1.0\mathbf{x}+18.809)-0.261$	1.300	$-1.271\sin(1.201\mathbf{x}+6.256)^2-$ $0.495\sin(3.242\mathbf{x}+10.966)+0.655$	0.409
15	$5.616\sin(0.508\mathbf{x}-3.803)^2+$ $2.295\sin(0.973\mathbf{x}+10.024)+7.281$	1.811	$7.123\sin(\mathbf{x}-18.942)-0.025$	1.256	$-0.016\mathbf{x}^2\sin(2.849\mathbf{x}+1.598)+$ $0.053\mathbf{x}\sin(1.122\mathbf{x}-0.075)$ $+0.762\sin(1.745\mathbf{x}+4.757)$	0.277
20	$10.002-4.987\sin(0.995\mathbf{x}-5.861)$	1.866	$7.202\sin(0.999\mathbf{x}-18.820)-0.190$	1.232	$0.993\sin(0.199\mathbf{x}^2+1.612)+0.003$	0.220
25	$10.004-5.001\sin(0.996\mathbf{x}-5.860)$	1.967	$7.149\sin(0.999\mathbf{x}-18.831)-0.128$	1.185	$0.994\sin(0.200\mathbf{x}^2+1.608)+0.004$	0.222
30	$10.037-4.998\sin(0.996\mathbf{x}-6.696)$	1.988	$7.217\sin(0.999\mathbf{x}-0.033)-0.218$	1.206	$0.991\sin(0.199\mathbf{x}^2+1.623)$	0.223
35	$9.987+5.014\sin(0.999\mathbf{x}+16.138)$	2.061	$7.138\sin(1.000\mathbf{x}+18.835)-0.093$	1.223	$-0.997\sin(0.200\mathbf{x}^2+17.315)$	0.221
40	$10.002+5.003\sin(0.997\mathbf{x}-15.285)$	2.043	$7.128\sin(1.000\mathbf{x}+18.838)-0.071$	1.216	$0.986\sin(0.198\mathbf{x}^2+1.688)+0.004$	0.223
45	$10.011+5.043\sin(1.000\mathbf{x}-9.003)$	2.051	$-7.097\sin(1.000\mathbf{x}-3.131)-0.074$	1.237	$1.0\sin(0.200\mathbf{x}^2+1.612)+0.002$	0.228
50	$10.055-5.005\sin(0.999\mathbf{x}+12.997)$	2.055	$-7.150\sin(1.000\mathbf{x}+15.689)-0.135$	1.244	$0.990\sin(0.199\mathbf{x}^2-17.217)+0.004$	0.231

integration of SeTGAP and ASPINN (Section 5.5.4), while highlighting their strengths, limitations, and implications for uncertainty-aware modeling.

5.6.1 Prediction-Interval Learning Results

The $Loss_{\text{DualAQD}}$ function was designed to minimize the estimation error and produce narrow PIs simultaneously while using constraints that maximize the coverage probability inherently. From Tables 5.1 and 5.2, we note that DualAQD consistently produced significantly narrower PIs than the compared methods, according to the paired t-test performed at the 0.05 significance level, except for the Protein dataset, where QD+ obtained comparable PI widths. Simultaneously, we yielded $PICP_{val}$ values of at least 95% and

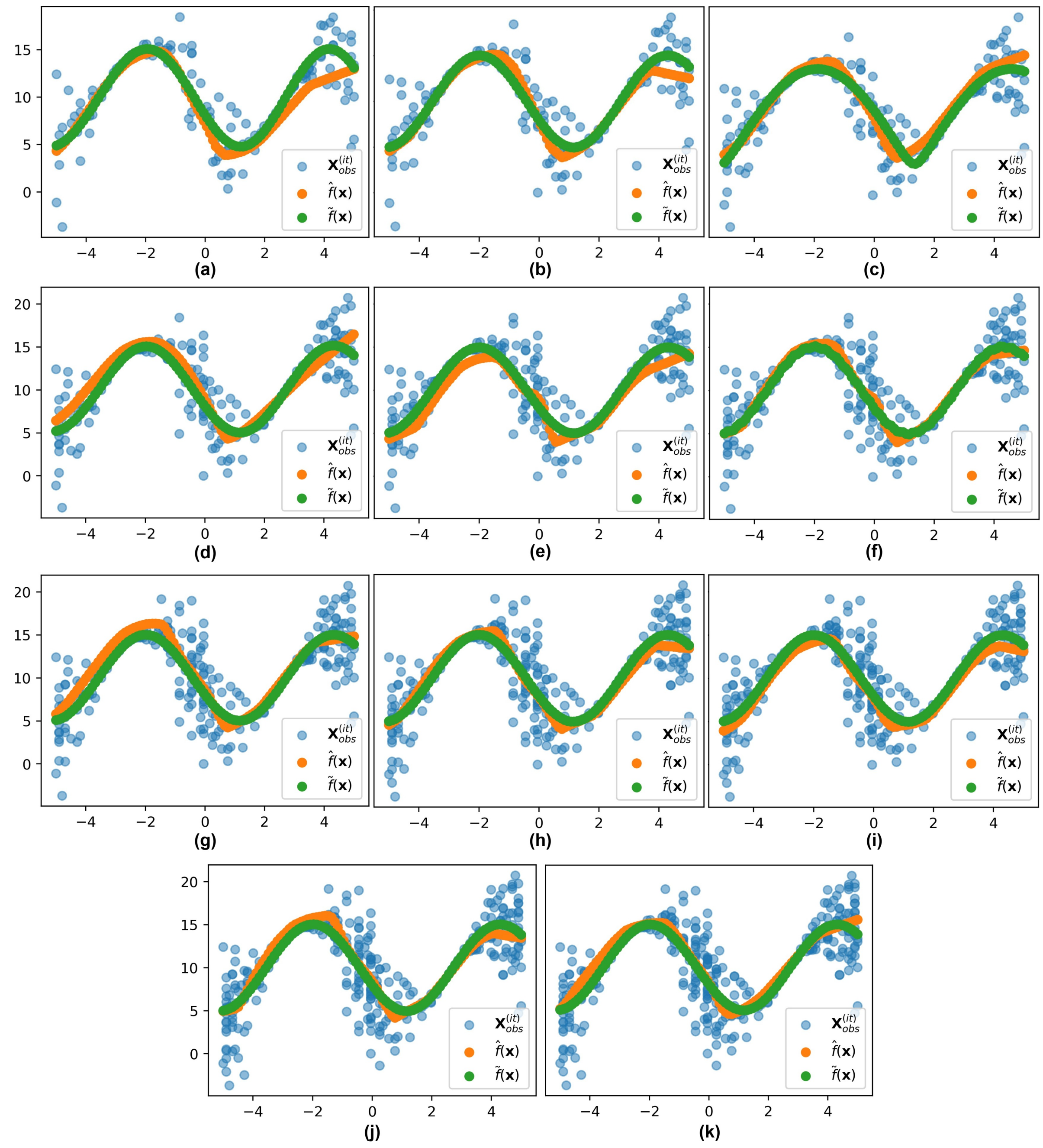


Figure 5.15: Comparison of $\hat{f}_{it}(\mathbf{x})$ vs. $\tilde{f}_{it}(\mathbf{x})$ throughout the AS process for problem `cos`. **(a)** $it = 1$. **(b)** $it = 5$. **(c)** $it = 10$. **(d)** $it = 15$. **(e)** $it = 20$. **(f)** $it = 25$. **(g)** $it = 30$. **(h)** $it = 35$. **(i)** $it = 40$. **(j)** $it = 45$. **(k)** $it = 50$.

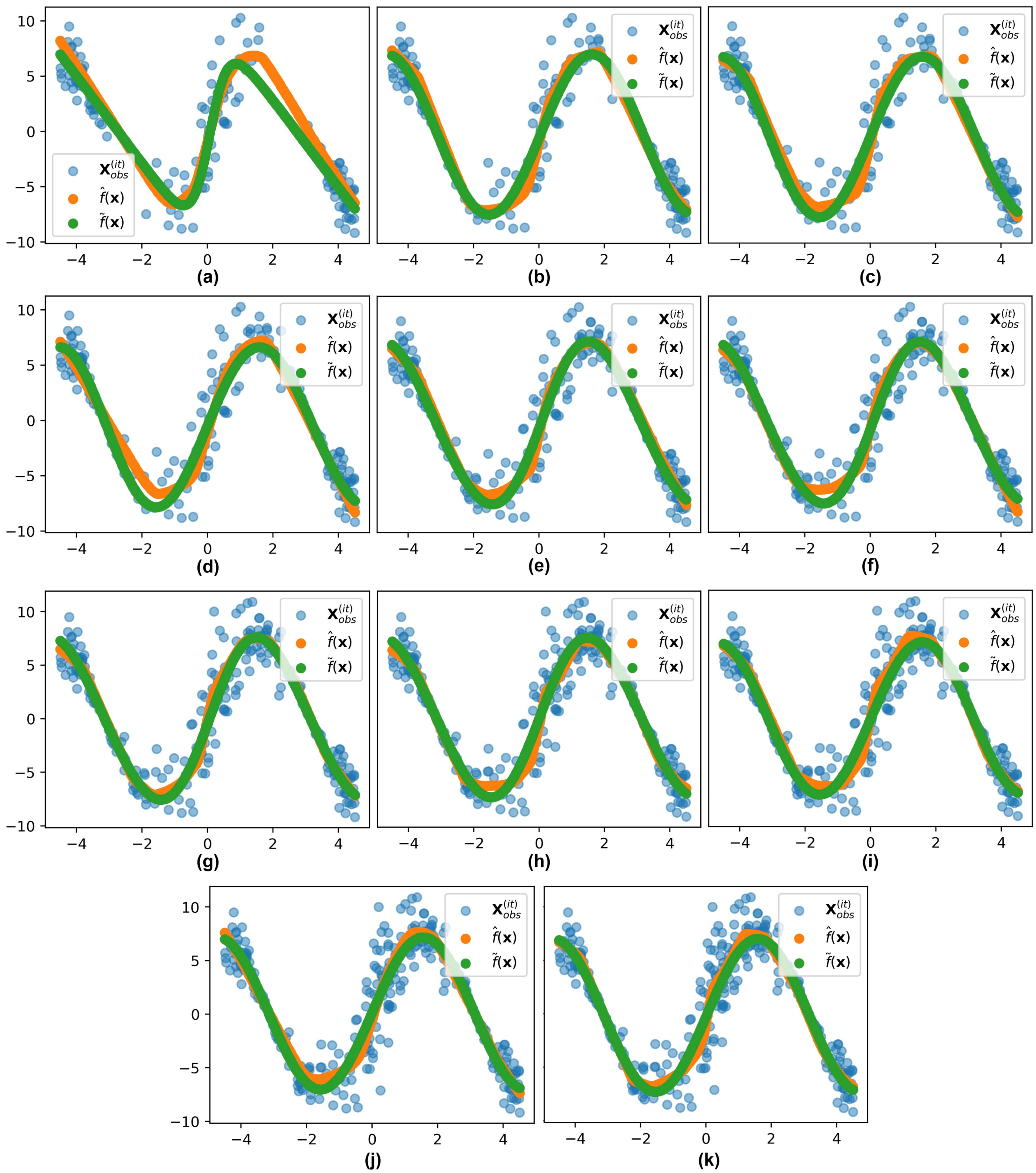


Figure 5.16: Comparison of $\hat{f}_{it}(\mathbf{x})$ vs. $\tilde{f}_{it}(\mathbf{x})$ throughout the AS process for problem `hetero`. **(a)** $it = 1$. **(b)** $it = 5$. **(c)** $it = 10$. **(d)** $it = 15$. **(e)** $it = 20$. **(f)** $it = 25$. **(g)** $it = 30$. **(h)** $it = 35$. **(i)** $it = 40$. **(j)** $it = 45$. **(k)** $it = 50$.

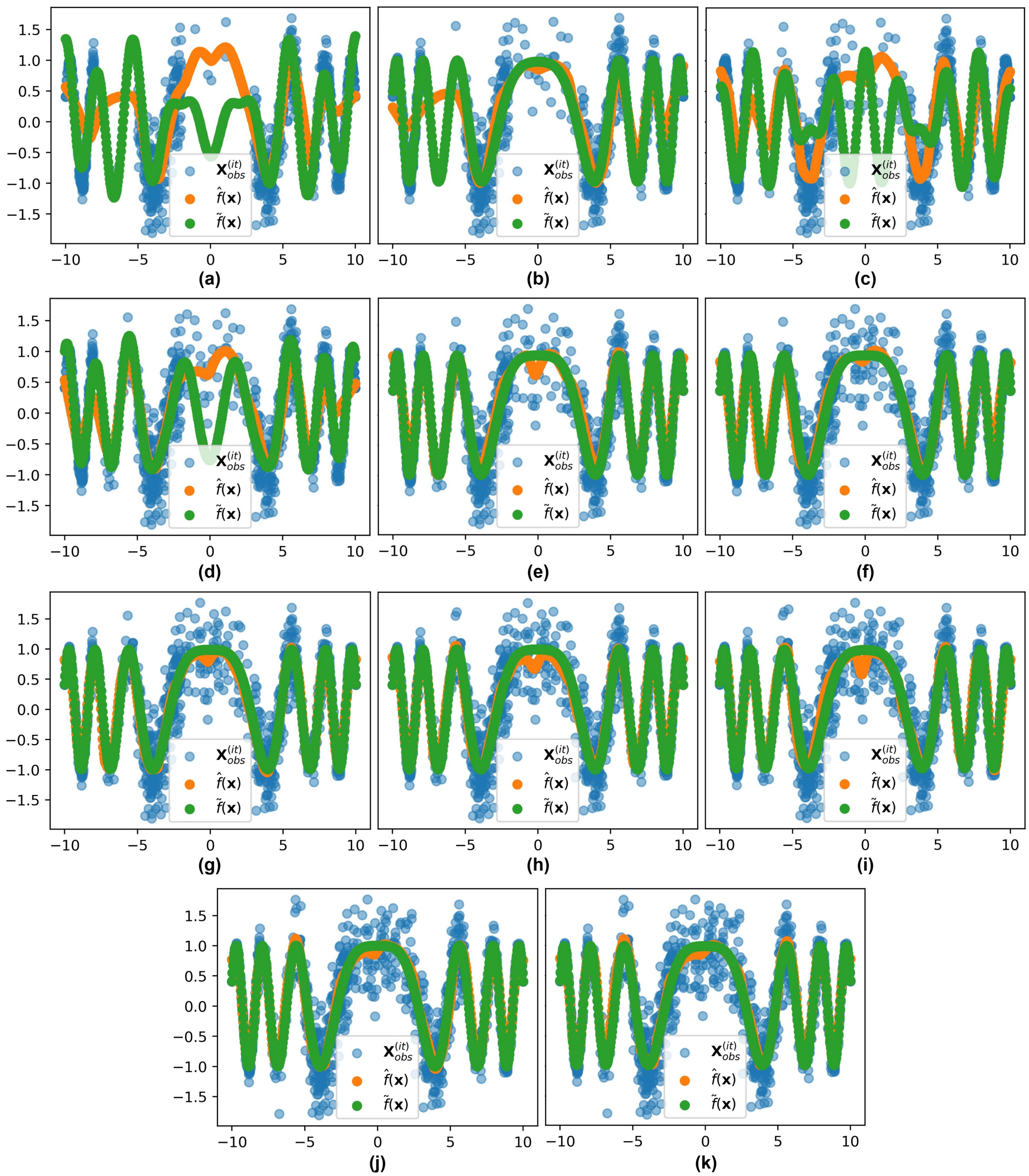


Figure 5.17: Comparison of $\hat{f}_{it}(\mathbf{x})$ vs. $\tilde{f}_{it}(\mathbf{x})$ throughout the AS process for problem `cosqr`. **(a)** $it = 1$. **(b)** $it = 5$. **(c)** $it = 10$. **(d)** $it = 15$. **(e)** $it = 20$. **(f)** $it = 25$. **(g)** $it = 30$. **(h)** $it = 35$. **(i)** $it = 40$. **(j)** $it = 45$. **(k)** $it = 50$.

better or comparable MSE_{val} values. In addition, the $PI_{\delta val}$ values reported in Table 5.1 demonstrate that DualAQD is the method that best adapts to the highly varying uncertainty levels of our synthetic dataset. Thus, the PI bounds generated by DualAQD were the closest to the ideal 95% PIs.

Notice that DualAQD obtains lower MSE_{val} values than QD+ consistently despite the fact that QD+ also includes an objective function that minimizes the error of the target predictions. The reason is that our method uses a NN (i.e., $f(\cdot)$) that is specialized in generating accurate target predictions, and its optimization objective does not compete with others. Conversely, QD+ uses a loss function that balances four objective functions: minimizing the PI widths, maximizing PI coverage probability, minimizing the target prediction errors, and ensuring PI integrity. The NN used by QD-Ens, on the other hand, only generates the upper and lower bounds of the PIs. The target estimate is then calculated as the central point between the PI bounds. As a consequence of not using a NN specialized in minimizing the target prediction error, QD-Ens achieved the worst MSE_{val} values of the compared methods, except for the Year dataset.

One of the advantages of using DualAQD over QD+ and QD-Ens is that we achieved better PIs while requiring less computational complexity. That is, our method requires training only two NNs and uses MC-Dropout to account for the model uncertainty while QD+ and QD-Ens require training ensembles of five NNs. In addition, QD+ requires extra complexity given that it uses a split normal aggregation method that involves an additional fitting process for each data point during testing. Note that using deep ensembles of M models is expected to perform better or similar to MC-Dropout when using M forward passes [97]. In other words, using an ensemble of five NNs, as QD and QD+ do, is expected to perform better than using five forward passes through the NN using MC-Dropout. Nevertheless, during inference, we are able to perform not only five but 100 passes through the NN without significantly adding computational cost. Our method becomes more practical in the sense that, even when it uses the rough estimates of model uncertainty provided by MC-Dropout, it is still able to generate significantly higher-quality PIs.

In Fig. 5.8, we see the effect of using different scale factors η to update the balancing coefficient λ of $Loss_{\text{DualAQD}}$. DualAQD produced wide PIs at the beginning of the training process in order to ensure PI integrity; as a consequence, the $PICP_{train}$ and $PICP_{val}$ values improved drastically. Once the generated PIs were wide enough to cover most of the samples in the training set (i.e., $PICP_{train} \approx 1$), DualAQD focused on reducing the PI widths until $PICP_{train}$ reached the nominal probability coverage α. The rate at which $PICP$ and $MPIW$ were reduced was determined by the scale factor η.

Furthermore, Fig. 5.8a ($\eta = 0.01$) and Fig. 5.8b ($\eta = 0.1$) show that both models converged to a similar $MPIW_{val}$ value (~ 15) despite having improved at different rates. It is worth noting that we did not find a statistical difference between the results produced by the different η values that were tested on all the datasets (i.e., $\eta \in [0.001, 0.1]$), except for the case of Kin8nm. When various η values were considered equally as good for a given dataset, we selected the η value that yielded the lowest average $MPIW_{val}$, which was $\eta = 0.01$ for Boston, Concrete, and Yacht, $\eta = 0.005$ for Kin8nm, and $\eta = 0.05$ for the rest of the datasets. This is significant because it shows that the sensitivity of our method to the scale factor η is low, unlike the hyperparameters required by QD-Ens (Section 5.5.1.2). What is more, our method requires a single hyperparameter, η, while QD-Ens requires two: λ and a softening factor used to enforce differentiability of its loss function; and QD+ requires four: λ_1, λ_2, and λ_3, and the same softening factor used by QD-Ens. DualAQD does not need an additional softening factor given that its operations are differentiable.

5.6.2 Adaptive Sampling Results

When evaluating ASPINN on the tested 1-D problems, as shown in Fig. 5.14, we observed that it produced learning curves with faster convergence rates and lower standard deviation than the other methods. Although the confidence bands exhibit some overlap, this is attributed to outliers with high $PI_{\delta}^{(it)}$ values generated by other methods (e.g., GP), which increase the variance. Nevertheless, the learning curves for ASPINN consistently remain below those of the other methods across all iterations and have narrower confidence bands. Thus, the difference in AUUC values is shown to be statistically significant according to the t-test, as shown in Table 5.4. Also from Fig. 5.14, we notice that ASPINN generated constantly decreasing and smoother learning curves. Conversely, other methods, such as MC-Dropout, tend to oversample certain regions of the input domain, which results in overfitting in those regions while causing a poor fit in others, producing unstable learning curves.

One limitation of our approach is that it does not handle multi-modal aleatoric noise inherently. Multi-modal noise indicates that the data variability comes from different underlying sources, each contributing to a different mode in the noise distribution. In such cases, it would be necessary to use a PI-generation method capable of producing multiple upper and lower bounds based on the identified number of modes. In the presence of multiple PIs, we would need to adapt the epistemic uncertainty metric accordingly and execute the remaining steps similarly. Another limitation, which also applies to the compared methods, is the computational cost when dealing with high-dimensional problems due to the need to evaluate all potential locations in the input space.

5.6.3 Symbolic Regression and Adaptive Sampling Results

We evaluated the performance of SeTGAP adapted for solving 1-D problems under conditions of epistemic uncertainty. The results presented in Table 5.6 demonstrate that the estimated mathematical expressions began consistently matching the true functional form of the target function by iteration $it = 20$ across all tested cases. Notably, although the correct symbolic structure was occasionally identified at earlier stages (e.g., at $it = 5$), these early discoveries tended to be unstable, as subsequent iterations often produced alternative expressions. This instability highlights the inherent challenge of model identification when domain coverage is limited or when predictive uncertainty remains high.

In the early iterations of the AS process, the observed data are sparse and unevenly distributed across the input domain. For instance, Fig.5.17.b illustrates the prediction generated by the model $\hat{f}_{it}$ at $it = 5$, where limited observations in the central region of the domain result in a relatively smooth estimated response by the model $\hat{f}_{it}$. In such cases, SeTGAP can recover occasionally a correct or near-correct functional form because the prediction lacks fine-grained variations that would otherwise complicate the model discovery task. However, as sampling proceeds and additional points are collected in previously underrepresented regions, local variations in the curve predicted by $\hat{f}_{it}$ begin to emerge, as shown in Fig.5.17.c for $it = 10$. These local fluctuations can mislead the SR process, causing the symbolic models to temporarily favor more complex or distorted expressions that attempt to capture these finer structures.

This evolution highlights a key strength of integrating SeTGAP into the adaptive sampling framework. As the AS process strategically targets regions of high epistemic uncertainty, the overall domain coverage improves, progressively eliminating major gaps and inconsistencies in the input space. Consequently, SeTGAP gains access to increasingly informative and representative subsets of the domain, allowing it to generate symbolic models that are not only more accurate but also more stable over time. The convergence toward simpler, correct symbolic expressions in later iterations reflects the systematic reduction of epistemic uncertainty and the stabilization of the learned system behavior.

Importantly, this convergence is achieved despite the persistent presence of aleatoric uncertainty in the observations, highlighting the robustness of the approach. However, it is important to note that this was facilitated in these experiments by the specific characteristics of the aleatoric noise ε_a used during data generation, as summarized in Table 5.3. Although ε_a was heteroscedastic, meaning its variance depended on the input $\mathbf{x}$, its mean was zero across the domain. Under these conditions, a sufficiently trained neural network model $\hat{f}_{it}$ approximates the conditional expectation $\mathbb{E}[y|\mathbf{x}]$, which coincides with the underlying function $f(\mathbf{x})$. As a result, the model captures the mean behavior of the system, enabling SeTGAP to recover the correct functional form. In contrast, if the observational noise were biased (i.e., nonzero mean) or introduced systematic distortions, $\hat{f}_{it}$ would converge to a shifted version of $f(\mathbf{x})$, potentially preventing symbolic recovery of the true structure.

In certain cases, SeTGAP uncovered the correct underlying functional form even when the prediction model $\hat{f}_{it}$ exhibited notable inaccuracies. Continuing the analysis of Fig. 5.17.b, we observe that the prediction produced by $\hat{f}_{it=5}$ deviates significantly from the expected behavior in the region $\mathbf{x} \in [-7, -5]$, primarily due to sparse data coverage in that interval. Despite this local error, SeTGAP was able to identify the correct symbolic skeleton and, after fitting it to the available observations, produced a response closely resembling the true function. This robustness arises from SeTGAP's multi-set strategy, which involves randomly sampling diverse subregions across the domain under the assumption that they share a common functional form. In this example, the majority of sampled subregions exhibited a strong $\sin(\mathbf{x}^2)$ behavior, allowing the symbolic regression process to prioritize consistent patterns over localized uncertainties and diminishing the impact of poorly sampled regions during skeleton discovery.

5.7 Summary

This chapter addresses the critical challenge of uncertainty quantification and reduction in predictive modeling, with a focus on improving reliability in scenarios where data collection is costly or limited. Reliable prediction under uncertainty is essential for informed decision-making in high-stakes domains where model outputs influence critical actions and outcomes. The chapter introduces two complementary strategies: learning high-quality prediction intervals to represent total uncertainty, and adaptive sampling methods designed to reduce epistemic uncertainty by strategically choosing new data points to sample. Furthermore, it explores the integration of symbolic regression within the proposed adaptive sampling framework, enabling the discovery of interpretable mathematical expressions that become increasingly accurate and stable as uncertainty is reduced. These methods are motivated by the need for predictive models that not only perform well on average but also transparently express confidence levels across the input space.

To this end, the chapter presents DualAQD, a loss function for training companion neural networks that produce accurate target estimates and high-quality PIs. DualAQD improves over existing approaches by balancing two key objectives, minimizing interval width and maintaining target coverage probability, through a self-adaptive coefficient that reduces the need for hyperparameter tuning. Furthermore, the chapter presents ASPINN, an adaptive sampling framework that uses PI-based metrics to estimate and minimize epistemic uncertainty iteratively. Experiments demonstrate that DualAQD consistently yields tighter and more reliable PIs compared to state-of-the-art baselines. ASPINN, in turn, accelerates learning and avoids overfitting by maintaining balanced data acquisition across the domain.

To assess the interpretability and informativeness of the resulting models, we integrated symbolic regression into the adaptive loop using a modified version of our SeTGAP method. This integration allowed us to monitor the evolution of symbolic interpretations during the AS process. While early expressions were unstable due to localized uncertainty or sparse data, the method reliably converged to the correct symbolic skeletons as epistemic uncertainty decreased. By unifying uncertainty quantification, adaptive sampling, and symbolic regression, our approach facilitates both efficient data acquisition and scientific insight, making it broadly applicable to domains where accurate, interpretable modeling must be achieved with limited experimental effort.

CHAPTER SIX

REAL-WORLD APPLICATION — PRECISION AGRICULTURE

In recent years, the field of agriculture has been undergoing a significant transformation, driven by the convergence of cutting-edge technologies [74, 141, 180, 184] and a growing need for more sustainable and efficient farming practices [69, 114]. Precision agriculture (PA), a data-driven approach to farming, has emerged as a pivotal solution to address the challenges faced by the agricultural industry. At the heart of PA lies the ability to analyze and leverage data to optimize almost every aspect of farming, optimizing the use of the available farming resources (e.g., water, nutrients, and pesticides), minimizing environmental impact, and maximizing profit [31, 115, 164].

In order to accomplish these objectives, models are created that establish relationships between input covariate factors, which are gathered from a range of sensors and advanced agricultural machinery, and outcome variables, such as crop yield [129]. Subsequently, these models are used to predict and analyze how outcome variables change across different rates of spatially variable input factors. Thus, the purpose of these models is to provide valuable insights for making informed and optimal decisions in agricultural management [144, 145, 146]. A critical aspect in the development of these models and simulations is On-Farm Precision Experimentation (OFPE), a framework that yields site-specific data about how fields respond to various management practices [33, 69].

Within the realm of PA, one approach has been relatively underutilized despite its potential to aid the objectives of the field: Symbolic Regression. At its core, SR seeks to discover mathematical equations that encapsulate the underlying dynamics of a given system. These equations, which aim to be concise and interpretable tools, not only offer the means to make accurate predictions and informed decisions but also provide important insights into the system. In the context of PA, one of the most important challenges is to model the response of crops to certain input covariate factors precisely. Among these, the nitrogen fertilizer rate (N-rate) and seeding rate receive particular attention due to their crucial role in crop management and the optimization of resource utilization [19, 114].

SR, with its potential to be used as a scientific discovery tool, offers a promising solution to this problem. This chapter explores the application of SR methods to PA, explicitly focusing on modeling and interpreting site-specific N-response curves, with the expectation that the developed methods can also generalize to other inputs, such as seed rate. Specifically, the main objective consists of the discovery of accurate, data-driven mathematical equations that encapsulate N-response dynamics of a constructed fertilizer management zone (MZ). By taking advantage of the power of advanced machine learning techniques and the low-cost data available to the farmers (e.g., open-source satellite imagery and data gathered from farmers' equipment), we aim to equip farmers with valuable tools for informed decision-making, resource optimization, and sustainable farming practices.

Furthermore, despite the growing sophistication of modeling techniques in PA, uncertainty remains a critical factor. In practice, limited data availability, high variability in field conditions, noisy sensor data, and the complexity of biological processes introduce significant uncertainty into yield predictions and response models. As a result, quantifying and managing this uncertainty becomes essential to ensure the reliability of model outputs and the decisions derived from them. Uncertainty quantification enables practitioners to identify low-confidence regions within a field, anticipate risk, and allocate resources more strategically. This is particularly important in site-specific management, where decisions such as fertilizer application rates must be tailored to local conditions with high confidence.

To address this need, this chapter also demonstrates how the uncertainty management techniques introduced in Chapter 5 can be applied within the context of precision agriculture. Thus, we generate prediction interval maps for crop yield using DualAQD, allowing farmers to visualize spatial patterns of prediction uncertainty and make decisions accordingly. Second, we apply ASPINN, our adaptive sampling method, to a simulated field site from a given MZ to demonstrate how targeted data acquisition can reduce epistemic uncertainty in crop modeling. These contributions aim to enhance the interpretability and trustworthiness of data-driven tools in PA, offering new avenues for reliable, site-specific decision-making.

6.1 Background

In this section, we briefly outline key agricultural concepts relevant to our work, and the techniques used for yield prediction and management zone analysis, which are essential for enabling the application of SR methods in the experiments conducted in this chapter.

6.1.1 On-Field Precision Experimentation

The PA application discussed in this chapter was developed within the context of an On-Field Precision Experimentation project. The OFPE methodology [69] is a flexible, data-driven framework that supports sustainable and profitable agricultural management by combining precision agriculture tools with adaptive, field-scale experimentation. It enables the implementation of on-farm trials that capture spatial and temporal variability in crop responses to varying input rates, such as fertilizer and seed rates. Unlike traditional approaches, OFPE explicitly quantifies uncertainty in management outcomes and presents it as probabilities of achieving improvements over farmers' existing practices.

OFPE operates through a repeatable, model-based process that leverages both open-source data and information collected directly from farmers' operations. Its primary goals are to generate field-specific crop response models, develop input recommendations that account for local variability, and offer farmers probabilistic assessments of different management strategies. The framework ultimately supports adaptive management by continuously refining recommendations based on updated observations, thereby aligning agricultural

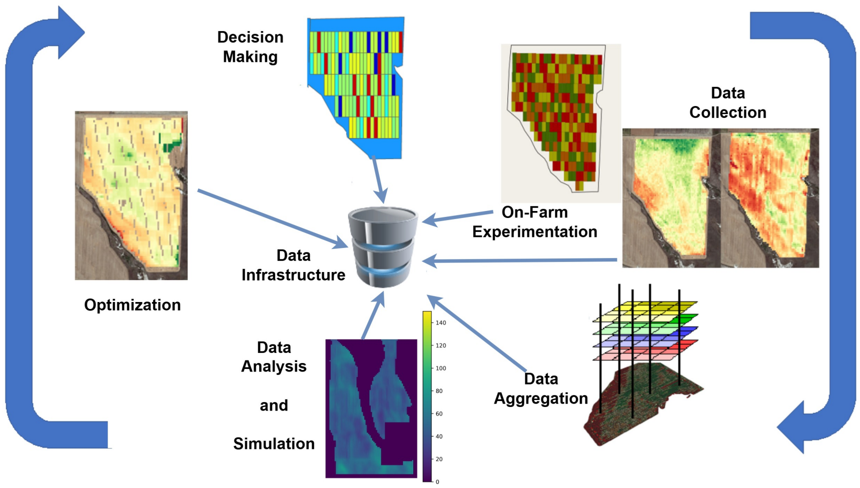


Figure 6.1: Steps of the OFPE framework.

decision-making with the evolving dynamics of each field.

The steps of the OFPE framework are outlined below and further explained in the following sections. As illustrated in Fig. 6.1, these steps form an iterative process.

- *Field Experimentation Setup:*
  - *Data Infrastructure:* Establish a database for managing field-specific data.
  - *On-Farm Experimentation:* Design experiments to study the interaction between crops, environment, and agronomic inputs.
- *Data Collection:* Apply experiments using PA equipment and gather data from both field sensors and open-source repositories.
- *Data Aggregation:* Combine on-farm and remote sensing data on a grid to produce analysis-ready datasets.
- *Ecological Modeling:* Train statistical and machine learning models to estimate crop responses based on environmental and input variables.
- *Results:*
  - *Simulation:* Simulate outcomes under various weather and economic scenarios.

  - *Optimization:* Identify site-specific optimal input rates based on predefined goals.
- *Decision Making and Iteration:* Deploy selected strategies while continuing experimentation to refine future recommendations.

Data Infrastructure: The OFPE methodology begins with the development of a centralized data management system to store and organize the spatiotemporal data collected through precision agriculture tools and open-source repositories. A secure spatial database was used to integrate farm-specific information (e.g., field boundaries, yield, protein levels, and input data) with external sources such as vegetation indices, weather, and soil characteristics. This system provides the digital infrastructure needed to support the analysis, modeling, and simulation of crop responses under varying conditions.

Experimentation: Field-specific experiments are designed to test different agronomic input rates, such as nitrogen fertilization or seeding density, across entire fields in representative patterns. The design is flexible to accommodate both research goals and equipment constraints. These on-field experiments are essential for generating the data required to model spatially varying crop responses [19]. Over time, the methodology aims to reduce the experimental footprint while maintaining statistical rigor. Farmers review and adjust the experimental designs before implementation to ensure agronomic relevance.

Data Collection: Once field experiments are implemented using variable rate application technology, actual input application data are retrieved from farm equipment and added to the database. Crop response data, such as yield or protein content, are collected at harvest using combine-mounted monitors.

Data Aggregation: All collected data must be aligned and harmonized into georeferenced datasets. This process involves resolving inconsistencies in spatial resolution across datasets to reduce uncertainty during analysis. For the development of OFPE, a standard 10-meter resolution was adopted, though this can be adjusted depending on equipment and field characteristics.

Data Analaysis: At the core of OFPE is the development of ecological models that quantify the relationship between crop responses, agronomic inputs, and environmental covariates. Due to the spatial and temporal variability in crop responses, no single model type suffices for all fields or years. Hence, models are selected on a per-field, per-year basis to address the bias–variance tradeoff and capture heteroscedastic behavior. The methodology avoids assuming fixed functional forms to reduce uncertainty and improve predictive performance. Since various types of prediction models can be considered, the model selection process aims to identify the best-performing model for each specific context.

Optimization: Since only one rate can be applied at a given location during an experiment, the optimal agronomic input rate must be inferred from the model rather than observed directly. Optimization goals are defined based on the user's priorities, which may be single-objective (e.g., maximizing protein for export markets) or multi-objective (e.g., balancing profit and environmental sustainability) [145]. The most common single-objective scenario involves maximizing net return by finding the rate at which added input cost no longer yields proportional profit. For sustainability-focused applications, optimization incorporates environmental costs, such as nitrogen losses through leaching or denitrification, and seeks to balance economic return with input minimization [146]. The result is a Pareto set of solutions that trade off between competing objectives.

Simulation: Due to the inherent uncertainty in weather and soil conditions, simulation is used to assess the robustness of management recommendations across possible future scenarios. Since optimal input rates may vary significantly across different weather years or price regimes, simulating outcomes under varying conditions provides a more reliable decision-support framework. This allows users to evaluate how a given strategy performs not only under average conditions but also under best- and worst-case scenarios, ultimately supporting more resilient on-farm decision making.

Decision Making and Iteration: In the final stage, farmers review the probabilistic outcomes generated by simulations and decide on an implementation strategy. They can fully deploy the selected management strategy, repeat full experimentation, or combine the two by deploying optimized rates while conducting additional low-density experiments. This mixed approach is encouraged, as ongoing experimentation enhances the statistical power of field-specific models and allows for refinement over time.

6.1.2 Crop Yield Prediction

PA has benefited recently from the confluence of the growing availability of sensors that can accurately and continuously collect information about fields [74, 184], the boom of machine learning, and the development of accessible and fast computational resources [141, 180]. In this context, crop yield prediction is one of the most beneficial areas of PA. It provides farmers with tools to make informed decisions, such as designing marketing plans [72] or determining the nitrogen fertilizer rates (N) needed to maximize farmer net return [19, 114].

The crop yield prediction task can be automated by using machine learning-based approaches. They generate computational models that attempt to approximate the field's behavior based on the observed relationships between multi-source covariate factors and crop yield, as noted in the simulation step of the OFPE framework (Section 6.1.1). Regardless of the data selected as explanatory variables, machine learning-based approaches pose the yield prediction problem as a regression task. As such, regression models are trained to estimate the crop yield response in terms of production, for example, bushels per acre (bu/ac), as accurately as possible. By doing so, the output model is treated as an implicit function of the input variables whose functional form is learned based on observed data. Previous works have proposed the use of regression models that process the information of each field site and (possibly) from its surroundings to estimate its corresponding yield value. Here, a **field site** is a small georeferenced region (e.g., a field site may represent an area of 10×10 m).

Yield values of neighboring regions are mutually dependent. Thus, it is natural to consider a crop yield prediction model that analyzes spatial neighborhoods of field sites [147]. What is more, it is feasible to consider a prediction model whose outputs correspond to the predicted yield values of all the sites within these neighborhoods, not only to the center of the neighborhood. Given that we are considering the use of models with two-dimensional (2D) inputs and 2D outputs, we refer to this problem as a two-dimensional regression.

Here, we discuss a yield prediction method we introduced in [129], which employs a Convolutional Neural Network (CNN) called Hyper3DNetReg. Hyper3DNetReg processes input data in the form of a multi-channel data cube, representing a small neighborhood of points in the field, where each channel corresponds to a different feature (e.g., nitrogen rate applied). Given a two-dimensional input with multiple channels, the network produces a two-dimensional raster output, with each pixel representing a predicted yield value for its corresponding input location. This approach is field-specific, meaning the models are trained on data from a given field and used to predict future yield maps for the same crop in that field using data from different years than those used for training. The motivation behind Hyper3DNetReg was that its two-dimensional output structure would enhance yield prediction accuracy compared to methods that generate single-output predictions, with accuracy assessed based on the ability to reproduce yield values from combine yield monitors.

<u>6.1.2.1 Dataset</u> For the experiments in [129], data were collected from four winter wheat dryland fields across two farms in Montana, each with distinct climates and soils. However, for the sake of brevity, this chapter focuses on two fields from different farms, referred to as

Field “A” and “B.” Yield maps were acquired during the harvest season, which corresponded to the month of August for all years of the study. Site-specific yield values were measured in bushels per acre (bu/ac) by a yield monitor mounted on a combine harvester. Every three seconds while traveling through the field, the yield monitor measured the volume rate of harvested material and integrated this flow rate to generate an estimate of the total amount of harvested material during that interval. The generated yield data values were then georeferenced using an onboard GPS. Finally, grid-like yield maps with equally spaced points were aggregated at a scale of 10 m. Considering that multiple points could be found within a 10×10m cell, we used the median to represent the yield value of that cell.

While the yield value represents the response variable in our regression problem, the explanatory variables correspond to a combination of remotely sensed data and on-ground data. Among the remotely sensed data, we considered SAR satellite images acquired by Sentinel-1. Sentinel-1 images contain two bands acquired using Vertical Transmit-Vertical Receive Polarization (VV) and Vertical Transmit-Horizontal Receive Polarization (VH). These images were obtained at the Ground Range Detected (GRD) level, which includes four pre-processing steps: speckle noise reduction, thermal noise removal, radiometric calibration, and ortho-rectification [50]. The resulting images have a spatial resolution of 10 m.

Besides the Sentinel-1 images, the collected dataset consists of a set of six raster features: nitrogen rate applied (lbs/ac), annual precipitation (mm), slope (degrees), elevation (m), topographic position index (TPI), and terrain aspect (radians). The nitrogen rate applied is gathered from the farmer’s application equipment and aggregated to the 10 m scale in the same manner as to yield responses, while precipitation is gathered at a 1 km scale from NASA’s Daymet V3 dataset and georeferenced to pixels via the intersection of 10 m points within the $\sim$ 1 km pixels. The amount of precipitation is measured during the current water year; that is, from November 1st to March 30th. Topographic variables are gathered at a 10 m scale from the USGS Digital Elevation Model. Thus, our resulting input can be viewed as an image data cube with eight channels in total, where each pixel has a resolution of 10×10 m (i.e., the same resolution as our yield maps). It is of particular importance to note that the nitrogen fertilizer was top-dress applied in March and the Sentinel-1 images were selected from the same month. Thus, we used data acquired in March to predict crop combine yield monitor values in August of the same year. Data were collected across three growing seasons (2016, 2018, and 2020).

In summary, the list of explanatory variables is as follows:

1. $\mathbf{x}^{Nr}$: Nitrogen rate (lb/ac).
2. $\mathbf{x}^{S}$: Topographic slope (degrees).
3. $\mathbf{x}^{E}$: Topographic elevation (m).
4. $\mathbf{x}^{TPI}$: Topographic position index.
5. $\mathbf{x}^{A}$: Topographic aspect (radians).
6. $\mathbf{x}^{P}$: Precipitation from the prior year (mm).
7. $\mathbf{x}^{VV}$ and $\mathbf{x}^{VH}$: Backscattering coefficients derived from Sentinel-I images.

Given that each field is represented with one large raster image, we must divide it into small 2D patches that our CNN regression model will process. To do this, we extract square patches using a 5×5 pixel window with a maximum overlap of 19 pixels (75% of the pixels of the window) so that we collect a sufficient number of training samples. Accordingly, the number of patches extracted from Field A in 2016, 2018, and 2020 was 484, 497, and 617, respectively, while Field B yielded 408, 316, and 317 patches for the same years.

6.1.2.2 Yield Prediction Model Let $\mathbf{X}$ and $\mathbf{Y}$ denote the input and target output of our prediction model, respectively. $\mathbf{X}$ is an image data cube of $W \times W$ pixels with n channels, where each channel corresponds to a different covariate. According to the dataset description given in Section 6.1.2.1, we set $W = 5$ and $n = 8$ for the datasets used in this work. Output $\mathbf{Y}$ is a two-dimensional image of $W_Y \times W_Y$ pixels ($W_Y \leq W$) that corresponds to the ground-

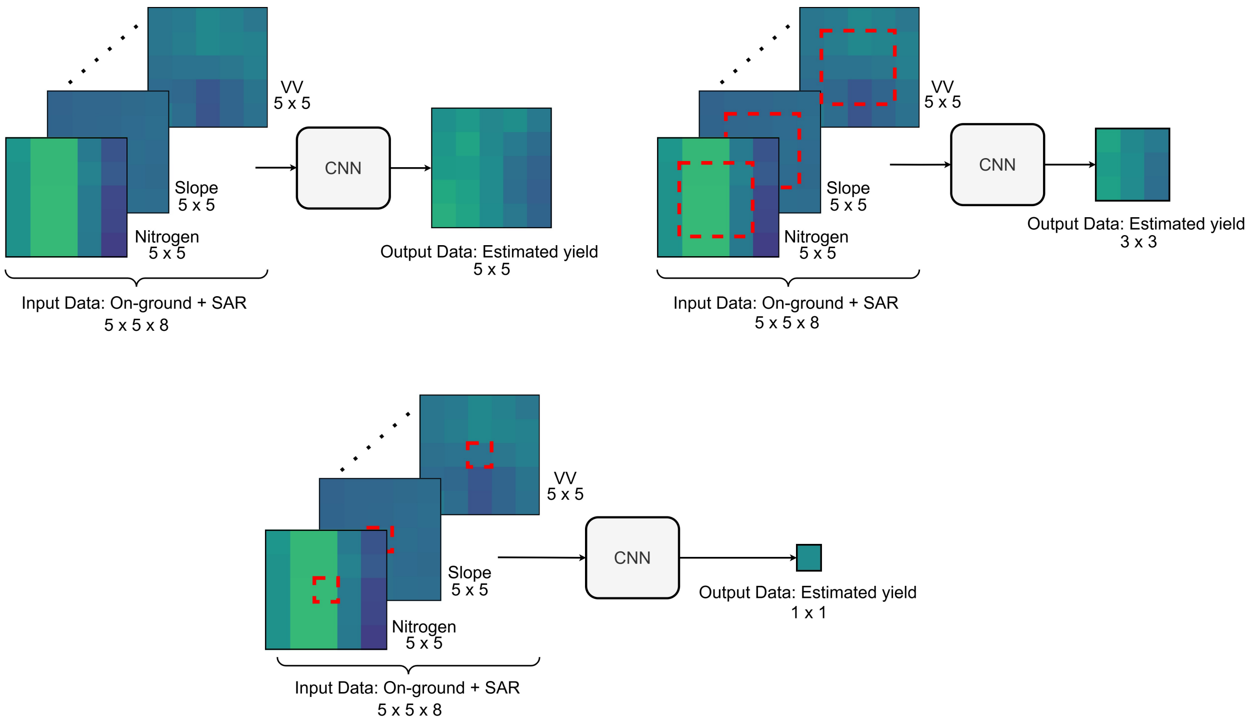


Figure 6.2: Yield prediction model using different output window sizes: **(a)** 5×5, **(b)** 3×3, and **(c)** 1×1 [129].

truth yield observed during the harvest season. Here, the value of the (i, j)–th pixel of Y is the observed yield, corresponding to the $(i + \frac{W-W_Y}{2}, j + \frac{W-W_Y}{2})$–th pixel of $\mathbf{X}$.

Our regression models are trained to capture the association between the target $\mathbf{Y}$ and the input $\mathbf{X}$ using a CNN architecture called Hyper3DNetReg. Let the function computed by Hyper3DNetReg and its corresponding weights be denoted by $\hat{f}(\cdot)$ and θ, respectively. Thus, the predicted yield $\hat{\mathbf{Y}}$ given input $\mathbf{X}$ is calculated as $\hat{\mathbf{Y}} = \hat{f}(\mathbf{X})$. We implemented models using three output window sizes (i.e., $W_Y = 5,\ 3,\ 1$), as depicted in Fig. 6.2, to evaluate the impact of W_Y on the effectiveness of Hyper3DNetReg. The output windows shown in Fig. 6.2.b and Fig. 6.2.c represent the predicted yield of the area enclosed by the dotted squares within the input images.

6.1.2.3 Hyper3DNetReg Architecture Hyper3DNetReg is a 3D—2D CNN architecture; that is, it incorporates three-dimensional and two-dimensional convolutional layers. It is based on our previously proposed network, Hyper3DNet [121], which was designed for hyperspectral classification and processes hyperspectral data cubes as input. Thus, Hyper3DNetReg adapts the Hyper3DNet architecture for 2D regression. Notably, two-dimensional inputs with multiple channels can be interpreted as data cubes (Fig. 6.2), as is the case with hyperspectral images. This allows us to leverage the 3D convolutional filters of Hyper3DNet to capture both spatial information from input neighborhoods and interactions between input covariates.

Table 6.1 shows the architecture of the Hyper3DNetReg network. Note that the general input shape of the network is $(5, 5, n, 1)$, which indicates that the input consists of a single data cube with a width and height of five pixels, and n input channels (i.e., the number of covariates). The Hyper3DNetReg network can be divided into two main modules: 3D feature extractor and 2D encoder. The former consists of four densely connected blocks [73] with 3D convolutional layers that are interconnected using skip connections to ensure maximum information flow throughout the network. The skip connections allow concatenating the outputs (using "CONCAT" layers) of the two preceding layers along the fourth dimension. Each convolutional layer is followed by a rectified linear unit activation layer, denoted as "ReLU" (where $\text{ReLU}(x) = \max(0, x)$), and a batch normalization layer, denoted as "BN".

The second module of the network, the 2D encoder, consists of five 2D separable convolution layers [168] (denoted as "SepConv2D"). These layers perform 2D convolution operations while minimizing the number of trainable parameters [29]. Since the 3D feature extractor outputs a tensor with three spatial dimensions, it must be reshaped into a 3D tensor suitable for processing by the SepConv2D layers. To mitigate overfitting, we include dropout units between layers in the 2D encoder. During training, these units randomly set elements of the preceding tensor to zero with a probability of 0.5 [170]. The final layer

Table 6.1: Hyper3DNetReg architecture.

Layer Name	**Kernel Size**	**Padding Size**	**Output Size**
Input	—	—	(5, 5, n, 1)
Conv3D + ReLU + BN	(3, 3, 3)	(1, 1, 1)	(5, 5, n, 32)
Conv3D + ReLU + BN	(3, 3, 3)	(1, 1, 1)	(5, 5, n, 32)
CONCAT	—	—	(5, 5, n, 64)
Conv3D + ReLU + BN	(3, 3, 3)	(1, 1, 1)	(5, 5, n, 32)
CONCAT	—	—	(5, 5, n, 96)
Conv3D + ReLU + BN	(3, 3, 3)	(1, 1, 1)	(5, 5, n, 32)
CONCAT	—	—	(5, 5, n, 128)
Reshape	—	—	(5, 5, $128 \cdot n$)
Dropout (0.5)	—	—	(5, 5, $128 \cdot n$)
SepConv2D + ReLU + BN	(3, 3)	(1, 1)	(5, 5, 512)
SepConv2D + ReLU + BN	(3, 3)	(1, 1)	(5, 5, 320)
Dropout (0.5)	—	—	(5, 5, 320)
SepConv2D + ReLU + BN	(3, 3)	(1, 1)	(5, 5, 256)
Dropout (0.5)	—	—	(5, 5, 256)
SepConv2D + ReLU + BN	(3, 3)	(1, 1)	(5, 5, 128)
SepConv2D + ReLU + BN	(3, 3)	(1, 1)	(5, 5, 32)
if $W_Y = 5$ or $W_Y = 3$:			
Conv2D + ReLU	(3, 3)	$\begin{cases} (1,1), & \text{if } W_Y = 5 \\ (0,0), & \text{elif } W_Y = 3 \end{cases}$	$\mathbf{(W_Y, W_Y, 1)}$
elif $W_Y = 1$:			
Conv2D + ReLU	(3, 3)	(0, 0)	(3, 3, 1)
Reshape	—	—	(9, 1)
FC	—	—	W_Y

depends on the output type: if the output is a 2D patch ($W_Y = 3$ or 5), a 2D convolutional layer reduces the number of channels to one and adjusts the output window size if needed; otherwise, if the output is a single predicted value ($W_Y = 1$), a fully connected layer produces the final prediction. Fig. 6.3 illustrates the network architecture for a 2D output.

For example, Fig. 6.4.b shows a predicted yield map for Field A in 2020, generated using a Hyper3DNetReg model trained on data from 2016 and 2018. This map was produced by applying a sliding window of 5×5 pixels ($W = 5$) across the entire field to estimate yield

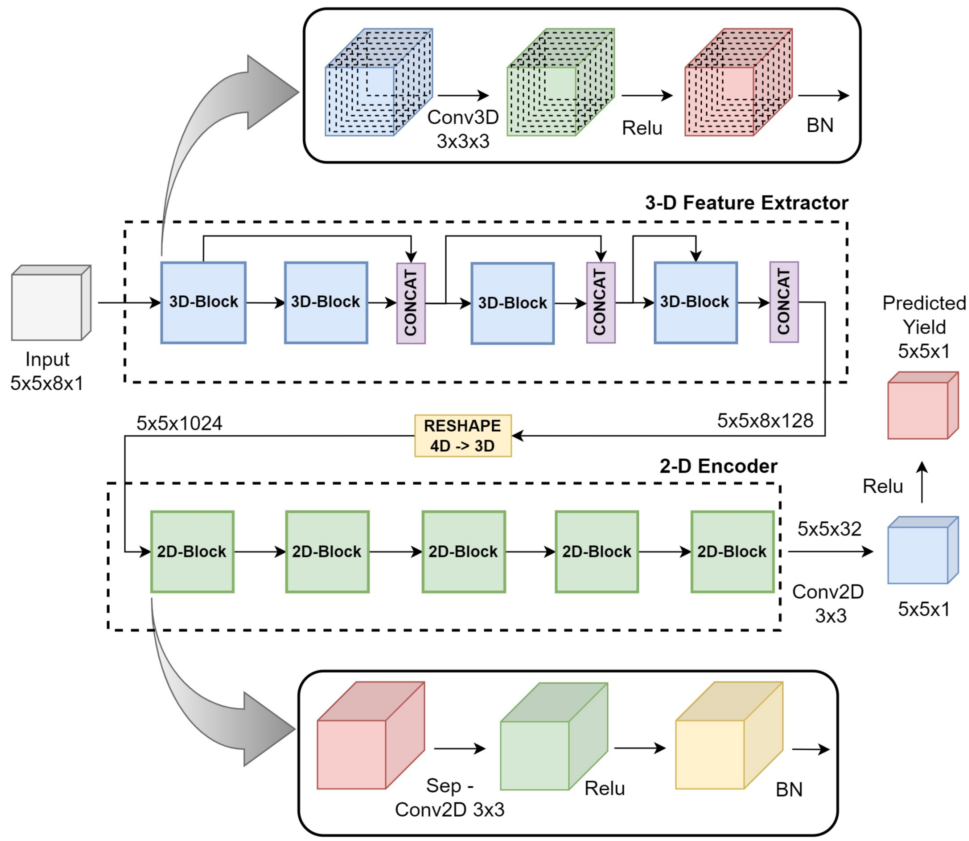


Figure 6.3: Hyper3DNetReg architecture with a $5\times5\times8\times1$ input and a $5\times5\times1$ output [129].

values at each cell. We set the output window size to $W_Y = 5$, constructing the final yield map by averaging the overlapping regions of neighboring predicted patches. Fig. 6.4.a presents the observed yield map after harvest, which serves as a reference for computing the square error map and structural similarity map against the predicted yield map, shown in Fig. 6.4.c and d, respectively. The error map was computed as the squared difference between the predicted and observed maps, from which we derive the root mean squared error (RMSE). The structural similarity map was obtained by calculating the structural similarity index metric (SSIM)[186] across all field locations. The SSIM index, a local metric for comparing the similarity between two small windows, ranges from -1, indicating perfect anti-correlation, to 1, indicating identical windows. In this example, we computed the SSIM

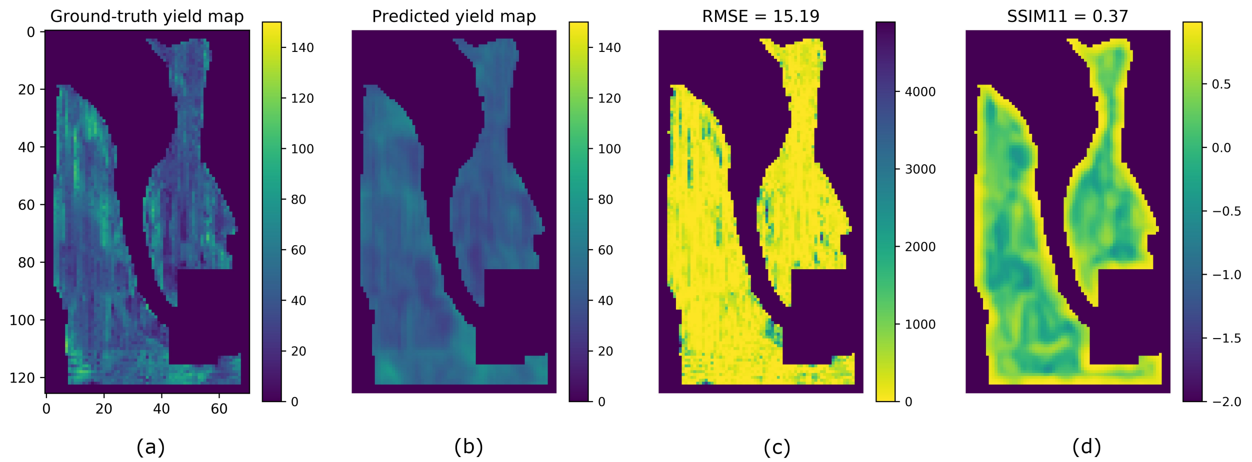


Figure 6.4: Yield prediction example of Field A for the year 2020. **(a)** Ground-truth yield map. **(b)** Predicted yield map using our Hyper3DNetReg with $W_Y = 5$. **(c)** Square error map. **(d)** Structural similarity map [129].

index using an 11-pixel window, with the average index value reported at the top of Fig. 6.4.c.

In [129], we conducted an extensive comparison across multiple prediction methods, additional winter wheat fields, and other Hyper3DNetReg configurations using $W_Y = 1$ and 3. Our results showed that Hyper3DNetReg with $W_Y = 5$ consistently achieves lower RMSE values and higher SSIM scores. As shown in Fig. 6.4, the predicted yield map exhibits low error values across the field. The structural similarity map confirms that regions of high and low predicted yield align well with actual high- and low-yield areas in the observed data.

6.1.3 Fertilizer Management Zones Clustering

In PA, management zones (MZs) are distinct sub-regions in a field with similar yield-influencing factors [83]. Different MZs account for the variability of factors within the field (e.g., soil composition) and, thus, vary in their requirements for specific treatments. These zones are areas with relative homogeneity where specific crop management practices are implemented, aiming to optimize crop productivity and reduce the environmental impact by reducing the overall fertilizer applied [36, 48]. Identifying MZs helps to constrain and homogenize the optimal treatment recommendations obtained during the optimization step of the OFPE framework discussed in Section 6.1.1.

Several methods have been proposed for the delineation of MZs. Some of them rely on historical yield data solely [2, 78] while others use information extracted from remote sensing data exclusively [55, 62]. Alternatively, certain approaches employ a combination of covariate factors, encompassing various soil properties, environmental factors, and topographic information [3, 112, 155, 166]. Most of these methods are based on unsupervised learning techniques; specifically, clustering methods such as k-means [77] and fuzzy c-means [3, 53, 155, 166], and principal component analysis (PCA) [112, 155, 166]. In addition, MZ delineation methods based on supervised learning, such as random forests (RFs) and support vector machines (SVMs) [55, 112] have emerged recently.

All previous works produce management zones using factors that are directly or indirectly related to crop productivity; that is, factors influencing the estimated total crop yield and the economic returns derived from it. In this context, we published in [124] the first work, to the best of our knowledge, that explicitly considers fertilizer responsivity as the main driver for defining MZs. Here, fertilizer responsivity refers to how the field reacts to different fertilizer rates, characterized by the shape of the response curve that describes yield variations as a function of applied nutrients. One of the key objectives of using MZs is to equip farmers with the necessary tools to make informed decisions about crop management, such as determining the appropriate amount of fertilizer required in each zone. As a consequence, our efforts should be directed toward establishing management zones where all included sites display comparable responsivity to varying fertilizer rates [78]. Note that this approach can be applied with any type of input responsivity that may be affected by multiple covariate inputs.

Fertilizer responsivity can be characterized using nitrogen fertilizer-yield response (N-response) curves. These curves exhibit the estimated crop yield values corresponding to a specific field site in response to all admissible fertilizer rates [123, 130]. For example, the rate values typically range between 0 and 150 pounds per acre (lbs/acre) for winter wheat. The shape of the N-response curve indicates the site's responsiveness to fertilizer, with a flat curve suggesting low responsivity and a steep curve suggesting high responsivity. Thus, in this section, we discuss the MZ clustering method we proposed in [124], which accounts for within-field variability of fertilizer responsivity based on approximated N-response curves.

To do this, we derive non-parametric response curves from observed data. Most response curves are generated based on parametric assumptions using methods such as linear regression or quadratic plateau regression; however, our experience suggests that N-response is much more complex than these models can capture. Our approach is based on a neural network (our Hyper3DNetReg model) acting as a regression model to map the covariate factors to crop yield, as suggested in [129]. Then, the network is used to generate approximated N-response curves across a range of admissible fertilizer rates [123]. The distinction in shape between two N-response curves is quantified by measuring the distance between the corresponding transformed curves in a reduced space, calculated using functional principal component analysis (fPCA) [153]. Hence, determining MZs relies on leveraging the shape dissimilarity of N-response curves as the key distance metric in cluster analysis.

It is worth pointing out that none of the existing methods for determining MZs attempted to provide explanations regarding the behavior of their results. This presents a significant limitation, particularly within the context of the growing area of explainable artificial intelligence. An approach to determining MZs that is inherently interpretable should enable farmers to discern cause-and-effect relationships between their inputs and outputs, enabling a more transparent decision-making process. Therefore, we used a *post-hoc* interpretability method to facilitate understanding the impact of various covariates on determining the MZ assignment for a given site. Specifically, we employed a counterfactual explanation (CFE) method, adapted from previous work [123], that solves a multi-objective optimization problem (MOO) using genetic algorithms. However, in this section, we focus exclusively on the clustering method for defining management zones, as the counterfactual explanation approach is not directly relevant to the application discussed in this chapter.

6.1.3.1 N-response Curve Generation N-response curves are one of the main tools used by agronomists to estimate the economic optimum nitrogen rate (EONR); that is, the N rate beyond which there is no actual profit for the farmers [20]. Over-application of N fertilizer can result in environmental pollution and waste of resources [146], while under-application can lead to reduced yields and economic losses [145]. Achieving the ideal balance in N application is contingent upon understanding the complex dynamics of N-response curves, which often exhibit non-linear and context-dependent behaviors. Another significant input factor affecting crop response is the seeding rate [106], which is used to determine the economic optimum seeding rate (EOSR). However, in the scope of this research, the fields from which we extracted data did not employ variable seeding rates. As a consequence, we face limitations in training accurate prediction models to approximate how different sections of the field respond to varying seeding rates.

Traditional methods of modeling N-response have frequently relied on simple models that fail to capture the complexities inherent to real-world agricultural systems. They often assume the use of plateau-type, quadratic, and exponential functions to model the relationship between N rate and crop yield [20, 80, 188]. Alternative approaches draw upon basic agronomic principles, such as Liebig response functions [1], to capture the nitrogen-yield relationship. It is worth pointing out that it has been shown that the selection of crop yield data models may yield drastically different EONR estimations [117]; thus, selecting a model that is faithful to the underlying dynamics of the field is a crucial task.

Conversely, our approach trains a regression CNN model to learn the mapping between covariate factors and crop yield values from observed data. To address this regression task, we use the Hyper3DNetReg architecture described in Section 6.1.2. Once trained, and assuming it effectively captures the underlying causal structure of the problem, the model can be used to generate approximate response curves. In previous work [123], we defined a response curve as a tool that allows for the responsivity analysis of a sensitive system to a particular "active feature". In addition, other stimuli that may influence the relationship between the response variable and the active feature were referred to as "passive features". In the context of the generation of N-response curves, the active feature corresponds to $\mathbf{x}^{Nr}$ (nitrogen rate), while the seven remaining variables represent the set of passive features.

An input data cube is represented as $\mathbf{X} = \{X_1, \ldots, X_n\}$, with X_1 corresponding to the $\mathbf{x}^{Nr}$ covariate, and the subsequent dimensions aligning with the remaining covariates based on the order outlined in Section 6.1.2.1. Let us denote the trained model as $\hat{f}(\cdot)$ so that, given an input $\mathbf{X}$, its estimated yield patch is denoted as $\hat{\mathbf{Y}} = \hat{f}(\mathbf{X})$. We produce estimated yield patches for all admissible values of $\mathbf{x}^{Nr}$ (bounded by $\mathbf{x}^{Nr}_{\text{min}} \leq \mathbf{x}^{Nr} \leq \mathbf{x}^{Nr}_{\text{max}}$) and stack them as a data cube $\hat{R}(\mathbf{X})$ (Fig. 6.5):

$$\hat{R}(\mathbf{X}) = \{\hat{f}(\mathbf{X}|\mathbf{x}^{Nr} = \mathbf{x}^{Nr}_{\text{min}}), \ldots, \hat{f}(\mathbf{X}|\mathbf{x}^{Nr} = \mathbf{x}^{Nr}_{\text{max}})\}, \tag{6.1}$$

As such, the (i, j)-th cell of the data cube $\hat{R}(\mathbf{X})$ (where $1 \leq i, j \leq 5$), denoted as $\hat{R}_{i,j}(\mathbf{X})$, represents the approximated response curve corresponding to the (i, j)-th cell of input $\mathbf{X}$.

The example in Fig. 6.5 shows the response curve generation process of all pixels within input $\mathbf{X}_{(lat,lon)}$, which represents the 5×5–pixel patch generated around coordinates (lat, lon) of the field. However, our goal is to generate N-response curves for all sites within the field. To do so, we move a 5×5–pixel sliding window throughout the entire field. Note that this approach results in overlapping predicted yield patches for consecutive points. Therefore, results obtained from neighboring points of the field must be aggregated.

Fig. 6.6 depicts the aggregation process of N-response curves obtained for a field point

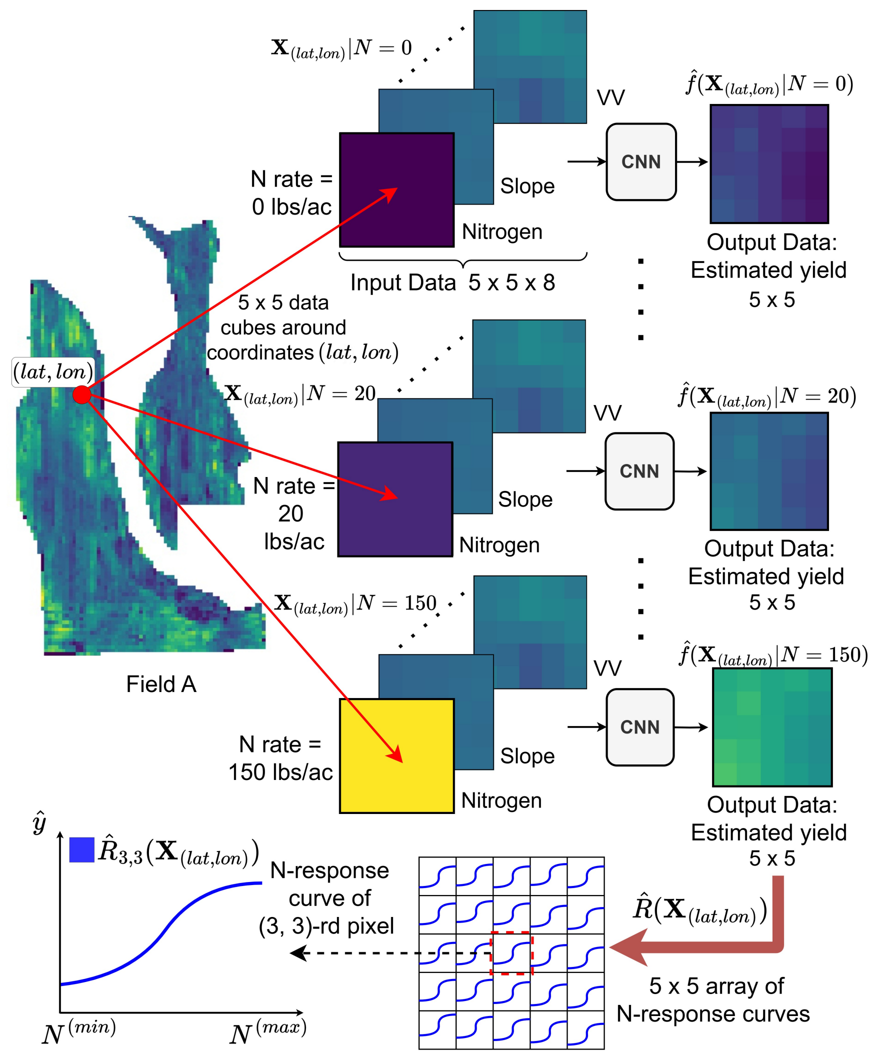


Figure 6.5: Generation of a 5×5 array of N-response curves generated around a field point at coordinates (lat, lon) [124].

at coordinates (lat, lon). This process involves considering all valid neighboring 5×5–pixel patches; i.e., patches whose centers are located within the field and that contain the point at (lat, lon), highlighted in red. The 9×9 window generated around the field point at (lat, lon) is denoted as $W_{(lat,lon)}$. For each of the valid patches in $W_{(lat,lon)}$, a 5×5 array of N-response curves is generated using Eq. 6.1. Then, the N-response curves corresponding to the field point at (lat, lon) are averaged, yielding a singular approximated N-response $\mathbf{r}(W_{(lat,lon)})$.

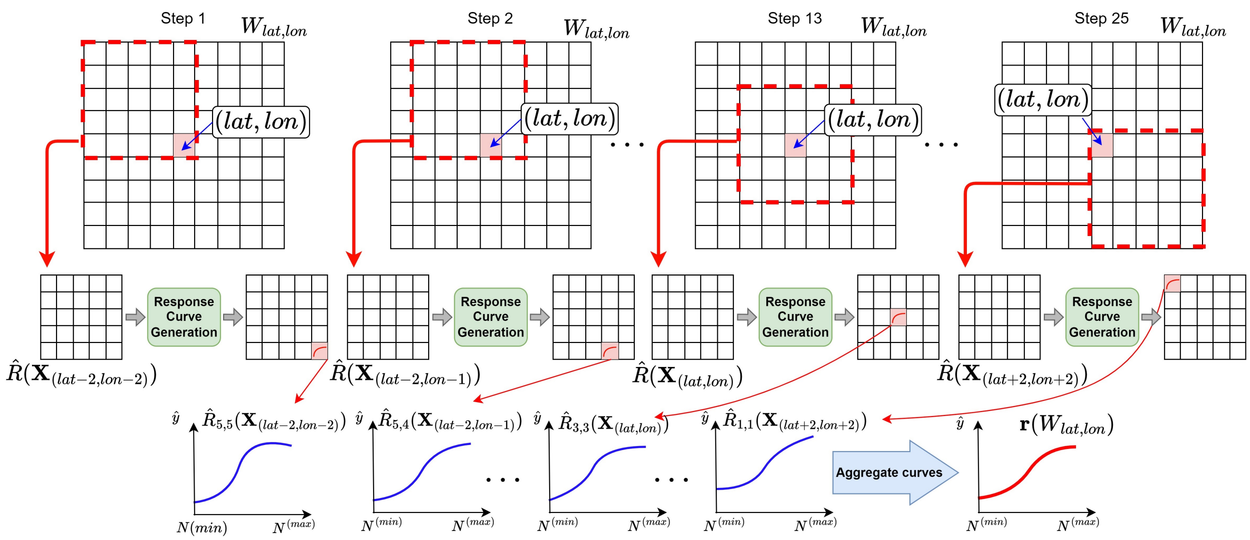


Figure 6.6: N-response curves aggregation for a field point at coordinates (lat, lon) [124].

This averaging process alleviates noisy outcomes and produces smoothed curves.

We state that the fertilizer responsivity of a given site is characterized by the shape of its N-response curve. As such, when comparing the shape of two or more N-response curves, the focus is not on their absolute estimated yield values. Hence, any vertical shifts are eliminated to obtain the aligned approximate N-response curve $\tilde{\mathbf{r}}(W_{(lat,lon)})$ as follows:

$$\tilde{\mathbf{r}}(W_{(lat,lon)}) = \mathbf{r}(W_{(lat,lon)}) - \min(\mathbf{r}(W_{(lat,lon)})).$$

6.1.3.2 Functional Principal Component Analysis We compute the set $\mathbf{R}$ comprising the aligned approximate N-response curves generated for all sites within the field. $\mathbf{R}$ constitutes a set of functional data whose samples are approximated N-response curves. Thus, fPCA can be applied to $\mathbf{R}$ to establish a distance metric conveying the difference in shape between N-response curves, as suggested in [123].

Functional Principal Component Analysis extends traditional PCA to analyze and represent variability in functional data [153]. As such, an N-response curve can be expressed as a linear combination of *functional* principal components (fPCs). Each fPC encapsulates a unique curve pattern, implying that curves with distinct shapes will be encoded using different fPC values. In this work, we suggest approximating an N-response curve using $K = 3$ fPCs, a choice justified by their ability to explain at least 99.5% of the variance of Field A. Thus, the proposed distance metric between curves $\mathbf{r}_1$ and $\mathbf{r}_2$ is:

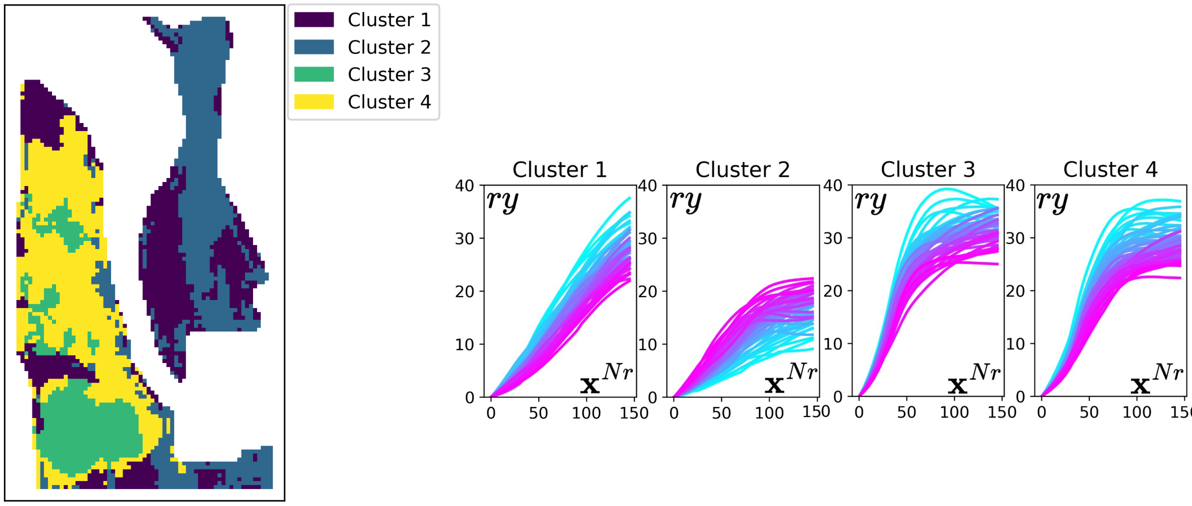
$$d(\mathbf{r}_1, \mathbf{r}_2) = \sqrt{\sum_{k=1}^{K} \left(v_k(\mathbf{r}_1) - v_k(\mathbf{r}_2)\right)^2}, \tag{6.2}$$

where $v_k(\mathbf{r}_j)$ is the value of the k-th principal component obtained after transforming $\mathbf{r}_j$.

6.1.3.3 Management Zone Clustering Using fuzzy c-means has become a prevalent approach in management zone delineation methods [155, 166, 199]. In fuzzy c-means, each data point is assigned a membership score indicating the extent to which it belongs to a specific cluster. A cluster centroid is computed as the mean of all data points, weighted by their respective cluster membership values.

Our approach consists of clustering all field sites based on their fertilizer responsivity so that each cluster corresponds to a distinct MZ. The process involves generating aligned approximate N-response curves for all field sites, followed by their transformation into a reduced 3D space through fPCA. Hence, the difference in fertilizer responsivity between curves (i.e., the difference in curve shape) is conveyed by their Euclidean distance in the transformed space. Therefore, the fertilizer responsivity distance (Eq. 6.2) serves as the distance metric for the fuzzy c-means algorithm.

Some approaches utilize indices such as the silhouette score, fuzziness performance index, and normalized classification entropy to determine the optimal number of clusters [3, 53]. However, these indices might face challenges in situations where clusters lack clear separation, as observed in the present context. Recall that all data points for clustering

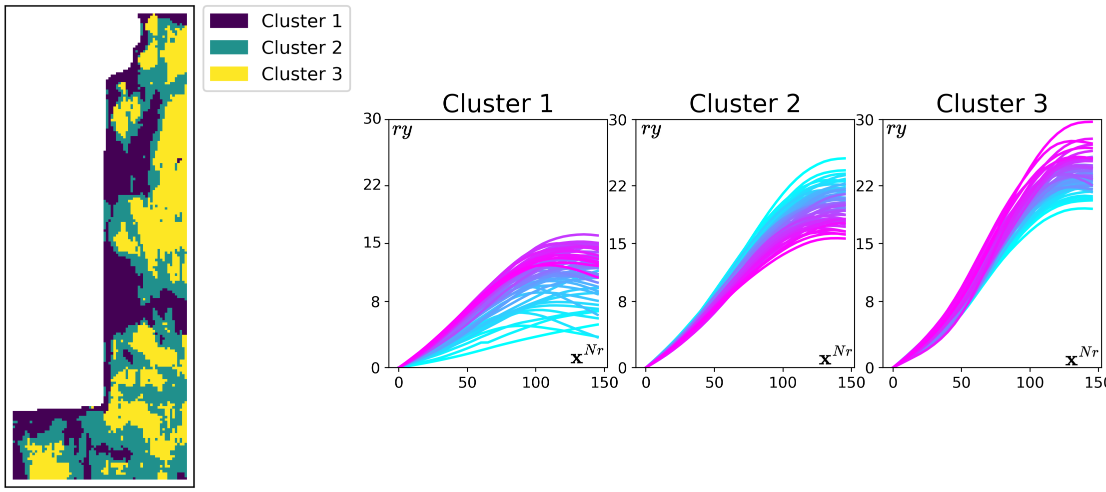


(a) (b)

Figure 6.7: Results obtained for Field A. **(a)** Delineated management zones and **(b)** Aligned approximated N-response curves for each MZ (N-rate $\mathbf{x}^{Nr}$ vs. relative yield ry) [124].

belong to the same field, leading to gradual changes in soil variability and, as a consequence, gradual changes in fertilizer responsivity. In addition, in PA, it is a common practice to specify between three and five MZs [55]. The decision to use up to five MZs is often influenced by practical considerations, such as the limitations of variable rate application machinery and the complexity of the field. For instance, using more than five zones may entail intricate zone boundaries, posing challenges for certain variable rate technologies to distinguish between closely situated zones. Following this convention, we chose a cluster count through visual inspection that minimizes the creation of redundant or highly variable MZs.

The MZ clustering method was evaluated on Fields A and B. Field A was divided into four MZs, whereas Field B was divided into three. This decision is justified by the fact that Field B is more homogeneous than Field A, thus their corresponding N-response curves show less variability. Note that an MZ does not need to form a contiguous region; rather, it may consist of scattered areas across the field that exhibit similar fertilizer responsivity. Fig. 6.7



(a) (b)

Figure 6.8: Results obtained for Field B. **(a)** Delineated management zones and **(b)** Aligned approximated N-response curves for each MZ [124].

shows the resulting MZs for Field A and Field B, respectively, along with fifty randomly selected, aligned approximate N-response curves from each MZ. As illustrated in Fig.6.7.b and Fig.6.8.b, the MZ delineation method effectively grouped N-response curves into clusters with consistent curve shapes. The variation in curve profiles across clusters reveals distinct patterns of fertilizer responsivity. Additionally, the approximate curves exhibit behavior consistent with agronomic expectations; i.e., sigmoid-*like* curves that capture an apparent yield loss after reaching a saturation point [188].

6.2 Prediction Intervals for Crop Yield Prediction

This section presents experiments aimed at producing prediction intervals for crop yield predictions in real-world agricultural fields. By visualizing not only predicted yield values but also the associated uncertainty across entire fields, these experiments enable the identification of areas with high or low predictive confidence. We evaluate and compare several PI-generation methods, highlighting their performance in capturing uncertainty in spatial crop yield predictions. The resulting PI maps provide an interpretable and informative layer of decision support, offering farmers insight into where predictions are most or least reliable.

For our experiments, we used data collected from Fields A and B. As previously described, three crop years of data were collected for each field. The information from the first two years was used to create the training and validation sets (90% of the data is used for training and 10% for validation). We use the same four PI-generation methods used for comparison in Section 5.5.1, namely, DualAQD [128], QD+ [160], QD-Ens [142], and MC-Dropout-PI [202]. These methods were compared using the results from the test set of each field, which consists of data from the last observed year and whose ground-truth yield map is denoted as Y. The test set was used to generate a predicted yield map of the entire field, $\hat{Y}$, and its corresponding lower and upper bounds, $\hat{Y}_L$ and $\hat{Y}_U$, respectively.

Fig. 6.9 shows the ground-truth yield map for Field A (darker colors represent lower yield values) along with the uncertainty maps obtained by the four compared methods and their corresponding $PICP$ and $MPIW$ values. Field A is used as a representative field for presenting our results, since we obtained similar results on the other fields. Here, we define the uncertainty map $U = \hat{Y}^u - \hat{Y}^\ell$ as a map that contains the PI width of each point of the field (darker colors represent lower PI width and thus lower uncertainty). That is, the wider the PI of a given point, the more uncertain its yield prediction.

We used four metrics to assess the behavior of the four methods (Table 6.2). First, we calculated the root mean square error ($RMSE_{test}$) between the ground-truth yield map Y and the estimated yield map $\hat{Y}$. Then, we considered the mean prediction interval width ($MPIW_{test}$) and prediction interval probability coverage ($PICP_{test}$). Note that k-fold or $k \times 2$ cross-validation cannot be used in this experimental setting. Thus, to help us explain the advantages of our method over the others, we introduce a new metric that summarizes the $MPIW_{test}$ and $PICP_{test}$ metrics shown in Table 6.2. Let $\overline{MPIW}_{test}$ represent the mean PI width after min-max normalization, using as upper bound the maximum $MPIW_{test}$ value

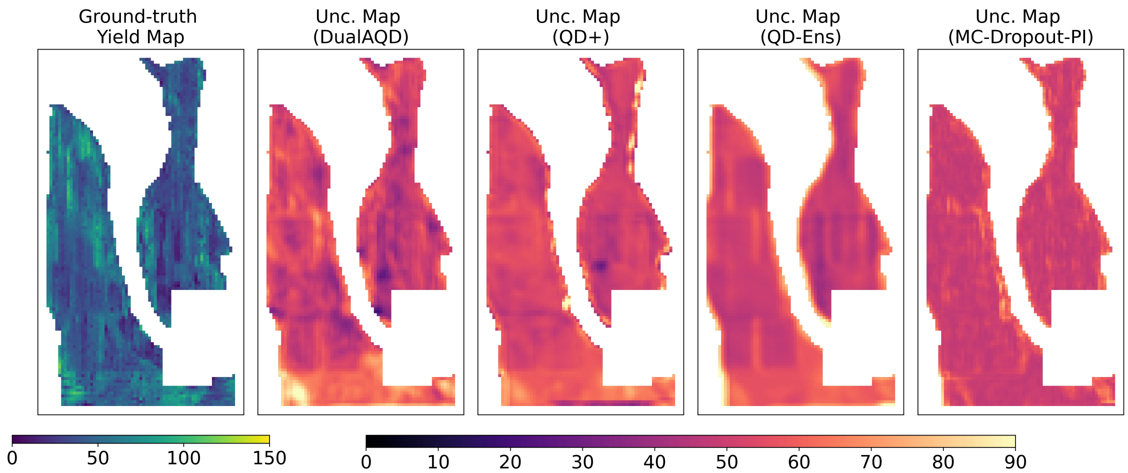


Figure 6.9: Uncertainty maps comparison for Field A.

among the four methods in each field. Let μ_ω denote the weighted geometric mean between $\overline{MPIW}_{test}$ and $(1 - PICP_{test})$ (i.e., the complement of the PI coverage probability) with $\omega \in [0, 1]$ being the relative importance between both terms. Then

$$\mu_\omega = (\overline{MPIW}_{test})^{\omega}(1 - PICP_{test})^{(1-\omega)}.$$

According to the HQ principle (Section 5.2) that aims to obtain narrow PIs and high probability coverage, low μ_ω values are preferable when comparing the performance of different PI-generation methods. Fig. 6.10 shows the comparison of the μ_ω metric obtained for each method on the three tested fields for different ω values. In order to summarize the behavior shown in Fig. 6.10 into a single metric, we calculated the integral $\mu = \int_0^1 \mu_\omega \, d\omega$. Since we seek to obtain low μ_ω values for various ω, low μ values are preferable. Bold entries in Table 6.2 indicate the method with the lowest μ.

We see in Table 6.2 that DualAQD yielded better $PICP_{test}$ values than the other methods for Field A, while, for Field B, QD-Ens had the highest $PICP_{test}$ value, albeit at the expense of generating excessively wide PIs. What is more, Fig. 6.10 shows that

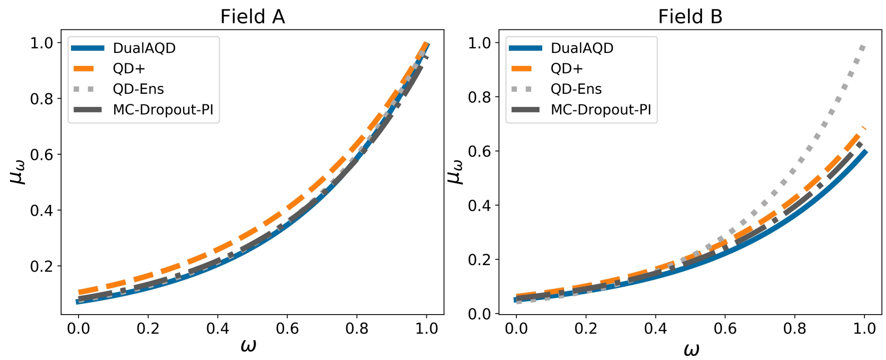


Figure 6.10: μ_ω vs. ω comparison on the yield prediction datasets.

Table 6.2: PI metrics evaluated on the yield prediction datasets.

Field	**Method**	$RMSE_{test}$	$MPIW_{test}$	$PICP_{test}$ (%)	μ
A	DualAQD	15.44	53.75	92.8	**.350**
	QD+	17.73	54.27	89.5	.397
	QD-Ens	15.55	53.99	92.3	.359
	MC-Dropout-PI	15.27	51.68	91.8	.355
B	DualAQD	11.16	43.45	94.9	**.221**
	QD+	11.83	50.17	93.7	.261
	QD-Ens	12.95	73.09	95.6	.306
	MC-Dropout-PI	10.83	47.18	94.4	.241

DualAQD obtained lower μ_ω values; as a consequence, it achieved the lowest μ value in each field (Table 6.2), which implies that it offers a better width-coverage trade-off in comparison to the other methods. Notice that Table 6.2 shows $PICP_{test}$ values lower than 95% for Field A. During training and validation, the coverage probability did reach the nominal value of 95%. Since the distribution of the test set (2020) differs from the one seen during training (2016 and 2018), the $PICP_{test}$ values may not be equal to those obtained during training. This illustrates the ability to show increased uncertainty when insufficient data is available for making reliable predictions, thus further motivating our adaptive sampling method (Section 5.1.2).

Fig. 6.9 shows that DualAQD was able to produce better distributed PIs for Field A (i.e., with a wider range of values) while achieving slightly better $PICP_{test}$ and $MPIW_{test}$

values than QD-Ens. This means that DualAQD is more dynamic in the sense that it outputs narrower PIs when it considers there is more certainty and wider PIs when there is more uncertainty. As a consequence, 54.4%, 44.3%, and 40.3% of the points processed by DualAQD on Field A have a smaller PI width than QD+, QD, and MC-Dropout, respectively, while still being able to cover the observed target values. Similarly, 88.7%, 65.3%, and 49.9% of the points processed by DualAQD on Field B have a smaller PI width than QD+, QD, and MC-Dropout while still covering the observed target values.

Finally, Fig. 6.9 shows that DualAQD indicates higher uncertainty in the lower region of the field, which received an N rate value that was not used in previous years (i.e., it was not available for training). Similarly, regions of high yield values are related to high nitrogen rate values; however, considerably fewer training samples of this type exist, which would logically lead to greater uncertainty. Thus, there is more uncertainty when predicting regions that received high nitrogen rate values, and this is represented effectively by the uncertainty map generated by DualAQD but not the compared methods. It is worth mentioning that even though DualAQD showed some degree of robustness empirically when given previously unseen samples, neural network-based PI generation methods do not offer any guarantee for the model's behavior for out-of-distribution samples.

6.3 Parametric N-response Curve Learning

In prior work [130], we proposed a counterfactual method that aims to identify the features with the highest impact on N responsivity at a local and global scale by solving an MOO problem using a GA-based approach. An impact on responsivity is related to a change in which a given location (local scale) or the entire field (global scale) reacts to different N rates. Based on this approach, we presented in [124], a *post-hoc* interpretability method that generates CFEs that reveal the influence of covariate factors on MZ assignments.

Although these are considered explainable methods since they allow us to understand the relevance of the input features, they do not provide a thorough understanding of the mathematical behavior of the field's N responsivity at various locations. What is more, to the best of our knowledge, no previous work has studied the problem of learning the mathematical expressions that describe site-specific N-response curves from data without pre-specifying the functional form of these expressions.

Let $y_{(lat,lon)}$ represent the observed yield at a field site with coordinates (lat, lon). Furthermore, let $\mathbf{X}_{(lat,lon)}$ represent a set of multiple covariate factors that describe the state of the field at position (lat, lon), and potentially its neighboring areas. The underlying yield function of the field is denoted as $f(\cdot)$ and $y_{(lat,lon)} = f(\mathbf{X}_{(lat,lon)})$. In practice, f is a complex multivariate system with unknown functional form. Nevertheless, tasks like N-rate optimization, which allows for profit maximization and environmental impact maximization [69], do not require estimating the full functional form of $f(\mathbf{X}_{(lat,lon)})$. Instead, N-rate optimization only analyzes the functional relationship between the N-rate variable and the predicted yield values. This relationship is typically represented using N fertilizer-yield response curves, also known as N-response curves, as explained in Section 6.1.3.1.

The experiments in this section aim to estimate the functional form of N-response curves for one winter wheat dryland field, Field A. Traditionally, N-response curves are modeled using a single parametric function for the entire field [20, 81]. However, previous studies suggest that the functional form of these curves varies across different field regions due to factors such as terrain slope and soil composition [123, 124]. In particular, our prior research [124] provided evidence, some of which is revisited in Section 6.1.3, that fields can be divided into MZs based on fertilizer responsivity, where the shape of the estimated N-response curves differentiates field sites for clustering.

Building on the results from Section 6.1.3, which clusters Field A into four MZs, we assume that all sites within a given MZ share the same functional form due to the shape similarity of the curves within each cluster (Fig. 6.7). An N-response curve consists of input–

response pairs, where the nitrogen rate $\mathbf{x}^{Nr}$ serves as the input and relative yield ry as the response. Consequently, estimating the functional form of N-response curves for an MZ can be formulated as a multi-set symbolic skeleton problem (MSSP).

Given the pre-computed N-response curves for all field sites, $\mathbf{R}$ (Section 6.1.3.2), we describe the procedure for generating skeleton expressions that capture the mathematical behavior of the N-response curves associated with the z-th management zone (MZ), denoted by $\mathbf{R}_z$. This procedure is outlined in Algorithm 6.1 and builds upon the skeleton generation method introduced in Algorithm 4.1. In Line 4, the function $\texttt{selectRandomCurves}(\mathbf{R}_z, N_s)$ is used to randomly select N_s N-response curves, forming the collection $\tilde{\mathbf{D}}_{\mathbf{R}_z}$. These selected curves are then input to the Multi-Set Transformer g to generate n_B skeletons via a diverse beam search strategy. The curves are standardized using z-score normalization, and the N rates are rescaled to the range $[-10, 10]$ to match the input format required by the Multi-Set Transformer. This process is repeated n_{cand} times, each time with a newly sampled $\tilde{\mathbf{D}}_{\mathbf{R}_z}$ composed of curves from different sites within the MZ, thereby enhancing input diversity and promoting the discovery of more generalizable skeletons. Then, any duplicate or mathematically equivalent skeletons are removed. The resulting list of skeletons is denoted by $\texttt{genSks}_z = \hat{\mathbf{e}}_1(\mathbf{x}^{Nr}), \dots, \hat{\mathbf{e}}_{|\texttt{genSks}_z|}(\mathbf{x}^{Nr})$, where $|\texttt{genSks}_z| \leq n_{\text{cand}} n_B$. These skeletons are selected based on their ability to maximize the Pearson correlation with a randomly selected test collection $\tilde{\mathbf{D}}^{(\text{test})}_{\mathbf{R}_z}$ using the GA-based optimization function `fitCoefficients`.

For our experiments, we set $n_B = 3$ and $n_{\text{cand}} = 5$, as higher values did not yield more distinct skeletons across all tested problems. Table 6.3 shows the skeletons derived by the MST for each MZ. Furthermore, we evaluated the suitability of the obtained skeletons for each MZ and compared them to two traditional N-response models: quadratic-plateau [20] and exponential [81]. For each method, field site, and MZ, we fit the skeleton's coefficient values to minimize the distance to the corresponding N-response curve. To do this, we use the GA-based optimization function $\hat{\mathbf{r}}_j^{(z)} = \texttt{fitCoefficients}_{\text{MSE}}(\hat{\mathbf{e}}(\mathbf{x}^{Nr}), \tilde{\mathbf{r}}_j^{(z)})$ modified to

Algorithm 6.1 N-response Curve Skeleton Generation

Input: Aligned N-response curves of the z-th MZ, $\mathbf{R}_z$; number of input sets N_S; Multi-Set Transformer g; number of skeleton candidates n_{cand}; beam size n_B

Output: Generated list of candidate skeletons for the z-th MZ $\texttt{genSks}_z$; corresponding correlation values $\texttt{corrVals}_z$

function $\textsc{generateNrespSks}(\mathbf{R}_z, N_s, g, n_{\text{cand}}, n_B)$
  $\text{genSks}_z \leftarrow [\,]$
  for each $i \in (1, n_{\text{cand}})$ **do**
    $\tilde{\mathbf{D}}_{\mathbf{R}_z} \leftarrow \texttt{selectRandomCurves}(\mathbf{R}_z, N_s)$
    $\text{genSks}_z.\texttt{append}(g(\tilde{\mathbf{D}}_{\mathbf{R}_z}, \Theta; n_B))$
  $\text{genSks}_z \leftarrow \text{removeDuplicates}(\text{genSks}_z)$ ▷ $\text{genSks}_z = \{\hat{\mathbf{e}}_1(\mathbf{x}^{Nr}), \ldots, \hat{\mathbf{e}}_{|\text{genSks}_z|}(\mathbf{x}^{Nr})\}$
  $\tilde{\mathbf{D}}^{(\text{test})}_{\mathbf{R}_z} \leftarrow \texttt{selectRandomCurves}(\mathbf{R}_z, N_s)$
  $\text{corrVals}_z \leftarrow \text{zeros}(|\text{genSks}_z|)$
  for each $k \in (1, n_{\text{cand}})$ **do**
    $\text{corrVals}_z[k] \leftarrow \texttt{fitCoefficients}(\hat{\mathbf{e}}_k(\mathbf{x}^{Nr}), \tilde{\mathbf{D}}^{(\text{test})}_{\mathbf{R}_z})$
  $\text{genSks}_z \leftarrow \texttt{sortSkeletons}(\text{genSks}_z, \text{corrVals}_z)$
  if $|\text{genSks}_z| > n_{\text{cand}}$ **then**
    $\text{genSks}_z, \text{corrVals}_z \leftarrow \text{genSks}_z[1 : n_{\text{cand}}], \text{corrVals}_z[1 : n_{\text{cand}}]$
  return $\text{genSks}_z, \text{corrVals}_z$

return the fitted function $\hat{\mathbf{r}}_j^{(z)}$ that minimizes the MSE error with respect to the j-th aligned curve in $\mathbf{R}$, $\tilde{\mathbf{r}}_j^{(z)}$. We report the mean error $\bar{r}$ obtained considering all sites within each MZ:

$$\bar{r} = \frac{1}{|\mathbf{R}_z|} \sum_{\tilde{\mathbf{r}}_j^{(z)} \in \mathbf{R}_z} \left| \hat{\mathbf{r}}_j^{(z)} - \tilde{\mathbf{r}}_j^{(z)} \right|.$$

Unlike traditional approaches that assume a single form for the entire field, modeling each MZ with a distinct functional form is agronomically justified, as different regions often exhibit varying soil types and terrain characteristics. This hypothesis is supported by the results in Table 6.3, which show that using tailored skeletons leads to lower fitting errors, indicating their greater suitability for modeling the field's N-response curves.

Fig. 6.11 shows two fitted curves for each of the four MZs in Field A, derived from the skeleton expressions identified by our symbolic regression approach. Additionally, Fig. 6.12

Table 6.3: Comparison of skeleton prediction results for Field A

Method	MZ	Functional Form	$\bar{r}$	Method	MZ	Functional Form	$\bar{r}$
Quadratic-plateau	1	$c_1 + c_2\,(\min(\mathbf{x}^{Nr}, c_3) + c_3)^2$	0.0695	Quadratic-plateau	3	$c_1 + c_2\,(\min(\mathbf{x}^{Nr}, c_3) + c_3)^2$	0.1028
Exponential		$c_1(1 - \exp(c_2 + c_3\,\mathbf{x}^{Nr})) + c_4$	0.2303	Exponential		$c_1(1 - \exp(c_2 + c_3\,\mathbf{x}^{Nr})) + c_4$	0.1965
SeTGAP		$c_1 + c_2 \tanh(c_3 + c_4\,\mathbf{x}^{Nr}))$	**0.0620**	**SeTGAP**		$c_1 + c_2\,\mathbf{x}^{Nr} + c_3 \cos(c_4 + c_5\,\mathbf{x}^{Nr})$	**0.0683**
Quadratic-plateau	2	$c_1 + c_2\,(\min(\mathbf{x}^{Nr}, c_3) + c_3)^2$	0.0725	Quadratic-plateau	4	$c_1 + c_2\,(\min(\mathbf{x}^{Nr}, c_3) + c_3)^2$	0.0615
Exponential		$c_1(1 - \exp(c_2 + c_3\,\mathbf{x}^{Nr})) + c_4$	0.1825	Exponential		$c_1(1 - \exp(c_2 + c_3\,\mathbf{x}^{Nr})) + c_4$	0.2249
SeTGAP		$c_1 + c_2\,\tanh(c_3 + c_4\,\mathbf{x}^{Nr})$	**0.0355**	**SeTGAP**		$c_1 + c_2 \exp(c_3 \sin(c_4 + c_5\,\mathbf{x}^{Nr}))$	**0.0448**

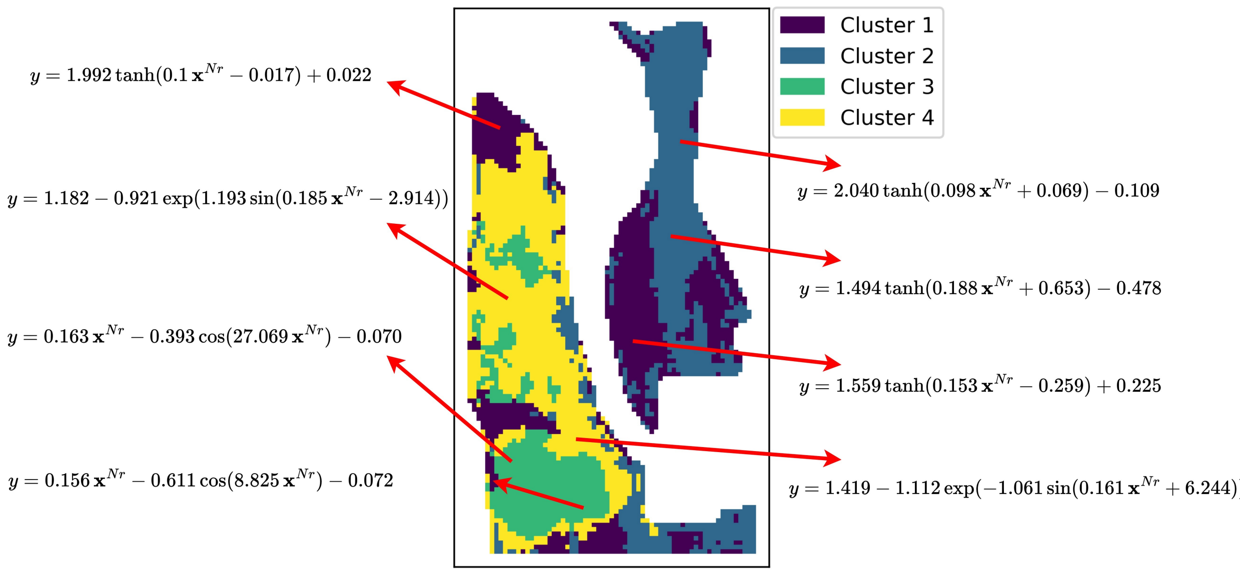


Figure 6.11: Example of fitted N-response curves using the identified skeleton for each MZ.

compares these fitted curves against the corresponding aligned N-response curves. The close agreement in shape between the reference and fitted curves confirms that the identified functional forms were appropriate for all cases. It is worth noting that although Fig. 6.7b reveals distinct fertilizer response patterns between the N-response curves of MZ 1 and MZ 2, both can be described by the same skeleton expression: $c_1 + c_2 \tanh(c_3 + c_4, \mathbf{x}^{Nr})$.

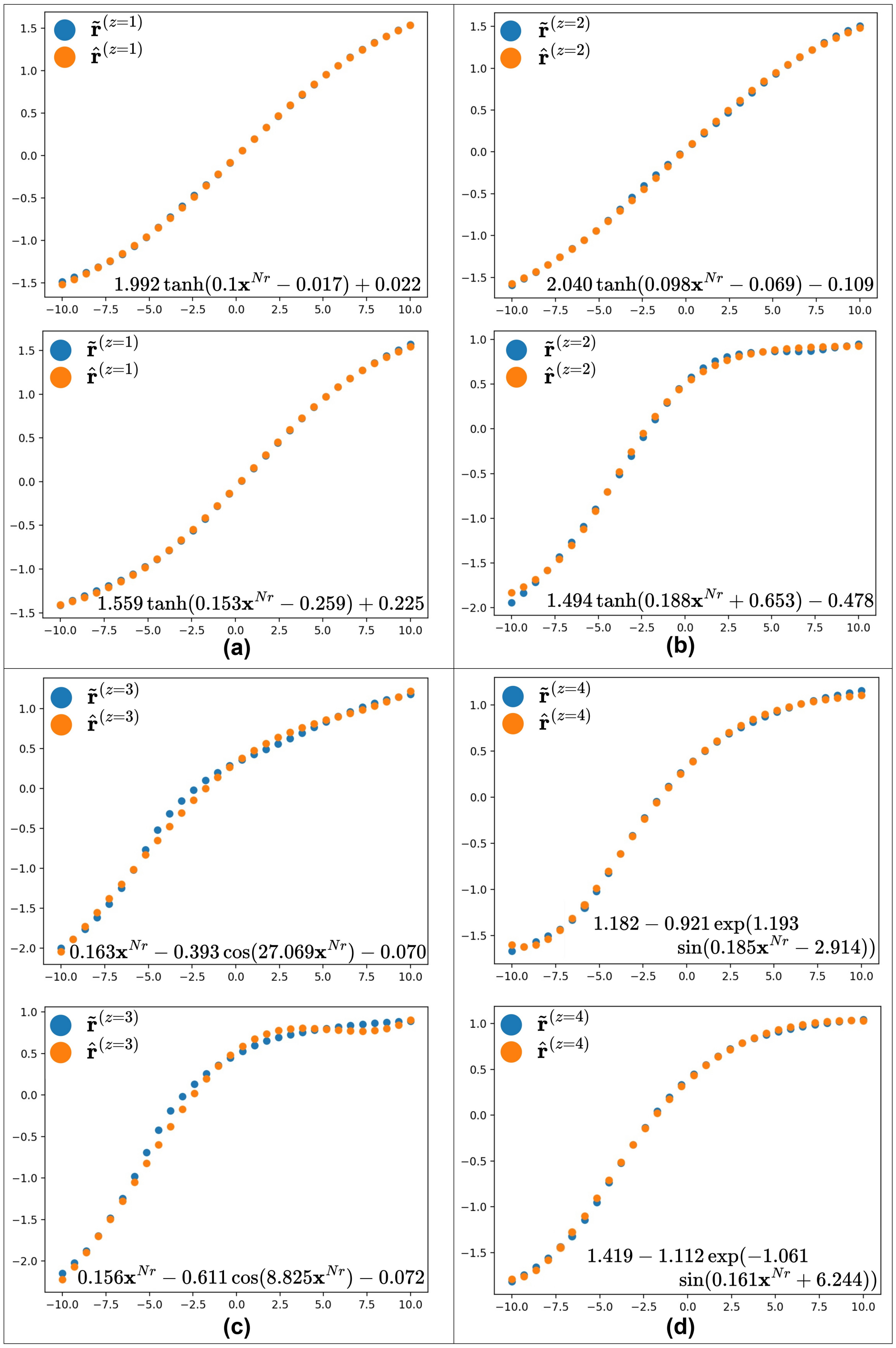


Figure 6.12: Comparison between NN-generated N-response curves $\tilde{\mathbf{r}}$ and fitted curves $\hat{\mathbf{r}}$ from **(a)** MZ 1, **(b)** MZ 2, **(c)** MZ 3, and **(d)** MZ 4. Equation $\hat{\mathbf{r}}$ at the bottom of each plot.

6.4 Adaptive Sampling with Simulated Field Data

Motivated by the fertilizer management zones analysis discussed in Section 6.1.3, this section introduces a multi-dimensional problem that simulates an agricultural field site. Note that actual real-world data cannot be considered for a comparative AS study. There are multiple reasons for this. First, a given field site receives a single experimental rate during the fertilization stage and its effects are observed during the harvest season (e.g., five months for winter wheat). Second, additional samples at the same site require collecting data over multiple years. Third, when comparing different AS methods, each may recommend distinct experimental rates, which cannot be implemented simultaneously within a single growing season. Fourth, real-world conditions, such as unforeseen environmental factors and concept drift, introduce additional complexity that prevents the isolation of the effects attributable to the AS strategies. Therefore, simulations based on the characteristics of a real field offer a controlled environment in which different AS methods can be evaluated under identical conditions, enabling fair and consistent comparisons.

6.4.1 Simulated Field Data for Adaptive Sampling

In Section 6.1.3, the functional forms of N-response curves for different MZs within an actual winter wheat field were derived as symbolic skeleton expressions using the SeTGAP methodology introduced in Chapter 4. As previously defined, a symbolic skeleton expression is a representation of a mathematical expression that does not specify numerical values.

The experiments presented in this section simulate a field site whose behavior is modeled based on the symbolic skeleton extracted from MZ 2 of Field A (Table 6.3). For instance, the relationship between yield, y, and N rate, $\mathbf{x}^{Nr}$, at a given site is given by the skeleton

$$y = c_1 + c_2 \tanh(c_3 + c_4 \, \mathbf{x}^{Nr}), \tag{6.3}$$

where c_1–c_4 are placeholder constants. Based on this skeleton, we consider the yield function:

$$y = f(\mathbf{x}) = \frac{\mathbf{x}^P}{15} + \left(\frac{\mathbf{x}^A}{\pi} + 1\right) \tanh\left(\frac{0.1\,\mathbf{x}^{Nr}}{3\mathbf{x}^{VH} + 2}\right) + \varepsilon_a(\mathbf{x}), \tag{6.4}$$

where $\mathbf{x} = [\mathbf{x}^P, \mathbf{x}^A, \mathbf{x}^{VH}, \mathbf{x}^{Nr}]$ comprises the following site-specific covariates: annual precipitation (mm), terrain aspect (radians), Sentinel-1 backscattering coefficient from the Vertical Transmit-Horizontal Receive Polarization band, and applied N rate (lbs/ac), respectively. The aleatoric noise is modeled as $\varepsilon_a(\mathbf{x}) = \mathcal{N}(0, (\mathbf{x}^P + \mathbf{x}^{Nr})/150)$.

Let us justify the selection of these underlying and noise functions. From comparing Equations 6.3 and 6.4, it is observed that the constant placeholders were assigned the following values: $c_1 = \frac{\mathbf{x}^P}{15}$, $c_2 = (\frac{\mathbf{x}^A}{\pi} + 1)$, $c_3 = 0$, and $c_4 = (\frac{0.1}{3\mathbf{x}^{VH}+2})$. Below, we analyze each of these expressions. It is important to clarify that our goal is not to derive precise functional expressions for the coefficients c_1–c_4 in order to model the underlying function of the field accurately. Rather, our aim is to design a yield function that exhibits behavior consistent with agronomic principles, informed by past observations of an actual field.

In previous work [123], we utilized counterfactual explanations to analyze the influence of a set of "passive features" over the shape of the response curves generated for the response variable and a selected "active feature." In the context of this work, we are interested in the analysis of N-response curves, which allow for the analysis of the site-specific responsivity to all admissible values of the N fertilizer rate, which serves as the selected "active feature." Nevertheless, the shape of the N-response curves may be influenced not only by the relationship between the response variable and the active feature but also by other factors, termed "passive features." In this case, we consider the variables $\mathbf{x}^P$, $\mathbf{x}^A$, and $\mathbf{x}^{VH}$ as passive features that may affect a field site's responsivity.

In [123], we studied an early-yield prediction dataset of winter wheat. The findings indicate that, although precipitation $\mathbf{x}^P$ is a critical factor for crop production, it has minimal impact on N responsivity. This suggests that $\mathbf{x}^P$ is independent of the other features and only shifts the N-response curves vertically without altering their shape. In Eq. 6.3, c_1 acts as an independent term responsible for vertical shifts, which is why it is modeled as a function of $\mathbf{x}^P$. In addition, variables $\mathbf{x}^A$ and $\mathbf{x}^{VH}$ were identified as having a significant impact on the shape of N-response curves, making them key factors in this study.

Furthermore, c_2 stretches the N-response curves vertically. We argue this behavior corresponds to that of the terrain aspect $\mathbf{x}^A$ (i.e., the slope orientation). In terrain with varying elevations located in the Northern Hemisphere, regions that are facing north and east have limited sunlight during the day and are more prone to snow retention. These are factors that may affect the responsiveness of the fertilizer. For instance, we observed that regions facing north ($\mathbf{x}^A = 0$) correspond to flatter N-response curves than those facing south ($\mathbf{x}^A = \pi$). Our simulated field site is being modeled as a field site that is located within a sub-region of an actual field whose $\mathbf{x}^A$ values vary between $\pi/4$ and $\pi/2$. Within this sub-region, we found that considering $c_2 = (\frac{\mathbf{x}^A}{\pi} + 1)$ adjusts reasonably well to the variation in vertical stretching of the estimated N-reponse curves.

Coefficient c_3 causes horizontal shifts, which are not observed in the estimated N-response curves obtained for the studied area. Hence, for the sake of simplicity, we select c_3 to be equal to 0. Finally, c_4 controls the horizontal stretching of the curve. A lower c_4 value causes the output of the function to increase more gradually as $\mathbf{x}$ increases. Conversely, a higher value of c_4 leads to a steeper increase, causing the function to reach its saturation point more rapidly. Dry soil has a lower capacity to retain and absorb nutrients and, thus, reaches the saturation point more quickly than moist soil when applying N fertilizer. Therefore, we model c_4 as an inverse function of $\mathbf{x}^{VH}$: $c_4 = (\frac{0.1}{3\mathbf{x}^{VH}+2})$.

On the other hand, the heteroscedastic aleatoric noise is modeled as $\varepsilon_a(\mathbf{x}) = \mathcal{N}(0, (\mathbf{x}^P + \mathbf{x}^{Nr})/1500)$. This formulation reflects the observation that both precipitation ($\mathbf{x}^P$) and nitrogen fertilizer rate ($\mathbf{x}^{Nr}$) contribute to variability in agricultural yield. Increased precipitation can introduce uncertainty through effects such as runoff and fluctuating soil moisture, which influence nutrient availability and plant health. Similarly, nitrogen application does not always yield consistent improvements; excessive levels may lead to diminishing returns, nutrient imbalances, or plant stress. These nonlinear and site-specific responses introduce additional uncertainty, particularly at higher fertilizer rates.

Although the yield regression problem includes four explanatory variables, only the nitrogen rate $\mathbf{x}^{Nr}$ is controllable by the farmers. Consequently, the AS process focuses on the $\mathbf{x}^{Nr}$ axis to identify the optimal experimental rate for reducing epistemic uncertainty. Each field site within an MZ receives a single fertilizer treatment ($B = 1$). The AS process spans 50 iterations, with each iteration representing a distinct year or growing season, characterized by a randomly sampled precipitation value $\mathbf{x}^P \sim U(75, 150)$. All methods use the same sequence of precipitation values to ensure a fair comparison. The variable $\mathbf{x}^A$, which encodes topographic features, is assumed constant across iterations, whereas $\mathbf{x}^{VH}$, representing soil moisture, is modeled as a function of both precipitation and topographic aspect.

We considered $\mathbf{x}^P \in [75, 150]$, $\mathbf{x}^A \in [\pi/4, \pi/2]$, $\mathbf{x}^{VH} \in [0.5, 1]$, and $\mathbf{x}^{Nr} \in [0, 30, 60, 90, 120, 150]$. These values were selected to reflect realistic conditions based on past observations from the sub-region of the field used to model our simulated field site. During each iteration of the process, we sample a new precipitation value such that $\mathbf{x}_t^P \sim \mathcal{U}(75, 150)$. Similarly, we accounted for variations in the terrain aspect by modeling $\mathbf{x}_t^A$ as $\mathcal{U}(\pi/4, \pi/2)$. This assumption reflects slight alterations in the landscape each growing season, influenced by factors such as weather conditions and the use of heavy machinery.

Variable $\mathbf{x}^{VH}$ is associated with soil moisture content, where lower values correspond to drier soil conditions. Given the absence of additional variables to model soil moisture accurately and since this is beyond the scope of our study, we developed a simplified moisture function that incorporates precipitation and terrain aspect. In particular, we consider $\mathbf{x}_t^{VH} = \frac{\mathbf{x}_\tau^P}{150}\mathbf{x}_t^A$, reflecting that higher precipitation and terrain aspect values result in greater soil

Table 6.4: AUUC comparison for the simulated field site

MCDropout	**GP**	**NF-Ensemble**	**ASPINN**
614.68 ± 112.48	593.54 ± 107.42	730.80 ± 74.63	$\mathbf{496.85 \pm 71.65}$

Table 6.5: Statistical significance tests between ASPINN and the compared methods.

Compared Method	p-value
NF-Ensemble	1.3E-4 ($\uparrow$)
GP	4.4E-2 ($\uparrow$)
MC-Dropout	6.7E-3 ($\uparrow$)

moisture content. Based on this parameterization, the initial dataset $\mathbf{X}_{obs}^{(\tau=0)}$ is generated by randomly sampling 50 data points, each representing a distinct growing season.

6.4.2 Adaptive Sampling Experiments

We applied the AS process ten times. At each iteration, we used a unique initialization seed and evaluated the epistemic uncertainty along the allowed N rates for winter wheat (i.e., 0, 30, 60, 90, 120, and 150 lbs/ac) under the current field conditions. Table 6.4 presents the average AUUC values and corresponding standard deviations, highlighting the best-performing method in bold. Figure 6.13 depicts the evolution of the mean $PI_{\delta}^{(it)}$ values, calculated based on the results from the ten repetitions. Finally, we assessed the differences in AUUC values achieved by ASPINN across the ten AS iterations compared to those obtained by the other methods. Table 6.5 reports the p-values from the paired t-tests comparing ASPINN with the alternative approaches. The results indicate that the differences in AUUC values are statistically significant (i.e., p-value < 0.05).

The experiments conducted on the simulated field data exhibit consistent behavior with the results from the 1-D problems In particular, Table 6.4 demonstrates that ASPINN

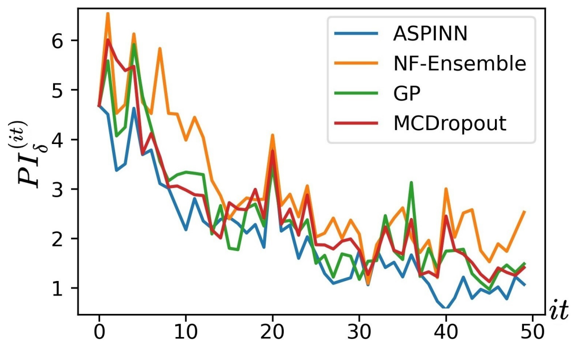


Figure 6.13: Evolution of the mean $PI_\delta^{(it)}$ for the simulated field site.

achieves the lowest AUUC values, and the differences between ASPINN and the compared methods are statistically significant. Given that the precipitation values vary at each iteration, the resulting learning curves are expected to exhibit multiple peaks and valleys rather than a smooth, consistently decreasing trend, as observed in Fig 6.13. This variability arises because higher precipitation values are associated with increased uncertainty levels, leading to more pronounced fluctuations in the learning curves. Considering that the sequence of precipitation values is not the same for all AS repetitions, Fig 6.13 reports only the mean curve and not the confidence bands. This is because the $PI_\delta^{(t)}$ values obtained by a method across different iterations are generated from contexts that could correspond to extreme opposites, leading to high variance values that do not necessarily reflect the method's performance. Despite this behavior, we observed that ASPINN consistently produced learning curves that remained below those of the compared methods.

6.5 Summary

This chapter demonstrated the practical application of the core areas studied in this dissertation: symbolic regression, prediction-interval generation, and adaptive sampling, within the context of precision agriculture. We worked with real-world on-farm experimental data of winter wheat to explore how these techniques can improve both the interpretability and efficiency of agronomic decision-making. The chapter builds upon the OFPE framework and addresses key challenges in modeling crop responses, managing uncertainty, and guiding data collection under resource constraints.

The chapter outlined the theoretical and methodological foundations that support this application, including the principles of OFPE, the importance of crop yield prediction, and the design of MZ clustering algorithms based on the shape dissimilarity of N-response curves. To address the need for uncertainty quantification in crop yield modeling, we incorporated PI-generation techniques, enabling field-wide visualization of predictive uncertainty. Our approach, DualAQD, demonstrated a consistent ability to produce narrower intervals while maintaining high coverage, outperforming other state-of-the-art approaches.

The proposed SR framework advances the current state of PA by offering a data-driven yet interpretable alternative to opaque models, capturing complex local variations in N-response without predefining functional forms. Then, we applied our symbolic regression method, SeTGAP, to model N-response curves at the MZ level. These MZs were derived based on shape dissimilarities in fertilizer responsivity rather than yield productivity. These MZ-specific symbolic expressions captured diverse crop responses across spatial subregions, offering interpretable and data-driven alternatives to traditional parametric models. The resulting equations not only reflect localized agronomic dynamics but also serve as functional foundations for further analysis and simulation.

Finally, we addressed the challenge of data acquisition in real agricultural environments by evaluating adaptive sampling strategies in a controlled simulation based on symbolic skeletons learned from the field. Simulated yield responses allowed fair comparison of AS methods under identical conditions. Our approach, ASPINN, converged faster to minimum epistemic uncertainty levels than other methods, demonstrating its capacity to prioritize informative sampling under uncertain and variable environmental conditions. Together, these findings show how the methods introduced throughout this dissertation can be deployed in real-world scenarios to support transparent, data-efficient, and uncertainty-aware agricultural decision-making.

CHAPTER SEVEN

CONCLUSIONS

For our concluding remarks, we summarize the contributions of the dissertation and identify directions for future work.

7.1 Contributions

This dissertation addressed the central challenge of producing interpretable data-driven models through the lens of symbolic regression. SR represents a promising avenue for building interpretable models, a key aspect of modern machine learning. By seeking to uncover mathematical equations that represent the relationships between input variables and their response, resulting equations offer transparency and clear insights into model behavior. This interpretability is vital in various domains, including healthcare, finance, and scientific research, where understanding the underlying mechanisms is essential for informed decision-making and building trust in machine learning-based systems. In addition, SR plays a crucial role in scientific discovery by enabling researchers to unveil fundamental laws and relationships governing natural phenomena. However, despite its appeal, symbolic regression remains a difficult problem due to the vastness of its search space, the need for generalization, and the tendency to overfit when models are not adequately regularized or guided.

To tackle these challenges, we presented a decomposable neuro SR approach called SeTGAP. The core contribution of this work was the formulation of the SR problem as a Multi-Set Symbolic Skeleton Prediction problem, which enables the decomposition of a complex multivariate system into univariate skeletons that can be independently predicted and subsequently recombined into a full multivariate model. As such, SeTGAP constitutes a *post-hoc* interpretability tool since, given an opaque model that approximates the system's behavior, it distills mathematical expressions that serve as interpretations of the functional relationships between input variables and the system's response embedded within the opaque model's learned function.

To address the MSSP problem, we proposed the Multi-Set Transformer, a specialized neural network architecture based on the transformer model. This architecture is designed to process multiple input–response sets simultaneously and infer a symbolic skeleton expression that captures the underlying mathematical structure shared across all sets. Trained on a large corpus of synthetically generated expressions, the Multi-Set Transformer serves as a general-purpose model that can be applied to new regression problems without requiring retraining. Hence, when presented with an observed dataset and an opaque predictive model, the Multi-Set Transformer generates univariate symbolic skeletons that approximate the functional relationships between each input variable and the system's response. Experimental results showed that this method consistently recovered the correct functional form of all system variables across all tested problems. To support this evaluation, we presented a skeleton performance evaluation methodology based on genetic algorithms that tests how well a given skeleton's functional form matches the system's underlying functional form. In contrast, compared methods, including evolutionary approaches and end-to-end neural models, often prioritized global error minimization or focused on only the most influential variables, leading to incomplete or inaccurate structural representations.

The predictive modeling of univariate skeletons was complemented by a symbolic merging algorithm designed to recombine the predicted univariate expressions into a coherent and accurate multivariate model. This merging step was implemented as a cascade process using evolutionary techniques; specifically, genetic programming and genetic algorithms guided by structural constraints. Notably, the merging process was not treated as a generic search, but rather as a structurally-aware operation that preserves the internal structures of the subexpressions discovered during the univariate skeleton prediction. By maintaining these discovered components, the method avoids redundant rediscovery and ensures that the final multivariate expressions remain interpretable and modular. Among all tested SR approaches, SeTGAP was the only method that consistently reconstructed the expected functional forms of the underlying generating functions across all evaluated problems. Further experiments demonstrated its robustness, as SeTGAP continued to identify the correct functional structures even under varying levels of noise, highlighting its reliability in uncovering meaningful symbolic interpretations.

While our symbolic regression approach demonstrates a certain level of robustness to aleatoric uncertainty (i.e., noise inherent in the data), we recognize that the accurate recovery of symbolic representations is influenced by both aleatoric and epistemic uncertainty. Understanding and quantifying these two types of uncertainty is essential, especially in application domains where ensuring the reliability of AI-powered systems is critical. Therefore, this dissertation also tackles the challenge of uncertainty quantification and management in regression models. In many scientific and engineering contexts, understanding the confidence of a model's predictions is as important as the predictions themselves. To this end, we introduced DualAQD, a neural-network-based method for prediction interval generation. DualAQD employs two companion networks: one dedicated to estimating the target response and another responsible for producing high-quality PIs. The training objective uses a custom loss function that simultaneously minimizes the mean prediction interval width while enforcing coverage constraints to maximize the prediction interval coverage probability implicitly. Empirical results across multiple datasets revealed that DualAQD consistently maintained a nominal coverage level while producing significantly narrower intervals than three state-of-the-art PI-generation methods, all without compromising prediction accuracy.

The uncertainty modeling capabilities achieved with DualAQD were leveraged in the development of an adaptive sampling strategy called ASPINN. The primary motivation behind ASPINN lies in the practical challenges of many real-world systems where data collection is costly, time-consuming, or constrained by limited resources. In such scenarios, it becomes critical to identify sampling locations that most effectively reduce model uncertainty. ASPINN addresses this by focusing on epistemic uncertainty reduction in regression problems, using NN-generated PIs to guide adaptive data acquisition. Specifically, ASPINN estimates potential epistemic uncertainty by evaluating the distance between the predicted PI bounds and the observed data, both at candidate locations and their neighborhoods. This local discrepancy informs where the model lacks information and is likely to benefit from new data. To make batch acquisitions, ASPINN employs a Gaussian Process as a surrogate model of the neural networks trained at each iteration of the adaptive sampling process. This enables the formulation of an acquisition function that selects a diverse and informative set of new sampling points. Experimental results demonstrated that ASPINN consistently outperforms state-of-the-art methods in convergence rate, achieving lower epistemic uncertainty with fewer samples across the tested problems.

A practical application of the developed methods was demonstrated in the domain of precision agriculture. In this setting, accurately modeling the effect of nitrogen application on crop yield under varying environmental conditions (e.g., precipitation, terrain, soil moisture) is important for confidently designing optimization strategies that maximize profit while minimizing environmental impact. To apply symbolic regression meaningfully, we employed a zone-based modeling strategy where fields are divided into management zones based on the fertilizer responsivity similarity rather than raw productivity. This responsivity-based clustering enabled the identification of functionally coherent subregions that are suitable for local model learning. Symbolic regression was then applied independently in each MZ using SeTGAP's MSSP pipeline. This approach yielded interpretable and low-complexity models that captured the functional behavior of crop response to N fertilizer rate in each region.

Furthermore, we assessed the applicability of our uncertainty management techniques in the context of PA. In agricultural systems, where ground truth data are scarce and environmental factors can be highly heterogeneous, it is important to understand the reliability of model predictions. Therefore, we incorporated PI generation into the crop yield prediction process by applying DualAQD in 2-D regression convolutional neural networks. This enabled spatially explicit visualization of predictive uncertainty across the field, allowing farmers and stakeholders to assess not just expected yields but also the associated confidence. Our results indicated that DualAQD outperformed other methods by maintaining high probability coverage while offering significantly narrower PIs, making it well-suited for guiding interventions and management practices in heterogeneous field conditions.

Adaptive sampling holds significant promise for precision agriculture, where experimental results often take an entire growing season to materialize, and trials are typically confined to only a portion of the field. This challenge is further exacerbated by constraints such as delayed data collection due to crop rotation practices. These limitations motivate the adoption of efficient sampling strategies to minimize the time and resources required for reliable experimentation. In this context, we tested the performance of our adaptive sampling technique, ASPINN, in a controlled simulation of adaptive sampling on an agricultural site. By simulating crop yield responses based on symbolic skeletons that model an MZ extracted from real field data, we created an evaluation framework to compare different sampling strategies under fair conditions. ASPINN showed the fastest convergence to minimal epistemic uncertainty levels across all trials, demonstrating its ability to prioritize data acquisition in regions where uncertainty reduction is most beneficial.

To summarize, the contributions presented in this dissertation include the following:

- We developed SeTGAP, a decomposable symbolic regression framework that learns univariate skeleton expressions via a pre-trained transformer model and incrementally merges them into multivariate expressions using structurally guided evolutionary techniques. It preserves interpretability and reduces the search space effectively.

- We introduced the Multi-Set Symbolic Skeleton Prediction problem, which enables the extraction of a shared symbolic structure from multiple input–response sets governed by the same underlying functional form but differing in parameterization.

- We designed and pre-trained the Multi-Set Transformer model, a novel transformer-based architecture tailored to solve the MSSP problem. The model was pre-trained on a large corpus of synthetic symbolic skeleton expressions, with training data produced dynamically during the training process using a specialized data generation framework.

- We introduced a symbolic skeleton performance metric that quantifies the fidelity of predicted univariate skeletons in relation to the true underlying functions.

- We presented DualAQD, a loss function and training scheme that uses two companion neural networks: one for accurate prediction and another for PI generation, designed to minimize interval width and implicitly maximize PI coverage simultaneously.

- We designed ASPINN, an adaptive sampling framework that produces potential epistemic uncertainty estimates from NN-generated PIs to guide data acquisition. It selects informative and diverse sampling points by using Gaussian Process surrogates.

- We demonstrated DualAQD's effectiveness in crop yield prediction, where it provides high-quality prediction intervals that enable spatial visualization of uncertainty across agricultural fields, supporting risk-aware decision-making in agricultural management.

- We applied SeTGAP to study functional variation in fertilizer nitrogen responsivity across the fields, allowing the discovery of interpretable mathematical expressions that parameterize site-specific N-response curves within management zones.

- We evaluated ASPINN's performance in an agricultural simulation, where it was shown to accelerate uncertainty reduction under realistic field constraints, outperforming existing sampling methods in convergence rate and sample efficiency.

7.2 Future Work

Several research questions remain open and will be explored in future work. In particular, we plan to investigate the components of SeTGAP and their roles in the successful recovery of the underlying function. One key area of interest is the influence of the number of input sets, denoted by N_S, on the accuracy of the predicted univariate skeletons in the context of the multi-set symbolic skeleton prediction problem. In this dissertation, experiments were limited to $N_S = 10$ and test problems with up to four input variables. However, during univariate skeleton prediction, where the goal is to model the relationship between a specific variable and the system's response, it is reasonable to hypothesize that higher-dimensional systems may require more input sets to capture the variable's functional form adequately across a sufficiently diverse range of conditions influenced by the remaining variables. Future experiments will vary N_S across problems of increasing dimensionality to systematically quantify its effect on prediction accuracy and generalization.

We will investigate how the order of skeleton merging affects the learned expressions and their predictive performance. In the current design, skeletons are merged based on their individual performance, with less reliable skeletons merged later to minimize error propagation. However, this approach does not account for potential interdependencies between variables. For example, if two variables interact strongly, merging their skeletons early might improve the accuracy of the modeled interaction. We plan to explore alternative merging heuristics, including strategies informed by variable interaction analysis.

We intend to expand SeTGAP's expressiveness by supporting more complex functional constructs beyond unary and binary operators. This includes differential operators, integral transforms (such as Fourier and Laplace transforms), and wave transforms; i.e., mathematical tools that frequently arise in signal processing, dynamical systems, and quantum physics.

While these extensions would increase the search space significantly, we plan to adapt the Multi-Set Transformer architecture to accommodate domain-specific prior knowledge, helping to constrain and guide the exploration. This will enable a human-in-the-loop symbolic regression framework, in which experts can specify known or preferred functional forms, operators, or constraints to bias the search toward plausible hypotheses. This capability will ensure that the generated expressions remain aligned with domain-specific knowledge. In parallel, we aim to augment the framework with robust uncertainty quantification mechanisms. By integrating efficient uncertainty estimates into the symbolic regression process, we will enable a more rigorous assessment of the quality and stability of the identified skeleton expressions and their combinations throughout the merging process.

Future work will also broaden the application of SeTGAP to problems in scientific discovery across various domains. In optics, for instance, symbolic regression could be used to derive analytical expressions for calibrating microbolometer thermal images, which are critical for improving temperature measurements and compensating for sensor non-linearities. In physics, symbolic regression offers a promising path for discovering or approximating governing equations for complex phenomena. For example, it could help uncover interpretable models of neutrino oscillation behavior based on data from next-generation telescopes and simulations. As high-quality experimental and synthetic datasets continue to grow, symbolic regression methods such as SeTGAP have the potential to play a key role in advancing theoretical insights and facilitating data-driven discovery.

In the presented experiments, the PI-generation model used in DualAQD was trained with the same architecture as the target-prediction model, except for the final layer, to facilitate the use of transfer learning and accelerate the learning process. Future work will explore the impact of employing specialized architectures for the PI-generation model. Moreover, DualAQD can be extended to handle more complex uncertainty scenarios, particularly in multi-modal settings where aleatoric uncertainty arises from distinct, overlapping sources of variability. In such cases, a single PI may be insufficient to characterize the range of plausible outcomes. To address this, future work will focus on adapting DualAQD to generate multiple intervals that reflect the multi-modal structure of the data. One possible direction is to incorporate neural architectures that predict the parameters of a mixture of probability distributions, allowing the model to represent several distinct modes in the output space. Another approach is to use latent variable models enhanced with attention mechanisms to learn context-dependent representations that help identify and separate the different sources of variability. These enhancements would enable DualAQD to generate multiple PIs per input when needed, offering a faithful representation of uncertainty in systems where noise is inherently multi-modal.

Furthermore, to strengthen the theoretical grounding of DualAQD, we plan to explore the incorporation of principles from conformal prediction. Conformal methods offer distribution-free, finite-sample guarantees for PIs and can complement DualAQD's architecture by providing rigorous confidence levels under minimal assumptions. This integration could help quantify uncertainty with provable guarantees, making the framework more reliable while still producing narrow PIs, especially in applications where rigorous statistical bounds are required to validate empirical findings or support theoretical insights.

Future work on ASPINN will focus on dealing with multi-modal scenarios and improving scalability. Similar to DualAQD, ASPINN is limited by its inability to handle multi-modal aleatoric noise natively. As such, future extensions will adapt ASPINN to support more complex uncertainty modeling by leveraging advances developed for DualAQD in handling multi-modal noise. In particular, ASPINN's potential epistemic uncertainty metric would be revisited to support multiple PIs per input and guide sampling accordingly.

Another limitation involves the computational cost of evaluating epistemic uncertainty across the entire input space, which becomes prohibitive as dimensionality increases. To mitigate this, we aim to explore dimensionality reduction techniques and surrogate models that can approximate the epistemic uncertainty landscape more efficiently. Additionally, we will explore surrogate modeling approaches, such as sparse Gaussian processes, which model uncertainty over fewer representative points to reduce computational cost. Diffusion models may also be used to learn smooth latent representations of the data distribution; in this lower-dimensional latent space, sampling and uncertainty estimation can be performed more efficiently while preserving the structure of the original problem. Together, these strategies aim to make ASPINN practical and effective even in challenging high-dimensional settings.

7.3 Concluding Remarks

This dissertation investigated the discovery of mathematical expressions as interpretable models to advance transparency and trust in AI-driven systems. While symbolic regression was employed to derive human-understandable representations of complex systems, complementary techniques for uncertainty quantification and adaptive sampling were developed to enhance the trustworthiness of opaque models. The methods developed here highlight that model interpretability, uncertainty quantification, and adaptive sampling are not isolated challenges but deeply interconnected aspects of building trustworthy models. Throughout, the emphasis has been on creating solutions that are both practical for real-world applications, such as precision agriculture, and grounded in theoretical principles.

The ideas presented here suggest several directions for future work. For instance, our symbolic regression strategies could be extended to address more complex systems, while uncertainty quantification and adaptive sampling techniques may be adapted to multi-modal and high-dimensional settings. Further study is needed to better understand the interactions between these elements. The broader goal is to advance machine learning models that are not only powerful but also interpretable, reliable, and adaptable to the needs of scientific discovery and practical applications.

REFERENCES CITED

[1] Christopher Ackello-Ogutu, Quirino Paris, and William A. Williams. Testing a von Liebig crop response function against polynomial specifications. *American Journal of Agricultural Economics*, 67(4):873–880, 1985.

[2] A. Ali, R. Martelli, E. Scudiero, Lupia F., Falsone G., Rondelli V., and L. Barbanti. Soil and climate factors drive spatio-temporal variability of arable crop yields under uniform management in northern Italy. *Archives of Agronomy and Soil Science*, 69(1):75–89, 2023.

[3] Abid Ali, Valda Rondelli, Roberta Martelli, Gloria Falsone, Flavio Lupia, and Lorenzo Barbanti. Management zones delineation through clustering techniques based on soils traits, NDVI data, and multiple year crop yields. *Agriculture*, 12(2), 2022.

[4] Maryam Amir Haeri, Mohammad Mehdi Ebadzadeh, and Gianluigi Folino. Statistical genetic programming for symbolic regression. *Applied Soft Computing*, 60:447–469, 2017.

[5] Peter J. Angeline. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In *Annual Conference on Genetic Programming*, pages 21—-29, 1996.

[6] Peter J. Angeline. Subtree crossover: Building block engine or macromutation? In *Annual Conference on Genetic Programming*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997.

[7] Anastasios N Angelopoulos, Amit Pal Kohli, Stephen Bates, Michael Jordan, Jitendra Malik, Thayer Alshaabi, Srigokul Upadhyayula, and Yaniv Romano. Image-to-image regression with distribution-free uncertainty quantification and applications in imaging. In *International Conference on Machine Learning*, volume 162, pages 717–730, 17–23 Jul 2022.

[8] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Genetic and Evolutionary Computation*, pages 879—886, 2014.

[9] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[10] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.

[11] Tommaso Bendinelli, Luca Biggio, and Pierre-Alexandre Kamienny. Controllable neural symbolic regression. In *International Conference on Machine Learning*, ICML'23, 2023.

[12] Lucas Berry and David Meger. Normalizing flow ensembles for rich aleatoric and epistemic uncertainty modeling. *AAAI Conference on Artificial Intelligence*, 37(6):6806–6814, Jun. 2023.

[13] Amanda Bertschinger, James Bagrow, and Joshua Bongard. Evolving Form and Function: Dual-Objective Optimization in Neural Symbolic Regression Networks. In *Genetic and Evolutionary Computation Conference*, pages 277–285, July 2024.

[14] Amanda Bertschinger, Q. Tyrell Davis, James Bagrow, and Joshua Bongard. The metric is the message: Benchmarking challenges for neural symbolic regression. In *Machine Learning and Knowledge Discovery in Databases*, pages 161–177, 2023.

[15] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, volume 139, pages 936–945, 2021.

[16] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[17] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[18] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *Machine Learning and Knowledge Extraction*, 3(4):966–989, 2021.

[19] David S. Bullock, Maria Boerngen, Haiying Tao, Bruce Maxwell, Joe D. Luck, Luciano Shiratsuchi, Laila Puntel, and Nicolas F. Martin. The data-intensive farm management project: Changing agronomic research through on-farm precision experimentation. *Agronomy Journal*, 111(6):2736–2746, 2019.

[20] Donald G. Bullock and David S. Bullock. Quadratic and quadratic-plus-plateau models for predicting optimal nitrogen rate of corn: A comparison. *Agronomy Journal*, 86(1):191–195, 1994.

[21] Guendalina Caldarini, Sardar Jaf, and Kenneth McGarry. A literature survey of recent chatbots. *Information*, 13(1), 2022.

[22] Gustau Camps-Valls, Andreas Gerhardus, Urmi Ninad, Gherardo Varando, Georg Martius, Emili Balaguer-Ballester, Ricardo Vinuesa, Emiliano Diaz, Laure Zanna, and Jakob Runge. Discovering causal relations and equations from data. *ArXiv*, abs/2305.13341, 2023.

[23] Eduardo G. Carrano, Carlos M. Fonseca, Ricardo H. C. Takahashi, Luciano C. A. Pimenta, and Oriane M. Neto. A preliminary comparison of tree encoding schemes for evolutionary algorithms. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1969–1974, 2007.

[24] William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[25] Lucy R Chai. Uncertainty estimation in bayesian neural networks and links to interpretability. Master's thesis, Department of Engineering, University of Cambridge, 2018.

[26] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 77–85, 2017.

[27] Boyuan Chen, Kuang Huang, Sunand Raghupathi, Ishaan Chandratreya, Qiang Du, and Hod Lipson. Automated discovery of fundamental variables hidden in experimental data. *Nature Computational Science*, 2(7):433–442, Jul 2022.

[28] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014.

[29] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[30] Xieting Chu, Hongjue Zhao, Enze Xu, Hairong Qi, Minghan Chen, and Huajie Shao. Neural symbolic regression using control variables. *ArXiv*, abs/2306.04718, 2023.

[31] I. Cisternas, I. Velásquez, A. Caro, and A. Rodríguez. Systematic literature review of implementations of precision agriculture. *Computers and Electronics in Agriculture*, 176:105626, 2020.

[32] Matthew J. Colbrook, Vegard Antun, and Anders C. Hansen. The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and smale’s 18th problem. *National Academy of Sciences*, 119(12):e2107151119, 2022.

[33] S. Cook, M. Lacoste, F. Evans, M. Ridout, M. Gibberd, and T. Oberthür. An on-farm experimental philosophy for farmer-centric digital innovation. In *International Conference on Precision Agriculture*, 2018.

[34] Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl. *ArXiv*, abs/2305.01582, 2023.

[35] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. In *Neural Information Processing Systems*, 2020.

[36] N. Davatgar, M.R. Neishabouri, and A.R. Sepaskhah. Delineation of site-specific nutrient management zones for a paddy cultivated area based on soil fertility using fuzzy clustering. *Geoderma*, 173–174:111–118, 2012.

[37] Ana Maria Delgado, Digvijay Wadekar, Boryana Hadzhiyska, Sownak Bose, Lars Hernquist, and Shirley Ho. Modelling the galaxy–halo connection with machine learning. *Monthly Notices of the Royal Astronomical Society*, 515(2):2733–2746, 07 2022.

[38] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, volume 80, pages 1184–1193, 10–15 Jul 2018.

[39] Francesco Di Fiore, Michela Nardelli, and Laura Mainini. Active learning and bayesian optimization: A unified perspective to learn with a goal. *Archives of Computational Methods in Engineering*, April 2024.

[40] Jason G. Digalakis and Konstantinos G. Margaritis. An experimental study of benchmarking functions for genetic algorithms. *International Journal of Computer Mathematics*, 79(4):403–416, 2002.

[41] Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *Genetic and Evolutionary Computation*, pages 1588—-1595. Association for Computing Machinery, 2007.

[42] Virginia Dignum. *Responsible Artificial Intelligence: How to Develop and Use AI in a Responsible Way*. Springer Verlag, 2019.

[43] Dheeru Dua and Casey Graff. UCI machine learning repository, 2019. https://archive.ics.uci.edu/ml/index.php.

[44] M. Ebner. On the search space of genetic programming and its relation to nature's search space. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1357–1361, 1999.

[45] Harrison Edwards and Amos Storkey. Towards a neural statistician. In *International Conference on Learning Representations*, Toulon, France, 2017.

[46] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *IEEE/CVF International Conference on Computer Vision*, pages 6824–6835, October 2021.

[47] Sebastian Farquhar, Michael A. Osborne, and Yarin Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In Silvia Chiappa and Roberto Calandra, editors, *International Conference on Artificial Intelligence and Statistics*, volume 108, pages 1352–1362, 26–28 Aug 2020.

[48] R. B. Ferguson, G. W. Hergert, J. S. Schepers, C. A. Gotway, J. E. Cahoon, and T. A. Peterson. Site-specific nitrogen management of irrigated maize. *Soil Science Society of America Journal*, 66(2):544–553, 2002.

[49] Renato Filho, Anisio Lacerda, and Gisele Pappa. Explaining symbolic regression predictions. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020.

[50] Federico Filipponi. Sentinel-1 GRD preprocessing workflow. In *International Electronic Conference on Remote Sensing*, 2019.

[51] Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.

[52] B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *IEEE International Conference on Evolutionary Computation*, pages 616–621, 1996.

[53] Jon J. Fridgen, Newell R. Kitchen, Kenneth A. Sudduth, Scott T. Drummond, William J. Wiebold, and Clyde W. Fraisse. Management zone analyst (mza). *Agronomy Journal*, 96(1):100–108, 2004.

[54] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 20–22 Jun 2016.

[55] Diego José Gallardo-Romero, Orly Enrique Apolo-Apolo, Jorge Martínez-Guanter, and Manuel Pérez-Ruiz. Multilayer data and artificial intelligence for the delineation of homogeneous management zones in maize cultivation. *Remote Sensing*, 15(12), 2023.

[56] Inés M. Galván, José M. Valls, Alejandro Cervantes, and Ricardo Aler. Multi-objective evolutionary optimization of prediction intervals for solar energy forecasting with neural networks. *Information Sciences*, 418-419:363–382, 2017.

[57] M.A. Ganaie, Minghui Hu, A.K. Malik, M. Tanveer, and P.N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

[58] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *International Conference on Neural Information Processing Systems*, pages 7587—-7597, Red Hook, NY, USA, 2018.

[59] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[60] Robin Gebbers and Viacheslav I. Adamchuk. Precision agriculture and food security. *Science*, 327(5967):828–831, 2010.

[61] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, ICML'17, pages 1243–1252, 2017.

[62] Claudia Georgi, Daniel Spengler, Sibylle Itzerott, and Birgit Kleinschmit. Automatic delineation algorithm for site-specific management zones based on satellite remote sensing data. *Precision Agriculture*, 19(4):684—-707, November 2017.

[63] Ning Gong and Nianmin Yao. A generalized decoding method for neural text generation. *Computer Speech & Language*, 81:101503, 2023.

[64] Javier Gonzalez, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch bayesian optimization via local penalization. In *International Conference on Artificial Intelligence and Statistics*, volume 51, pages 648–657, Cadiz, Spain, 09–11 May 2016.

[65] Alex Graves. Generating sequences with recurrent neural networks. *ArXiv*, abs/1308.0850, 2013.

[66] Daria Grechishnikova. Transformer neural network for protein-specific de novo drug generation as a machine translation problem. *Scientific Reports*, 11(1):321, Jan 2021.

[67] Baihe He, Qiang Lu, Qingyun Yang, Jake Luo, and Zhiguang Wang. Taylor genetic programming for symbolic regression. In *Genetic and Evolutionary Computation Conference*, pages 946—954, 2022.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[69] Paul Hegedus, Bruce Maxwell, John Sheppard, Sasha Loewen, Hannah Duff, Giorgio Morales, and Amy Peerlinck. Towards a low-cost comprehensive process for on-farm precision experimentation and analysis. *Agriculture*, 13(3), 2023.

[70] Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res.*, 13:1809—-1837, jun 2012.

[71] John H. Holland. Genetic algorithms. *Scientific American*, July 1992.

[72] T. Horie, M. Yajima, and H. Nakagawa. Yield forecasting. *Agricultural Systems*, 40(1):211–236, 1992.

[73] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2261–2269, July 2017.

[74] E.R. Hunt and Craig S. T. Daughtry. What good are unmanned aircraft systems for agricultural remote sensing and precision agriculture? *International Journal of Remote Sensing*, 39(15–16):5345–5376, 2018.

[75] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, March 2021.

[76] Pavel Izmailov, Wesley Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Wilson. Subspace inference for bayesian deep learning. In *Uncertainty in Artificial Intelligence Conference*, pages 1169–1179, Jul 2020.

[77] S. Hamed Javadi, Angela Guerrero, and Abdul M. Mouazen. Clustering and smoothing pipeline for management zone delineation using proximal and remote sensing. *Sensors*, 22(2):645, January 2022.

[78] D. B. Jaynes, T. C. Kaspar, T. S. Colvin, and D. E. James. Cluster analysis of spatiotemporal corn yield patterns in an Iowa field. *Agronomy Journal*, 95(3):574–586, 2003.

[79] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. Bayesian symbolic regression. *ArXiv*, abs/1910.08892, 2020.

[80] Lucie A. Kablan, Valérie Chabot, Alexandre Mailloux, Marie-Ève Bouchard, Daniel Fontaine, and Tom Bruulsema. Variability in corn yield response to nitrogen fertilizer in eastern canada. *Agronomy Journal*, 109(5):2231–2242, 2017.

[81] Shunkei Kakimoto, Taro Mieno, Takashi Tanaka, and David Bullock. Causal forest approach for site-specific input management via on-farm precision experimentation. *Computers and Electronics in Agriculture*, 199:107164, 2022.

[82] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and Francois Charton. End-to-end symbolic regression with transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Neural Information Processing Systems*, volume 35, pages 10269–10281, 2022.

[83] R. Khosla, K. Fleming, J. A. Delgado, T. M. Shaver, and D. G. Westfall. Use of site-specific management zones to improve nitrogen management for precision agriculture. *Journal of Soil and Water Conservation*, 57(6):513–518, 2002.

[84] Abbas Khosravi, Saeid Nahavandi, Doug Creighton, and Amir F. Atiya. Comprehensive review of NN-based prediction intervals and new advances. *IEEE Transactions on Neural Networks*, 22(9):1341–1356, 2011.

[85] Abbas Khosravi, Saeid Nahavandi, Douglas C. Creighton, and Amir F. Atiya. Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE Trans. Neural Networks*, 22(3):337–346, 2011.

[86] Abbas Khosravi, Saeid Nahavandi, Dipti Srinivasan, and Rihanna Khosravi. Constructing optimal prediction intervals by using neural networks and bootstrap method. *IEEE Transactions on Neural Networks and Learning Systems*, 26(8):1810–1815, 2015.

[87] Sehoon Kim, Amir Gholami, Albert Shaw, Nicholas Lee, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, and Kurt Keutzer. Squeezeformer: An efficient transformer for automatic speech recognition. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Neural Information Processing Systems*, volume 35, pages 9361–9373, 2022.

[88] K.E. Kinnear. Evolving a sort: lessons in genetic programming. In *IEEE International Conference on Neural Networks*, volume 2, pages 881–888, 1993.

[89] David Klahr and Herbert A. Simon. Studies of scientific discovery: Complementary approaches and convergent findings. *Psychological Bulletin*, 125(5):524–543, 1999.

[90] Blanka Klimova, Marcel Pikhart, Alice Delorme Benites, Caroline Lehr, and Christina Sanchez-Stockhammer. Neural machine translation in foreign language teaching and learning: a systematic review. *Education and Information Technologies*, 28(1):663–682, Jan 2023.

[91] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *International Joint Conference on Artificial Intelligence*, volume 1, pages 768—-774, 1989.

[92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[93] O. Kramer and H.-P. Schwefel. On three new approaches to handle constraints within evolution strategies. *Natural Computing*, 5(4):363–385, Nov 2006.

[94] Oliver Kramer. *Genetic Algorithm Essentials*. Springer International Publishing, 2017.

[95] Gabriel Kronberger, Bogdan Burlacu, Michael Kommenda, Stephan M. Winkler, and Michael Affenzeller. *Symbolic Regression*. Chapman and Hall/CRC, New York, August 2024.

[96] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[97] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *31st Int. Conf. on Neural Information Processing Systems*, page 6405–6416, 2017.

[98] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.

[99] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The Evolution of Size and Shape. In *Genetic Programming*. The MIT Press, 07 1999.

[100] Patrick G. Lawrence, Lisa J. Rew, and Bruce D. Maxwell. A probabilistic Bayesian framework for progressively updating site-specific recommendations. *Precision Agriculture*, 16(3):275–296, June 2015.

[101] Joon-Yong Lee, Min-Soeng Kim, Cheol-Taek Kim, and Ju-Jang Lee. Study on encoding schemes in compact genetic algorithm for the continuous numerical problems. In *SICE Annual Conference*, pages 2694–2699, 2007.

[102] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, volume 97, pages 3744–3753, 09–15 Jun 2019.

[103] Kin Long Kelvin Lee and Nalini Kumar. Artificial intelligence for scientific discovery at high-performance computing scales. *Computer*, 56(4):116–122, 2023.

[104] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *International Conference on Learning Representations*, 2017.

[105] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1), 2021.

[106] Laura E. Lindsey, Allen W. Goodwin, S. Kent Harrison, and Pierce A. Paul. Optimum seeding rate and stand assessment of soft red winter wheat. *Agronomy Journal*, 112(5):4069–4075, 2020.

[107] Siyan Liu, Pei Zhang, Dan Lu, and Guannan Zhang. PI3NN: Out-of-distribution-aware prediction intervals from three neural networks. In *International Conference on Learning Representations*, 2022.

[108] M. Lones. *Enzyme Genetic Programming*. PhD thesis, Department of Electronics, University of York, 2004.

[109] Jun Lu, Jinliang Ding, Xuewu Dai, and Tianyou Chai. Ensemble stochastic configuration networks for estimating prediction intervals: A simultaneous robust training algorithm and its application. *IEEE Transactions Neural Networks and Learning Systems*, 31(12):5426–5440, 2020.

[110] Jun Lu, Jinliang Ding, Changxin Liu, and Tianyou Chai. Hierarchical-bayesian-based sparse stochastic configuration networks for construction of prediction intervals. *IEEE Transactions on Neural Networks and Learning Systems*, 33(8):3560–3571, 2022.

[111] Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: A review. *ArXiv*, abs/2211.10873, 2022.

[112] Sedigheh Maleki, Alireza Karimi, Amin Mousavi, Ruth Kerry, and Ruhollah Taghizadeh-Mehrjardi. Delineation of soil management zone maps at the regional scale using machine learning. *Agronomy*, 13(2), 2023.

[113] Georg Martius and Christoph H. Lampert. Extrapolation and learning equations. *ArXiv*, abs/1610.02995, 2016.

[114] B. Maxwell, P. Hegedus, P. Davis, A. Bekkerman, R. Payn, J. Sheppard, N. Silverman, and C. Izurieta. Can optimization associated with on-farm experimentation using site-specific technologies improve producer management decisions? In *International Conference on Precision Agriculture*, 2018.

[115] A. McBratney, B. Whelan, Tihomir Ancev, and J. Bouma. Future directions of precision agriculture. *Precision Agriculture*, 6:7–23, 2005.

[116] Esther D. Meenken, Christopher M. Triggs, Hamish E. Brown, Sarah Sinton, Jeremy Bryant, Alasdair D.L. Noble, Martin Espig, Mostafa Sharifi, and David M. Wheeler. Bayesian hybrid analytics for uncertainty analysis and real-time crop management. *Agronomy Journal*, 113(3):2491–2505, 2021.

[117] J. J. Meisinger, J. S. Schepers, and W. R. Raun. *Crop Nitrogen Requirement and Fertilization*, chapter 14, pages 563–612. John Wiley & Sons, Ltd, 2008.

[118] Fernando E. Miguez and Hanna Poffenbarger. How can we estimate optimum fertilizer rates with accuracy and precision? *Agricultural & Environmental Letters*, 7(1):e20075, 2022.

[119] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2013.

[120] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric semantic genetic programming. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, pages 21–31, 2012.

[121] G. Morales, J. Sheppard, B. Scherrer, and J. Shaw. Reduced-cost hyperspectral convolutional neural networks. *Journal of Applied Remote Sensing*, 14(3):036519, 2020.

[122] Giorgio Morales and John W. Sheppard. Two-dimensional deep regression for early yield prediction of winter wheat. In *SPIE Future Sensing Technologies 2021*, volume 11914, pages 49–63, November 2021.

[123] Giorgio Morales and John W. Sheppard. Counterfactual explanations of neural Network-Generated response curves. In *International Joint Conference on Neural Networks*, Queensland, Australia, June 2023.

[124] Giorgio Morales and John W. Sheppard. Counterfactual analysis of neural networks used to create fertilizer management zones. In *International Joint Conference on Neural Networks*, Yokohama, Japan, June 2024.

[125] Giorgio Morales and John W. Sheppard. Univariate skeleton prediction in multivariate systems using transformers. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 107–125, Vilnius, Lithuania, 2024.

[126] Giorgio Morales and John W. Sheppard. Adaptive sampling to reduce epistemic uncertainty using prediction interval-generation neural networks. In *AAAI Conference on Artificial Intelligence*, volume 39, pages 19546–19553, 2025.

[127] Giorgio Morales and John W. Sheppard. Decomposable symbolic regression using Multi-Set Transformers and genetic programming. In *European Conference on Machine Learning and Knowledge Discovery in Databases (submitted)*, Porto, Portugal, 2025.

[128] Giorgio Morales and John W. Sheppard. Dual accuracy-quality-driven neural network for prediction interval generation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(2):2843–2853, 2025.

[129] Giorgio Morales, John W. Sheppard, Paul Hegedus, and Bruce D. Maxwell. Improved yield prediction of winter wheat using a novel two-dimensional deep regression neural network trained via remote sensing. *Sensors*, 23(1):489, January 2023.

[130] Giorgio Morales, John W. Sheppard, Amy Peerlinck, Paul Hegedus, and Bruce D. Maxwell. Generation of site-specific nitrogen response curves for winter wheat using deep learning. In *International Conference on Precision Agriculture*, 2022.

[131] T. Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio Santiago, Daniel Faissol, and Brenden Petersen. Symbolic regression via neural-guided genetic programming population seeding. In *Neural Information Processing Systems*, 2021.

[132] Kajsa Møllersen, Jon Yngve Hardeberg, and Fred Godtliebsen. A probabilistic bag-to-class approach to multiple-instance learning. *Data*, 5(2), 2020.

[133] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[134] Vu Nguyen, Sunil Gupta, Santu Rana, My Thai, Cheng Li, and Svetha Venkatesh. Efficient Bayesian Optimization for Uncertainty Reduction Over Perceived Optima Locations. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1270–1275, November 2019.

[135] Vu-Linh Nguyen, Sébastien Destercke, and Eyke Hüllermeier. Epistemic Uncertainty Sampling. In Petra Kralj Novak, Tomislav Šmuc, and Sašo Džeroski, editors, *Discovery Science*, pages 72–86, Cham, 2019. Springer International Publishing.

[136] Vu-Linh Nguyen, Mohammad Hossein Shaker, and Eyke Hüllermeier. How to measure uncertainty in uncertainty sampling for active learning. *Machine Learning*, 111(1):89–122, January 2022.

[137] D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *IEEE International Conference on Neural Networks*, volume 1, pages 55–60, 1994.

[138] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 224—-230, USA, 1987.

[139] Patryk Orzechowski, William La Cava, and Jason H. Moore. Where are we now? A large benchmark study of recent symbolic regression methods. In *Genetic and Evolutionary Computation Conference*, page 1183–'1190, 2018.

[140] Soumyasundar Pal, Antonios Valkanas, Florence Regol, and Mark Coates. Bag graph: Multiple instance learning using bayesian graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 7922–7930, June 2022.

[141] D. Patrício and R. Rieder. Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review. *Computers and Electronics in Agriculture*, 153:69 – 81, 2018.

[142] Tim Pearce, Alexandra Brintrup, Mohamed Zaki, and Andy Neely. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *35th Int. Conf. on Machine Learning*, pages 4072–4081, 2018.

[143] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009.

[144] A. Peerlinck. *Multi- and Many-Objective Factored Evolutionary Algorithms*. PhD thesis, Gianforte School of Computing, Montana State University, 2023.

[145] Amy Peerlinck, Giorgio Morales, John Sheppard, Paul Hegedus, and Bruce Maxwell. Optimizing nitrogen application to maximize yield and reduce environmental impact in winter wheat production. In *International Conference on Precision Agriculture*, 2022.

[146] Amy Peerlinck and John Sheppard. Addressing sustainability in precision agriculture via multi-objective factored evolutionary algorithms. In *Metaheuristics International Conference*, pages 391–405, 2023.

[147] Amy Peerlinck, John Sheppard, and Bruce Maxwell. Using deep learning in yield and protein prediction of winter wheat based on fertilization prescriptions in precision agriculture. In *International Conference on Precision Agriculture*, 2018.

[148] Jonas Peters, Dominik Janzing, and Bernhard Schlkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. The MIT Press, 2017.

[149] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *Proc. of the International Conference on Learning Representations*, 2021.

[150] Riccardo Poli, William B. Langdon, and Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Genetic Programming*, pages 193–204, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[151] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[152] Kamrul Hasan Rahi, Hemant Kumar Singh, and Tapabrata Ray. Partial evaluation strategies for expensive evolutionary constrained optimization. *IEEE Transactions on Evolutionary Computation*, 25(6):1103–1117, 2021.

[153] J. Ramsay and B. Silverman. *Functional Data Analysis*. Springer, 2005.

[154] Patrick A. K. Reinbold, Logan M. Kageorge, Michael F. Schatz, and Roman O. Grigoriev. Robust learning from noisy, incomplete, high-dimensional experimental data via physically constrained symbolic regression. *Nature Communications*, 12(1):3219, 05 2021.

[155] Javier Reyes, Ole Wendroth, Christopher Matocha, and Junfeng Zhu. Delineating site-specific management zones and evaluating soil water temporal dynamics in a farmer's field in Kentucky. *Vadose Zone Journal*, 18(1):180143, 2019.

[156] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[157] Jakob Runge, Sebastian Bathiany, Erik Bollt, Gustau Camps-Valls, Dim Coumou, Ethan Deyle, Clark Glymour, Marlene Kretschmer, Miguel D. Mahecha, Jordi Muñoz-Marí, Egbert H. van Nes, Jonas Peters, Rick Quax, Markus Reichstein, Marten Scheffer, Bernhard Schölkopf, Peter Spirtes, George Sugihara, Jie Sun, Kun Zhang, and Jakob Zscheischler. Inferring causation from time series in earth system sciences. *Nature Communications*, 10(1), 2019.

[158] Annachiara Ruospo and Ernesto Sanchez. On the reliability assessment of artificial neural networks running on AI-oriented MPSoCs. *Applied Sciences*, 11(14), 2021.

[159] Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4442–4450, 10–15 Jul 2018.

[160] Tárik Salem, Helge Langseth, and Heri Ramampiaro. Prediction intervals: Split normal mixture from quality-driven deep ensembles. In Jonas Peters and David Sontag, editors, *36th Conf. on Uncertainty in Artificial Intelligence*, volume 124, pages 1179–1187, 03–06 Aug 2020.

[161] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[162] Jordan Schupbach, John W. Sheppard, and Tyler Forrester. Quantifying uncertainty in neural network ensembles using u-statistics. In *International Joint Conference on Neural Networks*, pages 1–8, 2020.

[163] Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *International Conference on Learning Representations*, 2022.

[164] U. Shafi, R. Mumtaz, J. García-Nieto, S. Hassan, S. Zaidi, and N. Iqbal. Precision agriculture techniques and practices: From considerations to applications. *Sensors*, 19(17), 2019.

[165] Zhuchen Shao, Hao Bian, Yang Chen, Yifeng Wang, Jian Zhang, Xiangyang Ji, and yongbing zhang. Transmil: Transformer based correlated multiple instance learning for whole slide image classification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Neural Information Processing Systems*, volume 34, pages 2136–2147. Curran Associates, Inc., 2021.

[166] B. Shashikumar, S. Kumar, K. George, and A. Singh. Soil variability mapping and delineation of site-specific management zones using fuzzy clustering analysis in a Mid-Himalayan watershed, india. *Environment, Development and Sustainability*, 25(8):8539—-8559, 2022.

[167] Durga L. Shrestha and Dimitri P. Solomatine. Machine learning approaches for estimation of prediction interval for the model output. *Neural Networks*, 19(2):225–235, 2006.

[168] L. Sifre. *Rigid-Motion Scattering For Image Classification*. PhD thesis, Ecole Polytechnique, 2014.

[169] Eli Simhayev, Gilad Katz, and Lior Rokach. Piven: A deep neural network for prediction intervals with specific value prediction. *cs.LG*, abs/2006.05139, 2020.

[170] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[171] Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *International Conference on Learning Representations*, 2023.

[172] Gilbert Syswerda. Simulated crossover in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 2, pages 239–255. Elsevier, 1993.

[173] Dustin Tran, Jeremiah Liu, Michael W Dusenberry, Du Phan, Mark Collier, Jie Ren, Kehang Han, Zi Wang, Zelda Mariet, Huiyi Hu, et al. Plex: Towards reliability using pretrained large model extensions. *CoRR*, abs/2207.07411, 2022.

[174] Leonardo Trujillo, Luis Muñoz, Edgar Galván-López, and Sara Silva. neat genetic programming: Controlling bloat naturally. *Information Sciences*, 333:21–43, 2016.

[175] Ho Fung Tsoi, Vladimir Loncar, Sridhara Rao Dasu, and Philip Harris. Symbolnet: Neural symbolic regression with adaptive dynamic pruning for compression. *Machine Learning: Science and Technology*, 2025.

[176] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

[177] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: Application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, Jun 2011.

[178] M. Valdenegro-Toro and D. Mori. A deeper look into aleatoric and epistemic uncertainty disentanglement. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 1508–1516, Los Alamitos, CA, USA, jun 2022.

[179] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. In *Neural Information Processing Systems: Workshop on Efficient Natural Language and Speech Processing*, 2022.

[180] T. Van Klompenburg, A. Kassahun, and C. Catal. Crop yield prediction using machine learning: A systematic literature review. *Computers and Electronics in Agriculture*, 177:105709, 2020.

[181] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Neural Information Processing Systems*, volume 30, 2017.

[182] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R. Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *ArXiv*, abs/1610.02424, 2018.

[183] Marco Virgolin and Solon P Pissis. Symbolic regression is NP-hard. *Transactions on Machine Learning Research*, 2022.

[184] M.C. Vuran, A. Salam, R. Wong, and S. Irmak. Internet of underground things in precision agriculture: Architecture and technology aspects. *Ad Hoc Networks*, 81:160 – 173, 2018.

[185] Kenneth F. Wallis. The Two-Piece Normal, Binormal, or Double Gaussian Distribution: Its Origin and Rediscoveries. *Statistical Science*, 29:106–112, 2014.

[186] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[187] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, volume 70, pages 3627–3635, 2017.

[188] K. B. Watkins, J. A. Hignight, R. J. Norman, T. L. Roberts, N. A. Slaton, C. E. Wilson, and D. L. Frizzell. Comparison of economic optimum nitrogen rates for rice in arkansas. *Agronomy Journal*, 102(4):1099–1108, 2010.

[189] Matthias Werner, Andrej Junginger, Philipp Hennig, and Georg Martius. Informed equation learning. *ArXiv*, abs/2105.06331, 2021.

[190] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[191] M.-J. Willis, H.G. Hiden, P. Marenbach, B. McKay, and G.A. Montague. Genetic programming: an introduction and survey of applications. In *International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications*, pages 314–319, 1997.

[192] Michael Winikoff and Julija Sardelić. Artificial intelligence and the right to explanation as a human right. *IEEE Internet Computing*, 25(2):116–120, 2021.

[193] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard Turner, José Hernández-Lobato, and Alexander Gaunt. Deterministic variational inference for robust Bayesian neural networks. In *International Conference on Learning Representations*, 2019.

[194] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

[195] J. Yao, W. Pan, S. Ghosh, and F. Doshi-Velez. Quality of uncertainty quantification for bayesian neural network inference. In *International Conference on Machine Learning: Workshop on Uncertainty & Robustness in Deep Learning*, 2019.

[196] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[197] Ashkan Zarnani, Soheila Karimi, and Petr Musilek. Quantile regression and clustering models of prediction intervals for weather forecasts: A comparative study. *Forecasting*, 1(1):169–188, 2019.

[198] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method, 2012. arXiv:1212.5701.

[199] Mojtaba Zeraatpisheh, Esmaeil Bakhshandeh, Mostafa Emadi, Tengfei Li, and Ming Xu. Integration of PCA and fuzzy clustering for delineation of soil management zones and cost-efficiency analysis in a citrus plantation. *Sustainability*, 12(14), 2020.

[200] Wang Zhang, Ziwen Martin Ma, Subhro Das, Tsui-Wei Lily Weng, Alexandre Megretski, Luca Daniel, and Lam M. Nguyen. One step closer to unbiased aleatoric

uncertainty estimation. *AAAI Conference on Artificial Intelligence*, 38(15):16857–16864, Mar. 2024.

[201] Xiaoyu Zhang, Zhe Shu, Rui Wang, Tao Zhang, and Yabing Zha. Short-term load interval prediction using a deep belief network. *Energies*, 11(10), 10 2018.

[202] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at Uber. In *IEEE Int. Conf. on Data Mining Workshops (ICDMW)*, pages 103–110, 2017.

APPENDIX: SETGAP COMPARISON RESULTS

Table A.1: Comparison of skeleton prediction results (E1–E9)

Prob.	Met. / Var.	PySR	TaylorGP	NeSymReS	E2E	MST	Target $\mathbf{e}(x)$
E1	x_1	$c_1\,x_1$	$c_1\,x_1$	$c_1\,x_1+\cos(c_2(c_3+c_4\,x_1)^2)$	$c_1+c_2\,x_1+c_3\sin((c_4+c_5\,x_1)^2)$	$c_1+c_2\,x_1+c_3\sin(c_4+c_5\,x1)$	$c_1\,x_1+c_2\,\sin(c_3(c_4+x_1))$
	x_2	$c_1\,x_2$	$c_1\,x_2$	$c_1\,x_2+\cos(c_2(c_3+x_2)^2)$	$c_1+c_2(c_3+c_4\,x_2)$	$c_1+c_2\,x_2+c_3\sin(c_4+c_5\,x_2)$	$c_1\,x_2+c_2\,\sin(c_3(c_4+x_2))$
E2	x_1	$c_1+\lvert c_2+\lvert c_3+x_1\rvert\rvert$	$c_1+c_2\,x_1$	$c_1+c_2\,x_1$	$c_1+c_2\,x_1+c_3(c_4+c_5\,x_1)^2$	$c_1+c_2(c_3+c_4\,x_1)^2$	$c_1+(c_2+c_3\,x_1)^2$
	x_2	c_1	c_1	$c_1+e^{e^{c_2\,x_2}}$	$c_1+c_2\,(c_3+c_4\,x_2)$	$c_1\sqrt{c_2\,x_2+c_3}+c_4$	$c_1\sqrt{x_2+c_2}+c_3$
	x_3	$c_1+c_2\,x_3$	$c_1+c_2\,x_3$	$c_1+c_2\,x_3$	$c_1+c_2\,x_3+c_3(c_4+c_5\cos(c_6+c_7\,x_3))$	$c_1+c_2\sin(c_3\,x_3+c_4)$	$c_1+c_2\sin(c_3\,x_3)$
E3	x_1	$c_1\,e^{x_1}\,\lvert\sinh(c_2\,x_1)\rvert$	$c_1\,e^{c_2\,x_1}$	$c_1\,e^{c_2\,x_1}$	$c_1+c_2e^{c_3\,x_1}$	$c_1+c_2e^{c_3\,x_1}$	$c_1+c_2\,e^{c_3\,x_1}$
	x_2	c_1	c_1	$c_1\,\cos(c_2\,x_2)$	$c_1+c_2\,\cos(c_3+c_4\,x_2)$	$c_1+c_2\,\cos(c_3+c_4\,x_2)$	$c_1+c_2\,\cos(c_3\,x_2)$
E4	x_1	$c_1+c_2\,x_1^2$	$c_1\,x_1^2$	—	$c_1+c_2\lvert c_3+c_4\,x_1+c_5\,x_1^2+c_6\,x_1^3+c_7\,x_1^4\rvert$	$c_1(c_2\,x1+c_3)^4+c_4$	$c_1+c_2\,x_1+c_3\,x_1^2+c_4\,x_1^4$
	x_2	$c_1+c_2\,x_2$	c_1	—	$c_1+c_2\lvert c_3+c_4\,x_2+c_5\,x_2^2\rvert$	$c_1(c_2\,x2+c_3)^2+c_4$	$c_1+c_1\,x_2+c_3\,x_2^2$
	x_3	$c_1+c_2\,x_3^2$	c_1	—	$c_1+c_2\lvert c_3+c_4\,x_3+c_5\,x_3^2+c_6\,x_3^3+c_7\,x_3^4\rvert$	$c_1(c_2\,x3+c_3)^4+c_4$	$c_1+c_2\,x_3+c_3\,x_4^2+c_4\,x_3^4$
	x_4	$c_1+c_2\,x_4$	c_1	—	$c_1+c_2\lvert c_3+c_4\,x_4+c_5\,x_4^2\rvert$	$c_1(c_2\,x4+c_3)^2+c_4$	$c_1+c_2\,x_4+c_3\,x_4^2$
E5	x_1	$c_1+\sin(c_2+c_3\,x_1)$	c_1	—	$c_1+c_2\cos(c_3+c_4\,x_1)$	$c_1\sin(c_2\,x1+c_3)+c_4$	$c_1+\sin(c_2+c_3\,x_1)$
	x_2	$c_1+\sin(c_2+c_3\,x_2)$	c_1	—	$c_1+c_2\,x_2$	$c_1\sin(c_2\,x2+c_3)+c_4$	$c_1+\sin(c_2+c_3\,x_2)$
	x_3	$c_1+\sin(c_2+c_3\,x_3)$	c_1	—	$c_1+c_2\,x_3+c_3\,x_3^2+c_4\,x_3^3$	$c_1\sin(c_2\,x3+c_3)+c_4$	$c_1+\sin(c_2+c_3\,x_3)$
	x_4	$c_1+e^{c_2\,x_4}$	$c_1\,e^{x_4}\,e^{\sin(c_2\,x_4)}$	—	$c_1+c_2\,e^{c_3\,x_4}$	$c_1e^{c_2\,x4}+c_3$	$c_1+e^{c_2\,x_4}$
E6	x_1	c_1	c_1	$c_1+c_2\,x_1$	$c_1+c_2\,\mathrm{atan}(c_3+c_4\,x_1)$	$c_1+c_2\tanh(c_3\,x_1)$	$c_1+\tanh(c_2\,x_1)$
	x_2	c_1	c_1	$c_1+x_2\,\sin(c_2/(c_3+x_2))$	$c_1+c_2\,x_2+c_3\,x_2\,\cos(c_4+c_5\,x_2)$	$c_1+c_2\,\lvert x_2\rvert$	$c_1+c_2\,\lvert x_2\rvert$
	x_3	$\tanh(\exp(x_3))$	$c_1\,\sin(c_2\,x_3^2)/(c_3\,\sqrt{x_3}+\sin(\sqrt{x_3}))$	c_1	c_1	$c_1\,\cos(c_2\,(c_3+x_3)^2)+c_4$	$c_1+c_2\,(\cos(c_3\,x_3^2))$
E7	x_1	$c_1+x_1^2$	c_1	$c_1+c_2\,x_1$	$c_1+c_2\,\sin(c_3+c_4\,x_1)^2+c_5\,\sin(c_6+c_7\,x_1)$	$c_1+c_2/(c_3+\sin(c_4\,x_1+c_5))$	$c_1/(c_2+\sin(c_3\,x_1))$
	x_2	$c_1/\sinh(\sinh(\tanh(e^{\sinh(\sin(c_2\,x_2))})))$	$c_1\,x_2^2+\sqrt{\lvert x_2\rvert}$	$(c_1+x_2^2)/(c_2+\cos(c_3\,(c_4+c_5\,x_2)^3))$	$c_1\,(c_2+c_3\,x_2)^2$	$c_1+c_2\,x_2^2$	$c_1+c_2\,x_2^2$
E8	x_1	$c_1+\tanh(c_2+\cosh(x_1))$	c_1	$e^{c_1\,\cos(c_2/x_1)}$	$c_1+c_2\,ec_3\,\lvert c_4+c_5\,x_1\rvert$	$c_1+c_2/(c_3\,x_1^4+c_4)$	$c_1+c_2\,x_1^4/(c_3+c_4\,x_1^4)$
	x_2	$c_1+\cos(\tan(\tanh(c_2/x_2^2)))$	c_1	$c_1+\cos(1/x_2)$	$c_1+c_2\,e^{c_3\,\lvert c_4+x_2\rvert}$	$c_1+c_2/(c_3\,x_1^4+c_4)$	$c_1+c_2\,x_2^4/(c_3+c_4\,x_2^4)$
E9	x_1	$\log(c_1/(c_2+c_3\,x_1^2))$	$c_1+\log(c_2/\lvert x_1\rvert)$	$\log(c/\lvert x_1\rvert)$	$c_1+c_2\,\log(c_3+c_4\,x_1+c_5\,x_1^2)$	$c_1+\log(c_2\,x_1^2+c_3)$	$c_1+c_2\,\log(c_3+c_4\,x_1^2)$
	x_2	$\log(c_1+c_2\,x_2)$	$c_1\exp(c_2\,x_2)$	$\log(c\,\lvert x_2\rvert)$	$c_1+c_2\,\log(c_3+c_4/(c_5+c_6\,x_2+c_7\,x_2^2))$	$c_1+\log(c_2+c_3\,x_2)$	$c_1+\log(c_2+c_3\,x_2)$

Table A.2: Comparison of skeleton prediction results (E10–E13)

Prob.	**Var.**	**PySR**	**TaylorGP**	**NeSymReS**	**E2E**	**MST**	**Target $\mathbf{e}(x)$**
E10	x_1	$\sin(c_1\, x_1)$	$c_1\, x_1$	$\sin(c_1\, x_1)$	$c_1 + c_2\, \sin(c_3 + c_4\, x_1)$	$c_1 + c_2\, \sin(c_3\, x_1)$	$\sin(c_1\, x_1)$
	x_2	$\sin(c_1\, e^{x_2})$	$c_1\, e^{c_2\, \sqrt{\lvert x_2\rvert}}$	$\sin(c_1\, e^{x_2})$	$c_1 + c_2\, \sin(c_3 + c_4\, e^{c_5\, x_2})$	$c_1 + \sin(c_2\, e^{c_3\, x_2})$	$\sin(c_1\, e^{x_2})$
E11	x_1	$c_1\, x_1$	$c_1\, x_1$	$c_1\, x_1$	$c_1\, x_1$	$c_1 + c_2\, x_1$	$c_1\, x_1$
	x_2	$c_1\, \log(x_2^4)$	$c_1\, \log(\lvert x_2\rvert)$	$c_1\, \log(x_2^4)$	$c_1 + c_2\, \log(c_3 + c_4\, \lvert c_5 + c_6\, x_2\rvert)$	$c_1 + c_2\, \log(c_3\, x_2^2)$	$c_1\, \log(x_2^4)$
E12	x_1	$c_1 + \sin(x_1)\, \lvert x_1\rvert$	$c_1 + c_2\, x_1$	$c_1 + c_2\, x_1$	$c_1 + c_2\, x_1$	$c_1 + c_2\, x_1$	$c_1 + c_2\, x_1$
	x_2	$c_1 + c_2\, \sin(c_3/x_2)$	$c_1\, \sin(c_2/x_2)+$ $\sqrt{x_2}\, \sin(c_3/x_2)$	$c_1\, \sin(1/x_2)+$ $x_2\, \sin(1/x_2)$	$c_1\, \sin(c_2/(c_3 + c_4\, x_2))$	$c_1 + c_2\, \sin(c_3/x_2)$	c_1+ $c_2\, \sin(1/x_2)$
E13	x_1	$c_1\, \sqrt{x_1}$	$c_1 + c_2\, \sqrt{e^{\sqrt{x_1}}}$	$c_1 + c_2\, x_1$	$c_1 + c_2\, \log(c_3 + c_4\, x_1)$	$c_1 + c_2\, \sqrt{c_3 + x_1}$	$c_1\, \sqrt{x_1}$
	x_2	$c_1\, \log(x_2^2)$	$c_1 + \log(\lvert x_2\rvert)+$ $c_2\, \log(\lvert x_2\rvert)$	$c_1\, \log(x_2^2)$	$c_1 + c_2/(c_3 + c_4\, \lvert c_5 + c_6\, x_2\rvert)$	$c_1 + c_2\, \log(c_3\, (c_4 + x_2)^2)$	$c_1\, \log(x_2^2)$

Table A.3: Comparison of predicted expressions — Iteration 1

Eq.	**PySR**	**TaylorGP**	**NeSymReS**	**E2E**	**SeTGAP**
E1	$0.61x_0x_1$	$0.64x_0x_1$	$0.59x_0x_1+$ $\cos(0.01(x_1 - x_0 - 0.08)^2)$	$1.08(0.56x_0x_1 - 0.03x_0 + 0.02x_1-$ $\sin(0.01x_0^2 + 8.6x_0 + 0.45) - 0.01)$	$0.61x_0x_1+$ $1.15\sin((2.24x_0 - 1.5)(x_1 - 0.68))$
E2	$0.41x_2+$ $\lvert\lvert x_0 - 3.51\rvert{-1.95}\rvert$ $+4.11$	$-0.5x_0 + 0.001x_1$ $+0.39x_2 + 8.62$	$-x_0 + 0.40x_2+$ $e^{e^{-0.001x_1}} + 5.88$	$0.06x_0^2 - 0.51x_0 - 0.22x_1\cos(0.18x_2$ $+1.43) - 0.01x_1 + 0.01x_2-$ $3.25\cos(0.18x_2 + 1.43) + 6.56$	$0.06x_0^2 - 0.5x_0 + (3.37\sqrt{0.1x_1 + 1}$ $-0.19)(\sin(0.2x_2) + 0.01) + 6.49$
E3	$0.34e^{x_0}\lvert\sinh(0.47x_0)\rvert$	$0.23x_0e^{x_0}$	$9.10e^{0.72x_0}\cos(0.15x_1)$	$0.14e^{1.52x_0}+$ $0.52\cos(3.45x_1 + 0.05) + 0.11$	$0.15e^{1.5x_0} + 0.5\sin(3x_1 - 4.71)$
E4	$0.21x_0^2 - 0.18x_1+$ $0.21x_2^2 - 0.18x_3 - 0.76$	$0.29x_0^2$	—	$0.001\lvert 8.99(-0.88x_1 + (x_0 + 0.01)^2$ $+0.62)^2 + 9.72(-x_3+$ $0.98(x_2 + 0.01)^2 + 0.01)^2\rvert{+0.0023}$	$0.01x_0^4 - 0.02x_0^2x_1 - 0.001x_0 + 0.01$ $x_1^2 + 0.01x_2^4 + 0.01x_3^2-$ $(0.02x_2^2 + 0.004)(x_3 - 0.11) - 0.02$
E5	$e^{1.2x_3} + \sin(x_0 + x_1x_2)$	$0.51e^{x_3}e^{\sin(0.87x_3)}$	—	$e^{1.2x_3} - 0.91\cos((2.62x_0 + 0.15)$ $(24.66x_1 + 1.24)) - 0.05$	$0.999e^{1.2x_3} - \sin(x_0 + x_1x_2 + 9.42)$
E6	$\tanh(e^{x_2})$	$-\frac{sin(0.34x_2^2)}{-\sqrt{\lvert x_2\rvert}+\sin(\sqrt{\lvert x_2\rvert})}$	$-0.39x_0 + x_1\sin(\frac{x_0}{x_1} - 0.001x_2)$	$0.01x_1(-7.5\cos(15.41x_1 + 0.21)-$ $0.18) + 0.69\,\mathrm{atan}(0.75x_0 + 0.05) + 0.47$	$\cos(0.2x_2^2 + 0.05)\lvert x_1\rvert{+}\tanh(0.5x_0)$
E7	$(0.56 - 0.59x_0^2)/(\sinh(\sinh$ $(\tanh(e^{\sinh(\sin(6.28x_1))}))))$	$\sqrt{\lvert x_1\rvert} - x_1^2$	$\frac{0.12x_0+x_1^2}{\cos(3.1(-0.02x_1-1)^2)-0.31}$	$(-0.03x_1 - 0.03)(0.34x_1 - 0.35)$ $(41.59(1 - 0.5\sin(6.74x_0 + 0.23))^2 + 40)$	$\frac{4.53-4.54x_1^2}{4.54\sin(6.28x_0+6.28)+6.81)}$
E8	$(\tanh(\cosh(x_0) - 1.04)+$ $\tanh(\cosh(x_1) - 1.04))$	2	$\cos(\sin(1.69x_0)/(x_0x_1)) + 0.71$	$2.01 - 1.05e^{-0.06\lvert x_0 2.73-0.14\rvert\lvert 0.59x_1+0.1\rvert}$	$2 - \frac{19.76}{19.31x_0^4+0.12x_0^3+0.42x_0^2+19.72}-$ $\frac{5.33}{5.44x_1^4-0.09x_1^2+5.34}$
E9	$\log(\frac{x_1+0.5}{0.5+2x_0^2})$	$\log(\frac{0.79}{\lvert x_0\rvert})-$ $2.36e^{-x_1}$	$1.12\log(\lvert x_1/x_0\rvert) - 1.37$	$2 - 0.60\log(13.36(0.004 - x_0)^2$ $(1 - 0.13/(-0.06x_1 - 0.02))^2 + 0.8)$	$-\log(13.95x_0^2 + 3.48)+$ $\lvert\log(8.32x_1 + 4.18)\rvert{-0.18}$
E10	$\sin(x_0e^{x_1})$	$x_0e^{-\sqrt{\lvert x_1\rvert}}$	$\sin(x_0e^{x_1})$	$-0.98\sin((0.06 - 2.86e^{1.03x_1})$ $(0.32x_0 + 0.002)) - 0.007$	$\sin(x_0e^{0.999\,x_1})$
E11	$x_0\log(x_1^4)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$-0.74x_0(-5.62\log(0.07\lvert -6.94x_1 + 0.13\rvert$ $+0.01) - 3.74)$	$1.998x_0\log(x_1^2)$
E12	$\sin(\frac{x_0}{x_1/0.12})\lvert x_0\rvert{+0.99}$	$(x_0 + \sqrt{\lvert x_1\rvert}-$ $0.91)\sin(\frac{0.73}{x_1})$	$(x_0 + x_1)\sin(1/x_1)$	$(0.79x_0 - 0.04)\sin(4.5/(3.4x_1 + 0.08)$	$x_0\sin(1/x_1) + 1$
E13	$\sqrt{x_0}\log(x_1^2)$	$\sqrt{e^{\sqrt{x_0}}}\log(\lvert x_1\rvert)+$ $\log(\lvert x_1\rvert) + 0.58$	$0.31x_0 + 3.19\log(x_1^2) - 3.25$	$(-90.0 + \frac{9}{0.12\lvert 3.4x_1+0.12\rvert+0.04})$ $(0.09 - 0.1\log(0.17x_0 + 3.29))$	$2\sqrt{x_0}\log\lvert x_1\rvert$

Table A.4: Comparison of predicted expressions — Iteration 2

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.607x_0x_1$	$0.597x_0x_1$	$0.586x_0x_1 + \cos(0.593(x_0 - 0.979x_1)^2)$	$(0.081x_0 - 17.975)(0.051\sqrt{0.008x_1 + 1} + 0.006)$ $\cos(1.744(0.045(0.001\|15810201.601(0.292x_0 + 1)^3$ $+0.016\|+1)^2 + 1)^{0.5} - 0.007) + (2.709x_0 - 0.014)$ $(0.223x_1 + 0.002)$	$0.61x_0x_1+$ $1.041\sin((2.243x_0 - 1.512)(x_1 - 0.669) + 0.005)$
E2	$0.726x_0 - 0.759x_2 + 12.944\tanh(0.107x_2)-$ $4.572 + 11.225e^{-0.099x_0}$	$2.95(\sqrt{e^{\sqrt{\log(2x_1)}} + 0.206 + e^{-0.294x_0}}$ $+e^{\tanh(x_2)} + e^{\tanh(\log(x_0))}+$ $\sin^{0.5}(1.821((-e^{\log(x_0\tanh(x_0)+x_1)^{0.5}} - e^{-0.294x_0})^{0.5} + 0.179)^{0.5}$ $+e^{\tanh(x_2)} + e^{\tanh(\log(x_0))} + \sin^{0.5}(x_2) + e^{-0.294x_0}) + e^{-0.294x_0})^{0.5}$	$-x_0 + 0.001e^{0.597x_1} + e^{\sin(0.175x_2)} + 7.267$	$-0.002x_1 + ((0.016x_0 + 0.004)(1.231x_0 - 9.407)+$ $\sin(0.235x_2 - 0.031) - 0.007)$ $(0.004x_0 + 0.087x_1 - 0.081x_2 + 3.316) + 6.19$	$-0.497x_0 + 0.063x_0^2 + (0.648\sqrt{0.1x_1 + 1}+$ $0.002)(4.913\sin(0.205x_2) + 0.003) + 6.497$
E3	$0.15e^{1.5x_0} + 0.5\cos(3.0x_1)$	$-x_0 - (-x_0 - 0.868(e^{x_0})^{0.5} + e^{x_0})^{0.5}+$ $e^{x_0} - \cos(\cos^{0.5}(\frac{x_1}{x_0})) + 0.092$	$0.581e^{x_0} - 0.463\sin(1.104(0.003x_1 + 1)^2)$	$0.177\sqrt{(0.58e^{3.132x_0} + 1}+$ $0.486\cos(4.706(x_1 + 0.016)^2 - 0.066) - 0.104$	$0.151e^{1.499x_0} + 0.499\sin(3x_1 + 1.574)$
E4	$0.091\cosh(x_2) + \cosh(0.064x_0^2 - 0.052x_1-$ $0.052x_3 + 1.154) - 1.914$	$\sqrt{x_0}\log(x_0) + x_2 - e^{\tanh(\log(x_2))}$	—	$0.002\|31.163x_3 + 5.524(x_0 + 0.023)^4+$ $5.498(0.985x_1 - (x_2 - 0.038)^2 + 0.042)^2 - 1.464\|$	$-0.02x_0^2x_1 - 0.004x_0^2 + 0.01x_0^4 + 0.01x_1^2$ $-0.02x_2^2x_3 + 0.01x_2^4 + 0.01x_3^2 + 0.01$
E5	$e^{1.2x_3}$	$x_3^{0.5}e^{x_3} + \sin^{0.5}(0.056e^{x_3})$	—	$0.96e^{1.218x_3} + 0.927\sin(221.795x_1 + 27.668) - 0.004$	$e^{1.2x_3} - \sin(x_0 + x_1x_2 + 9.43)$
E6	$\frac{\cos(0.2x_2^2)}{\cos(1.927\tanh(\tanh(0.281x_1)))} + \tanh(x_0)$	$\tanh(x_0) + 0.327 + \frac{4.831\sin(x_2)}{x_2}$	$0.149x_0 + x_1\sin(0.29x_1 + \frac{x_2}{x_1})$	$(5.971\|0.169x_1 - 0.005\|+0.14)$ $\cos((1.531 - 17.089x_2)(-0.012x_2 - 0.009))+$ $0.8\arctan(0.695x_0 + 0.196) - 0.01$	$-(0.997\|x_1\|+0.008)\sin(0.201x_2^2 - 1.611)+$ $\tanh(0.492x_0) - 0.008$
E7	$-x_1^2\sinh(0.492\cos(6.274x_0 + 1.574)+$ $0.778) + 1.049$	$-0.974x_1^2 + \|x_1\|^{0.5}+0.008$	$\frac{0.289x_0+x_1^2}{\cos(2.667(0.131x_1-1)^2)-2.044}$	$(0.009\sin(7.746x_0 + 0.025) + 7.454)(0.002(0.055x_0 - 1)^3-$ $0.863(x_1 - 0.001)^2 + 0.696)/(6.13\sin(7.746x_0 + 0.025) + 9.52)$	$\frac{2.785x_1^2+0.05\sin(6.283x_0-3.149)-2.793}{2.77\sin(6.283x_0-3.149)-4.167}$
E8	$2.036\cosh(\tanh(x_0))+$ $3.34\cosh(\tanh(\tanh(x_1\tanh(x_1)))) - 5.5$	$\frac{\tanh(x_1)}{\cos(\tanh(x_0))}$	$\cos(\frac{\sin(\frac{x_0}{x_1})}{x_0}) + 0.629$	$-1.35\arctan(-0.096\|2.433x_1 + (0.328x_0 - 0.01)$ $(-2.094x_1 + (0.008 - 1.796x_0)(0.465 - 13.81x_1)$ $(-0.835x_1 - 0.038) + 0.128) + 0.113\|-0.04) - 0.035$	$2 - \frac{12.162}{-0.026x_1+0.132x_1^2+0.084x_1^4+12.008x_1^4+12.144}-$ $\frac{14.373}{14.113x_0^4+0.232(-x_0)^2+14.361}$
E9	$\tan(1.044\cos(0.627x_0) + 0.242)+$ $\tan(\cos(e^{\cos(0.893x_1^{0.5})})) - 2.117$	$-0.906\log(\|x_0\|) + \log(\|x_1\|)\tanh(x_1) - \|x_0 + 0.119\|^{0.5}$	$\log(\|\frac{x_1}{x_0}\|) - 1.272$	$\frac{(1.9-0.6\log(13.214(x_0+0.001)^2+0.6))(0.23x_1+0.231)-0.6}{0.23x_1+0.231}$	$-\log(30.042x_0^2 + 7.509)+$ $\|\log(13.19x_1 + 6.59)\|+0.13$
E10	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$-0.02x_0 - 0.982\sin((29.612x_0 + 0.106)(10.592e^{0.571x_1}$ $-0.033)(0.017\cos(0.07e^{0.902x_1} - 4.53) + 0.002)) - 0.003$	$\sin(x_0e^{x_1})$
E11	$2x_0\log(x_1^2)$	$4x_0\log(\|x_1\|)$	$x_0\log(x_1^4)$	$x_0(6.669\log(0.19(\|22.158x_1 + 0.255\|+0.111)^{0.5}$ $-0.01) + 0.1)$	$2x_0\log(x_1^2)$
E12	$x_0\sin(\frac{1.0}{x_1}) + 1.0$	$0.752x_0\tanh(\frac{0.985}{x_1}) + 0.752$	$(x_0 + x_1)\sin(\frac{1}{x_1})$	$0.997 - 7.62\sin(\frac{(0.131x_0-0.001)(0.004x_1-6.7)}{6.661x_1+0.137})$	$x_0\sin(\frac{1}{x_1}) + 1.0$
E13	$\frac{x_0(\tan(\sinh(\sinh(\cosh(\tanh(0.309\cosh(x_1))))))+7.703)}{x_0+8.766}$	$2\sqrt{x_0}\log(\|x_1\|)$	$0.231x_0 + \log(x_1^2)$	$(0.091\log(0.174x_0 + 0.772) + 0.086)$ $(27.6\log(0.711(x_1 + 0.043)^2 + 0.13) - 0.099)$	$2.0\sqrt{x_0}\log(\|x_1\|)$

Table A.5: Comparison of predicted expressions — Iteration 3

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.607x_0x_1+$ $1.1\cos(-2.25x_0x_1+1.5x_0+1.5x_1+0.571)$	$0.6x_0x_1$	$0.586x_0x_1+\cos(0.585(0.991x_0-x_1)^2)$	$0.255x_0(2.399x_1+0.034)+$ $1.11\cos(3089.014x_0+1035.398)-0.006$	$0.607x_0x_1+$ $1.123\sin((2.243x_0-1.496)(x_1-0.654)+0.014)$
E2	$e^{e^{-\sin(0.128x_0)}}+$ $1.214e^{\sqrt{e^{\tanh(x_2)}}}+\tanh(\frac{x_1}{x_2})$	$-0.503x_0+0.001x_1+0.211x_2+8.607$	$-x_0+0.384e^{0.024x_1}+e^{\sin(0.175x_2)}+6.813$	$(0.776-0.111x_0)(0.171x_1+(0.174x_0+0.019)$ $(0.001x_2-0.07)(-0.499x_0+0.042\cos(1.689x_1+50.388)$ $+42.271)+0.017)-2.93\cos((2.184x_1+86.243)$ $(0.025\sin(0.116x_2-0.038)+0.016))+6.62$	$-0.5x_0+0.062x_0^2+$ $(3.163\sqrt{0.1x_1+1}+0.001)\sin(0.2x_2)+6.513$
E3	$0.15e^{1.499x_0}+$ $0.497\cos(3.001x_1+\frac{1.557}{x_0+22.291})$	$0.346e^{x_0}\log(1.348\lvert(-0.247x_0+e^{x_0}$ $-0.247e^{0.5\lvert\sin(x_1)\rvert^{0.5}})^{0.5}\rvert)+0.107$	$0.582e^{x_0}-0.749\sin(0.57(1-0.001x_1)^2)$	$0.113e^{1.594x_0}+$ $0.511\cos(3.08(x_1+0.032)^2+0.008)+0.136$	$0.151e^{1.497x_0}-0.5\sin(3.004x_1-7.854)$
E4	$-0.02x_2^2(\frac{2.628x_1\sin(x_2)}{x_2}+x_3)-$ $0.007(x_1-13.401)(\cosh(x_0)+\cosh(x_2))$	$x_0+x_2-\lvert x_0\rvert^{\frac{1}{4}}\lvert x_2\rvert^{\frac{1}{4}}+0.778$	—	0.0	$-0.004x_0-0.02x_0^2x_1+$ $0.017(-x_0)^2+0.009(-x_0)^4+0.008(-x_1)^2+$ $0.01(-x_2)^4+0.011(-x_3)^2-(0.013x_2^2+0.003)$ $(1.575x_3+0.614)-0.024$
E5	$e^{1.2x_3}+\sin(x_0+x_1x_2)$	$2e^{x_3}-\log(3e^{x_3}+0.856\sqrt{e^{(x_3)}\lvert x_3\rvert}-$ $\log(2.32e^{x_3}+0.856\sqrt{e^{(x_3)}\lvert x_3\rvert})-0.26)-0.26$	—	$0.871e^{1.253x_3}+0.981\cos(88.561(0.014\lvert 3.137x_1+$ $31752866.296(0.104\arctan(0.087x_2+$ $5.508)-1)^2-2.465\rvert-1)^{0.5}-0.006)+0.13$	$1.006e^{1.201x_3}+1.006\sin(1.008x_0+$ $(1.005x_1-0.006)(x_2+0.029)+12.451)-0.007$
E6	$\lvert x_1\rvert\cos(0.2x_2^2)+\tanh(0.502x_0)$	$\frac{x_1\sin(x_2)}{x_2\tanh(x_1)}+\cos(x_2)+\tanh(x_0)$	$0.144x_0+x_1\sin(0.065x_1+\frac{x_2}{x_1})$	$(0.002\sin(0.816x_1+0.616)+0.151)$ $(0.036x_1+\cos(0.711(0.014-x_2)^2-0.573)+3.995)$	$1.001\cos(0.2x_2^2+0.01)\lvert x_1\rvert+1.0\tanh(0.5x_0)+0.002$
E7	$-0.412x_1^2\cosh(1.055e^{-0.743\sin(6.278x_0)})+1$	$-0.974x_1^2+\sqrt{\lvert x_1\rvert}$	$\frac{0.072x_0+x_1^2}{\cos(3.099(0.012x_1-1)^2)-0.252}$	$(0.002(1-0.256\sin(0.183\lvert-32.468x_0+$ $0.047x_1+14.284\rvert-0.293))^3-0.026)(-0.043x_0+$ $19.294(x_1+0.017)^2-0.543)$	$\frac{5.18x_1^2-5.18}{5.18\sin(6.283x_0+3.142)-7.77}$
E8	$2.046\cosh(\tanh(x_0))+$ $2.046\tanh(\frac{x_1}{\tanh(\sinh(1.875x_1))})-3.196$	$1.148\lvert x_0\rvert^{\frac{1}{4}}\lvert x_1\rvert^{\frac{1}{4}}-0.105$	$\cos\left(\frac{\sin(\frac{x_0}{x_1})}{x_0}\right)+0.69$	0.865	$2.0-\frac{14.485}{0.142x_0^2+14.37(-x_0)^4+14.466}+$ $\frac{13.074}{-13.081x_1^4-13.074}$
E9	$1.172\sqrt{\lvert x_1\rvert}-1.518\sinh(\sinh(\tan(\tanh($ $1.739e^{\cos(\cos(0.217x_0)}+\cos(\tanh(x_0)))))))+2.232$	$-\log(\lvert x_0\rvert)+\tanh(\log(\lvert x_1\rvert))-\sqrt{\lvert\log(\frac{0.568}{\lvert x_0\rvert})\rvert}$	$\log\left(\frac{0.286\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-0.098-0.052(-0.003(0.029x_1-1)^3+0.005+$ $\frac{6.93}{-0.001x_0-51.126x_1+(3.21-0.085\lvert 32.081x_0-1.081\rvert)(2.068x_1-69.713)+126.399})$	$-0.999\log(11.668x_0^2+2.91)+$ $1.006\lvert\log(13.4x_1+6.834)\rvert-0.861$
E10	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$1.0\sin(0.035x_0(26.26e^{1.16x_1}-0.1))-0.001$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2x_0\log(x_1^2)$	$4.545x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$(0.058\lvert 15.4\arctan(0.258x_1+0.008)-0.009\rvert-0.005)$ $(-0.006x_0+59.4\sin(0.168x_0-0.001)+0.011)$	$2.0x_0\log(x_1^2)$
E12	$x_0\sin\left(\frac{1.0}{x_1}\right)+1$	$2.77\log\left(\sqrt{\lvert x_0+x_1\rvert}\right)$ $\tanh\left(\frac{0.5(x_0+x_1)}{x_1}\right)$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	$7.5\sin\left(0.001x_0\left(0.1-\frac{24.6}{-0.159x_1-0.005}\right)\right)+1.0$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$2\log(\lvert x_1\rvert)\sqrt{\lvert x_0\rvert}$	$0.241x_0+\log(x_1^2)$	$1.08\log(0.02(-x_0-0.825)^2(-x_1-0.057)^2$ $(-x_1-0.056)^2+0.101)+0.09$	$\sqrt{1.0x_0}\log(x_1^2)$

Table A.6: Comparison of predicted expressions — Iteration 4

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.608x_0x_1+$ $1.1\cos(1.076(1.045x_0-0.697)(2x_1-1.333)-1.571)$	$0.602x_0x_1$	$0.586x_0x_1+\cos(0.586(x_0-0.988x_1)^2)$	$0.03x_0(19.977x_1+0.327)-$ $1.1\cos(31.768x_1+2.118)+0.016$	$0.607x_0x_1+$ $1.1\sin((2.25x_0-1.489)(x_1-0.663))$
E2	$(0.198x_1+2.985)\sin(0.201x_2)+$ $\cosh\left(0.836e^{\tanh\left(2.02e^{0.162x_0}\right)}-4.76\right)$	$1.936\sqrt{x_0(\tanh(x_0)-0.837)}+$ $1.936(\log(e^{x_2}+11.111\sin(\cos(e^{x_0}))+$ $11.111\sqrt{\left\lvert x_0\left(x_2+3.147\sqrt{\lvert x_0\rvert}+0.984\right)\right\rvert}))^{0.5}$	$-x_0+e^{\sin(x_2)}+7.289$	$0.075x_0^2-0.443x_0+0.014x_1x_2+$ $0.002x_1-0.002x_2^2+0.083x_2+$ $3.07\sin(0.227x_2+0.025)+6.23$	$-0.5x_0+0.063x_0^2+$ $\left(3.148\sqrt{(0.1x_1+1)}+0.012\right)\sin(0.204x_2)+6.493$
E3	$13.457e^{\frac{1.753x_0-5.251}{\cosh(\tanh(\cos(1.525x_1)))}}$	$-x_0+e^{x_0}-$ $\sqrt{\left\lvert x_0-e^{x_0}-\sqrt{\lvert x_0\rvert}+0.785\sqrt{\lvert e^{x_0}-0.083\rvert}\right\rvert}$	$0.584e^{x_0}-0.411$	$0.107x_0+0.156(0.318x_0+1)^2(x_0-0.002)^2$ $(\lvert 0.279x_0+0.578\rvert+0.133)^2-$ $0.501\sin(0.091x_0+3.572x_1-1.273)+0.152$	$0.147e^{1.508x_0}+0.5\sin(2.999x_1+1.571)+0.002$
E4	$0.091x_1(\cos(0.643x_0)-1.803)-0.044x_3\cosh(0.65x_2)+$ $0.091\cosh(x_0)+0.091\cosh(x_2)-0.027$	$\frac{x_2}{\tanh(x_2)}+$ $\log(\lvert x_0\rvert)\sqrt{\lvert x_0\rvert}-1.571$	—	$(0.095-0.001x_1)(0.1\left(0.559x_1-(x_0-0.003)^2+0.037\right)^2$ $+0.073)+0.036(-0.524x_3+(0.204x_0-0.001)(0.142x_2+0.013)+$ $(0.017x_2+0.001)(32.565x_2+1.169)+0.024)^2-0.001$	$-0.02x_0^2x_1+0.011x_0^4+0.009x_1^2+$ $0.011x_3^2-0.013(-x_0)^2+0.009(-x_2)^4-$ $(0.035x_2^2-0.002)(0.584x_3-0.46)-0.016$
E5	$e^{1.2x_3}+\sin(x_0+x_1x_2)$	$e^{x_3}\sqrt{\lvert x_3\rvert}+\tanh(0.515e^{x_3}-$ $0.331\sin(\sqrt{e^{x_3}\lvert x_3-0.662\rvert}))$	—	$0.96e^{1.21x_3}-0.963\sin(-0.001x_0\left(0.03+\frac{90.6}{0.216x_1-0.235}\right)+$ $0.555x_1+0.03x_2+(0.737-6.778x_0)(0.007x_1-1.38)+$ $0.898)+0.055$	$0.997e^{1.201x_3}+0.999\sin(x_0+$ $1.0x_1x_2+0.017)+0.004$
E6	$\sqrt{(x_1^2)}\cos(0.2x_2^2)+\tanh(0.5x_0)$	$\left(-\log(\lvert x_2\rvert)+\lvert x_1\rvert^{0.5}\right)(\cos(x_2)+0.604)+$ $\cos(x_2)+\tanh(0.142x_0)$	$0.173x_0+x_1\sin\left(0.284x_1+\frac{x_2}{x_1}\right)$	$0.011x_1-(1.57-0.001x_2)\arctan(-0.155x_0+$ $(0.055\sin(0.173x_1+0.031)-0.005)$ $(53.8\cos(0.171x_1-0.172(x_2+0.075)^2+$ $0.011)+0.006)+0.004)-0.006$	$0.997\cos(0.202x_2^2+6.236)\lvert x_1\rvert+1.0\tanh(0.5x_0)$
E7	$\frac{-1.0x_1^2+\cos(0.011x_1\sin(6.283x_0))}{\sin(6.283x_0)+1.5}$	$-0.974x_1^2+\lvert x_1\rvert^{0.5}$	$\frac{0.059x_0+x_1^2}{\cos\left(1.577(-0.668x_1-1)^3\right)-1.862}$	$\left(11.393(x_1-0.128)^2+11.849\right)$ $(-0.02(-0.021\cos(1254.632(1-0.528x_0)^2+$ $0.173)-1)^2-0.022)$	$\frac{5.94x_1^2-5.979}{-5.941\sin(6.283x_0-6.283)-8.91}-0.004$
E8	$4.404\tanh\left(\cosh(0.869x_0)\sqrt{\tanh(\cosh(x_1))}\right)$ $\tanh(\cosh(0.869x_1))-2.427$	$1.108\lvert x_0\rvert^{\frac{1}{4}}\lvert x_1\rvert^{\frac{1}{4}}$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.644$	$2.0-\frac{0.9}{0.006\left\lvert(3.121x_1+0.11)\left(24.018(x_0-0.003)^2+6.0\right)\right\rvert+0.5}$	$2.0-\frac{15.123}{15.039x_1^4+15.129}-\frac{15.086}{15.06x_0^4+15.09}$
E9	$1.141\sqrt{x_1}+1.141(-0.895x_0+$ $(0.059x_0^2-1.08)\tanh(3.763x_0))\tanh(x_0)$	$0.26x_1+\log\left(\frac{0.984}{\lvert x_0\rvert}\right)-$ $\sqrt{\left\lvert\log\left(\frac{0.593}{\lvert x_0\rvert}\right)+\log\left(\left\lvert\tanh\left(\frac{x_1}{x_0}\right)\right\rvert\right)\right\rvert}$	$\log\left(\frac{0.282\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-0.6\log\left(10.525(0.003-x_0)^2+0.6\right)+1.9-\frac{0.6}{0.229x_1+0.235}$	$-1.0\log(14.156x_0^2+3.539)+$ $1.0\lvert\log(9.499x_1+4.75)\rvert-0.294$
E10	$\sin\left(x_0e^{x_1}\tanh\left(e^{e^{\tanh(\cosh(\tan(e^{x_1}+0.775)))}}\right)\right)$	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$0.001-1.0\sin(0.561x_0(-0.002+$ $\frac{17.5}{\left(-88.065-\frac{0.009}{7.629x_0+19.891}\right)(0.001-0.025x_1)+0.364(0.008x_1-1)^2-7.514(0.732x_1-1)^2-5.172}))$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2x_0\log(x_1^2)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$-2.266x_0(0.018-0.26\log(13.784(0.002-(x_1-0.024)^2)^2+0.008))$	$2.0x_0\log(x_1^2)$
E12	$x_0\cos\left(1.659-\frac{0.088x_0x_1+x_0}{x_0x_1}\right)+1.0$	$\left(x_0\tanh\left(\frac{0.664}{x_1}\right)+1\right)\sqrt{\lvert\tanh(x_1)\rvert}$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	$0.998-6.35\sin\left(\left(0.006+\frac{0.673}{0.173x_1+0.007}\right)(-0.04x_0-0.002)\right)$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$2\log(\lvert x_1\rvert)\sqrt{\lvert x_0\rvert}$	$0.234x_0+\log(x_1^2)$	$\left(5.75-\frac{8.24}{1.265(\lvert x_1\rvert+0.008)^{0.5}+0.135}\right)(2.72\log(0.174x_0+0.991)+0.188)$	$\sqrt{1.0x_0}\log(x_1^2)$

Table A.7: Comparison of predicted expressions — Iteration 5

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.608x_0x_1 +$ $1.1\sin(2.25x_0(x_1-0.667)-1.5x_1+1.0)$	$0.616x_0x_1$	$0.59x_0x_1 +$ $\cos\left(0.584(0.989x_0-x_1)^2\right)$	$0.021x_0(29.332x_1-0.194)+1.1\cos(30.949x_1-0.049)+0.01$	$0.607x_0x_1 -$ $1.1\sin((2.252x_0-1.492)(x_1-0.666)+9.432)$
E2	$-0.5x_0-(-0.179x_1-2.984)\sin(0.2x_2)$ $+e^{e^{\cos(\cos(0.111x_0))}}+1.041$	$-0.494x_0-0.005x_1+0.204x_2+8.593$	$-x_0+e^{\sin(x_2)}+7.456-0.05e^{-0.102x_1}$	$-0.016x_0+0.049x_2+(0.086x_0-0.704)(0.75x_0-0.015)+$ $(0.003x_1+0.052)(-0.97x_1+$ $53.2\sin(0.221x_2+0.035)+0.064)+6.444$	$-0.5x_0+0.063x_0^2+$ $3.16\sqrt{0.1x_1+1}\sin(0.202x_2)+6.499$
E3	$0.15e^{1.5x_0}+0.5\cos(3.0x_1)$	$-x_0+e^{x_0}-$ $\sqrt{\left\lvert x_0+0.838\sqrt{e^{x_0}}-e^{x_0}-\sqrt{\lvert x_0\rvert}\right\rvert}$	$0.585e^{x_0}-0.437\sin\left(1.612(0.04x_1+1)^2\right)$	$0.224e^{0.188x_0+(0.004x_0+0.045)(22.712x_0+0.009x_1-5.15)}$ $+0.514\cos(3.152x_1+0.082)+0.043$	$0.15e^{1.5x_0}+0.5\sin(3.0x_1+1.571)$
E4	$(\cosh(x_0)+\cosh(x_2))$ $(-0.005x_1-0.005x_3+0.091)$	$\frac{x_2}{\tanh(x_2)}+\log\left(\left\lvert\frac{x_0}{\tanh(x_0)}\right\rvert\right)\sqrt{\lvert x_0\rvert}-1.797$	—	$0.002\lvert 30.73x_3+5.334(x_0+0.019)^4+$ $6.811(x_1-0.92(x_2-0.021)^2+0.028)^2+1.185\rvert$	$-0.02x_0^2x_1+0.001x_0^2+0.01x_0^4+$ $0.01x_1^2-0.02x_2^2x_3+0.01x_2^4+$ $0.01(-x_3)^2-0.001$
E5	$e^{1.2x_3}+\sin(x_0+x_1x_2)$	$e^{x_3}\lvert x_3\rvert^{0.5}+0.487$	—	$0.903e^{1.231x_3}+0.105+$ $1.0\cos\left(\frac{0.764}{0.062\sin((-0.619x_0-0.079)(0.344x_1-0.212x_2-0.857))-0.007}\right)$	$0.977e^{1.201x_3}+$ $1.0\sin(x_0+x_1x_2+18.85)+0.001$
E6	$0.52\cos(0.2x_2^2)+\tanh(x_0)$ $\tan\left(\cos\left(\tanh\left(\frac{1.9}{x_1}\right)\right)+0.534\right)$	$\frac{x_1\sin(x_2)}{x_2\tanh(x_1)}-1.185\cos\left(e^{\lvert x_2\rvert^{0.5}}\right)$	$-0.376x_0+x_1\sin\left(\frac{x_0}{x_1}\right)$	$(-0.001\lvert 4.547x_1-0.068\rvert-0.601)$ $\arctan\left(8.32x_0-242.537(0.016x_0+1)^3+7.105\right)+$ $0.005\cos\left(0.667(-x_2-0.162)^2-4.352\right)-0.004$	$1.0\cos(0.2x_2^2+6.282)\lvert x_1\rvert+1.0\tanh(0.5x_0)$
E7	$4.26x_1^2(\tanh(\sin(6.283x_0)+1.8)-1.1)$ $+0.967$	$-x_1^2+\sqrt{\lvert x_1\rvert}$	$\frac{-3967.593x_0+x_1^2}{\cos\left(65757.558(-x_1-0.157)^3\right)+28400.518}$	$\left(0.09-0.083(x_1-0.002)^2\right)$ $\left(6.724(1-0.756\sin(6.11x_0-0.012))^2+4.0\right)$	$(0.927-0.948x_1^2)$ $\left((\sin(6.283x_0)+1.478)^{-1.0}+0.025\right)+0.02$
E8	sinh(0.406 cosh(tanh(x_0 sinh(tanh(sinh (x_0)))))+ $\tanh(\cosh(0.895x_1))+0.69)-3.031$	$1.07\lvert x_0x_1\rvert^{\frac{1}{4}}$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.653$	$0.536-0.903\arctan((0.046x_1-0.991)$ $(-0.367\sin(2.487\sqrt{(-\lvert 32.358x_1+1.7\rvert-0.017)^2+0.006)}$ $-49.4)+0.017\lvert(0.053(x_0-0.007)^2+0.016)$ $(0.001x_1+425.696(x_1+0.011)^2-22.805)\rvert-0.286))$	$\frac{15.997}{(0.026x_1^3-15.998x_1^4-15.994)}-$ $\frac{16.587}{(-0.001x_0+0.077x_0^2+0.012x_0^3+16.459x_0^4+16.583)}+2.0$
E9	$\sqrt{1.148x_1}+\cosh(1.354\cos(x_0)\cos(\tanh(x_0)))-$ $2.202\cosh(1.613\tanh(0.481x_0))$	$-\log(\lvert x_0\rvert)+\tanh(\log(\lvert x_1\rvert))-$ $\sqrt{\left\lvert\log\left(\frac{0.603}{\lvert x_0\rvert}\right)\right\rvert}$	$\log\left(\frac{0.28\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-0.6\log\left(12.904(x_0+0.008)^2+0.6\right)+1.9-\frac{0.6}{0.227x_1+0.236}$	$-1.001\log(18.044x_0^2+4.522)+$ $0.994\lvert\log(18.83x_1+9.24)\rvert-0.705$
E10	$\sin(x_0e^{x_1})$	$\tanh\left(\sin(e^{x_1}\sin(x_0))\sqrt{\lvert\cos(\sin(x_1))\rvert}\right)$	$\sin(x_0e^{x_1})$	$0.001-1.0\sin(0.257x_0\cdot(0.1-3.597e^{1.08x_1}))$	$1.0\sin(1.0x_0e^{x_1})$
E11	$x_0(0.943-$ $2.768\log\left(\sqrt{\log(\cosh(1.303\tanh(0.652x_1)))}\right))+$ $3.385x_0\log(\log(\cosh(x_1)))$	$3.927x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$0.257x_0(30.0\log(0.19$ $\sqrt{\lvert 21.905x_1+0.102\rvert+0.001}-0.01)+0.01)$	$2.0x_0\log(x_1^2)$
E12	$1.0x_0\sin\left(\frac{1.0}{x_1}\right)+1.0$	$\frac{1.217x_0}{1.229x_1+\frac{0.471}{x_1}}+0.961$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	1	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$2\log(\lvert x_1\rvert)\sqrt{\lvert x_0\rvert}$	$0.252x_0+\log(x_1^2)$	$(0.031-0.44\log(23.696(-x_1-0.009)^2$ $(x_1+0.201)^2(0.033x_0+0.086\lvert 0.501x_0-7.823\rvert-1)^2+$ $0.02))(-0.001x_0-8.705)(0.032x_0+0.057)$	$1.0\sqrt{x_0}\log(x_1^2)$

Table A.8: Comparison of predicted expressions — Iteration 6

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.607x_0x_1$	$0.62x_0x_1$	$0.667x_0x_1+\cos\left(0.066\left(-x_0-0.869x_1\right)^2\right)$	$0.024x_1\left(25.373x_0-0.001x_1+0.057\right)-$ $1.1\cos\left(76.2\tan\left(2.983x_1-4.272\right)+0.767\right)+0.002$	$0.608x_0x_1-$ $1.099\sin\left(\left(2.24x_0-1.544\right)\left(x_1-0.67\right)-9.46\right)$
E2	$-0.499x_0+e^{\tanh\left(e^{x_1}\right)}\tanh\left(x_2\right)+$ $\log\left(\cosh\left(x_0\right)+29.144\right)+3.287$	$-0.497x_0-0.002x_1+0.208x_2+8.571$	$93.845e^{0.002x_2}+e^{\sin\left(\frac{x_0}{x_1}\right)}-86.601$	$(35.4-0.001x_0)\left(0.022(0.299x_0-1)^2+\right.$ $(0.013x_1+0.301)\left(0.319\sin((0.145x_1+3.599)\right.$ $\left.\left.(0.052x_2-0.004))+0.061\right)+0.163\right)$	$-0.5x_0+0.063x_0^2+$ $3.162\sqrt{0.1x_1+1}\sin\left(0.2x_2\right)+6.498$
E3	$0.15e^{1.5x_0}+0.5\cos\left(3x_1\right)$	$-x_0+e^{x_0}-\sqrt{\lvert x_0\rvert}-0.506+$ $\tanh\left(x_0-e^{x_0}+\sqrt{\lvert e^{x_0}-0.977\rvert}+0.977\right)$	$0.58e^{x_0}-0.464\sin\left(1.482\left(-0.071x_1-1\right)^2\right)$	$0.212e^{1.379x_0}+0.475\cos\left(3.108x_1-0.004\right)-0.013$	$0.149e^{1.502x_0}-0.5\sin\left(3.0x_1+4.712\right)$
E4	$\left(0.09-0.009x_1\right)\cosh\left(x_0\right)+$ $\left(0.091-0.009x_3\right)\cosh\left(x_2\right)$	$\sqrt{\log\left(\lvert x_0\rvert\right)}\sqrt{\left\lvert\frac{x_2\left(-x_2\sqrt{\lvert x_0\rvert}+\log\left(\lvert x_2\rvert\right)+0.553\right)}{\tanh\left(x_2\right)}\right\rvert}$	—	$0.018(0.573x_1-(0.033x_0-0.002)(21.569x_0-1.005)+0.036)^2+$ $2.72\lvert(0.01\arctan(-(0.079e^{\frac{0.005}{18.338x_2-4.735}}-199.276)$ $e^{-\frac{0.005}{18.338x_2-4.735}})+0.001)((1.851x_2-2.372)$ $(0.283x_3-0.03)+0.25((x_2+0.005)^2-0.003)^2+0.075)\rvert+0.001$	$-0.004x_0-0.02x_0^2x_1+0.017\left(-x_0\right)^2+$ $0.009\left(-x_0\right)^4+0.008\left(-x_1\right)^2+0.01x_2^4+0.011x_3^2-$ $\left(0.013x_2^2+0.003\right)\left(1.575x_3+0.614\right)-0.023$
E5	$e^{1.2x_3}+\sin\left(x_0+x_1x_2\right)$	$e^{x_3}\lvert x_3\rvert^{0.5}+\tanh\left(\left(x_3-0.891\right)e^{x_3}+1.718e^{x_3}\right)$	—	$0.002x_2+1.106e^{1.147x_3}+0.774\cos\left(0.002x_1\right)+$ $0.942\cos\left(28.703x_2-0.956\right)-0.977$	$0.995e^{1.201x_3}+1.0\sin\left(x_0+x_1x_2-6.283\right)+0.002$
E6	$2.535e^{\tan\left(1.058\cos\left(2.528\tanh\left(\frac{x_2}{x_1}\right)\right)\right)}$ $\cos\left(0.2x_2^2\right)+\tanh\left(x_0\right)$	$-1.185\cos\left(e^{\lvert x_2\rvert^{0.5}}\right)+\frac{4.854\sin\left(x_2\right)}{x_2}$	$-0.368x_0+x_1\sin\left(\frac{x_0}{x_1}-0.001x_2\right)$	0.932	$0.999\cos\left(0.2x_2^2-0.011\right)\lvert x_1\rvert+1.0\tanh\left(0.5x_0\right)$
E7	$\frac{1.0-1.0x_1^2}{\sin\left(6.283x_0\right)+1.5}$	$-x_1^2+$ $\log\left(\left\lvert x_1^2-\log\left(\left\lvert x_1^2-\sqrt{\lvert x_1\left(x_1-0.073\right)\rvert}\right\rvert\right)\right\rvert\right)$	$\frac{0.184x_0+x_1^2}{\cos\left(1.965x_0-x_1\right)-1.93}$	$\left(6.284\left(x_1-0.007\right)^2-7.49\right)$ $\left(-0.082\left(0.01x_1+\sin\left(6.457x_0+0.14\right)-0.705\right)^2-0.098\right)$	$\left(-0.001+\frac{0.999}{\sin\left(6.283x_0-9.424\right)-1.5}\right)$ $\left(x_1^2-0.977\right)+0.018$
E8	sinh(sinh(sinh(tanh(sinh(cosh($\tanh(x_0\tanh(x_0)))$ $\cosh(\tanh(x_1\tanh(x_1)))-0.718)))))$	$1.102\lvert x_0x_1\rvert^{\frac{1}{4}}$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.643$	$2.06-$ $\frac{30.5}{6.892(0.047-x_0)^2(x_1-0.003)^2+18.24\left((1-0.006x_0)^2\right)^2-9.27\cos\left(1.329x_1-44.203\right)+10.058}$	$-\frac{16.085}{\left(16.038x_1^4+2.016.09\right)}+2.0+$ $\frac{7.918}{\left(0.158x_0^2-8.067x_0^4-7.933\right)}$
E9	$\sinh(\sinh(2.7e^{-\frac{0.259x_0^2}{x_1+0.727}-\frac{0.178x_0}{(x_1+1.017)\tanh\left(x_0\right)}}$ $-1.154))-1.787$	$-0.683\log\left(\lvert x_0\rvert\right)-1.618\sqrt{\lvert x_0\rvert}+\sqrt{\lvert x_1\rvert}$	$\log\left(\frac{0.281\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$2.964\left(0.2x_1-1\right)^3+4.352-$ $0.792\log\left(166.831\left(x_0-0.052\right)^2+17.6\right)$	$-1.0\log\left(11.241x_0^2+2.809\right)+$ $1.0\left\lvert\log\left(9.824x_1+4.914\right)\right\rvert-0.559$
E10	$\sin\left(1.0x_0e^{x_1}\right)$	$\sin\left(x_0e^{x_1}\right)$	$\sin\left(x_0e^{x_1}\right)$	$1.0\sin\left(0.005x_0\left(167.868e^{1.143x_1}-0.01\right)\right)+0.001$	$1.0\sin\left(1.0x_0e^{x_1}\right)$
E11	$2x_0\log\left(x_1^2\right)$	$2.458x_0\log\left(\lvert x_1\rvert\right)+\left(1.632x_0-0.908\right)\log\left(\lvert x_1\rvert\right)+$ $0.875\cos(3.872x_0+\left(2.732x_0-2.481\right)$ $\log\left(\left\lvert 0.754x_1\tanh\left(x_1\right)\lvert x_0\rvert^{1.5}+0.307\right\rvert\right)+$ $\left(4.458x_0-2.481\right)\log\left(\lvert x_1\rvert\right))$	$x_0\log\left(x_1^4\right)$	$\left(2.024x_0-0.006\right)\left(0.75\log\left(1.926\left(-x_1-0.005\right)^2+0.05\right)+0.068\right)$	$2.0x_0\log\left(x_1^2\right)$
E12	$x_0\sin\left(\frac{1}{x_1}\right)+1.0$	$0.806x_0\sin\left(\frac{1.229}{x_1}\right)+0.806$	$\left(x_0+x_1\right)\sin\left(\frac{1}{x_1}\right)$	$7.12\sin\left(0.001x_0\left(0.003+\frac{31.2}{0.173x_1+0.001}\right)\right)+0.996$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log\left(x_1^2\right)$	$\left(\log\left(\lvert x_0\rvert\right)\log\left(\lvert x_1\rvert\right)+\tanh\left(\log\left(\lvert x_1\rvert\right)\right)\right)$ $(\log(\lvert x_0\tanh\left(\log\left(\lvert x_1\rvert\right)\right)+\log\left(\lvert x_1\rvert\right)-0.13\rvert^{0.5})+$ $\lvert\tanh\left(x_0\tanh\left(\log\left(6.162\lvert\log\left(\lvert x_1\rvert\right)\rvert\right)\right)\right)\rvert^{0.5})$	$0.245x_0+\log\left(x_1^2\right)$	$(0.004-5.12\log(0.175x_0+1.109))$ $\left(0.945\sqrt{\sqrt{\left\lvert 0.004-\frac{49.1}{0.326x_1-0.001}\right\rvert}+0.045-0.898}-3.44\right)$	$2.0\sqrt{x_0}\log\left(\lvert x_1\rvert\right)$

Table A.9: Comparison of predicted expressions — Iteration 7

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.608x_0x_1+$ $1.1\sin\left((x_0-0.667)(x_1-0.667)\right)$	$0.607x_0x_1$	$0.61x_0x_1-\sin\left(0.001\left(0.965x_0-1\right)^2\right)$	$0.598x_0x_1-0.004x_0+0.001x_1+0.143$	$0.607x_0x_1+$ $1.098\sin\left((2.24x_0-1.544)(x_1-0.67)\right)-0.002$
E2	$3.017\sin(0.2x_2)+$ $3.017\cosh(0.162x_0-0.723)+2.818$	$-0.499x_0-0.008x_1+0.207x_2+8.586$	$96.368e^{0.002x_2}+e^{\sin(x_0x_1)}-89.13$	$(0.011x_2+0.445)(0.174x_1-0.051x_2+5.52)$ $\left(0.387(0.235x_0-1)^2+\sin(0.225x_2+0.023)+0.144\right)+4.88$	$-0.501x_0+0.064x_0^2+$ $\left(3.166(0.1x_1+1)^{0.5}+0.001\right)\sin(0.201x_2)+6.452$
E3	$0.15e^{1.5x_0}+0.5\cos(3.0x_1)$	$-2x_0+e^{x_0}-0.363$	$0.579e^{x_0}-0.437\sin\left(1.633\left(0.043x_1+1\right)^2\right)$	$\frac{\left(7.16(0.001x_0-1)^2+0.01\right)\left(0.198e^{1.42x_0}+0.519\cos(3.037x_1-0.034)-0.072\right)+0.057}{7.16(0.001x_0-1)^2+0.01}$	$0.151e^{1.498x_0}-0.5\sin(3.001x_1+4.712)$
E4	$-0.02x_0^2x_1-0.02x_2^2x_3+$ $0.091\cosh(x_0)+0.091\cosh(x_2)$	$0.756x_2\tanh(x_2)+0.756\log(\lvert x_0\rvert)\sqrt{\lvert x_0\rvert}$	—	$0.002\lvert 5.04\left((x_0+0.033)^2-0.001\right)^2+$ $5.21\left(0.818x_1+0.67x_3-(x_2+0.025)^2+0.01\right)^2-0.11\rvert+0.001$	$-0.02x_0^2x_1+0.01x_2^4-0.021x_3(-x_2)^2+$ $0.005x_3+0.01(-x_0)^4+0.01(-x_1)^2-$ $(0.004x_0-0.708)(0.014x_3^2-0.019)+0.028$
E5	$e^{1.2x_3}+\sin(x_0+x_1x_2)$	$2e^{x_3}-\log(\lvert -3e^{x_3}+\log(\lvert 2.287e^{x_3}-$ $1.447\log(\lvert\sin(x_3)\rvert)\rvert)+1.447\log(\lvert\sin(x_3)\rvert)\rvert)$	—	$1.138e^{1.163x_3}-0.123+0.953\cos(22.045x_2-14.032+$ $\frac{1.11}{-0.742x_0+0.002x_1+19.256}+\frac{2.11}{(31.854x_2+0.793)(0.001x_3+2.26)}-\frac{0.033}{3.382x_2+0.147})$	$0.999e^{1.201x_3}+1.0\sin(x_0+x_1x_2-12.566)+0.003$
E6	$x_1\cos(0.2x_2^2)\tanh(236.582x_1)+$ $\tanh(0.496x_0)$	$\frac{x_1\sin(x_2)}{x_2\tanh(x_1)}+\cos(x_2)+\tanh(x_0)$	$-0.356x_0+x_1\sin\left(\frac{x_0}{x_1}-0.009x_2\right)$	0.806	$1.0\cos(0.2x_2^2-6.275)\lvert x_1\rvert+1.0\tanh(0.501x_0)$
E7	$(1.863x_1^2-1.843)(\tanh(\sinh($ $\sin(6.281x_0))+1.345)-1.215)$	$-x_1^2+\log\left(\left\lvert x_1^2-\log\left(\left\lvert x_1^2-0.828\sqrt{\lvert x_1^2\rvert}\right\rvert\right)\right\rvert\right)$	$\frac{0.08x_0+x_1^2}{\cos\left(2.445(0.924x_1-1)^2\right)-1.916}$	$\left(0.368-\frac{4.78}{8.105-0.291x_1}\right)(0.338x_1+23.594)(0.348x_1-4.637)$ $\left(0.136(0.019-x_1)^2-0.117\right)$ $\left(-0.219\left(1-0.329\cos(6.186x_0-7.784)\right)^3-0.046\right)$	$\frac{2.907x_1^2-2.919}{2.904\sin(6.283x_0+3.148)-4.358}$
E8	$1.644\log(0.976/(\cos(\tanh(1.094x_0$ $\tanh(\sinh(\sinh(x_0)))))$ $\cos(\tanh(0.812\sinh(x_1)))))$	$\tanh((x_0+0.504)\tanh(x_0))+$ $\tanh(x_1^2-0.017)$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.695$	$2.1-\frac{0.9}{0.03\lvert(0.309x_0+0.074)(0.687x_0-0.059)(2.086x_1-0.139)(17.421x_1-0.429)\rvert+0.6}$	$2.0-\frac{14.033}{13.947x_0^4-0.122x_0^3+14.032}-$ $\frac{14.438}{14.532x_1^4+14.434}$
E9	$0.226\cos(\cos(x_0))+\sinh(\sinh(\sinh($ $0.974\cos(0.643x_0)+0.167)))-\frac{8.714}{x_1+2.537}$	$0.26x_1+\log\left(\frac{0.952}{\lvert x_0\rvert}\right)-$ $\sqrt{\left\lvert\log\left(\frac{0.593}{\lvert x_0\rvert}\right)+\log\left(\left\lvert\tanh\left(\frac{x_1}{x_0}\right)\right\rvert\right)\right\rvert}$	$\log\left(\frac{0.274\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$2.0-0.6\log\left(6.862\left(1-\frac{0.129}{-0.056x_1-0.02}\right)^2(x_0-0.009)^2+0.8\right)$	$-1.0\log(2.785x_0^2+0.696)+$ $1.0\lvert\log(13.845x_1+6.92)\rvert-2.296$
E10	$\sin(1.0x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$-0.998\sin\left((0.214-33.248x_0)(0.026e^{1.125x_1}+0.001)\right)$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2.0x_0\log(x_1^2)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$0.241x_0(31.0\log(0.19\sqrt{\lvert 25.419x_1+0.12\rvert+0.011}$ $-0.01)+0.5)$	$4.0x_0\log(\lvert x_1\rvert)$
E12	$x_0\sin\left(\frac{1}{x_1}\right)+1$	$\frac{x_0\tanh(x_1^2)}{x_1}+1$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	$\left(0.629-4.58\sin\left(\left(-0.007+\frac{9.78}{0.014-3.043x_1}\right)(0.042x_0+0.001)\right)\right)$ $(1.581-0.006x_0)$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$2\log(\lvert x_1\rvert)\sqrt{\lvert x_0\rvert}$	$0.23x_0+\log(x_1^2)$	$\left(-0.052+\frac{0.004}{-0.019-\frac{4.486}{5.98x_0-68.5}}\right)(48.802\log(0.012\lvert 0.001-$ $\frac{6.156}{(0.243x_1-0.002)(0.806\sin(0.085x_0-0.168)+0.054)}\rvert+0.025)+0.04)$	$2.0\sqrt{x_0}\log(\lvert x_1\rvert)$

Table A.10: Comparison of predicted expressions — Iteration 8

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.609x_0x_1 + 0.931\cos(1.488x_0-$ $1.126x_1(2x_0 - 3.019) - 1.9x_1 + 0.556)$	$0.613x_0x_1$	$0.666x_0x_1 + \cos\left(0.068(-x_0 - 0.83x_1)^2\right)$	$(0.001x_0 - 1.463)\sin\left(0.482\cos\left(\frac{0.121x_0+78.526}{0.007x_0+0.29}\right) - 0.004\right)+$ $(3.222x_0 + 0.007)(0.186x_1 + 0.003)$	$0.608x_0x_1+$ $1.099\sin((2.251x_0 - 1.52)(x_1 - 0.655))$
E2	$2.278\tanh(0.356x_2) + 6.388$ $\sqrt{(0.016x_0^2 + e^{-0.402x_0})^{0.5} + 0.127\tanh(x_1x_2)}$	$-0.506x_0 - 0.001x_1 + 0.206x_2 + 8.575$	$e^{\sin(x_0x_1)} + 124.329 - 116.894e^{-0.002x_2}$	$(0.01x_0 - 0.099)(5.434x_0 + 0.438)+$ $3.0\sin((0.001x_1 + 0.02)(10.39x_2 + 0.291)) + 6.3$	$-0.5x_0 + 0.062x_0^2+$ $\sqrt{0.1x_1 + 1}(3.166\sin(0.201x_2) - 0.033) + 6.541$
E3	$0.15e^{1.499x_0} - 0.502\cos(2.999x_1 + 3.143)$	$-0.612\left(e^{x_0}e^{2\tanh(x_0)}\right)^{0.5} + e^{x_0}$	$0.579e^{x_0} - 0.444\cos(0.228x_1)$	$0.143e^{1.497x_0} + 0.506\cos(3.559x_1 + 0.061) + 0.043$	$0.15e^{1.5x_0} + 0.5\cos(3.001x_1)$
E4	$(\cosh(x_0) + \cosh(x_2))$ $(-0.005x_1 - 0.005x_3 + 0.091)$	$\left(-\sin(x_0) + \sqrt{\lvert x_2\rvert}\right)e^{\tanh\left(\log(\sqrt{\lvert x_0\rvert\lvert x_2\rvert})\right)}$	—	$0.079\lvert 0.162(-x_1 + 0.885(x_0 + 0.05)^2 + 0.031)^2+$ $0.123(0.135x_0x_1 + 0.004x_0 + 0.011x_1-$ $x_2^2 - 0.022x_2 + 0.884x_3 + 0.024)^2 + 0.523\rvert - 0.048$	$-0.005x_0^2 + 0.01x_0^4 - 0.02x_1(-x_0)^2+$ $0.01x_1^2 + 0.01x_2^4 - 0.02x_3(-x_2)^2$ $+0.01x_3^2 + 0.015$
E5	$e^{1.2x_3} + \sin(x_0 + x_1x_2)$	$0.855e^{x_3}\log\left(\left\lvert 2x_3 + \sqrt{\lvert\tanh(e^{2x_3})\rvert} + 0.748\right\rvert\right)$	—	$1.392e^{1.085x_3}+$ $0.949\sin\left(-25.875x_2 + 100.707 + \frac{0.007}{0.055-0.1x_1}\right) - 0.484$	$0.999e^{1.2x_3} + 1.0\sin(x_0 + x_1x_2 + 12.567) + 0.003$
E6	$0.279e^{3.915\cos\left(\frac{\tanh(1.761\tanh(\sin(x_1)))}{x_1}\right)\cos\left(\tanh\left(\frac{4.667}{x_1}\right)\right)}$ $\cos(0.2x_2^2) + \tanh(x_0)$	$\cos(x_2) + 0.422 + \frac{5.65\log\left(\lvert x_1\rvert^{0.5}\right)\sin(x_2)}{x_2}$	$0.126x_0 + x_1\sin\left(0.294x_1 + \frac{x_2}{x_1}\right)$	$(6.048\lvert 0.168x_1 + 0.001\rvert + 0.14)$ $\cos((0.21 - 17.355x_2)(-0.012x_2 - 0.01))+$ $0.8\arctan(0.699x_0 + 0.269) - 0.01$	$1.0\cos(0.2x_2^2 + 12.564)\lvert x_1\rvert+$ $1.0\tanh(0.501x_0)$
E7	$\cos(x_1) + \sinh(\sin(6.285x_0)+$ $\cos(0.35x_1) + \cos(0.668x_1) - 2.288)$	$-x_1^2 + \log(\lvert x_1^2-$ $\log(\lvert x_1^2 - \log(\lvert x_1^2 + 0.943\rvert)\rvert)\rvert)$	$\frac{0.042x_0+x_1^2}{\cos\left(5.419(x_1-0.517)^2\right)-1.896}$	$(-10.369(1 - 0.002x_0)^2(0.346x_1 + (0.01 - 0.003x_1)$ $(0.029x_0 - 0.003) + 0.018)^2 + 1.7)$ $\left(0.373(\sin(6.398x_0 + 0.342) - 0.814)^2 + 0.42\right)$	$\frac{3.767-3.787x_1^2}{(3.786\sin(6.283x_0)+5.68)} + 0.003$
E8	$\log(0.705\sinh(1.975$ $\cosh(\tanh(x_0)) - \frac{1.184}{\sinh(\cosh(x_1))}))$	$\frac{0.829\lvert x_0\rvert^{\frac{1}{4}}}{\cos(\tanh(x_1))}$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right) + 0.644$	$2.065(-1 - 0.556/(-0.361(x_0 + 0.028)^2$ $(0.001x_1 - 1)^2(x_1 + 0.046)^2 + 0.409\cos(2.271x_1 + 0.076)$ $-1.369))^2 + 0.001$	$\frac{14.946}{0.121x_0^3-15.228x_0^4+0.176(-x_0)^2-14.95}-$ $\frac{18.513}{0.135x_1-0.322x_1^3+18.515(-x_1)^4+18.515} + 2.0$
E9	$0.843e^{1.412\cos(x_0)} + \log(x_1 + 0.504) - 3.313+$ $0.843\cos(\cos(0.866x_0)) - 0.036\cosh(x_0)$	$\log\left(\frac{0.764}{\lvert x_0\rvert}\right) + \tanh(x_1)-$ $\sqrt{\left\lvert\log\left(\frac{0.792}{\lvert x_0\rvert}\right) + \log\left(\left\lvert\tanh\left(\frac{x_1}{x_0}\right)\right\rvert\right)\right\rvert}$	$\log\left(\frac{0.282\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-6.44\lvert 0.81\arctan(0.295x_0 + 0.004) + 0.001\rvert+$ $3.33 + \frac{96.1}{(0.021x_0-47.199)(0.062x_1+0.262)(0.023x_0+0.692x_1+2.901)}$	$-0.999\log(3.882x_0^2 + 0.963)+$ $0.999\lvert\log(10.173x_1 + 5.052)\rvert - 1.656$
E10	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$\sin(x_0e^{x_1})$	$0.994\sin((-5.26x_0 - 0.008)(0.273e^{0.573x_1} + 0.006)$ $(-0.31x_1 + 0.118\log(0.055x_1 + 0.756) - 0.782))$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2x_0\log(x_1^2)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$(0.331x_0 + 0.001)(20.8$ $\log\left(0.327\sqrt{\lvert 20.18x_1 + 0.198\rvert + 0.009} - 0.003\right) - 4.11)$	$2.0x_0\log(x_1^2)$
E12	$x_0\sin\left(\frac{\cos\left(\frac{0.001}{\frac{1581.494x_1}{\cosh(x_1)}+\cosh(x_1)}\right)}{x_1}\right) + 1$	$0.938 + \frac{-x_0x_1-0.344}{-x_1^2-0.307}$	$(x_0 + x_1)\sin\left(\frac{1}{x_1}\right)$	$7.72\sin\left(0.003x_0\left(0.008 + \frac{6.35}{0.173x_1+0.002}\right)\right)+$ $0.078\lvert 0.006x_1 - 0.247\rvert + 0.987$	$1.0x_0\sin\left(\frac{1}{x_1}\right) + 1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$0.255x_0\log(\lvert x_1\rvert) + 2.807\log(0.943\lvert x_1\rvert)$ $\lvert\log(0.826\sqrt{\lvert x_0\rvert}\lvert\log(4.902\lvert\log(4.902$ $\lvert\log(\lvert x_1\rvert)\rvert)\rvert)\rvert) + 0.13\rvert^{0.5}$	$0.239x_0 + \log(x_1^2)$	$\left(0.034 - 0.052\sqrt{\lvert 0.603x_0 + 1\rvert}\right)$ $(0.08 - 57.9\log(\lvert 7.09\lvert 0.162x_1 - 0.004\rvert - 0.001\rvert))$	$1.0\sqrt{x_0}\log(x_1^2)$

Table A.11: Comparison of predicted expressions — Iteration 9

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.607x_0x_1- 1.1\cos(x_0(2.25x_1-1.5)-1.5x_1+2.571)$	$0.613x_0x_1$	$0.61x_0x_1+\sin\left(0.002(-x_0-0.698)^2\right)$	$0.021x_1(29.356x_0+0.027)+ 1.1\cos(30.874x_1-1.378)-0.001$	$0.608x_0x_1+1.1\sin((2.256x_0-1.491) (x_1-0.655)+0.047)-0.001$
E2	$0.178x_1\sin(0.199x_2)+2.983\sin(0.2x_2)+ 2.983\log(\cosh(0.464x_0-1.809)+5.42)$	$\tanh(x_2)+\lvert(x_0(-x_0+(e^{x_0})^{\frac{1}{4}}+\log(55.556\lvert x_0\rvert)+1.704+ \cos\left(e^{1.424\log(55.556\lvert x_0\rvert)^{\frac{1}{4}}}\right)))(\tanh(x_0))\rvert^{0.5}+1.305\lvert-0.587x_0+ 0.587(e^{x_0})^{\frac{1}{4}}+0.587\log(55.556\lvert x_0\rvert)+0.587\cos(x_0)+1\rvert^{0.5}$	$93.728e^{0.002x_2}+e^{\sin(x_0x_1)}-86.434$	$0.019x_1+(0.007x_0-0.056)(8.915x_0+0.563)+ (0.437x_1+8.091) (0.371\sin(0.194x_2+0.034)-0.007)+6.279$	$-0.499x_0+0.064x_0^2+ \sqrt{0.1x_1+1}(3.161\sin(0.2x_2)+0.011)+6.46$
E3	$0.15e^{1.5x_0}+0.5\cos(3x_1)$	$-x_0+e^{x_0}+\cos(x_0)-0.815$	$0.583e^{x_0}-0.465\cos(0.204x_1)$	$0.042x_0-0.004x_1+(0.001x_0+0.216)(2.33\cos(3.43 x_1+0.075)-0.07)+0.166\sqrt{0.779e^{2.954x_0}+1}-0.008$	$0.15e^{1.5x_0}+0.5\cos(3.0x_1)$
E4	$(0.092-0.008x_1)(\cosh(x_0)+6.478)+ (0.092-0.009x_3)\cosh(x_2)-0.643$	$x_0+0.118x_2^2-0.412$	—	$0.004\lvert 2.358(x_3-0.048)^2+2.231((x_0+0.028)^2+0.039)^2+ 2.177(-0.921x_1+(x_2+0.01)^2+0.024)^2-0.255\rvert+0.003$	$-0.02x_0^2x_1+0.004x_0^2+0.01x_0^4+0.01x_1^2- 0.02x_2^2x_3+0.01x_2^4+0.01x_3^2-0.009$
E5	$(e^{2.4x_3})^{0.5}+\sin(x_0+x_1x_2)$	$(\cos(\sin(\tanh(1.515e^{x_3})))+\tanh(1.119\lvert(0.799\cos(\sin(\tanh(1.515e^{x_3})))+0.799\tanh\left(0.534e^{0.5(x_3)}\right))e^{x_3}-e^{x_3}\rvert^{0.5}))e^{x_3}$	—	$-0.001x_1+0.002x_3+0.023(0.024-x_3)^2+ 0.947e^{1.206x_3}+0.933\cos(12297.635(0.404x_1+1)^2+ 0.225(0.001x_0-0.001x_2+1)^{0.5}-15.41)+0.032$	$1.001e^{1.2x_3}+ 1.0\sin(x_0+x_1x_2+12.566)-0.008$
E6	$9.715\cos(0.2x_2^2) \cos\left(\frac{5.902}{\tanh(\cosh(0.095x_1))}\right)+\tanh(x_0)$	$\cos(x_2)+\tanh(x_0)+\frac{7.407\log\left(\lvert x_1\rvert^{0.5}\right)\sin(x_2)}{x_2}$	$0.145x_0+x_1\sin\left(0.302x_1+\frac{x_2}{x_1}\right)$	$(0.035x_1-0.002)(4.984\arctan(\left(-0.03-\frac{4.462}{-0.539x_0-0.118}\right) (0.004x_0+0.001))-0.16)+0.929$	$-1.0\cos(0.2x_2^2-9.425)\lvert x_1\rvert+ 1.0\tanh(0.5x_0)$
E7	$(2.482x_1^2-2.467) \left(\cos\left(0.435e^{\cos(6.278x_0+1.57)}\right)-1.16\right)$	$-x_1^2+\log\left(\left\lvert x_1^2+\frac{-x_1^2+\sqrt{\lvert x_1\rvert}+\sqrt{\left\lvert x_1-0.479\tanh\left(e^{-0.538x_1^2}\right)\right\rvert}}{x_1^2+0.392}\right\rvert\right)$	$-0.05632x_0-0.8403x_1^2$	$\left(0.09-0.082(x_1+0.007)^2\right) \left(6.131(0.766-\sin(6.516x_0+0.216))^2+5.0\right)$	$\frac{8.18x_1^2-8.247}{(-8.185\sin(6.283x_0-6.283)-12.273)}-0.006$
E8	$0.8\tan(\sinh(\tanh(\cosh(0.827 \cosh(\tan(\tanh(x_1))))\tanh(\cosh(x_0))))) \cosh(\tanh(x_0))-0.957$	$e^{\tanh\left(\tanh\left(\log\left(0.864\sqrt{\lvert x_0\rvert\lvert x_1\rvert}\right)\right)\right)}$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.634$	$2.1-\frac{0.9}{0.002\left\lvert(3.087x_1-0.168)\left(37.54(x_0+0.029)^3+6.0\right)\right\rvert+0.6}$	$\frac{16.205}{\left(-0.015x_0-16.207x_0^4-0.033(-x_0)^3-16.206\right)}+2.0+ \frac{11.914}{\left(0.013x_1-0.118x_1^2-0.016x_1^3-11.817x_1^4-11.901\right)}$
E9	$\log\left(\frac{x_1+0.5}{2x_0^2+0.5}\right)$	$-\log(\lvert x_0\rvert)+\tanh(x_1)-0.298- \sqrt{\left\lvert\log\left(\frac{0.873}{\lvert x_0\rvert}\right)+\tanh(x_1)-\sqrt{\lvert x_0\rvert}\right\rvert}$	$\log\left(\frac{0.281\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-0.6\log\left(10.053(0.009-x_0)^2+0.6\right)+1.9-\frac{0.7}{0.274x_1+0.301}$	$-1.001\log(13.041x_0^2+3.264)+ 0.998\lvert\log(7.365x_1+3.647)\rvert-0.108$
E10	$\sin(x_0e^{x_1})$	$\sin(\tanh(x_0))\cos\left(\sqrt{\lvert x_1\rvert}\right)$	$\sin(x_0e^{x_1})$	$-0.994\sin((3.608x_0+0.013) \left(-0.234e^{(0.595-0.053x_1)(1.976x_1+0.061)}-0.009\right))-0.001$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2.0x_0\log(x_1^2)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$(2.922x_0+0.006)(4.14\arctan(0.336- 0.008\left\lvert 0.26x_1+0.695-\frac{84.4}{3.242x_1-0.071}\right\rvert)-0.047)$	$4.0x_0\log(\lvert x_1\rvert)$
E12	$x_0\sin\left(\frac{1}{x_1}\right)+1.0$	$0.998x_0\sin\left(\frac{x_0}{x_1(x_0+0.048)}\right)\sqrt{\left\lvert\log\left(\sqrt{\lvert x_0\rvert}\right)\right\rvert}+0.998$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	$0.588+7.1\sin(0.058- \frac{9.51}{\left(0.01-\frac{103.409}{13.96x_0-0.071}\right)(-0.001x_0+10.994x_1-0.059)})$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\sqrt{x_0}\log(x_1^2)$	$\left(\sqrt{x_0\log\left(\sqrt{\lvert x_1\rvert}\right)^2}+0.689\right)\left(\log\left(\sqrt{\lvert x_1\rvert}\right)+0.446\right)+ \log(\lvert x_0\rvert)\log(\lvert x_1\rvert)+\log(\lvert x_1\rvert)+\log(\lvert x_1\tanh(x_1)\rvert)$	$0.243x_0+\log(x_1^2)$	$(0.741-4.65\arctan(0.175x_0+0.35))(1.92- 0.292\sqrt{0.001+\frac{1}{0.008\lvert 2.504x_1-0.051\rvert+0.003}})(0.174e^{0.175x_0}-20.8) (0.008\log(0.001x_1+12.9)+0.072)$	$2.0\sqrt{x_0}\log(\lvert x_1\rvert)$

Table A.12: Comparison of predicted expressions — Iteration 10

Eq.	PYSR	TaylorGP	NESYMRES	E2E	SeTGAP
E1	$0.608x_0x_1$	$\frac{0.604x_0x_1\left(x_0^2x_1^2\right)^{\frac{1}{4}}}{\sqrt{\lvert x_0x_1\rvert}}$	$0.589x_0x_1+\cos\left(0.598\left(0.966x_0-x_1\right)^2\right)$	$(23.064x_0-0.034)(0.026x_1+0.001)-0.001+1.09\sin(0.158x_0$ $+(1.872x_1+0.125)(0.003\lvert 0.021x_1-2.239\rvert+81.7)+0.021)$	$0.607x_0x_1+$ $1.099\sin((2.242x_0-1.492)(x_1-0.685))$
E2	$x_0+2.969\sin(0.2x_2)+$ $\cosh(0.083x_0-3.519)-10.302$	$-0.513x_0-0.001x_1+0.204x_2+8.572$	$-x_0+e^{\sin(x_1)}+63.876-56.599e^{-0.004x_2}$	$-0.008x_0(0.172x_0-1.607)(0.007x_1-40.299)+$ $6.26+(0.008x_1+0.149)$ $(19.2\sin((0.001x_2+0.073)(3.077x_2+0.001))+0.003)$	$-0.5x_0+0.062x_0^2+$ $3.162\sqrt{0.1x_1+1}\sin(0.201x_2)+6.501$
E3	$0.15e^{1.5x_0}+0.5\sin(3.0x_1+1.571)$	$-x_0\tanh(0.632e^{x_0}-1)+e^{x_0}-$ $\log(e^{x_0})-0.682$	$0.575e^{x_0}-0.418\sin\left(1.451\left(0.052x_1+1\right)^2\right)$	$0.003x_0+0.06e^{1.804x_0}-$ $0.499\cos(0.003x_0-2.985x_1+59.557)+0.159$	$0.147e^{1.507x_0}-0.5\sin(3.001x_1+4.712)$
E4	$-0.165x_1-0.165x_3+1.564\log($ $(\cosh(\sinh(0.083x_0^2))-0.553)$ $\cosh(x_2))+1.564\cos(x_2)$	x_0+x_2-2	—	$0.843\lvert 0.07x_3-0.013(0.836x_1-(x_0+0.059)^2$ $(0.004x_1-1)^2+0.039)^2-$ $0.01\left(0.96x_3-(x_2-0.046)^2+0.046\right)^2+0.017\rvert$	$-0.02x_0^2x_1+0.01x_0^4+0.01x_1^2+$ $0.01x_2^4+0.01x_3^2-$ $(0.02x_2^2+0.002)(x_3+0.212)+0.009$
E5	$1.0e^{1.2x_3}+$ $\sin(x_0+x_1x_2)$	$2e^{x_3}-\tanh(e^{x_3}+\sin((1.598-0.126\sqrt{\lvert x_3\rvert})$ $(e^{x_3}-0.24)))\sqrt{\lvert x_3+e^{x_3}\rvert}-0.24$	—	$0.002x_2+0.845e^{1.25x_3+0.008\arctan(-0.065x_0+0.002x_1-14.761)}+$ $0.993\cos(-16.985x_2+(-0.008x_0-37.102)$ $\left(67.0\left\lvert 0.925+\frac{0.008}{-0.573x_3-0.015}\right\rvert-59.1\right)+$ $0.088+\frac{46.3}{0.012-0.001x_0})+0.273$	$1.002e^{1.2x_3}+$ $0.999\sin(x_0+x_1x_2-6.281)$
E6	$\lvert x_1\rvert\cos(0.2x_2^2)+\tanh(0.5x_0)$	$\cos(x_2)+\tanh(x_0)+\frac{4.95\sin(x_2)}{x_2}$	$-0.368x_0+x_1\sin\left(\frac{x_0}{x_1}+0.005x_2\right)$	$(-0.007x_1-0.015)(9.191\cos((0.021-0.906x_2)(21.206$ $\lvert 3.697x_0+11.042(-x_2-0.077)^2+0.689\rvert+0.008))-0.121)-$ $0.219\arctan(-0.489x_0-0.094)+0.566$	$0.998\cos(0.201x_2^2-18.884)\lvert x_1\rvert+$ $1.0\tanh(0.501x_0)$
E7	$\frac{2.119-2.118x_1^2}{2.118\sin(6.283x_0)+3.176}$	$-x_1^2+0.802\sqrt{\lvert x_1^2+\sin(x_0)\rvert}$	$\frac{0.166x_0+x_1^2}{\cos\left(2.498(0.137x_1-1)^2\right)-2.073}$	$0.003x_0+0.144+$ $\frac{0.002-0.005(-x_1-0.014)^2}{0.006-0.001(-\sin(7.343x_0+3.499)-0.628)^2}$	$\frac{1.005x_1^2-0.873}{(-\sin(6.283x_0-6.283)-1.507)}+0.084$
E8	$\log(\tan(1.442(\tanh(\cosh(x_1)\tanh($ $\sqrt{\cosh(x_0)}))\tanh(\cosh(x_0)))^{0.5})-0.635)$	$2.037\tanh\left(0.521\sqrt{\lvert x_0\rvert\lvert x_1\rvert}\right)$	$\cos\left(\frac{\sin\left(\frac{x_0}{x_1}\right)}{x_0}\right)+0.64$	$2.0-\frac{0.9}{0.006\lvert(2.068x_0-0.4)(3.413x_0+0.229)(2.073x_1+0.038)(3.109x_1+0.206)\rvert+0.5}$	$-\frac{11.534}{(11.542x_1^4+11.535)}+2.0-$ $\frac{11.534}{(0.045x_0+0.062x_0^2+11.474x_0^4+11.528)}$
E9	$\sqrt{1.148x_1}-4.723+$ $\frac{4.49}{\cosh\left(x_0\tanh\left(\frac{2.296}{1.849-0.771\cos(x_0)}\right)\right)}$	$\log\left(\sqrt{\left\lvert\frac{x_1}{x_0}\right\rvert}\right)-1.486\sqrt{\lvert x_0\rvert}+0.889$	$\log\left(\frac{0.284\lvert x_1\rvert}{\lvert x_0\rvert}\right)$	$-0.986\log(26.495\left(-0.179-\frac{1}{0.691x_1+7.596}\right)^2$ $\left(-0.063-\frac{1}{0.691x_1+0.626}\right)^2(0.031-x_0)^2+0.224)-0.087$	$-1.0\log(10.831x_0^2+2.707)+$ $1.0\lvert\log(21.911x_1+10.961)\rvert-1.399$
E10	$\sin(x_0e^{x_1})$	$\cos\left(\sqrt{\lvert x_1\rvert}\right)\tanh(\tanh(x_0))$	$\sin(x_0e^{x_1})$	$-1.0\sin(0.026x_0(1.0-36.638e^{1.091x_1}))-0.001$	$1.0\sin(1.0x_0e^{x_1})$
E11	$2x_0\log(x_1^2)$	$4x_0\log(\lvert x_1\rvert)$	$x_0\log(x_1^4)$	$(-0.214x_0-0.001)(6.0\log(1742.4$ $\left(0.02-\frac{1}{1.244x_1+0.023}\right)^2+0.01)-50.0)$	$4.0x_0\log(\lvert x_1\rvert)$
E12	$1.0x_0\sin\left(\frac{1.0}{x_1}\right)+1.0$	$\log\left(e^{\tanh\left(\frac{x_0}{x_1}\right)}\right)+\tanh\left(\frac{x_0}{x_1}\right)+$ $\sqrt{\lvert\tanh(x_1)\rvert}$	$(x_0+x_1)\sin\left(\frac{1}{x_1}\right)$	$0.996-7.12\sin((0.01+$ $\frac{19.5}{0.006x_0-9.741x_1-0.305})(0.067x_0+0.001))$	$1.0x_0\sin\left(\frac{1}{x_1}\right)+1.0$
E13	$\log\left(3.711\sinh\left(\sqrt{\lvert\log(\lvert\cosh(0.381x_1)\rvert)\rvert}\right)\right)$ $\tanh\left(\sqrt{e^{-2.032\cosh(0.209x_1)}}+2.45\right)2.015\sqrt{\lvert x_0\rvert}$	$2\log(\lvert x_1\rvert)\sqrt{\lvert x_0\rvert}$	$0.23x_0+\log(x_1^2)$	$(2.27-0.465(0.086(\lvert(92.8\arctan(0.172x_0+20.71)+29.3)$ $(-0.015x_1+(0.913+\frac{61.081}{x_1})(0.056x_0-2.916)+0.736)\rvert+0.001)^{0.5}$ $-1)^{0.5})(3.87\log(\lvert 21.928\lvert 0.027x_0-1\rvert^{0.5}-26.9\rvert)+0.05)$	$1.0\sqrt{x_0}\log(x_1^2)$

Table A.13: Comparison of expressions learned by SeTGAP Under Noisy Conditions

Eq.	$\sigma_a = 0.01$	$\sigma_a = 0.03$	$\sigma_a = 0.05$
E1	$0.608\,x_0\,x_1 + 1.089\sin((2.263\,x_0 - 1.525)(x_1 - 0.665) - 0.011)$	$(0.611\,x_0 + 0.001)(x_1 - 0.043) - 0.885\sin((2.253\,x_0 - 1.544)(x_1 - 0.721) - 3.214) + 0.006$	$0.61\,x_1(x_0 - 0.006)-$ $0.47\sin((2.381\,x_0 - 1.171)(x_1 - 0.798) + 8.501) + 0.006$
E2	$0.063\,x_0^2 - 0.499\,x_0 + (3.078\sqrt{0.098\,x_1 + 1} + 0.062)$ $(\sin(0.199\,x_2) - 0.002) + 6.481$	$0.063\,x_0^2 - 0.499\,x_0 + (3.222\sqrt{0.099\,x_1 + 1} - 0.058)(\sin(0.201\,x_2) - 0.007) + 6.511$	$\frac{-(0.507\,x_0 - 6.557)(0.181\,x_1 - 18.566) - (0.374\,x_0^2 + 18.893\sin(0.198\,x_2))\log(x_1 + 19.824)}{0.181\,x_1 - 18.566}$
E3	$0.143e^{1.516\,x_0} + 0.506\sin(2.999\,x_1 + 1.572) + 0.009$	$0.148e^{1.505\,x_0} - 0.498\sin(3.0\,x_1 - 1.572) + 0.003$	$0.149e^{1.502\,x_0} - 0.496\sin(3.0\,x_1 - 1.581) + 0.003$
E4	$0.01x_0^4 - 0.02x_0^2x_1 - 0.007x_0^2 + 0.009x_1^2+$ $0.011x_2^4 - 0.02x_2^2x_3 - 0.02x_2^2 + 0.008x_3^2 + 0.113$	$0.01x_0^4 + 0.01x_2^4 - 0.02x_2^2x_3 + 0.01x_3^2 + 0.242\cosh(10.69\sqrt{1 - 0.016x_1} - 8.186) - 1.473$	$0.01x_1^4 - 0.021x_3^2x_4 + 0.01x_3^4 + 0.011x_4 + 0.01x_4^2+$ $(0.093x_1^2 + 0.007\lvert x_4 + 1\rvert)\sin(0.248x_2 - 2.981) + 0.124$
E5	$1.004e^{1.199\,x_3} + 0.997\sin(1.0\,x_0 + 1.0\,x_1\,x_2) - 0.005$	$(-(0.015\sin(0.508\,x_2 + 1.574) - 0.001)(\lvert 4.779\sin(1.0\,x_0 + 3.141) - 6.283\rvert - 6.283)+$ $(6.226\sin(0.508\,x_2 + 1.574) - 6.283)(0.002e^{1.202\lvert x_3 + 5.282\rvert} + 0.145\sin(1.0\lvert x_1 - 6.283\rvert - 1.847)+$ $0.067\sin(1.0\lvert x_1 + 6.202\rvert - 4.425)\lvert 2.649\sin(1.0\,x_0 + 3.24) - 2.097\rvert + 0.005))/$ $(6.226\sin(0.508\,x_2 + 1.574) - 6.283)$	$0.996e^{1.202\,x_3} + 1.0\sin(1.0\,x_0 + 1.0\,x_1\,x_2) + 0.004$
E6	$1.0\cos(0.2\,x_2^2 + 0.008)\lvert x_1\rvert + 1.0\tanh(0.5\,x_0) + 0.005$	$\cos(0.199\,x_2^2 + 0.028)(1.001\lvert x_1\rvert - 0.001) + 1.002\tanh(0.486\,x_0) + 0.02$	$1.0\cos(0.2\,x_2^2)\lvert x_1\rvert + 0.997\tanh(0.502\,x_0)$
E7	$\frac{1.001\,x_1^2 - 0.9914}{\sin(6.283\,x_0 + 3.15) - 1.501} + 0.011$	$\frac{-1.001\,x_1^2 + 1.006}{\sin(6.283\,x_0) + 1.501} + 0.007$	$-0.982\frac{x_1^2 - 0.993}{\sin(6.283\,x_0 + 3.139) - 1.492} + 0.019$
E8	$2 - \frac{11.356}{11.313x_1^4 - 0.116x_1^3 + 0.047x_1^2 + 0.078x_1 + 11.352} -$ $\frac{10.636}{10.804x_0^4 - 0.113x_0^2 + 10.646}$	$2 - \frac{15.531}{15.414x_1^4 - 0.051x_1^3 + 0.165x_1^2 + 0.018x_1 + 15.517} -$ $\frac{12.561}{12.523x_0^4 - 0.085x_0^3 + 0.062x_0^2 + 0.061x_0 + 12.553}$	$2 - \frac{12.837}{12.956x_0^4 + 0.007x_0^3 - 0.125x_0^2 + 12.861} -$ $\frac{16.19}{16.217x_1^4 - 0.007x_1^3 + 16.202}$
E9	$-1.0\log(4.884\,x_0^2 + 1.22) + \lvert\log(9.415\,x_1 + 4.709)\rvert - 1.35$	$-1.0\log(17.004\,x_0^2 + 4.251) + 0.999\lvert\log(10.158\,x_1 + 5.072)\rvert - 0.176$	$5.965\sqrt{0.625\log(9.36\,x_1 + 6.35) + 1}-$ $0.999\log(11.259\,x_0^2 + 2.817) - 7.713$
E10	$1.0\sin(1.0\,x_0e^{x_1})$	$1.0\sin(1.0\,x_0e^{x_1})$	$1.0\sin(1.0\,x_0e^{x_1})$
E11	$2.0\,x_0\log(x_1^2)$	$(4.001\,x_0 - 0.004)\log(\lvert x_1\rvert) + 0.003$	$(4.001\,x_0 - 0.008)\log(\lvert x_1\rvert) + 0.006$
E12	$1.0\,x_0\sin(1/x_1) + 1.0$	$1.0\,x_0\sin(1/x_1) + 1.0$	$1.0\,x_0\sin(1/x_1) + 1.0$
E13	$(1.001\sqrt{x_0} - 0.001)\log(x_1^2)$	$(0.993\sqrt{x_0} - 0.002)\log(x_1^2) + 0.029$	$(1.001\sqrt{x_0} - 0.003)\log(x_1^2)$