Physical System Knowledge Extraction and Transfer Using Machine Learning

by

Haoran Li

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2022 by the
Graduate Supervisory Committee:

Yang Weng, Chair
Hanghang Tong
Gautam Dasarathy
Lalitha Sankar

ARIZONA STATE UNIVERSITY

August 2022

ABSTRACT

Modern physical systems are experiencing tremendous evolutions with growing size, more and more complex structures, and the incorporation of new devices. This calls for better planning, monitoring, and control. However, achieving these goals is challenging since the system knowledge (e.g., system structures and edge parameters) may be unavailable for a normal system, let alone some dynamic changes like maintenance, reconfigurations, and events, etc.

Therefore, extracting system knowledge becomes a central topic. Luckily, advanced metering techniques bring numerous data, leading to the emergence of Machine Learning (ML) methods with efficient learning and fast inference. This work tries to propose a systematic framework of ML-based methods to learn system knowledge under three what-if scenarios: (i) What if the system is normally operated? (ii) What if the system suffers dynamic interventions? (iii) What if the system is new with limited data? For each case, this thesis proposes principled solutions with extensive experiments.

Chapter 2 tackles scenario (i) and the golden rule is to learn an ML model that maintains physical consistency, bringing high extrapolation capacity for changing operational conditions. The key finding is that physical consistency can be linked to convexity, a central concept in optimization. Therefore, convexified ML designs are proposed and the global optimality implies faithfulness to the underlying physics.

Chapter 3 handles scenario (ii) and the goal is to identify the event time, type, and locations. The problem is formalized as multi-class classification with special attention to accuracy and speed. Subsequently, Chapter 3 builds an ensemble learning framework to aggregate different ML models for better prediction. Next, to tackle high-volume data quickly, a tensor as the multi-dimensional array is used to store and

process data, yielding compact and informative vectors for fast inference. Finally, if no labels exist, Chapter 3 uses physical properties to generate labels for learning.

Chapter 4 deals with scenario (iii) and a doable process is to transfer knowledge from similar systems, under the framework of Transfer Learning (TL). Chapter 4 proposes cutting-edge system-level TL by considering the network structure, complex spatial-temporal correlations, and different physical information.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Yang Weng, for providing me the opportunity to work with him on exploring the cutting edge Machine Learning techniques in physical systems. I sincerely thank him for his invaluable guidance and constant encouragement. His critical insights and enthusiasm about research have been most helpful in the completion of this work.

I want to thank my committee members, Dr. Hanghang Tong, Dr. Gautam Dasarathy, and Dr. Lalitha Sankar. Their valuable feedback and guidance in directing me to helpful resources helped me complete this work successfully. Also, I like to thank my co-authors Dr. Yizheng Liao, Dr. Evangelos Farantatos, and Dr. Erik Blasch, for their support and critical validation of the research work. I would also like to thank all the faculty members in Arizona State University and University of Illinois at Urbana-Champaign for teaching me and equipping me with the skill set needed to research.

I would like to thank my family for faltering love and support. I would especially like to thank my parents Mr. Xianbin Li and Mrs. Yanxia Chen for their unwavering supporting and encouragement, despite being far away. Finally, I would like to thank my cat, Dandan, for her continuous love and loyalty to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

GENERAL INTRODUCTION

Modern physical system is extending its territory to better serve the society. For example, Internet of Everything (IoE) connects new devices into an intelligent web at an unprecedented speed (Miraz et al. 2015). If utilized efficiently and systematically, IoE has the potential to create much more efficient productivity via coordinated efforts across humans, processes, data, and things. However, the prerequisite is to own enough system knowledge, e.g., system connectivity and the line parameters. Such a requirement is often not met in real-world applications. The reasons can be summarized as follows. (1) Sensors usually can not cover the complete system due to the sensor cost and low investment interest for some system regions (Bhela, Kekatos, and Veeramachaneni 2018). (2) Some devices, such as plug-and-play components, are private, and the system operators do not have the accessibility (Bordel, De Rivera, and Alcarria 2016). (3) Many dynamic changes like disturbances, events, reconfiguration, or maintenance may not be timely reported to the system operator (Li et al. 2021). Therefore, it's highly demanded to develop a systematic framework to obtain the system knowledge.

To seek this goal, one cheap and efficient approach is to analyze data and recognize the underlying patterns and knowledge. Such a data-driven solution is significantly boosted with the growing techniques in Machine Learning (ML) and Data Mining (DM), equipped with the efficient learning and fast inference (Salih, Ahmed, and Saeed 2021). Therefore, this thesis develops advanced learning-based models to distill knowledge from system data. In particular, the roadmap provides systematic and principled

designs for normal systems, systems with dynamic interventions, and new systems with few measurements because they cover most scenes in real-world applications. In this chapter, a brief summary is given for each scenario. To cope with key problems, the following chapters will present more specific reviews, insightful frameworks with high performances, and the potential future work for further investigations.

1.1 Model A Normal System

With the era of the IoE coming to physical systems, there is an increasing need to extend monitoring and control to system edges, where traditional monitoring and control are unavailable. For example, power engineers nowadays try to provide a similar level of monitoring in its distribution grid when compared to the legendary transmission system (J. Yu, Y. Weng, and R. Rajagopal 2017). However, the parameters and connectivity of edge systems are often missing. Thus, ML tools are recognized as a viable way to conduct cost-efficient inferences to model a normal system as its operational knowledge. Essentially, the goal is to reconstruct the map between system variables, e.g., mapping from voltage phasor to power injections like power flow equations in electrical distribution systems (Li et al. 2021).

In ML perspectives, the system modeling can be interpreted as a regression problem whose target is to predict the outcome given input data. In general, existing work can be categorized into the following groups. First, some pioneer studies utilize ML models as a black box to take place of system physical equations, including linear regression with Least Absolute Shrinkage and Selection Operator (LASSO) (Y. Liao et al. 2018), Support Vector Regression (SVR) (J. Yu, Y. Weng, and R. Rajagopal 2017), and Deep Neural Network (DNN) (Wang et al. 2020), etc. However, for physical

systems with an evolving operating point, this black box can't guarantee the model extrapolation capacity.

To tackle this issues, the second group of researchers considers a grey box by enforcing some physical constraints to the ML model. For instance, (Hu et al. 2020) encodes Kirchhoff's laws to the DNN model to represent power system operations. (Pan et al. 2020) also embed the equality and inequality constraints to a DNN model. Despite the improvement, it's hard to theoretically guarantee that the learned ML model can maintain consistency and faithfulness to the system. Thus, the third approach is to directly learn the symbolic equations from data, which brings a white box and lies in the domain of Symbolic Regression (SR) (Udrescu and Tegmark 2020). SR frameworks typically include a search process to find the symbols and symbol operators and an estimation process to estimate the symbol coefficients, bringing explicit mathematical expressions to depict data (Petersen et al. 2019). Notwithstanding, the cutting-edge SR methods still can not guarantee that the search results are faithful to the true physical equations.

In conclusion, the key unsolved question is how we can guarantee that the learned ML model is consistent to the underlying physical equations. Such a goal is even tougher if the system lacks full observability. This is because hidden quantities with randomness in the system can create bias terms for a specific data set. Therefore, this thesis directly tackles this question for consistent physical knowledge extractions.

1.2 Identify System Dynamic Events

Physical system complexity is increasing with new technologies. For example, for power systems, intermittent renewable generation and new types of loads such as

electrical vehicles (EVs) (Lopes, Soares, and Almeida 2010) keep being integrated to the system. With the booming business, the transportation systems are also extending and may easily face the traffic congestion (Iqbal et al. 2018). To achieve grid reliability, new methods and tools for enhanced situational awareness are needed, with advanced functions including fast and accurate event identification. Luckily, the growing installation of system sensors like Phasor Measurement Units (PMUs) can enable the development of applications to find the knowledge of system event time, type, and locations.

To identify events, one idea is to use expert information. For example, one can use signal transformation or filtering to map the time series data into some physically meaningful domain for comparing with some predefined thresholds. These methods use wavelet transformation (Kim et al. 2015), Kalman filtering (Pérez and Barros 2008), and Swing Door Trending (SDT) (Cui et al. 2018), etc. For example, (Cui et al. 2018) utilizes a swing door to compress data with a pre-defined door width, and the detectable events must have a certain level of slope rate. However, as these methods need to pre-define some measures or thresholds, the usage may be biased because of the specific design and test cases. Therefore, can we have a general model?

For obtaining a general form, ML-based methods are extensively introduced. For example, previous work proposes to use existing events and their labels (time, type, and location information) to train a multi-class classification model. For instance, Decision Tree (DT) (Li et al. 2019a) treats each measurement as a factor to determine the final decision. Although transparent, such a method is inefficient to make use of complex measurement correlations. Therefore, (De Yong, Bhowmik, and Magnago 2015) proposes Support Vector Machine (SVM) to assign each input measurement a weight to form the final feature. There are also more complex and powerful models such

as Convolutional Neural Network (CNN) (Yuan et al. 2021), Graph Neural Network (GNN) (Yuan, Wang, and Wang 2020), and Long Short-Term Memory (LSTM) units (Zhang et al. 2018).

Despite great advances, the following problems remain challenging to some extent. (1) The learning models are biased with specific assumptions. (2) There are often insufficient labels for learning. (3) Advanced sensing technologies may bring a large volume of samples with high resolutions. In face of these challenges, how can we develop efficient models for fast learning and inference? This thesis tries to provide an answer by intellectually combing the physical properties with the powerful feature extraction ability in ML models.

1.3 Transfer Knowledge across Systems

The above knowledge-extraction ML models require a certain amount of training data. If the data are limited, the model training is likely to fail due to the curse of dimensionality. Unfortunately, data-limited scenarios often occur for a completely new grid or an old grid with increased metering, not to mention the common expansion process with new nodes/lines. Thus, it is urgent to employ new methods to transfer knowledge from the source grid with rich data to the target grid with limited data (Li, Weng, and Tong 2020; Li, Ma, and Weng 2022).

For this goal, Transfer Learning (TL) is defined conceptually as an efficient procedure to extract common knowledge from two different domains and boost the performance of the data-limited domain. Specifically, an efficient approach is Domain Adaptation (DA) (Pan, Kwok, Yang, et al. 2008; Long et al. 2013; Long et al. 2016) which minimizes the data distribution discrepancy of two domains, usually in a low-

dimensional space for domain invariant features. Therefore, DA promises that joint training using common knowledge can significantly boost the learning process. While many efficient DA models have been applied to computer vision (Long et al. 2013; Ganin et al. 2016a) and natural language process (Wu and Huang 2016), relatively few work has been done for graph data (Shen et al. 2021).

For physical systems, the widely placed sensors bring numerous networked measurements for nodes and edges with complex spatial-temporal correlations. Can we fully explore the correlations to improve DA methods? Further, nodes or edges can have different physical characteristics. Is this information beneficial to the distribution adaptation? This thesis answers above questions and provide accurate and scalable DA models between different systems.

1.4 Research Objectives

1. Develop novel ML models to learn physical equations for a normal system. Explore in-depth mathematical guarantees of learning physics-consistent model. Then, develop efficient and guaranteed learning models. Moreover, investigate how to maintain the consistency for partially-observable systems.
2. Construct fast and cost-efficient ML models to identify system event information. For real-world applications, the scenarios of limited labels and high volume measurements are considered. To tackle data challenges, leverage physical properties to boost the ML models.
3. Study the knowledge transfer between physical systems and propose cutting-edge solutions. In particular, utilize the graph data structure, complex spatial-

temporal correlations, and physical characters of different components to significantly enhance the knowledge migration.

1.5 Organization

The organization of this report is as follows:

1. Chapter 1 presents the general introduction, motivation, objectives, and organizations of the research.
2. Chapter 2 presents a literature review of equation learning. Then, this chapter investigates how to maintain the physics consistency for the learning model with mathematical guarantees. The key step is to link the physical consistency to the convexity of the optimization model to learn ML parameters. Such convex ML designs are significantly explored for both the fully-observable and partially-observable systems. Finally, this chapter also develops comprehensive experiments on diversified systems to illustrate the outstanding performance of proposed models.
3. Chapter 3 provides a literature summary of the physical system event identification. Then, two frameworks are designed, covering supervised and unsupervised learning. The first framework proposes an ensemble model to aggregate different supervised learning models with different biases. When no label information is available, the second framework leverages physical properties to provide labels. In addition, to specifically process high volume data due to recent sensing technologies, a tensor-learning model is proposed to convert large-scale tensors to compact and informative feature vectors for fast event identification. Finally, the

chapter presents extensive numerical validations to report the high performance of developed methods.

4. Chapter 4 conducts a survey of the transfer learning for networked data. Subsequently, the chapter studies how the network structure, complex spatial-temporal correlations, and physical characteristic information can be embedded to yield highly-efficient knowledge transfer processes.
5. Chapter 5 presents the conclusions and contributions of the research. Then, Chapter 5 discusses the future directions of the research work.

Chapter 2

LEARNING PHYSICAL EQUATIONS OF A NORMAL SYSTEM

2.1 Introduction

Modern physical systems like Internet of Everything (IoE) serves as a game-changer to network different devices and products and boost the economy. While IoE brings new opportunities for improving new inference and productivity, it remains an open question about how to efficiently monitor various grids with often limited sensors and system information. Such a problem is essential to maintain the grid stability. For example, the rooftop PhotoVoltaic (PV) panels can gather solar power but often generate the so-called inverse power flow (Weng, Liao, and Rajagopal 2017). If not timely monitored, the inverse flow may hurt the instant system power balancing (Lew et al. 2017). Thus, utilities and grid operators need new tools to monitor, control, and operate the systems under these profound changes.

As the foundation of the intelligent monitoring and control, the system knowledge, including the system connectivity and the parameters for system components, are not always available due to limited instrumentation and low investment interest (Bhela, Kekatos, and Veeramachaneni 2018). Further, the topology of the network may change frequently and cause the parameter matrix to have different structures and values. For example, the frequency of a topology change with PV panels (Jabr 2014) ranges from eight hours to once a month for medium-voltage grids (Fajardo and Vargas 2008). In addition, the unpredictable events (e.g., power outages) and regular maintenance could result in a topology reconfiguration, which may not be well synchronized with

all stakeholders. Finally, many devices, such as plug-and-play components, are not operated by the utilities and therefore, their connectives and status are not always observed by the distribution grid operators.

To obtain the system knowledge, Machine Learning (ML) techniques are widely utilized in the ear of big data. In general, this is a regression problem to find the best model to represent the physical relationship (i.e., physical equations) between system variables. Existing work can be categorized into the following groups. The first group (1) utilizes black-box ML models to capture the physical equations. Specifically, the employed ML models cover a wide range, including linear regression with Least Absolute Shrinkage and Selection Operator (LASSO) (Y. Liao et al. 2018), Support Vector Regression (SVR) (J. Yu, Y. Weng, and R. Rajagopal 2017), and Deep Neural Network (DNN) (Wang et al. 2020), etc. Although the representational power increases with more and more complex ML models, the low generalizability to unseen conditions significantly prevents the application of black-box ML models in physical systems. For instance, if the system operating point changes, the ML model can easily generate inaccurate output values.

To tackle this issues, the second group (2) considers a grey box by enforcing some physical constraints to the ML model. For instance, (Hu et al. 2020) encodes Kirchhoff's laws to the DNN model to represent power system operations. (Pan et al. 2020) also embed the equality and inequality constraints to a DNN model. These constraints can guarantee that some parameters and variables in the ML model should contain certain physical behaviors. Despite the improvement, therse models require physical prior to formalize constraints.

In addition, the above learning process focuses on learning the power flow equation in power systems. However, there are many other equations in power systems or

symbolic expressions in other physical systems. To learn these equations, we may not have as clear knowledge as power flow equations. Thus, our general goal is to understand the system connections, system symbols, and the parameters over these connections, which are hard and non-convex.

Thus, the third approach (3) makes the ML model a white box and directly learns the symbolic equations from data, which lies in the domain of Symbolic Regression (SR) (Udrescu and Tegmark 2020). Promising as it might be, SR is NP-hard (Petersen et al. 2019; Lu, Ren, and Wang 2016). Mathematically, one can cast SR as an optimization problem over both discrete variables to select symbols and continuous variables to represent the symbol coefficients. Traditional solutions employ evolutionary algorithms like Genetic Programming (GP) (Orzechowski, La Cava, and Moore 2018). These methods start from an initial set of expressions and continue evolving via operations like crossover or mutation. With fitness measures, GP-based algorithms can evaluate and select the best equations. However, these methods have poor scalability and limited theoretical guarantees (Petersen et al. 2019).

More recent SR studies leverage Neural Networks (NNs) with high representational power. For the NN-based SR, I mainly categorize them into two approaches based on the roles of the NNs. The first approach employs NNs to directly model the equations, where sparsity of the NN weights is enforced to select symbols (Sahoo, Lampert, and Martius 2018; Martius and Lampert 2016; Werner et al. 2021; Li and Weng 2021; Chen, Liu, and Sun 2021). Thus, the problem is transformed into training the designed NN with sparsity regularization. However, due to the non-convexity of NNs, the weight selection and updating can be easily stuck in local optima, failing to find the exact equations.

The second approach employs NNs to search the symbol connections, and non-

linear optimizations like BFGS (Fletcher 2013) can be employed to estimate symbol coefficients. For the searching procedure, (Petersen et al. 2019; Mundhenk et al. 2021) leverage a Recursive Neural Network (RNN) as a policy network to iteratively generate optimal actions that can select and connect symbols. (Biggio et al. 2021) employs large-scale pre-training to directly map from data to the symbolic equations. While these methods restrict the utilization of NNs in the search phase, the non-convexity of NNs can still suffer the risk of sub-optimal decisions to formulate equations. Finally, all above SR methods can hardly tackle the case when only partial variables are measurable in a system.

To summarize, it remains to be a problem to develop efficient equation learning models with provable performances to find the true physics. Next, when the system is partially measurable, the learning model should maintain certain physics consistency to the underlying equations. To fulfill these two requirements, I propose novel designs with cost-efficient learning processes and strict theoretical supports.

2.2 Problem Formulation

Many physical networks are graphs, which can be denoted as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathcal{V}$ to be the vertex set and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ to be the edge set. For some scenarios, limited sensors in the system pose challenges for the system knowledge extraction. Therefore, I define $\mathcal{V} = \{\mathcal{O} \cup \mathcal{H}\}$, where $\mathcal{O}$ represents the set of observable nodes (nodes with meters) and $\mathcal{H}$ represents the set of hidden nodes (nodes without meters). To represent the physical quantities in the system, I denote $\boldsymbol{x} \in \mathbb{R}^{|\mathcal{V}|}$ and $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{V}|}$ as the random variables of the input and output of physical equations, respectively, where $|\cdot|$ for a set is the cardinality of the set. Then, the physical equation can be formally defined

as $\boldsymbol{y} = g(\boldsymbol{x})$. Note that the dimension of $\boldsymbol{x}$ and $\boldsymbol{y}$ implies that there is one input and output variable for each node for the sake of convenience. However, this can be easily generalized into multiple variables.

Subsequently, I consider two cases in the thesis. First, if the system is fully-observable and $\mathcal{O} = \mathcal{V}$, I denote $\{\boldsymbol{x}^n, \boldsymbol{y}^n\}_{n=1}^N$ as the N samples obtained from meters. Then, the goal is to completely figure out the underlying physical equations $\boldsymbol{y} = g(\boldsymbol{x})$. Thus, the problem is:

- Problem 1: learn physical equations of a fully-observable system.
- Input: N measurements $\{\boldsymbol{x}^n, \boldsymbol{y}^n\}_{n=1}^N$ for system inputs and outputs.
- Output: a symbolic function f such that $\boldsymbol{y} = f(\boldsymbol{x})$. Further, after proper simplification, $f(\cdot)$ should approximately equal to $g(\cdot)$. Namely, $f(\cdot)$ should have the same physical symbols, connections, and parameters of the connections as $g(\cdot)$.

For the second case, the system is partially observable and $\mathcal{O} \subset \mathcal{V}$. Then, I denote $\{\boldsymbol{x}_{\mathcal{O}}^n, \boldsymbol{y}_{\mathcal{O}}^n\}_{n=1}^N$ as the N samples obtained from meters in observable nodes. With data, the target is to learn a model that maintains certain physics consistency to $g(\cdot)$. Thus, the problem is:

- Problem 2: learn a physics-consistent model of a partially-observable system.
- Input: N measurements $\{\boldsymbol{x}_{\mathcal{O}}^n, \boldsymbol{y}_{\mathcal{O}}^n\}_{n=1}^N$ for system inputs and outputs of observable nodes.
- Output: a function h such that $\boldsymbol{y}_{\mathcal{O}} = h(\boldsymbol{x}_{\mathcal{O}})$. Further, $h(\cdot)$ should maintain certain consistency to $g(\cdot)$ with (1) high extrapolation capacity when system operational conditions change and (2) accurate parameter estimation for a subset of parameters in $g(\cdot)$.

As illustrated in Section 2.1, the above problems are combinatorial optimizations as one needs to search correct symbols, estimate symbol coefficients (i.e., system parameters), and approximate unobservable quantities (in Problem 2). The problem is in general non-convex and can hardly produce global optimal solution that corresponds the true physics. Thus, this chapter tries to solve both problems with convexified relaxations and strict mathematical supports. Moreover, extensive experiments are implemented to show the superiority of proposed solutions.

2.3 Learn Physical Equations for A Fully-observable System via CoNSoLe: Convex Neural Symbolic Learning

2.3.1 Problem Scopes and Examples

In this section, I tackle Problem 1 in Section 2.2 to learn the complete physical equations. I assume the underlying equation $\boldsymbol{y} = g(\boldsymbol{x})$ follows compositionality and smoothness assumptions in (Udrescu and Tegmark 2020), which are often the case in physics and many other scientific applications. Namely, the system equations can be explicitly written by simple symbols and basic mathematical operators (i.e., plus, minus, multiply, and divide). For example, for a 2-node system, I can write down the equations as follows:

$$\begin{aligned} y_1 &= 3x_1^2 \cos(2.5x_2), \\ y_2 &= 2\big(\cos(3x_1) + 1.5x_2^2\big)^2. \end{aligned} \tag{2.1}$$

Given data from input $\boldsymbol{x} = [x_1, x_2]^\top$ and output $\boldsymbol{y} = [y_1, y_2]^\top$ variables, where $\top$ represents the transpose operator, the target is to find correct symbols like $\cos(x)$ and x^2 and estimate corresponding parameters like 3, 2.5, 2, and 1.5 in Equation (2.1).

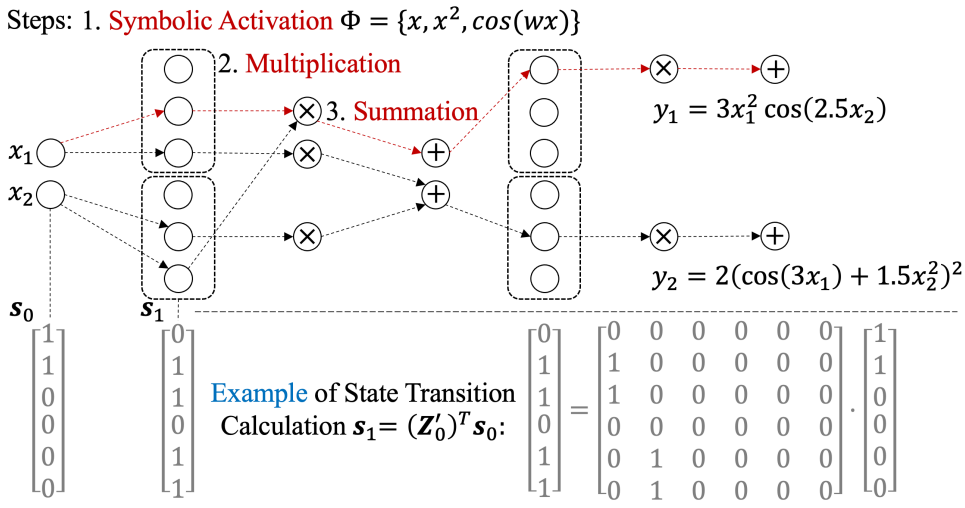


Figure 1. An example of LoCaL structure and the state transition calculation.

2.3.2 Construct A Neural Network to Represent Physical Equations

To tackle the above problem, the first step is to represent the multi-input multi-output symbolic equations. Specifically, I utilize a neural network due to the efficiency (Sahoo, Lampert, and Martius 2018). For example, Fig. 1 demonstrates how an NN can represent Equation (2.1). Although Section 2.1 presents that common NN models will lead to local-optimal solutions, I will prove that the the proposed NN in Fig. 1 has local convexity. Thus, I name the NN as Locally Convex Equation Learner (LoCaL).

Specifically, each input entry first goes through the activation functions from a symbol library Φ. For example, in Fig. 1, I denote $\Phi = \{x, x^2, \cos(wx)\}$ to correspond with three neurons from top to bottom in the dotted black box, where w is a learnable weight. As for weights of x and x^2, they only need to appear in the later summation layer. Then, some activation outputs will be selected as multipliers for the multiplication. Subsequently, the multiplied outputs are selected and summed together.

The repetition of symbolic activation, multiplication, and summation formulates final equations.

Mathematically, I denote $\boldsymbol{Z}_k \in \{0,1\}^{n_k \times n_{k+1}}$ and $\boldsymbol{W}_k \in \mathbb{R}^{n_k \times n_{k+1}}$ as the indicator and weight matrix between the k^{th} and the $(k+1)^{th}$ layer of the LoCaL, respectively. n_k is the number of neurons for the k^{th} layers. $\boldsymbol{Z}_k[i,j] = 1$ indicates that the connection exists between the i^{th} neuron in the k^{th} layer and the j^{th} neuron in the $(k+1)^{th}$ layer, where $\boldsymbol{Z}_k[i,j]$ is the $(i,j)^{th}$ entry of $\boldsymbol{Z}_k$. I assume there are $(K+1)$ layers in LoCaL and denote $\boldsymbol{h}_k \in \mathbb{R}^{n_k}$ to be the output of the k^{th} layer. Naturally, I have $\boldsymbol{h}_0 = \boldsymbol{x}$ and $\boldsymbol{h}_K = \boldsymbol{y}$. For the symbolic activation or summation layer, I have $\boldsymbol{h}_{k+1} = \boldsymbol{Z}_k^\top \circ \boldsymbol{W}_k^\top \boldsymbol{h}_k$, where $\circ$ represents the Hadamard product and helps to zero out some connections. For the multiplication layer, I have $\boldsymbol{h}_{k+1}[j] = \prod_{\boldsymbol{Z}_k[i,j]=1} \boldsymbol{h}_k[i]$. In general, I denote LoCaL as $f(\boldsymbol{x}; \{\boldsymbol{Z}_k\}_{k=0}^{K-1}, \{\boldsymbol{W}_k\}_{k=0}^{K-1})$. The search algorithm identifies the locations of the entry 1 in $\{\boldsymbol{Z}_k\}_{k=0}^{K-1}$ and the estimation process learns the corresponding weights in $\{\boldsymbol{W}_k\}_{k=0}^{K-1}$ using $\{\boldsymbol{x}^n, \boldsymbol{y}^n\}_{n=1}^N$.

2.3.3 Search LoCaL Structures with Double Convex Deep Q-Learning

The high dimensionality of the search space is a general problem for LoCaL and other SR models. For an efficient search, NN-based RL methods are often utilized, e.g., deep Q-learning (Fan et al. 2020). In this subsection, I show how to design a convexified deep Q learning to identify the true physics. The key for convexification is to employ the cutting-edge Input Convex Neural Network (ICNN) (Amos, Xu, and Kolter 2017) to approximate a convex function.

Markov Decision Process (MDP) to search the structure of LoCaL. The search procedure is an MDP where each layer of LoCaL can be treated as a state

and the connections between layers are actions. At each state, the deep Q-learning agent aims to find the optimal policy to decide the next action (Baker et al. 2016). Therefore, I denote $\boldsymbol{a}_k \in \{0,1\}^{n_a}$ as the k^{th} action vector where $\boldsymbol{a}_k[(i-1)n_k + j] = 1$ implies that the connection ij exists for the i^{th} neuron in the k^{th} layer and the j^{th} neuron in the $(k+1)^{th}$ layer. $n_a = \max\{n_k n_{k+1}\}_{k=0}^{K-1}$ is the dimensionality of the action space. Also, I denote $\boldsymbol{s}_k \in \mathbb{Z}^{n_s}$ as the state vector to represent the current state for the k^{th} layers, where $n_s = \max\{n_k\}_{k=0}^{K}$ is the dimensionality of the state space. For a known state and a fixed action, the next state is exactly known due to the exact neuron connections, which implies a deterministic state transition. Thus, I utilize a linear transformation to represent the state transition process. $\forall 0 \leq k \leq K-1$, I define

$$\boldsymbol{s}_0 = [\mathbf{1}, \mathbf{0}]^\top, \boldsymbol{s}_{k+1} = (\boldsymbol{Z}_k^{'})^\top \boldsymbol{s}_k = [\mathrm{Mat}(\boldsymbol{a}_k); \mathbf{0}]^\top \boldsymbol{s}_k, \tag{2.2}$$

where $\mathrm{Mat}(\cdot)$ is the operation to convert a vector to a matrix and I fill 0s to some $\boldsymbol{s}_k$, $\boldsymbol{a}_k$, and $\boldsymbol{Z}_k^{'}$ to maintain the fixed size. $[\cdot;\cdot]$ is the row concatenation process. I show this linear state transition is essential to guarantee the convexity of negative optimal Q-function in Theorem 1. For the calculation example of state transition, one can refer to Fig. 1. Finally, I can obtain $\boldsymbol{Z}_k$ from $\mathrm{Mat}(\boldsymbol{a}_k)$ by deleting the filled 0s. By the designed states and actions, the search will always start at $\boldsymbol{s}_0$ and end at $\boldsymbol{s}_K$ in one episode, thus formulating the LoCaL function $f(\boldsymbol{x}; \{\boldsymbol{Z}_k\}_{k=0}^{K-1}, \{\boldsymbol{W}_k\}_{k=0}^{K-1})$.

Double convexity for the negative reward and Q-function in the search model. Training LoCaL can bring a terminal reward (Petersen et al. 2019) to evaluate the episode. However, this reward is insufficient to enable a convex evaluation for instant state-action pairs. Thus, I propose to restrict the negative reward function to be convex. Based on Bellman equations (Bertsekas 2015), this design will ensure

the convexity of the negative optimal Q-function. More proof details can be referred to Theorem 1.

Specifically, I utilize an ICNN to model the reward function $-R(\boldsymbol{s}_k, \boldsymbol{a}_k)$ such that $-R(\boldsymbol{s}_k, \boldsymbol{a}_k)$ is convex in states and actions in continuous spaces. $-R(\boldsymbol{s}_k, \boldsymbol{a}_k)$ requires proper training to do the correct evaluation. In the t^{th} episode, I collect the training input of state-action pairs $\{\boldsymbol{s}_k^t, \boldsymbol{a}_k^t\}_{k=0}^{K-1}$. Then, the output can be defined as the terminal reward to evaluate the obtained LoCaL which is denoted as $f_t(\cdot)$. Basically, I calculate the Normalized Root-Mean-Square Error (NRMSE) (Petersen et al. 2019) of the trained $f_t(\cdot)$ function such that $\text{NRMSE}_t = \frac{1}{\sigma_y}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - f_t(\boldsymbol{x}^i))^2}$, where σ_y is the standard deviation of the outputs. Then, the output of the reward can be calculated as $R_t = \frac{1}{1+\text{NRMSE}_t}$. Therefore, I can train $-R(\cdot)$ using $\{\{\boldsymbol{s}_k^t, \boldsymbol{a}_k^t\}_{k=0}^{K-1}, -R_t\}$.

It's important to note that building this convex function $-R(\cdot)$ is better than directly leveraging the terminal reward for evaluations. The reason is that the former process not only provides theoretical guarantees in Theorem 1 but also evaluates the actions in a continuous space with convexity. Namely, one can utilize $R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ to evaluate a continuous action $\tilde{\boldsymbol{a}}_k$, given that there is a much higher probability to obtain $\tilde{\boldsymbol{a}}_k$ rather than the discrete action $\boldsymbol{a}_k$ in the deep Q-learning. Then, to update Q values, I have the following iterative computations based on the temporal difference (Baker et al. 2016):

$$Q_{t+1}(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) = Q_t(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) + \alpha\big(R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) + \gamma \max_{\boldsymbol{a}} Q_t(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}}) - Q_t(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)\big), \quad (2.3)$$

where $Q_t(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is the Q value at the t^{th} episode for state $\boldsymbol{s}_k$ and action $\tilde{\boldsymbol{a}}_k$. α and γ are pre-defined learning rate and discount factor, respectively. Similarly, I utilize another ICNN to represent $-Q(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ such that $-Q(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\tilde{\boldsymbol{a}}_k$ given fixed $\boldsymbol{s}_k$. Therefore, one can solve a convex optimization problem $\max_{\boldsymbol{a}} Q_t(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ to obtain the (approximately) global optimal action for Equation (2.3) in each iteration.

By definitions of the discrete actions, I can restrict the continuous action space in a hypercube $\text{conv}(\{0,1\}^{n_a})$, i.e., a convex hull of the discrete actions. Thus, the optimization problem is:

$$\tilde{\boldsymbol{a}}^* = \arg\min_{\tilde{\boldsymbol{a}}} -Q(\boldsymbol{s}, \tilde{\boldsymbol{a}}), \tilde{\boldsymbol{a}} \in \text{conv}(\{0,1\}^{n_a}). \tag{2.4}$$

Based on (Amos, Xu, and Kolter 2017), this convex optimization problem can be solved via a bundle entropy method. After obtaining a continuous solution $\tilde{\boldsymbol{a}}^*$, I discretize it to a discrete vector $\boldsymbol{a}^*$ to build LoCaL. One simple method is to enforce $\boldsymbol{a}^*[i] = 1$ if $\tilde{\boldsymbol{a}}^*[i] \geq 0.5$, and otherwise $\boldsymbol{a}^*[i] = 0$. Thus, both $Q(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ and $Q(\boldsymbol{s}_k, \boldsymbol{a}_k)$ can be trained using Equation (2.3). Practically, I introduce ϵ-greedy strategy (Hester et al. 2018), experience replay (Mnih et al. 2015) and a target Q-network (Fan et al. 2020) to update Q-function, thus boosting the convergence to the optimal Q-function. The specific algorithm can be seen in Algorithm 1. Finally, based on Theorems 1-2, if the optimal Q-function is found, solving the convex optimization via the optimal Q-function can generate the correct structures of LoCaL and exact equations.

Symbolic static and dynamic constraints. Constraints can be added to accelerate the search process (Petersen et al. 2019; Udrescu and Tegmark 2020). For example, in Algorithm 1, I propose a constraint checking program for the state-action pairs to avoid the invalid search, suitable for arbitrary restrictions. Then, I emphasize a general type of constraint, symbolic constraint, for the SR problem. The symbolic static constraint requires that each equation contains only a subset of symbols. For example, the equation $y_1 = x_1 x_2 \cdots x_{100}$ shouldn't exist since it is too complicated for real-world systems. This constraint eliminates part of the action space and can be checked by counting the number of 1s in the action vector. The symbolic dynamic constraint can gradually reduce the search space based on symbol

Algorithm 1 CoNSoLe: Convex Neural Symbolic Learning

Input: training dataset $\{\boldsymbol{x}^i, \boldsymbol{y}_i\}_{i=1}^N$.

Hyper-parameters: LoCaL layer number K, initial state $\boldsymbol{s}_0 = [\mathbf{1}, \mathbf{0}]^\top$, discount factor $\gamma \in (0, 1)$, ϵ for ϵ-greedy strategy, λ as a threshold to stop searching, ICNN for reward function $-R(\boldsymbol{s}, \boldsymbol{a})$, ICNN for Q-function $-Q(\boldsymbol{s}, \boldsymbol{a})$, replay buffer $B = \emptyset$, maximum episode T, target network $Q^{'}(\cdot) = Q(\cdot)$, and target network update interval T_0.

while $t \leq T$ **do**

  while $k \leq K$ **do**

    Solve Optimization in Equation (2.4) with $-Q(\boldsymbol{s}_k^t, \boldsymbol{a})$ to obtain $\tilde{\boldsymbol{a}}_k^*$.

    Use ϵ-greedy to select $\tilde{\boldsymbol{a}}_k^t$ from $\tilde{\boldsymbol{a}}_k^*$ and a random action. ▷ ϵ-greedy strategy.

    Discretize $\tilde{\boldsymbol{a}}_k^t$ to obtain $\boldsymbol{a}_k^t$.

    Execute $\boldsymbol{a}_k^t$ and use Equation (2.2) to obtain $\boldsymbol{s}_{k+1}^k$.

    Check if $\boldsymbol{a}_k^t$ and $\boldsymbol{s}_{k+1}^k$ satisfy certain constraints. Otherwise, delete this state transition and restart the iteration from $\boldsymbol{s}_k^t$. ▷ Constraint checking.

  Formulate LoCaL, train LoCaL with $\{\boldsymbol{x}^i, \boldsymbol{y}_i\}_{i=1}^N$, and calculate R_t.

  Train the reward function $-R(\cdot)$ using training data $\{\{\boldsymbol{s}_k^t, \boldsymbol{a}_k^t\}_{k=0}^{K-1}, -R_t\}$.

  $\forall 0 \leq k \leq K$, insert $(\boldsymbol{s}_k^t, \boldsymbol{a}_k^t, \boldsymbol{s}_{k+1}^t, R_t)$ and $(\boldsymbol{s}_k^t, \tilde{\boldsymbol{a}}_k^t, \boldsymbol{s}_{k+1}^t, R(\boldsymbol{s}_k^t, \tilde{\boldsymbol{a}}_k^t))$ to B_0.

  Sample a minibatch $B_0 \subset B$

  for $(\boldsymbol{s}_m, \boldsymbol{a}_m, \boldsymbol{s}_{m+1}, R_m) \in B$ **do** ▷ Experience replay.

    Solve Optimization in Equation (2.4) with $-Q^{'}(\boldsymbol{s}_{m+1}, \boldsymbol{a})$ to obtain $\tilde{\boldsymbol{a}}_{m+1}$.

    $y_m = R_m + \gamma Q^{'}(\boldsymbol{s}_m, \boldsymbol{a}_m)$

  Train $Q(\cdot)$ using training data $\{\boldsymbol{s}_{m+1}, \boldsymbol{a}_{m+1}, y_m\}_m$

  if $t \mod T_0 = 0$ **then**

    $Q^{'}(\cdot) = Q(\cdot)$ ▷ Update target Q-network.

  if $|R_t - 1| \leq \lambda$ **then**

    End the search process.

Output: LoCaL with the best performance and the corresponding equations.

correlations. Specifically, I investigate the $(K-1)^{th}$ layer's neurons that are linearly summed to form the equation. If some of these neurons have strong linear correlations to the output neuron (e.g., Pearson correlation coefficient larger than 0.99), they should be kept subsequently. Namely, I can maintain the path from input neurons to the neurons to be kept and reduce the search space.

2.3.4 Theoretical Analysis for CoNSoLe to Learn Physical Equations

I employ explorations, experience replay, and a target Q-network to boost the convergence to the optimal Q function. However, the extra requirement of the convex shape for the negative Q-function may deteriorate the convergence performance. Thus, I first prove that in CoNSoLe, the negative optimal Q-function is also convex so that the convex design doesn't affect the convergence. Then, I prove that the convexity of the negative optimal Q-function eventually yields the exact equations.

Theorem 1. *$\forall 0 \leq k \leq K-1$, the negative optimal Q-function $-Q^*(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ in the proposed* CoNSoLe *framework is convex in $\boldsymbol{s}_k$ and $\tilde{\boldsymbol{a}}_k$, where $\boldsymbol{s}_k$ is the discrete state and $\tilde{\boldsymbol{a}}_k$ is the continuous action at the k^{th} stage.*

The proof can be seen in Appendix A.1. Based on the convexity of negative optimal Q-function, the optimal sequence of states $(\boldsymbol{s}_0, \boldsymbol{s}_1^*, \cdots, \boldsymbol{s}_K^*)$ and actions $(\boldsymbol{a}_0^*, \boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_{K-1}^*)$ can be found via solving the convex optimization problem. Then, I have the following theorem.

Theorem 2. *Let $f^*(\cdot; W)$ denote the* LoCaL *constructed by the optimal sequences of states $(\boldsymbol{s}_0, \boldsymbol{s}_1^*, \cdots, \boldsymbol{s}_K^*)$ and actions $(\boldsymbol{a}_0^*, \boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_{K-1}^*)$ from $-Q^*(\cdot)$, where W is the set of weights of $f^*(\cdot; W)$. If $f^*(\cdot; W)$ can be trained with noiseless datasets and the*

training can achieve the global optimal weights W^*, $f^*(\cdot; W^*)$ represents the exact equations.

The proof can be seen in Appendix A.2. Theorem 2 requires that LoCaL can learn the global optimal weights. The requirement is generally hard to achieve due to the non-linearity and non-convexity of LoCaL. However, I show that with mild assumptions, there are local regions in the LoCaL loss surface with strict convexity. Then, if I have proper initializations, the gradient-based weight updating can find the global optimum. Specifically, I have the following theorem.

Theorem 3. *Assume the following conditions hold:* (1) *the equation $g(\boldsymbol{x})$ is C^2 smooth and has bounded second derivatives with respect to weights,* (2) $\exists \boldsymbol{x}$ *in the input measurement space, $g(\boldsymbol{x})$ has non-zero gradients with respect to weights,* (3) *the structure of* LoCaL *is correctly searched to exactly represent symbols and symbol connections in $g(\boldsymbol{x})$, and* (4) *the training dataset of* LoCaL *is noiseless. Then, for the MSE loss surface of* LoCaL*, each global optimal point has a strictly convex local region.*

The proofs can be seen in Appendix A.3. Note that Assumptions 1-2 easily hold for common physical equations in nature (Udrescu and Tegmark 2020). Assumption 3 relies on the search algorithm, and I show, both theoretically and numerically, that the double convex deep Q-learning has good performances. Assumption 4 relies on the quality of data and I focus on the noiseless data in this paper. What's more, Equation (A.3) in the proof suggests that if the absolute noise values are small, the locally convex region still exists. I also numerically prove CoNSoLe is robust under certain noise levels in Section 2.3.5.0.5. To summarize, these assumptions are acceptable. To

quantify the range of the local region for a LoCaL with a certain complexity, I have the following theorem.

Theorem 4. *Suppose Assumptions 1-4 in Theorem 3 hold. For a* LoCaL *with one symbolic activation, multiplication, and summation layer, the set of local convex regions with global optima is* $U = \{W \Big| \frac{\left|\frac{d}{dt}\Big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX)\right|^2}{\eta\left|\frac{d}{dt}\Big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\right|} > |\hat{y}(\boldsymbol{x}^k, W) - y_k|\}$*, where notations are defined in the proof.*

The proofs can be seen in Appendix A.4. I explain the region range is large for a stable system that satisfies all assumptions in Theorem 4.

Physical interpretations of the convex region size. Physical systems in scientific and engineering domains have certain stability that can withstand parameter changes to some extent. Further, this ability should hold for arbitrary $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, I can assume $\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX) \approx \frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX) \approx \frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^k, W+tX)$ for $\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^k$. Then, the inequality in set U can be approximately rewritten as $\frac{1}{\eta} > ||\boldsymbol{w} - \boldsymbol{w}^*||_2$, where $\boldsymbol{w}$ and $\boldsymbol{w}^*$ are the vectorized W and W^*, respectively. Namely, the distance between any point W in the region to the global optimal point W^* in the region is bounded by $\frac{1}{\eta}$. Based on Equation (A.16), η is the bound of the ratio of second derivative to the first derivative. For a stable system, this ratio should be small. Otherwise, the system can easily crash with a small parameter disturbance. Thus, $\frac{1}{\eta}$ is relatively large and so is the region of U. An example of the range is displayed in Section 2.3.5.0.2. Finally, the above analysis also holds when the LoCaL of the system equation has more than one symbolic activation, multiplication, and summation layers. This is because Equations (A.17) always hold as long as I can find a η to bound the ratio of second derivative to the first derivative, which is irrelevant to the structure of LoCaL.

2.3.5 Experimental Result

2.3.5.0.1 Settings

Datasets. I use the following datasets for testing. (1) **Synthetic datasets**. I create two datasets, Syn_1 and Syn_2, for testing. Syn_1 has the following equations: $y_1 = 3x_1^2\cos(2.5x_2)$, $y_2 = 4x_1x_3$, and $y_3 = 3x_3^2$. Syn_2 is more complex with the following equations $y_1 = \sqrt{2.2x_1}x_2 + x_1x_2^2$, $y_2 = \sin(1.8x_1)\big(\log(3x_2) + \sqrt{x_3}\big)$, $y_3 = \sqrt{3.7x_3}\log(1.6x_1) + x_1^2$. For the training data, each input variable is randomly sampled from a uniform distribution of $U(1, 2)$ to avoid invalid values like $\log(0)$. Totally, I create $2,000$ samples for training. Then, in the test phase, I utilize another $2,000$ samples whose input variables are sampled from $U(3, 4)$. The symbolic activation pools are $\{x, x^2, \cos(x)\}$ and $\{\sqrt{x}, x, x^2, \log(x), \sin(x)\}$ for Syn_1 and Syn_2, respectively.

(2) **Power system dataset.** Power flow equation determines the operations of electric systems (Jiafan Yu, Yang Weng, and Ram Rajagopal 2017). For node i in an M-node system, the equation can be written as $p_i = \sum_{m=1}^{|M|} G_{im}(u_iu_m + v_iv_m) + B_{im}(v_iu_m - u_iv_m)$ and $q_i = \sum_{m=1}^{M} G_{im}(v_iu_m - u_iv_m) - B_{im}(v_iu_m - u_iv_m)$, where u_i and v_i are the real and imaginary components of the voltage phasor at node i. p_i and q_i are the active and reactive power at node i. G_{im} and B_{im} represent the physical parameters of line im. If line im does not exist, $G_{im} = B_{im} = 0$. Therefore, I can treat $\boldsymbol{x} = [u_1, v_1, \cdots, u_M, v_M]^T$ and $\boldsymbol{y} = [p_1, q_1, \cdots, p_M, q_M]^T$. The target is to learn the underlying system topology and parameters, which has broad impacts on the power domains (Li et al. 2021). In this experiment, I implement simulation from a 5-node system using MATPOWER (MATPOWER community 2020) and two year's hourly data. The first $8,760$ points are used for training while the remaining samples

are used for testing. The symbolic activation pool is $\{x\}$. I denote this dataset as Pow.

(3) **Mass-damper system dataset.** Equations of the mass-damper system can be approximately written as: $\dot{\boldsymbol{q}} = -\boldsymbol{D}\boldsymbol{R}\boldsymbol{D}^\top\boldsymbol{M}^{-1}\boldsymbol{q}$, where $\dot{\boldsymbol{q}}$ is a vector of momenta, $\boldsymbol{D}$ is the incidence matrix of the system, $\boldsymbol{R}$ is the diagonal matrix of the damping coefficients for each line of the system, and $\boldsymbol{M}$ is the diagonal matrix of each node mass of the system. Thus, I can set $\boldsymbol{y} = \dot{\boldsymbol{q}}$ and $\boldsymbol{x} = \boldsymbol{q}$ and the goal is to learn the parameter matrix $-\boldsymbol{D}\boldsymbol{R}\boldsymbol{D}^\top\boldsymbol{M}^{-1}$. I conduct the simulation via MATLAB for a 10-node system and obtain $6,000$ points for 1min simulation with a step size to be 0.01s. The first $3,000$ samples are used for training while the rest samples are used for testing. The symbolic activation pool is $\{x\}$. Then, I denote the dataset as Mas.

Benchmark methods. The following benchmark methods are utilized. (1) Deep Symbolic Regression (**DSR**) (Petersen et al. 2019). DSR develops an RNN-based framework to search the expression tree. Especially, the risk-seeking policy gradient is utilized to seek the best performance. Then, BFGS (Fletcher 2013) can solve the non-linear optimization and estimate the symbol coefficients. (2) Vanilla Policy Gradient (**VPG**) (Petersen et al. 2019). VPG is a vanilla version of DSR with a normal policy gradient rather than the risk-seeking method. (3) Equation Learner (**EQL**) (Sahoo, Lampert, and Martius 2018; Martius and Lampert 2016). EQL creates an end-to-end NN to select symbols and estimate the coefficients. The sparse regularization is enforced for the NN weights to search symbols. (4) Multilayer Perceptron (**MLP**). I also employ a standard MLP to learn the regression from $\boldsymbol{x}$ to $\boldsymbol{y}$. I only evaluate the extrapolation capacity of MLP in the test dataset.

For DSR, VPG, and EQL methods, based on the input datasets, I adjust the symbol and operator library to enable the same searching space as CoNSoLe for

fair comparisons. I run the benchmark methods 5 times with different random seeds and present their best results. As for my method, I only run 1 time and obtain good results due to the convex design and the ϵ-greedy strategy.

Metrics for evaluation. I employ the following metrics. (1) Average coefficient estimation percentage error E_c. For an equation with H symbols, I calculate the error as $E_c = \frac{1}{H}\sum_h \text{PE}(w_h, \hat{w}_h)$, where w_h and $\hat{w}_h$ represents the true and the estimated coefficients for the h^{th} symbol, respectively. PE is the operation to calculate the percentage error. If there is no matched symbol for the h^{th} true symbol, I denote $\text{PE}(w_h, \hat{w}_h) = 100\%$. Note that when calculating E_c, proper simplifications may be needed. For example, $\cos(2.5(\sqrt{x})^2) = \cos(2.5x)$. (2) NRMSE in the test dataset. I measure the extrapolation capacity in the test dataset and utilize NRMSE employed in Section 2.3.3.

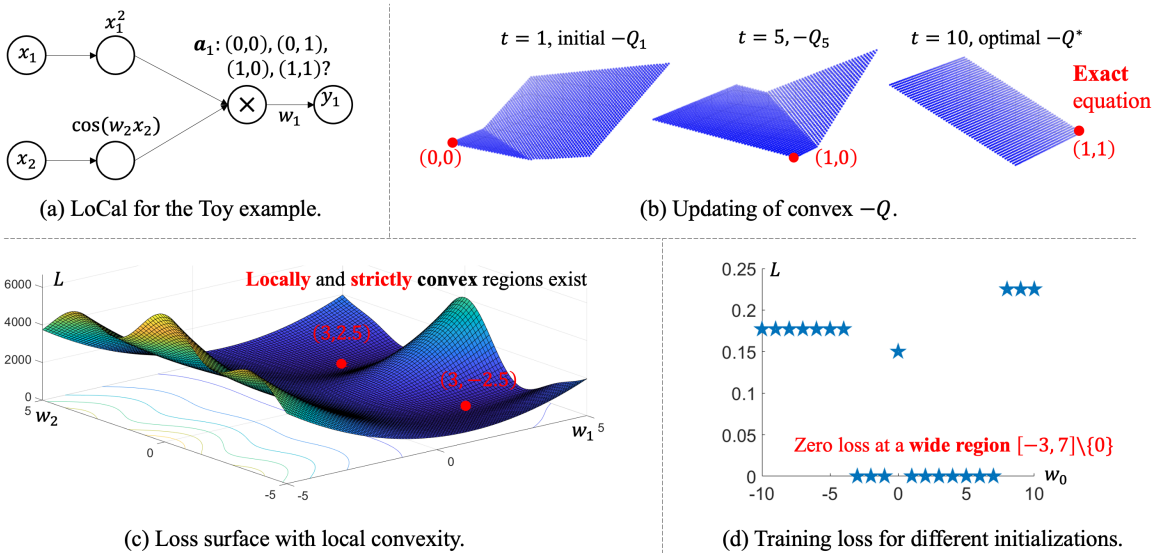
Implementing details of CoNSoLe. Hyper-parameters of CoNSoLe exist for both the double convex deep Q-learning and the LoCaL. In the deep Q-learning, I set $\gamma = 0.2$, $\epsilon = 0.4$, $T = 600$, $\lambda = 10^{-2}$, $T_0 = 10$ for Algorithm 1. Furthermore, to train the negative Q-function and the reward function, I set the learning rate to be 5×10^{-3} and the number of epochs for training to be 50. Then, I set the batch size for the negative Q-function to be 100. If the number of data in the replay buffer is less than 100, no training happens for the negative Q-function. Additionally, all the data gathered in one episode are used to train the negative reward function. As for the LoCaL, I set $K = 3$, the learning rate to be 1×10^{-2} and the number of training epochs to be 8. I make these training epochs to be small since training the LoCaL is the most time-consuming part of CoNSoLe. Furthermore, if the structure of LoCaL is correctly searched, a small number of iterations can help LoCaL to gain the global optimal weights. Finally, I initialize all trainable weights in LoCaL to be 1. The

following results show that a relatively large area is suitable for an initial guess of LoCaL.

2.3.5.0.2 Verification and 3-D Visualization of Convex Mechanisms

I first utilize a toy example to verify the benefits of convex designs for two sub problems. Specifically, I consider to learn $y_1 = 3x_1^2 \cos(2.5x_2)$ with the loss function $L = \sum_i^N (\boldsymbol{y}^i[1] - w_1 \boldsymbol{x}^i[1]^2 \cos(w_2 \boldsymbol{x}^i[2])^2$, where the data sample notations are defined in above. As shown in Fig. 2a, I design a 4-layer LoCaL with two learnable parameters w_1 and w_2 to represent the coefficients. Thus, in the search phase, the goal is to identify the action $\boldsymbol{a}_1$. I plot $-Q_t(\cdot)$ when $t = 1, 5, 10$ in Fig. 2b. The convexity always exists so that the algorithm can quickly find the global optimal solutions in red dots. Next, as the agent update Q values with true rewards, the Q-function converges to the optimal function within 10 episodes. Finally, the convex $-Q^*(\cdot)$ remains unchanged and can bring the optimal action and the true equation, which supports Theorem 2.

Subsequently, I plot the loss surface of L in Fig. 2c. I find that around the two global optima $(3, 2.5)$ and $(3, -2.5)$, there are convex regions that have an approximate quadratic shape. To further quantify the range for proper initialization, I vary the initial weight $w_0 \in \{-10, -9, \cdots, 10\}$ for w_1 and w_2 in LoCaL. Fig. 2 reports the final training loss with respect to different w_0, and I find the safe range for initialization is $[-3, 7] \setminus \{0\}$. This range is relatively large when compared to the optimal values. These observations support the Theorems 3 and 4. Finally, $w_0 = 0$ does not work since $\frac{\partial L}{\partial w_2}\big|_{w_2=0} = 0$ always holds, which prevents the weight updating using gradient methods.



(a) LoCal for the Toy example.

(b) Updating of convex $-Q$.

(c) Loss surface with local convexity.

(d) Training loss for different initializations.

Figure 2. Illustrations of convex mechanisms using a toy example.

2.3.5.0.3 Convexity Guarantees of CoNSoLe to Learn Correct Equations

In this subsection, I report the results of the equation learning. First, I list the learned equations for Syn_1 and Syn_2 in Tables 1 and 2. They present that CoNSoLe has the best performance in most of the equations while DSR ranks second. In particular, CoNSoLe can accurately learn all equations in Syn_1 and Syn_2. The superior performance is mostly due to the convex design of the search and the coefficient estimation process with provable guarantees. In addition, I observe that for the result of CoNSoLe in learning y_3 of Syn_1, I have $3x_3^2\cos(0.005x_1) \approx 3x_3^2$. This shows that there is a possibility that the search result of CoNSoLe might not be optimal (i.e., an extra consine term exists but is close to 1), but the learned equation is still highly accurate. Such an observation nonetheless guides further study of the coupling relationship between the search and the estimation procedures.

DSR doesn't perform well when the underlying equation is relatively complex,

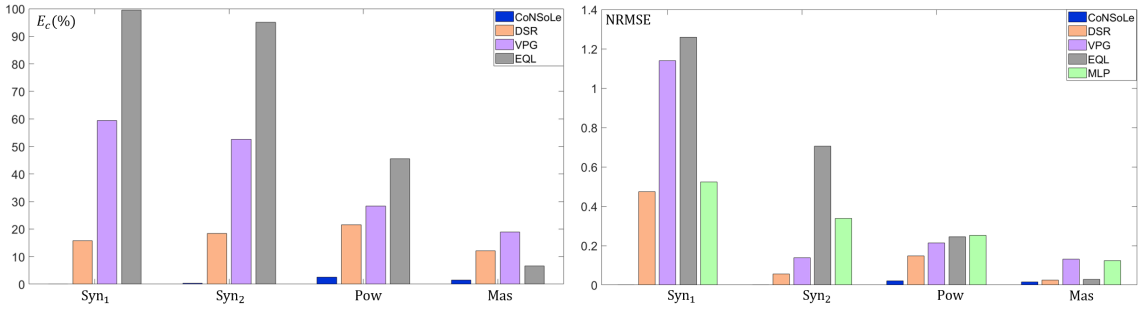
Table 1. The learned equations for Syn_1 and Syn_2, table 1.

	CoNSoLe
Syn_1	$y_1 = 3x_1^2 \cos(2.5x_2)$ $y_2 = 3.999x_1x_3$ $y_3 = 3x_3^2 \cos(0.005x_1)$
Syn_2	$y_1 = 1.47\sqrt{x_1}x_2 + 1.001x_1x_2^2$ $y_2 = \sin(1.8x_1)\big(\log(2.998x_2) + 0.998\sqrt{x_3}\big)$ $y_3 = 1.931\sqrt{x_3}\log(1.594x_1) + 1.007x_1^2$
	VPG
Syn_1	$y_1 = -0.364x_1^2 \cos(1.56x_3)$ $+4.707x_2 + 0.854x_2^2 \cos(1.98x_1)$ $y_2 = 3.293x_2 + 0.554\cos(2.82x_2)$ $y_3 = 3x_3^3$
Syn_2	$y_1 = 1.462\log(x_1)x_2 + 0.830x_1x_2^2$ $y_2 = \sin(1.220x_1)\log(3.024x_2)$ $+0.248\sin(1.454x_1)x_2^2 + 0.567\sin(1.56x_3)$ $y_3 = 2.081\sqrt{x_3}x_2 + 1.045x_1^2$

e.g., y_1 in Syn_1 and y_1 and y_2 in Syn_2. This is because DSR may still fall into a local optimal solution despite risk-seeking policy design. VPG method performs worse than DSR since VPG considers an expected reward (Petersen et al. 2019). Finally, EQL performs the worst as it merges the symbol search and the coefficient estimation in one NN model, which provides few guarantees of accurate learning. The above observations and analyses are consistent with the result of coefficient estimation errors and prediction errors in Fig. 3a and 3b. For the Pow and Mas, CoNSoLe doesn't learn completely exact equations within $T = 600$ episodes. This is because they have a large number of variables to be considered. However, CoNSoLe is still better than other methods.

Table 2. The learned equations for Syn_1 and Syn_2, table 2.

	DSR
Syn_1	$y_1 = 0.905x_2 + 3.88\cos(2.48x_2) + 1.74x_1^2\cos(1.98x_1)$ $y_2 = 4.02x_1x_3$ $y_3 = 3x_3^2$
Syn_2	$y_1 = 1.223\sqrt{x_1}x_2 + 0.181x_1\log(x_3) + 0.925x_1x_2^2$ $y_2 = \sin(1.633x_1)\log(2.965x_2)$ $+0.874\sin(1.723x_1)\sqrt{x_3}$ $y_3 = 2.081\sqrt{x_3}x_2 + 1.045x_1^2$
	EQL
Syn_1	$y_1 = 0.23x_1 + 0.021x_3^2 + 0.283x_3$ $y_2 = 0.03x_1x_3 + 0.488x_1$ $+0.045x_3^2 + 0.6x_3$ $y_3 = 0.366x_1 + 0.03x_3^2 + 0.45x_3$
Syn_2	$y_1 = 0.44x_2 + 0.2x_1^2 + 0.14x_1x_2^2 + 0.45x_1x_2$ $+0.51x_1 + 0.24x_2^2 + 0.55x_2 + 0.705$ $y_2 = 0.018x_1^2 + 0.012x_1x_2^2 + 0.0636$ $y_3 = 0.383x_2 + 0.357x_3 + 0.31x_1x_2 + 0.487$



(a) The average error $E_c(\%)$ of coefficients. (b) NRMSE of test datasets.

Figure 3. Results of equation learning for different methods and datasets.

2.3.5.0.4 Ablation Study: Exploration and Convex Search are Essential

I conduct an ablation study to further understand what factors are important in the CoNSoLe. I test the result with Syn_1 and Syn_1 and report the $E_c(\%)$ values. Specifically, I investigate the following cases. (1) No ablation. (2) Drop exploration

in deep Q-learning. I delete the ϵ-greedy strategy. (3) Drop double-convex deep-Q learning. I replace this design with a traditional deep-Q learning. (4) Drop coefficient estimation using LoCaL. After learning the structure of LoCaL, I reformulate a non-linear optimization and utilize BFGS (Fletcher 2013) in DSR, instead of gradient descent in LoCaL, to estimate the coefficient. (5) Drop static symbolic constraint. (6) Drop dynamic symbolic constraint. These two constraints are mentioned in the constraints of the search process.

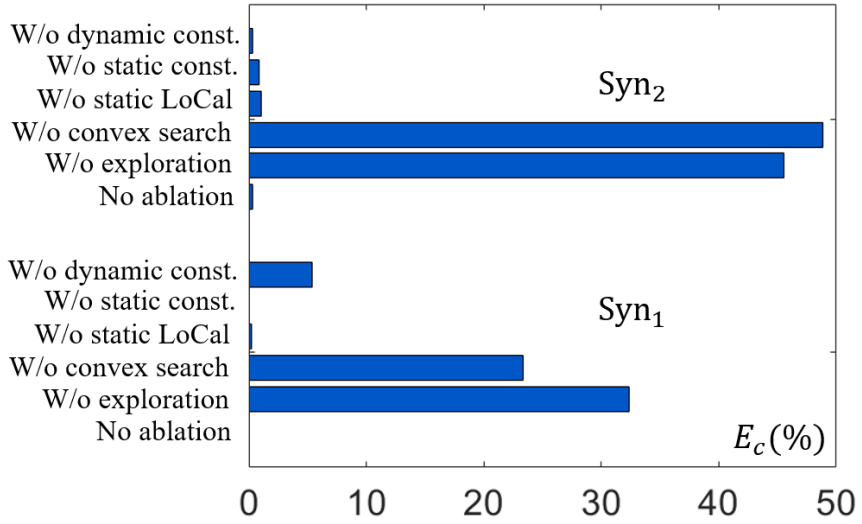
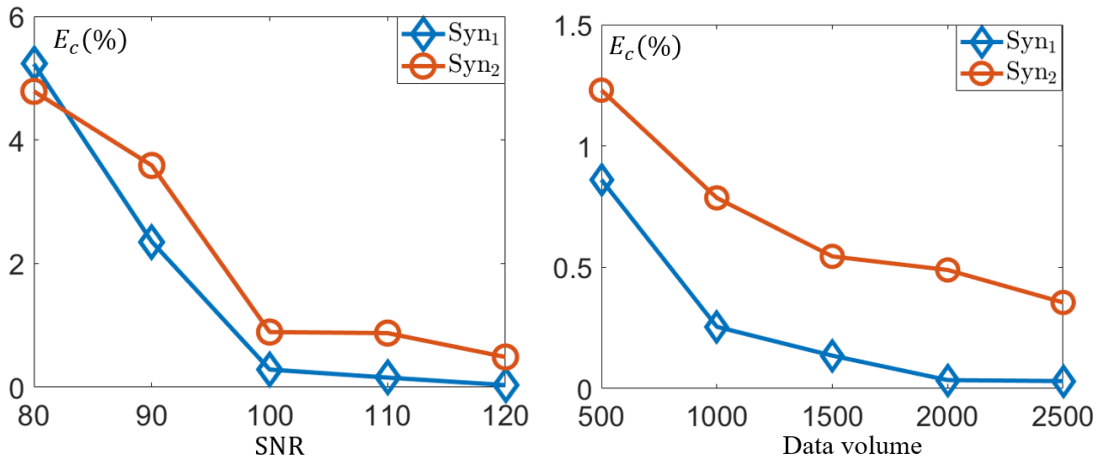


Figure 4. $E_c(\%)$ of ablation study.

Then, Fig. 4 shows that cases (2) and (3) cause large errors. For case (2), if no exploration strategy is added, the updating of the Q-function and the reward function is slow. For case (3), the non-convex search induces many sub-optimal actions in the search process. Thus, these two cases cause a slow search process and significant errors after $T = 600$ episodes. For symbolic constraints in (5) and (6), removing them slightly increases the error for Syn_1. This shows these constraints are beneficial to the search process. Finally, I find that utilizing BFGS in (4) can bring good results with initialization in the locally convex region. Since the non-linear optimization has the

same loss surface as LoCaL, the locally convex region in Theorems 3 and 4 can prove the good performance of BFGS.



(a) $E_c(\%)$ w.r.t. SNRs. (b) $E_c(\%)$ w.r.t. volumes.

Figure 5. $E_c(\%)$ of sensitivity analysis.

2.3.5.0.5 CoNSoLe is Robust with Changing Noise Levels and Data Volume

I utilize Syn$_1$ and Syn$_2$ to examine the robustness of the framework with changing noise levels and data volumes. For the noise level, I consider the Signal-to-Noise Ratio (SNR) such that SNR $\in \{80, 90, 100, 110, 120\}$. For the data volume, I fix SNR $= 100$ and vary $N \in \{500, 1000, 1500, 2000, 2500\}$. Fig. 5a and 5b demonstrate the results. I find that when SNR ≥ 100, the error can be less than 1%. This noise level is suitable to real-word systems. For example, SNR $= 125$ for electric measurements (Li et al. 2021). For the data volume, the overall error is less than 2% when $N \geq 500$, which shows a robust performance of CoNSoLe.

2.4 Learn A Physics-Consistent Model for A Partially-observable System via PCNN: Physics-Consistent Neural Network

2.4.1 Problem Scopes and Examples

In this section, I tackle Problem 2 in Section 2.2 to learn a physics-consistent model $h(\cdot)$ using measurements obtained from limited sensors in the system. The problem can still be generally decomposed into searching correct physical symbols and estimating the symbol coefficients. However, the existence of hidden quantities brings extra complexity to the problem. To simplify the problem, I assume the symbols for each quantity is known and focus on how to properly represent the hidden quantities and construct a physics-consistent model. Thus, in this section, I utilize $\boldsymbol{x}$ to represent the variable transformed from input measurements and known physical symbols for the convenience of later derivations. Consequently, the underlying physical equations $\boldsymbol{y} = g(\boldsymbol{x})$ can be written in a linear format:

$$\begin{bmatrix} \boldsymbol{y}_{\mathcal{H}} \\ \boldsymbol{y}_{\mathcal{O}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}_{\mathcal{H}\mathcal{H}} & \boldsymbol{L}_{\mathcal{H}\mathcal{O}} \\ \boldsymbol{L}_{\mathcal{O}\mathcal{H}} & \boldsymbol{L}_{\mathcal{O}\mathcal{O}} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_{\mathcal{H}} \\ \boldsymbol{x}_{\mathcal{O}} \end{bmatrix}, \tag{2.5}$$

where I utilize the subscripts $\mathcal{H}$ and $\mathcal{O}$ as partitions of the hidden and observed variables, respectively. $\boldsymbol{L}$ is the parameter matrix of the system to be figured out. However, the existence of $\boldsymbol{x}_{\mathcal{H}}$ makes it difficult to directly identify the non-zero values in $\boldsymbol{L}$, i.e., discover correct connections and edge parameters in the systems.

To guarantee the correct learning and maintain certain physics consistency, I borrow the idea of Network Reduction (NR) (Shi 2012) and propose to learn physical equations of reduced grids where virtual nodes are placed to represent the impacts of hidden nodes. In this thesis, I restrict the study scope: reduced grids should

maintain the inner connections and parameters the same as the true grids. The existence of hidden nodes only affects the connections and parameters to the boundary nodes. To mostly achieve this restriction, I propose in the next subsection that graph decomposition can help to model the reduced grid.

Thus, I propose to investigate the ability of estimating connections and parameters for small regions of the system. Specifically, I decompose the whole graph $\mathcal{G}$ into $|\mathcal{O}|$ unit-graphs $\{\mathcal{G}_i = \{\mathcal{V}_i, \mathcal{E}_i\}\}_{i=1}^{|\mathcal{O}|}$ with the graph center to be one observable node and the graph radius to be 1, where I define that the distance between every two connected nodes is 1. The concrete process is illustrated in the following subsection.

2.4.2 Graph Decomposition to Investigate the Fine-grained Capacity of Learning Physics

In this subsection, I show different types of unit graphs based on the observability, illustrated in Fig. 6.

Fully-observable unit-graph (F-Graph): This type of unit graph contains an observed node with all its 1-distance neighboring nodes observable. I denote the set of the central nodes in these unit graphs as $\mathcal{F}$. Therefore, any nodes $i \in \mathcal{F}$ with all of its 1-distance neighboring nodes, $\text{Neigh}(i)$, construct a fully-observable unit graph (F-Graph) $\mathcal{G}_i = \{i \cup \text{Neigh}(i), \mathcal{E}_i\}$. Then, the node i is *isloated* from $\mathcal{H}$. Based on equation (2.5), the topology and parameters of this sub graph can be accurately recovered via a linear layer of a neural network, i.e., a linear regression.

Partially-observable unit-graph (P-Graph): this type of area includes an observed node with at least one of its 1-distance neighboring nodes hidden. I denote the set of the central node set of these unit graphs as $\mathcal{P}$. Therefore, any node

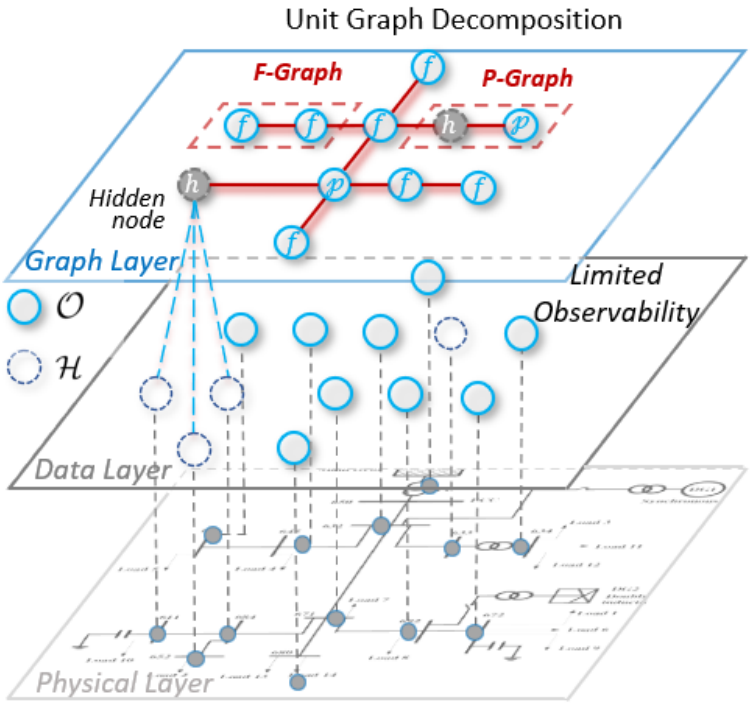


Figure 6. Physical system unit graph decomposition.

$j \in \mathcal{P}$ with nodes in $\text{Neigh}(j)$ construct a partially-observable sub graph (P-Graph) $\mathcal{G}_j = \{j \cup \text{Neigh}(j), \mathcal{E}_j\}$. Clearly, j has hidden boundary nodes $\mathcal{H} \cap \text{Neigh}(j)$. Thus, I need more layers instead of a linear layer to tackle the randomness from $\boldsymbol{x}_{\mathcal{H}\cap\text{Neigh}(j)}$, which requires multiple deep layers. Since the unknown $|\mathcal{H} \cap \text{Neigh}(j)|$ prevents the learning model construction, I aggregate $|\mathcal{H}\cap\text{Neigh}(j)|$ boundary nodes into K virtual nodes for $\mathcal{G}_i, \forall i \in \mathcal{P}$, where K is a hyper parameter. I show how to obtain a doable K value in Section 2.4.4. For $\mathcal{G}_j$, I denote the virtual node set to be $\mathcal{N}_j = \{j_k\}_{k=1}^K$.

With above categorization, I observe that the different types of uni-graphs require different NN layers to represent the corresponding physical equations. Therefore, I propose to represent $h(\cdot)$ as a Physics-Consistent Neural Network (PCNN) with a structural design of the deep NN and the shallow NN.

2.4.3 PCNN Structure: Deep-Shallow Hierarchy

F-Graph Layer: I utilize a linear layer to recover the topology and parameter of F-Graphs.

$$h_F(\boldsymbol{x}_{\mathcal{O}}) = \boldsymbol{\theta}_F \boldsymbol{x}_{\mathcal{O}}, \tag{2.6}$$

where $\boldsymbol{\theta}_F$ is the weights for the F-Graph layer. As discussed in Section 2.4.2, the equation of the center node in $\mathcal{F}$ can be completely represented via the linear layer. Thus, F-Graph layer can be pre-trained to identify the node set $\mathcal{F}$. As the input is all the data in $\mathcal{O}$, the pre-training should include a Least Absolute Shrinkage and Selection Operator (LASSO) regularization term $\lambda_1 ||\boldsymbol{\theta}_F||_1$ to guarantee the sparsity, where λ_1 is the hyper parameter for the penalty and $||\cdot||_p$ is the l_p norm. Namely, the pre-training tries to minimize:

$$\frac{1}{N}\sum_{n=1}^{N} ||\boldsymbol{y}_{\mathcal{O}}^n - h_F(\boldsymbol{x}_{\mathcal{O}}^n)||_2^2 + \lambda_1 ||\boldsymbol{\theta}_F||_1, \tag{2.7}$$

After the pre-training, I obtain $h_F^0(\boldsymbol{x}_{\mathcal{O}}) = \boldsymbol{\theta}_F^0 \boldsymbol{x}_{\mathcal{O}}$ as the pre-trained linear layer. This helps to identify the F-Graphs. Specifically, $\forall i \in \mathcal{O}$, the identification criteria is:

$$\frac{1}{N}\sum_{n=1}^{N} ||\boldsymbol{y}_{\mathcal{O}}^n[i] - h_F^0(\boldsymbol{x}_{\mathcal{O}}^n)[i]||_2 \leq \epsilon_1, \tag{2.8}$$

where $\boldsymbol{y}_{\mathcal{O}}^n[i]$ and $h_F^0(\boldsymbol{x}_{\mathcal{O}}^n)[i]$ are the i^{th} elements in $\boldsymbol{y}_{\mathcal{O}}^n$ and $h_F(\boldsymbol{x}_{\mathcal{O}}^n)$, respectively. ϵ_1 is a threshold. If equation (2.8) is satisfied, then $i \in \mathcal{F}$ since a linear equation can accurately represent $\boldsymbol{y}_{\mathcal{O}}[i]$. Based on the obtained $\mathcal{F}$, I can further compute $\mathcal{P} = \mathcal{O} \setminus \mathcal{F}$.

The pre-trained results can further bring the initialization of the F-Graph layers. Specifically, I have the following sequential **F-Graph initialization rule**:

1. $\forall i \in \mathcal{F}$, initialize $\boldsymbol{\theta}_F[i,:]$, i.e., the i^{th} row of $\boldsymbol{\theta}_F$, as the corresponding trained values $\boldsymbol{\theta}_F^0[i,:]$.

2. $\forall j \in \mathcal{P}, \forall i \in \mathcal{F}$, initialize $\boldsymbol{\theta}_F[j,i]$ as the trained value $\boldsymbol{\theta}_F^0[i,j]$.

3. $\forall j \in \mathcal{P}, \forall i \in \mathcal{P}$, initialize $\boldsymbol{\theta}_F[j,i]$ as 0 if $i \neq j$ and $-\sum_{k \neq j} \boldsymbol{\theta}_F[j,k]$ if $i = j$.

Rule 2 and 3 in the **F-Graph initialization rule** helps to fix some parameters for the P-Graph due the symmetry of the system. Namely, if $ij \in \mathcal{E}$, $i \in \mathcal{F}$ and $j \in \mathcal{P}$, the edge parameter of ij obtained from the F-Graph pre-training is fixed for both $\mathcal{G}_i$ and $\mathcal{G}_j$.

N-Approximation Layers: The initialization of F-Graph gives an accurate approximation of edge weights among nodes in the F-Graph. However, some edge parameters in the P-Graph $\mathcal{G}_j$ require further estimation due to the hidden quantities. Thus, for $\mathcal{G}_j$, I model the contributions of hidden quantities via K virtual nodes $\mathcal{N}_j$ as mentioned before. Though the input samples of the virtual nodes are unknown, I can approximate them using the observed nodes' input and a deep neural network (N-Approximation Layers) h_N:

$$\boldsymbol{x}_{\mathcal{N}} = h_N(\boldsymbol{x}_{\mathcal{O}}), \tag{2.9}$$

where $\mathcal{N} = \bigcup_{j=1}^{|\mathcal{P}|} \mathcal{N}_j$ represents the total set of virtual nodes.

P-Graph Layer: For a P-Graph $\mathcal{G}_j$, the contributions to the equation from nodes $\mathcal{F} \cap \text{Neigh}(j)$ are identified in the F-Graph Layer. Thus, I only need to consider the contributions from $\mathcal{N}_j \cup \text{Neigh}(j)$. Since I approximate the measurements from $\mathcal{N}_j$ using $h_N(\boldsymbol{x}_{\mathcal{O}})$, I can build another linear layer (P-Graph Layer) such that:

$$\boldsymbol{y}_{\mathcal{O}} - h_F(\boldsymbol{x}_{\mathcal{O}}) = h_P(\boldsymbol{x}_{\mathcal{N} \cup \mathcal{O}}) = \boldsymbol{\theta}_P \boldsymbol{x}_{\mathcal{N} \cup \mathcal{O}}, \tag{2.10}$$

where $h_F(\boldsymbol{x}_{\mathcal{O}})$ represents the output from the F-Graph Layer, $\boldsymbol{x}_{\mathcal{N} \cup \mathcal{O}} = [\boldsymbol{x}_{\mathcal{N}}; \boldsymbol{x}_{\mathcal{O}}]$ is the concatenation of $\boldsymbol{x}_{\mathcal{N}}$ and $\boldsymbol{x}_{\mathcal{O}}$, and $\boldsymbol{\theta}_P \in \mathbb{R}^{|\mathcal{O}| \times |\mathcal{N} \cup \mathcal{O}|}$ is the weight matrix of the P-Graph Layer.

In general, the above structural design utilize a DNN to approximate the hidden quantities. Despite the high approximation power, DNN models may bring sub-optimal results. To mitigate this issue, I propose in this Section that a convexity-based initialization can happen for the P-Graph layers. Specifically, I develop the following sequential **P-Graph initialization rule**:

1. $\forall j \in \mathcal{P}, \forall k \in \mathcal{N}$, if $k \in \mathcal{N}_j$ initializes $\boldsymbol{\theta}_P[j,k]$ from the optimal solution of a set of *convex optimizations* proposed in the next section. If $k \notin \mathcal{N}_j$, initialize $\boldsymbol{\theta}_P[j,k]$ to be 0.
2. $\forall j \in \mathcal{P}, \forall k \in \mathcal{N}, \forall i \in \mathcal{O}$, if $j = i$ initialize $\boldsymbol{\theta}_P[j,i]$ to be $-\sum_{k\in\mathcal{N}} \boldsymbol{\theta}_P[j,k]$. If $j \neq i$, initialize $\boldsymbol{\theta}_P[j,i]$ to be 0.
3. $\forall i \in \mathcal{F}, \forall k \in \mathcal{N} \cup \mathcal{O}$, initialize $\boldsymbol{\theta}_P[i,k]$ to be 0.

For rule 1, $\forall j \in \mathcal{P}, \forall k \in \mathcal{N}$, if $k \in \mathcal{N}_j$, the initial guess represents a good approximation for the weight of edge jk. In the next section, I propose a set of convex optimizations to obtain the optimal solution that both minimizes the squared loss and satisfies physical parameter constraints. I will theoretically prove that within these constraints, a globally optimal solution with zero loss for the noiseless data can be achieved due to convexity. Then, the global optimality represents the physics consistency. If $k \notin \mathcal{N}_j$, edge jk does not exist so that the initial value of $\boldsymbol{\theta}_P[j,k]$ is 0. Additionally, rules 2 and 3 are based on the symmetry and connectivity of the system.

The optimization also brings good estimation values for $\boldsymbol{x}_\mathcal{N}$, thus inducing the **N-Approximation initialization rule**:

1. Initialize parameters in $f_N(\boldsymbol{x}_\mathcal{O})$ via pre-training the network of f_N using input data from $\boldsymbol{x}_\mathcal{O}$ and estimated data of $\boldsymbol{x}_\mathcal{N}$ from the proposed *convex optimizations* in the next section.

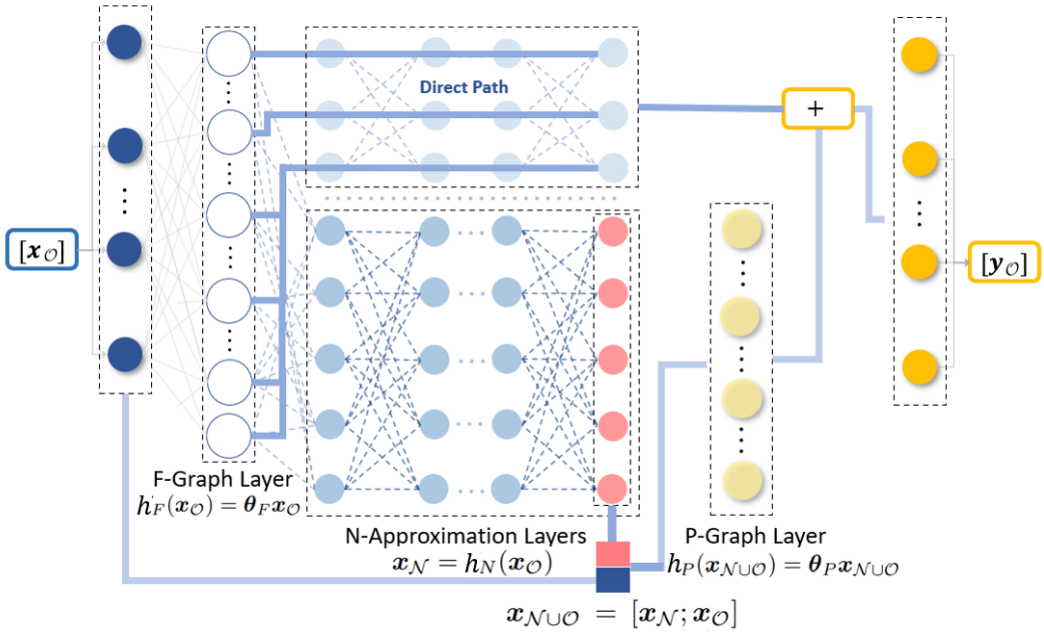


Figure 7. The design of the PCNN.

In conclusion, I show the proposed PCNN model in Fig. 7. The formulation is as follows:

$$\boldsymbol{y}_{\mathcal{O}} = h(\boldsymbol{x}_{\mathcal{O}}) = h_F(\boldsymbol{x}_{\mathcal{O}}) + h_P([f_N(\boldsymbol{x}_{\mathcal{O}}); \boldsymbol{x}_{\mathcal{O}}]). \tag{2.11}$$

Though I have good initial parameters for the PCNN, the retraining of the PCNN as a whole is still required for an end-to-end optimization to minimize the total loss. Finally, the complete algorithm for the pre-training and the retraining process can be summarized in Algorithm 2.

2.4.4 Theoretical Analysis for Convexity-based PCNN Initialization

The proposed PCNN embraces the deep-shallow structure where the deep NNs approximate the hidden variable, and the shallow NN formalizes all the variables into the physical-equation representation. Specifically, each output entry represents a

Algorithm 2 Training Algorithm for PCNN

Input: Measurements $\{\boldsymbol{x}^n\}_{n=1}^N$ and $\{\boldsymbol{y}^n\}_{n=1}^N$ from observed nodes.
Hyper-parameters: penalty term λ_1 for the pre-training of the F-Graph layer and threshold ϵ_1 for the identification of $\mathcal{F}$.
Pre-train the F-Graph Layer using Equation (2.7).
Obtain $\mathcal{F}$ set using Equation (2.8).
Compute $\mathcal{P} = \mathcal{O} \setminus \mathcal{F}$.
Initialize $\boldsymbol{\theta}_F$ using **F-Graph initialization rule**.
Solve the proposed optimization in equation (2.12) using Algorithm 3.
Initialize $\boldsymbol{\theta}_P$ using **P-Graph initialization rule**.
Initialize parameters in the deep layers $f_N(\boldsymbol{x}_{\mathcal{O}})$ using **N-Approximation initialization rule**.
Retrain PCNN using Stochastic Gradient Descent (SGD) algorithm.
Output: PCNN model.

nodal balance equation. In this section, I verify that the initialization rules for the F-Graph and P-Graph Layers can provide a physics-consistent solution. I first define this solution as follows.

Definition 1 (Physics-consistent solution)**.** *A physics-consistent solution (PCS) for the parameters of F-Graph and P-Graph Layers and the output of the N-Approximation Layer brings* 0 *loss for the noiseless data and satisfy all the physical parameter constraints. PCS is the solution of the optimization in Equation* (2.7) *and Equation* (2.12).

Based on the definition, the solutions for the F-Graph Layer can be obtained via optimizing Equation (2.7), which is a convex optimization. Next, I show the solutions for the P-Graph Layer and the output of the N-Approximation Layer can be obtained via the following optimization.

Specifically, I consider one P-Graph $\mathcal{G}_j, \forall j \in \mathcal{P}$ as an example. Based on the initialization rules, the parameter of $ji, \forall i \in \mathcal{F} \cap \text{Neigh}(j)$ has been quantified in the F-Graph Layer. Thus, I only need to discuss the parameter of $jk, \forall k \in \mathcal{N}_j \cap \text{Neigh}(j)$.

I denote w_k as the parameter of line jk, and w_k is one element in the parameter matrix $\boldsymbol{\theta}_P$ in the P-Graph Layer. Further, I let $p_n = \boldsymbol{y}^n_{\mathcal{O}}[j] - f^0_F(\boldsymbol{x}^n_{\mathcal{O}})[j]$ as the n^{th} sample net flow from $\mathcal{N}_j$, where the estimation of the net flow is guaranteed via accurate parameter estimation of the F-Graph Layer. Similarly, I let $x^n = \boldsymbol{x}^n_{\mathcal{O}}[j]$ as the n^{th} input measurement of node j. Finally, I denote x^n_k as the n^{th} approximated nodal measurements for the k^{th} virtual node in $\mathcal{N}_j$. Thus, x^n_k is a realization of one element in $\boldsymbol{x}^n_{\mathcal{N}}$. Here I eliminate the index j in w_k, p_n, x^n and x^k_n for simplicity. To find a good physics-consistent initialization, I propose to treat w_k and x^k_n as *variables* and formalize the following optimization $\mathbb{P}^j_K$.

$$\begin{aligned} \min_{w_k, x^n_k} L &= \sum_{n=1}^{N} (p_n - \sum_{k=1}^{K} w_k (x^n - x^n_k))^2 \\ s.t. \;\; &\{w_k, \{x^n_k\}_{n=1}^N\}_{k=1}^K \in \mathcal{C}_K, \end{aligned} \tag{2.12}$$

where L is the loss and I eliminate the index j, K for simplicity. $\mathcal{C}_K$ represents for K virtual nodes, the feasible region under a set of physical constraints. For example, the tolerance of the nodal devices requires x^n_k to have positive minimum and maximum values. Further, the capacity of the line jk limits the maximum values of the flow on that line, i.e., $|w_k(x^n - x^n_k)|$ has an upper bound. It can be easily proven that under the above constraints, $\mathcal{C}$ is convex. I assume this convexity holds in general for all the proposed physical constraints.

To prove the PCS exists with global optimality and under physical constraints, I propose the following theorem.

Theorem 5. *Physics-Consistent Solutions (PCS) defined in Definition 1 exist for the proposed PCNN.*

The proof can be seen in Section A.5. Further, to comupte the PCS, I propose the following theorem.

Theorem 6. *If the data number N is sufficiently large, Physics-Consistent Solutions (PCS) defined in Definition 1 can be obtained via solving convex optimizations.*

The proof can be seen in Section A.6. The process above presents for a P-Graph $G_j, \forall j \in \mathcal{P}$, the existence of the PCS for some Ks and the convexity of $\mathbb{P}_K^j$ for any K. Thus, I propose to iteratively solve $\mathbb{P}_K^j$ and evaluate if the solution is a PCS. Since the real-world data is not noiseless, I employ a threshold ϵ_2 for the evaluation. Then, the algorithm is shown in Algorithm 3 for the PCS for P-Graph Layer and N-Approximation Layer.

Algorithm 3 Training Algorithm for $\{\mathbb{P}_K^j\}, \forall j \in \mathcal{P}$

Input: Measurements $\{\boldsymbol{x}^n\}_{n=1}^N$ and $\{\boldsymbol{y}^n\}_{n=1}^N$ from observed nodes, and node set $\mathcal{P}$.
Hyper-parameters: threshold ϵ_2.
Initialize $K = 1$.
for j=1 to $|\mathcal{P}|$ **do**
  while $L_K^j > \epsilon_2$ **do**
    Use Gradient Descent (GD) to solve $\mathbb{P}_K^j$.
    Evaluate L_K^j, i.e., the loss of $\mathbb{P}_K^j$.
    $K = K + 1$.
Obtain the PCS as the optimal solutions of the above optimizations.
Output: A PCS for parameters in the P-Graph Layer and outputs of the N-Approximation Layer.

2.4.5 Experimental Result

2.4.5.0.1 Settings

Dataset Description. In the experiment, I introduce power systems, mass-damper systems, hydraulic networks, and the graph of large systems from the University of Florida (UF) sparse matrix collection (Davis and Hu 2011) as the underlying physical

system for model training and comparison. Specifically, the dataset descriptions are as follows.

(1) **IEEE Power Systems and PJM Load Data**. IEEE provides standard power system models, including the grid topology, parameters, and generation models, etc., for accurate simulations on the power domain. The model files and the simulation platform, MATPOWER (MATPOWER community 2020), are based on MATLAB. In this experiment, I incorporate IEEE 19-, 30-, 57-, 69-, and 85-systems for testing. To conduct the simulation, the load files are required as the input to the systems. Thus, I introduce real-world power consumptions in PJM Interconnection LLC (PJM) data (PJM Interconnection LLC 2018). The load files contain hourly power consumption in 2017 for the PJM RTO regions. With the above data, MATPOWER produces the system states of voltage angle $\boldsymbol{\phi}$ and system input active power flow $\boldsymbol{p}$, indicating the linearized power flow equations $\boldsymbol{p} = \boldsymbol{L}_A\boldsymbol{\phi}$, which can be formulated like Equation (2.5), and $\boldsymbol{L}_A$ is the weighted Laplacian matrix (i.e., the susceptance matrix) of the electric system.

(2) **Mass-damper system data**. The mass-damper systems can be represented with the physical equation $\dot{\boldsymbol{q}} = -\boldsymbol{D}\boldsymbol{R}\boldsymbol{D}^\top\boldsymbol{M}^{-1}\boldsymbol{q}$, where $\boldsymbol{q}$ is the vector of momenta of the masses, $\boldsymbol{D}$ is the incidence matrix of the graph, $\boldsymbol{R}$ is the diagonal matrix of the damping coefficients of the damper attached to the edges, and $\boldsymbol{M}$ is the diagonal mass matrix (Schaft 2017). Using MATLAB, I simulate the dynamic process of the mass-damper system with 10 buses and obtain $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

(3) **UF sparse matrix-based system**. The UF sparse matrix collection provides a lot of large sparse matrix-based networks. In this experiment, I utilize the 2003-bus system to test.

Therefore, I have three different systems, providing testing on 10-, 19-, 57-, 69-, 85-,

and 2003-node networks. To consider different system observability, I change the ratio of the number of the observed nodes to that of the total nodes $\gamma \in \{0.1, \cdots, 0.9\}$.

Benchmark Models. To fully investigate the strong interpretability and generalizability of the PCNN, I compare the proposed PCNN with other advanced DNN models. Specifically, I have the following benchmark models for comparison.

- **Resnet** (K. He et al. 2016). Deep Residual Network creates a shortcut connection to pass the deep information directly to the shallow layers. Such a skip-connection effect not only helps to avoid gradient vanishing issues in the training phase, but also contributes to the model generalization ability since the low-complexity features are connected to the output, thus decreasing the model complexity (Liu and Chen 2018).
- **SINDYs** (Brunton, Proctor, and Kutz 2016; Champion et al. 2019). The sparse identification of nonlinear dynamics (SINDy) utilizes the sparse regression technique to recover the parameters of the physical systems, while the base of the regression can be selected via DNNs. In the experiments, I consider systems with a fixed symbolic base due to the prior knowledge, and I eliminate the DNN part for simplicity.
- **DNNs with Dropout Method** (Srivastava et al. 2014). Dropout method randomly disables neurons in training, thus preventing the neurons from over co-adapting and increasing the model generalizability.
- **DeepLIFT** (Shrikumar, Greenside, and Kundaje 2017; Lundberg and Lee 2017). DeepLIFT is an advanced model to select important features of a well-trained DNN via calculating the importance signals from output to the input features. In this experiment, I calculate the SHAP (SHapley Additive exPlanations) values

of features in the trained DNN via DeepLIFT (Lundberg and Lee 2017). Thus, I can select the important features and evaluate the model interpretability.

Model Evaluations. I propose the following metrics to evaluate the generalizability and interpretability of PCNN and benchmark models.

(1) **Generalizability**. I conduct 5-fold cross-validation to evaluate the model generalizability. Mean square error (MSE) of the testing set is used to evaluate the model performance of predicting $\boldsymbol{y}_{\mathcal{O}}$.

(2) **Parameter estimation**. I utilize all the data to estimate the system parameters. To evaluate the model performance, I consider the following aspect: for lines among node set $\mathcal{O}$, both the line weight estimation error and the connectivity should be evaluated. Since the connectivity can be converted to the sparsity of the Laplacian matrix, I utilize the so-called normalized Total Vector Error ($nTVE$) (Li et al. 2021) to evaluate the difference between the estimated $\hat{\boldsymbol{L}}$ and the true Laplacian matrix $\boldsymbol{L}$:

$$nTVE = 100 \times \frac{||\hat{\boldsymbol{L}} - \boldsymbol{L}||_2}{||\boldsymbol{L}||_2}. \tag{2.13}$$

(3) **Interpretability**. The model interpretability determines the critical input features with respect to each output channel. For Resnet and DNNs with Dropout method, I utilize DeepLIFT (Lundberg and Lee 2017) for important feature selection. For the proposed PCNN, the sparsity of $\boldsymbol{\theta}_F$ illustrates the estimated topology within $\mathcal{O}$. Thus, for each $i \in \mathcal{O}$, the inputs in the neighboring nodes in the *estimated unit graph* are the important features. For the SINDy method, I denote input features with non-zero coefficients for one output feature as its important features.

In general, I denote the indices of the estimated important features as $Import(i)$ for the i^{th} output and the ground true indices are $Neigh(i) \cup \{i\}$. Thus, I introduce

the measure $h(\%)$:

$$h = \sum_{j \in \mathcal{O}} \frac{J(Import(j), Neigh(j) \cup \{j\})}{|\mathcal{O}|} \times 100\%,\ J(X,Y) = \frac{|X \cap Y|}{X \cup Y},$$

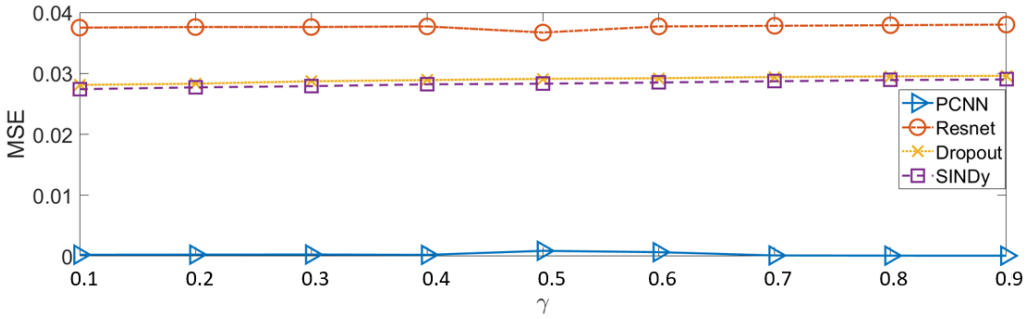
where $J(\cdot,\cdot)$ is the so-called Jaccard index.

2.4.5.0.2 Results for Model Generalizability

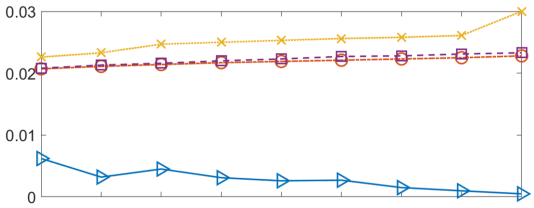
In the experiments, I test different systems with changing γ to comprehensively compare the model generalizability among different methods. 5-fold cross-validation is conducted. The results are shown in Fig. 19a to 8g. I find that for each trial, the PCNN **always achieves the lowest MSE value** in the validation dataset. Further, the MSE of the PCNN decreases as γ increases, while for other methods, the MSE increases. The lowest generalization error comes from (1) the well-extracted local governing equations that are generalizable to different datasets and (2) the physical constraints that enable the physical variables to be within the physical range. Secondly, the increasing of sensor penetration (γ) leads to more physical parameters to be captured, thus decreasing the MSE further. However, for other methods without physical consistency, MSE will increase due to the growth of the output dimensionality.

2.4.5.0.3 Results for Network Parameter Estimation

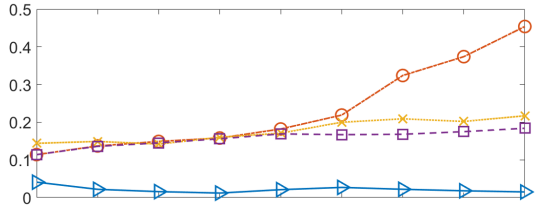
For the line parameters and connectivity among observed nodes $\mathcal{O}$, I calculate the $nTVE(\%)$ for evaluation. The comparison is between the PCNN and the SINDy since other DNNs can't estimate the physical equation parameters. The result is shown in Table 3 and 4. Generally, the PCNN far outperforms the SINDy method for all



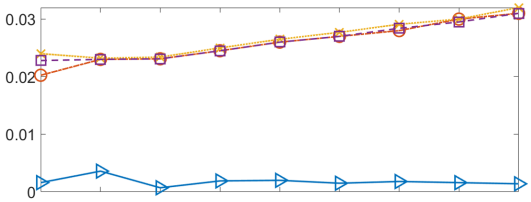
(a) MSE for different methods in the 10-bus system.



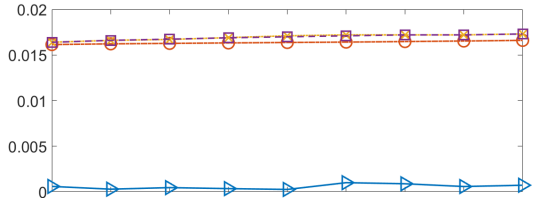
(b) MSE for different methods in the 19-bus system.



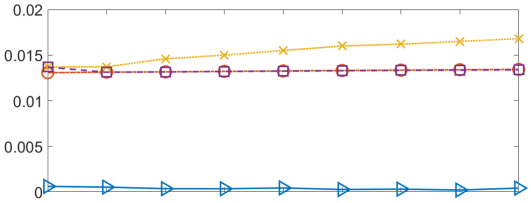
(c) MSE for different methods in the 30-bus system.



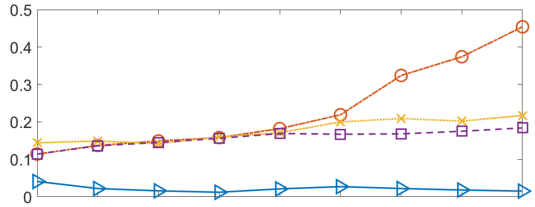
(d) MSE for different methods in the 57-bus system.



(e) MSE for different methods in the 69-bus system.



(f) MSE for different methods in the 85-bus system.



(g) MSE for different methods in the 2003-bus system.

Figure 8. The MSE value for different testing systems.

systems when $\gamma < 0.5$. Empirically, the PCNN's $nTVE$ is around $10\% \sim 25\%$ of the SINDy's $nTVE$. When γ increases, the performance of the PCNN and SINDy will become closer. However, PCNN's $nTVE$ still only covers around $40\% \sim 60\%$ of SINDy's $nTVE$. The reasons are as follows. (1) PCNN employs a testing criterion in equation (2.8) to decompose $\mathcal{O}$ into $\mathcal{F}$ and $\mathcal{P}$. Then, the initialization rule of the PCNN can enable the shared weights between $\mathcal{F}$ and $\mathcal{P}$ to always be accurately estimated in the pre-training of the F-Graph. For the SINDy method, however, the shared weight estimation incurs errors due to hidden quantities. (2) when $\gamma < 0.5$, the hidden nodes are dominant so that PCNN performs much better than SINDy. (3) when γ is increasing, the number of hidden nodes decreases so that the inaccurate estimation of the shared weights in SINDy decreases, forcing PCNN and SINDy to have closer performance. Secondly, I observe in Table 3 that for 19-bus system, PCNN and SINDy have relatively small $nTVE$ compared to other systems. This is because 19-bus system is radial so that a hidden node will only cause errors within one line for line parameter estimation.

Table 3. $nTVE(\%)$ error of parameter estimation for PCNN and SINDy methods, table 1.

	10-bus		19-bus		30-bus		57-bus	
γ	PCNN	SINDy	PCNN	SINDy	PCNN	SINDy	PCNN	SINDy
0.1	69	381	4.5	23	32	89	73	169
0.2	51	317	3.6	21	33	83	65	198
0.3	56	265	6.3	24	30	81	78	156
0.4	43	198	3.3	18	27	78	71	153
0.5	45	118	2.9	15	27	72	72	145
0.6	62	97	4.4	12	23	61	64	132
0.7	41	65	0.95	7.3	12	55	52	122
0.8	37	72	0.89	5.1	8.8	29	47	98
0.9	18	32	0.73	4.8	9.5	21	34	94

Table 4. $nTVE(\%)$ error of parameter estimation for PCNN and SINDy methods, table 2.

	69-bus		85-bus		2003-bus	
γ	PCNN	SINDy	PCNN	SINDy	PCNN	SINDy
0.1	65	648	31	139	89	399
0.2	68	723	34	121	74	421
0.3	68	614	24	118	61	406
0.4	54	598	19	123	59	385
0.5	79	470	16	121	78	335
0.6	76	423	13	104	66	299
0.7	69	327	9.8	96	64	301
0.8	43	211	10	93	59	276
0.9	22	108	10	71	57	283

2.4.5.0.4 Results for Model Interpretability

To test the model interpretability, I set $\gamma = 0.5$ and calculate the measure h in Equation (2.4.5.0.1) under different scenarios, as is shown in Table 5. The PCNN can always obtain 100% interpretable features, which show that the estimated topology within $\mathcal{O}$ is correct. The perfect performance essentially comes from the sparsity control when pre-training the F-Graph Layer. For SINDy method, the sparsity control also exists, thus yielding high h values. However, the hidden quantities bring some incorrect connectivity and prevent the h to be 100%. For the other two DNNs, h will decrease about $30\% \sim 80\%$ due to the complex correlations in the NN model.

Table 5. The $h(\%)$ value for different methods in different systems.

	PCNN	SINDy	Resnet	Dropout
10-bus	100	100	74.0	52.4
19-bus	100	93.2	69.0	36.7
30-bus	100	92.9	71.9	31.3
57-bus	100	85.3	57.4	12.8
85-bus	100	85.7	73.8	23.8
2003-bus	100	75.4	73.8	23.8

Chapter 3

IDENTIFYING SYSTEM EVENT INFORMATION

3.1 Introduction

Modern physical systems significantly incorporate highly uncertain components to facilitate clean and low-cost productions and consumptions. To better accommodate the growing uncertainty and maintain the system stability, the physical system requires advanced tools for system event identification. To capture the event dynamics, some advanced measuring techniques Phasor Measurement Units (PMUs) provide synchronized phasor measurements with high-granularity (e.g., 30 or 60 samples per second) (Yuan, Wang, and Wang 2020). Therefore, the data-driven event identification is one of the central topics to improve the system reliability. Many efforts are developed to analyze measurement patterns and identify when, where, and what type of events are. To find the event initialization time, methods like change point detection (Li et al. 2019b) can detect abnormal intervals that imply events. However, to know more information about event types and locations, how to analyze data streams in the best way becomes challenging.

One idea to find event types and locations is to use expert information. For example, one can use signal transformation or filtering to map the time series data into some physically meaningful domain for comparing with some predefined thresholds. These methods use wavelet transformation (Kim et al. 2015), Kalman filtering (Pérez and Barros 2008), and Swing Door Trending (SDT) (Cui et al. 2018), etc. For example, (Cui et al. 2018) utilizes a swing door to compress data with a pre-defined door width,

and the detectable events must have a certain level of slope rate. However, as these methods need to pre-define some measures or thresholds, the usage may be biased because of the specific design and test cases. Therefore, can I have a general model?

For obtaining a general form, previous work proposes to use existing events and their labels to train in a Supervised Learning (SL) manner. Such Machine Learning (ML) models typically extract features for minimizing the loss function. For instance, Decision Tree (DT) (Li et al. 2019a) treats each measurement as a factor to determine the final decision. Although transparent, such a method is inefficient to make use of complex measurement correlations. Therefore, (De Yong, Bhowmik, and Magnago 2015) proposes Support Vector Machine (SVM) to assign each input measurement a weight to form the final feature. There are also more complex and powerful models such as Convolutional Neural Network (CNN) (Yuan et al. 2021) and Graph Neural Network (GNN) (Yuan, Wang, and Wang 2020). They consider the spatial correlations with square and graph convolutions, respectively.

One can also couple the temporal information in Long Short-Term Memory (LSTM) units. For example, (Zhang et al. 2018) uses LSTM to extract periodic patterns and data inertia in time. However, for physical data, it's desirable to simultaneously consider correlations among spatial, temporal, and measurement type dimensions. So, one can keep on increasing the model complexity. But, some physical measurement streams accumulate quickly into terabyte (TB) level for training due to high volumes, large dimensionality, and complex correlations among the space, time, and measurement type (e.g., voltage magnitude, angle, frequency, etc.) dimensions.

In general, the above models generally face the challenges of (1) limited interpretability, (2) biased learning models, (3) large computational cost, and (4) insufficient labeled data for learning. For example, many event records are missing in utility logs.

For example, (Brahma et al. 2017) claims $1,013$ events could be identified based on a power utility's data records from 2001 to 2010, but only 84 events were reported in the utility log files. Therefore, this chapter tries to tackle above challenges by proposing (i) efficient and unbiased Supervised Learning framework based on ensemble techniques, (ii) Physics-guided Unsupervised Learning to tackle the scenario without labels, and (iii) Tensor-based Semi-supervised Learning to handle large data volumes.

3.2 Problem Formulation

In order to formulate the learning methods for event identification, I need to describe the time series data set provided by system sensors and the associated label data set. In this thesis, I consider two cases. The first case is when the data volume is not large, and I can utilize a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times A}$ to represent the system event data, where N is the number of time slots and A represent all the number of measurement columns for the data. The second case is when the data volume is large, and I utilize a tensor $\mathcal{X}' \in \mathbb{R}^{T \times L \times M}$ to represent a time slot's system measurement, where T denotes the number of time slots for each window, L denotes the number of sensors, and M denotes the number of measurement types (e.g., Voltage Magnitude (VM), Voltage Angle (VA), Frequency (F), etc.). Further, the total N tensors are denoted as the total event tensor $\mathcal{X} \in \mathbb{R}^{N \times T \times L \times M}$.

As for the labels, I utilize a vector $\boldsymbol{y} \in \mathbb{Z}^{H \times 1}$ to represent the label vector (i.e., the event type and location). If $H = 0$, it's the Unsupervised Learning setting. If $H < N$, it's the Semi-supervised Learning setting. Finally, if $H = N$, it's the supervised learning setting. With above definitions, I propose the following problem formulations.

- Problem 1: supervised event identification.

- Given: an event matrix $\boldsymbol{X}$ and a label vector $\boldsymbol{y}$.
- Find: an abstract mapping $f(\boldsymbol{X}) = \boldsymbol{y}$.

- Problem 2: unsupervised event identification.
- Given: an event matrix $\boldsymbol{X}$.
- Find: an abstract mapping $f(\mathcal{X})$ to identify event types and locations.

- Problem 3: semi-supervised event identification using tensor data.
- Given: a total event tensor $\mathcal{X}$ and a label vector $\boldsymbol{y}$.
- Find: an abstract mapping $f(\mathcal{X}) = \boldsymbol{y}$ to compress the information in $\mathcal{X}$ and use the compressed information to identify event labels in $\boldsymbol{y}$.

3.3 Supervised Event Identification with Unbiased Learning

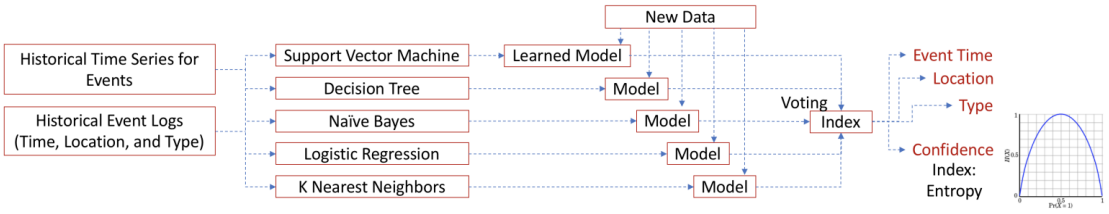


Figure 9. An illustration of the flow chart on the proposed ensemble learning for event identification.

In this section, I tackle Problem 1 in Section 3.2. Specifically, I employ different learning methods as candidates for making event decisions based on incoming sensor data. As shown in Fig. 9, each machine learning model trains its classifier separately. To avoid biases from different learning models, the process not only combines the result of different models for event identification, but also provides an entropy-based

index to measure the confidence of the voted label. The index E is shown below:

$$E = 1 + \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} p_{(n,k)} \log p_{(n,k)} / \log(M), \tag{3.1}$$

where M is the number of classifiers, N is the total number of testing samples, K is the number of voting types and $p_{(n,k)}$ is the percentage of votes for label k in the n^{th} testing samples. For each testing example, if different machine learning methods vote the same, I will obtain an entropy of 0. So, the index is 1, giving 100% confidence. If each classifier vote for a different label, I will obtain an entropy of $\log(M)$. With normalization and subtraction in (3.1), the index is 0 or 0%, showing that I do not have confidence in this estimate.

One big advantage of the proposed framework is the use of different ML models to obtain an improved prediction. Therefore, more advanced ML models can be embedded to the framework to improve the final performance, including the proposed Unsupervised Learning model and Semi-supervised Learning model in the following sections.

3.3.0.1 Numerical Result

3.3.0.1.1 Settings

To validate the results, I use the simulation tool of Positive Sequence Load Flow (PSLF) (General Electric Energy Consulting 2018) software with high-grade dynamic simulations. When running PSLF, I consider faults such as line trip, three-phase short circuit, and single-line to ground fault, etc. The Illinois 200-bus system (Engineering Texas A&M University 2016a), known as ACTIVSg200 case, is utilized to demonstrate the result.

With simulated data, I conduct 5-fold cross validation and utilize the test accuracy to evaluate the results. Finally, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), Naive Bayes (NB), Decision Tree (DT), and the ensemble method (Hybrid) are conducted for comparisons.

3.3.0.1.2 Results of Test Accuracy for Event Identification

In this subsection, I show the benchmark of supervised learning with the hybrid machine learning process. Specifically, I build the learning model as a multi-label classification model. In one case, 6 line trips are conducted, namely the lines (25, 64), (42, 44), (44, 200), (172, 180) and (172, 199). All the experiments are conducted under multiple loading conditions to mimic the reality.

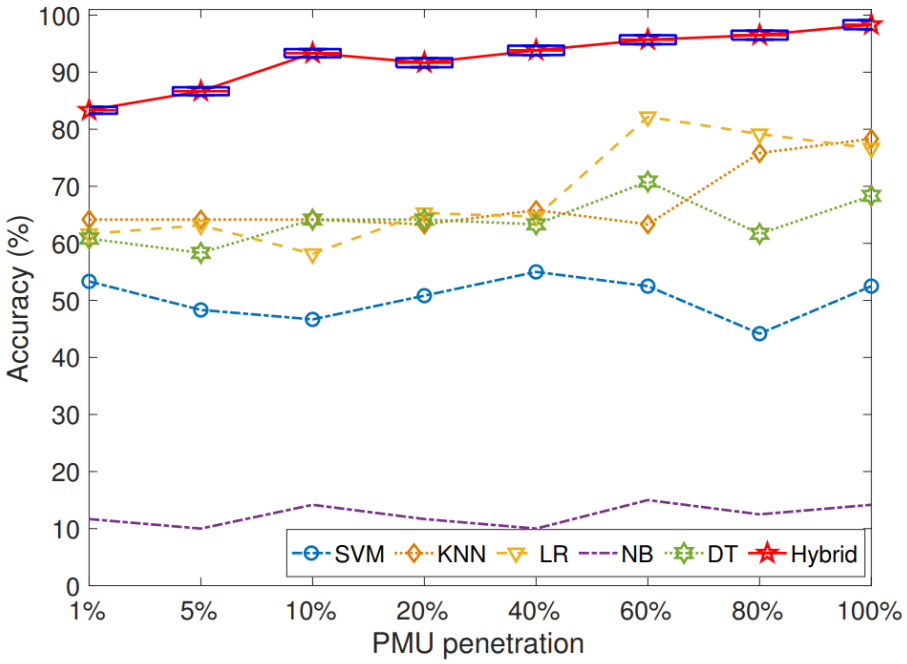


Figure 10. The comparison of individual SL mehtods and the proposed ensemble method.

To test the robustness of the supervised learning method against PMU number, I consider different PMU penetration levels of the grid. For example, 50% on the x-coordinate of Fig. 10 means that half of the buses are equipped with PMUs. For each penetration level, I randomly select the PMUs and use the selected PMU variables to create a feature pool. Such pool is then used to conduct the feature selection. After the feature extraction, the final features are vectorized into one sample vector. I collect 100 vectors for each event and train them through 5 machine learning methods and the hybrid method.

I plot the average accuracy of 5 machine learning methods and the hybrid machine learning method with respect to the penetration level of PMUs. The result is shown in Fig. 10. I can find that the accuracy varies among different machine learning methods and the penetration level of PMUs does not affect too much on different machine learning methods. Furthermore, the proposed hybrid machine learning method (i.e., the ensemble method) outperforms other supervised learning methods and its entropy based variance is relatively small under different PMU penetration levels. This result shows that the hybrid machine learning method successfully reduces the biases from different leaner and provides a confidence index for system operators.

3.4 Unsupervised Event Identification with Physics-guided Labeling

In this section, I tackle Problem 2 in Section 3.2. Specifically, I carefully study the different physical behaviors of the system under different events. Then, I propose physics-guided labeling for unlabeled data by linking the event severity to the cluster number and compactness in the feature space.

3.4.1 Event Type Identification via Clustering and Compactness

Since there are no labels indicating the event types, the unsupervised learning methods use two stages to narrow down the possible event type with physical understanding. In the first stage, I conduct Principal Component Analysis (PCA) to form clusters and focus on the most important features. By counting the cluster numbers, I can roughly classify the possible event types. For example, in power systems, line trip and generator trip have 2 clusters and the line faults have 3 clusters. In the second stage, I evaluate the compactness of normal and event data distribution as different events have different levels of severity. Such a two-stage quantification helps us obtain the event type estimation.

3.4.1.1 Principal Component-Based Data Pre-Processing

The event severity depend on the dynamic process of an event. Therefore, different events will have different dynamic deviations due to different grid parameters in the dynamic process. For example, Fig. 11 presents the dynamic voltage magnitude signals in one node. To capture this dynamic deviation, the event data matrix $\boldsymbol{X}$ defined in Section 3.2 should capture the pre-event, during-event, and post-event intervals together.

For extracting the most important features for the two-stage event type identification, I utilize PCA to find the major-variance directions. Without loss of generality, I assume $\boldsymbol{X}$ is centralized, leading to the direct application of the singular value

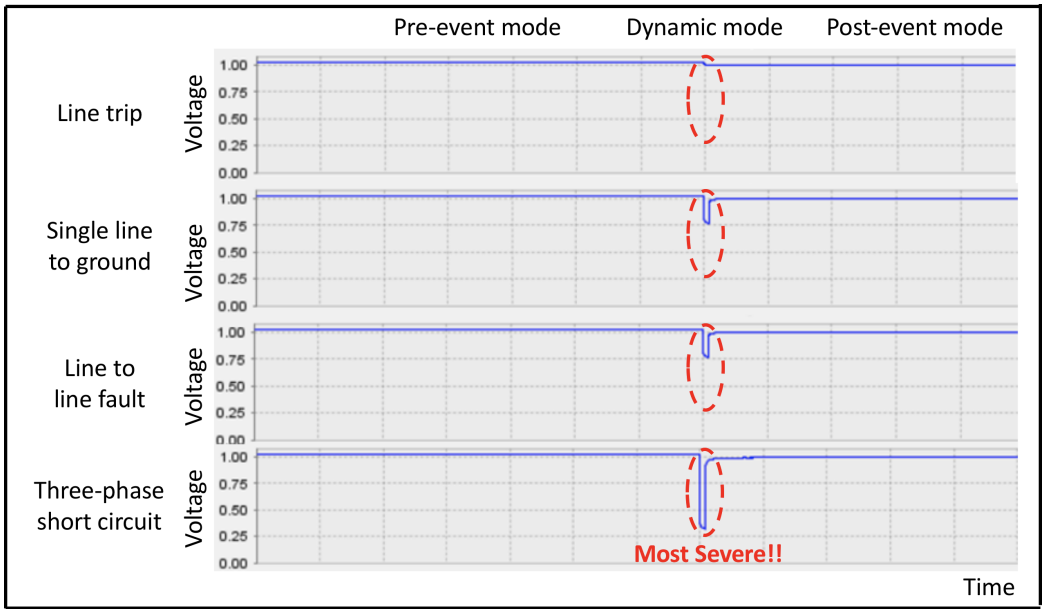


Figure 11. PSLF simulations for different events.

decomposition (SVD) for PCA:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\top}, \boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}. \tag{3.2}$$

The columns of $\boldsymbol{V}$ represent principal components and I can project the original matrix into the PCA-based space to obtain data matrix $\boldsymbol{Y}$. For example, Fig. 12 visualizes the top three principals' data points in $\boldsymbol{Y}$, which correspond with the first three columns in $\boldsymbol{Y}$. Green nodes represent the pre-event mode data; orange nodes represent the dynamic data and blue nodes represent the post-event mode data. The events include line trip, generator trip, single-phase-to-ground fault, phase-to-phase fault, and three-phase fault. From the figure, I observe the following patterns.

- Pattern 1: line trip and generator trip have 2 clusters and other line faults have 3 clusters, where the 3^{rd} cluster lies far away from the 1^{st} and 2^{nd} clusters in the horizontal plane, spanned by the 1^{st} and 2^{nd} components.

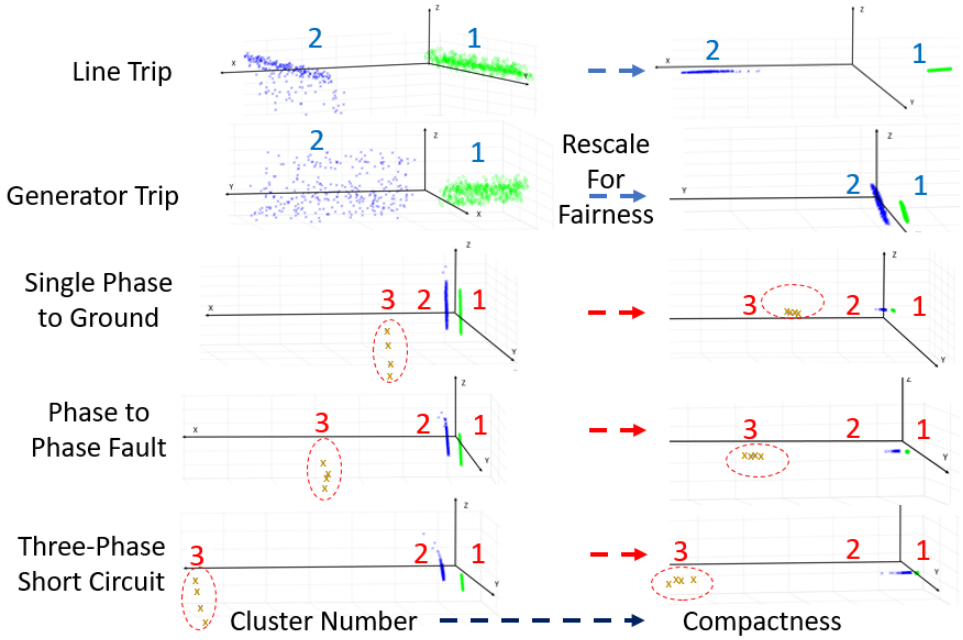


Figure 12. PCA and rescaling for event type differentiation.

- Pattern 2: the data distribution has different compactness, including the distances between different clusters and the density of each cluster.

3.4.1.2 Stage 1: Clustering According to Dynamic Modes

Pattern 1 suggests that the dynamic deviation for line faults and the post-event mode deviation for line trip or generator trip account for the maximum variance of $\boldsymbol{X}$, i.e., the deviation in the horizontal plane. Since line trip and generator trip come from switching off a breaker, they have relatively small transient processes in Fig. 11. Therefore, the 3^{rd} cluster does not exist in Fig. 12. Furthermore, since different events have different deviations, the compactness of data is different with respect to the deviation, leading to pattern 2.

To obtain clusters in the PCA-based space, I measure the distance between data points. However, the weights for different principals are different in terms of distance calculation. For example, in Fig. 12, the cluster under normal operation in the line faults stretches mostly in the 3^{rd} component, but this disturbs the aim to capture the dynamic deviation that mainly lies in the 1^{st} component. In this sense, the length in the horizontal plane should have a bigger weight, since horizontal plane represents the most data variance due to the first and second principles. Therefore, a rescaling is introduced to weight points in $\boldsymbol{Y}$ matrix:

$$\Phi(\boldsymbol{Y})(a) = \lambda_a \boldsymbol{Y}(a),\ 1 \leq a \leq A, \tag{3.3}$$

where $\boldsymbol{Y}(a)$ and $\Phi(\boldsymbol{Y})(a)$ are the a^{th} column of $\boldsymbol{Y}$ and $\Phi(\boldsymbol{Y})$ matrices, respectively. λ_a is the corresponding eigenvalue for the a^{th} feature. The mapped data is shown in Fig. 12, I find that the cluster extension in the 1^{st} component is emphasized.

Subsequently, the stage 1 clustering is introduced to find the number of clusters in PCA-based space. Specifically, I utilize a fixed benchmark $\boldsymbol{X}^B$ that contains normal states (e.g., voltage magnitude measurements in PMUs) under various loading conditions. Then, I combine $\boldsymbol{X}^B$ with the dynamic segment and post-event segment of an event $\boldsymbol{X}^E$ (obtained via change point detection) into the training data matrix $\boldsymbol{X} = [\boldsymbol{X}^B, \boldsymbol{X}^E]^T$. The rescaling mapping transforms the $\boldsymbol{X}$ matrix into $\Phi(\boldsymbol{Y}) = [\Phi(\boldsymbol{Y}^B), \Phi(\boldsymbol{Y}^E)]^T$ matrix. Since $\Phi(\boldsymbol{Y}^B)$ represents the normal cluster, I only need to do clustering for $\Phi(\boldsymbol{Y}^E)$. Thus, a top-down hierarchical clustering is introduced to find the number of clusters and divide the event type.

Basically, the algorithm views initial points in $\Phi(\boldsymbol{Y}^E)$ as one big cluster and recursively splits the cluster(s) via a k-means clustering where $k = 2$. For the i^{th} iteration, I obtain $N^i \leq 2^i$ clusters (some clusters may stop division in the previous iterations). For the j^{th} $(1 \leq j \leq N^i)$ cluster that can be further divided into 2

sub-clusters with centers $\mathbf{c}_{j_1}$ and $\mathbf{c}_{j_2}$, I introduce the stopping criterion:

$$\frac{\max\{||\mathbf{c}_{j_1} - \mathbf{c}_b||, ||\mathbf{c}_{j_2} - \mathbf{c}_b||\}}{\min\{||\mathbf{c}_{j_1} - \mathbf{c}_b||, ||\mathbf{c}_{j_2} - \mathbf{c}_b||\}} \geq \epsilon_1, \tag{3.4}$$

where $||\cdot||$ represents the Euclidean distance, $\mathbf{c}_b$ is a vector representing the cluster center under normal operations and ϵ_1 is a constant threshold. (3.4) illustrates that if the distances from j_1^{th} and j_2^{th} cluster centers to the normal centers are close (i.e., these 2 clusters have similar severity from the normal operation), the algorithm will stop dividing the j^{th} cluster. With this criterion, all the clusters are grouped via the severity level: the generator and line trip have only 1 severity level (post-event mode) I will obtain 2 clusters (with the normal cluster). However, the line faults have 2 severity level (dynamic mode and post-event mode) and I will obtain 3 clusters.

3.4.1.3 Stage 2: Compactness Classification via Severity Levels

From Fig. 11, I learn that the three-phase fault has the severest signal changes relatively. So, such information can be used to refine the event type further. Such a severity can be interpreted as compactness of data distribution in Fig. 12. Quantifying the compactness of $\Phi(\boldsymbol{Y})$, I implement a weighted-vectorization approach. Specifically, top 3 columns, in $\Phi(\boldsymbol{Y})$, are vectorized into one vector $\mathbf{z} = [\mathbf{z}^B, \mathbf{z}^E]^T$. Since the dynamic process is short, the number of points in the 3^{rd} cluster in Stage 2 is small (e.g., 4 points in Fig. 12). Thus, points in the 3^{rd} cluster should account for more weights to emphasize the dynamic deviation and avoid being treated as insignificant outliers. Therefore, I reweight the components in $\mathbf{z}$ based on the number of points in the same cluster: $\phi(\mathbf{z})(j) = \mathbf{z}(j)/N_i, i \in \{1, 2, 3\}$. If the j^{th} component $\mathbf{z}(j)$ in $\mathbf{z}$ belongs to the i^{th} cluster, I divide $\mathbf{z}(j)$ by the number of points in the i^{th} cluster, i.e., N_i. Finally, I do normalization to $\phi(\mathbf{z})$ to limit the range in $[0, 1]$.

The above process from X to $\phi(\mathbf{z})$ can be viewed as a physical knowledge guided mapping, in which the event is projected differently according to different severity levels. Thus, I utilize $\phi(\mathbf{z}_i)$ to denote the projection for the i^{th} event.

With these projections, I denote $S = \{S_1, S_2, \cdots, S_L\}$ as the event type set, where L is the number of event types. Then, an off-line k-means ($k = L$) clustering can be employed to find the event centers: $\text{argmin}_S \sum_{l=1}^{L} \sum_{\phi(\mathbf{z}_l) \in S_l} ||\phi(\mathbf{z}_l) - \boldsymbol{\mu}_l||^2$, where $\boldsymbol{\mu}_j$ is the centroid of event S_j. This optimization can be solved via Lloyd's algorithm.

The following observations can be utilized in the training and validation process: 1) the value of k can be denoted as a fixed number according to the number of common events in the grid, and 2) the number of points in the cluster represents the event frequency. For example, line trip is the most common event in this paper, and therefore, the cluster that contains the most points should be the line trip.

Finally, online testing is implemented in real time. For L event types, I calculate the distance between the new data $\phi(\mathbf{z}_i)$ to the event cluster center $\boldsymbol{\mu}_j$, $d(i,j) = ||\phi(\mathbf{z}_i) - \boldsymbol{\mu}_j||,\ 1 \leq j \leq L$. Thus, the new event will be classified into the event cluster that has the minimal distance.

3.4.2 Event Localization

3.4.2.1 Change-Point-based Localization

For event localization, one idea is to see which sensor has the most significant change. Then, such a sensor can infer the area that an event is likely to happen. For example, I can use the result from change-point detection to rank the relative change at different sensors. Let $\hat{t}_1$ represent the event changes from normal mode. Thus, I

use the criterion below to select the sensors with large changes at $\hat{t}_1$:

$$\sum_{n=1}^{N} \left| \frac{X(\hat{t}_1 + n, a) - X(\hat{t}_1 - n, a)}{N \cdot X(\hat{t}_1 - n, a)} \right| \geq \epsilon_2, \ \forall 1 \leq a \leq A, \tag{3.5}$$

where N points before and after $\hat{t}_1$ are used to calculate the percentage change and ϵ_2 is a constant. In a $A \times 1$ index vector **ind**, if a^{th} variable is selected, the a^{th} component is 1, and otherwise 0. I denote $\mathbf{ind}_i$ as the index vector of the i^{th} event. Thus, the event location is linked to the sensor locations.

3.4.2.2 Increase Robustness Via Grouping sensors

While the change-point-based localization is a good starting point for studying the localization problem, it is non-robust due to factors such as imbalanced system setup and increasing intermittent renewables in the power grid. This makes sensor measurements with large variance unnecessary locate near the event location. For example, if generators are far away from the load area in a loopy structure, the measurement at the far away generator may change significantly during a line trip event at the load area. This makes change-point-based method prone to error, calling for robustness.

For this purpose, I combine the event vector $\phi(\mathbf{z}_i)$ with the sensor-index vector $\mathbf{ind}_i$ to obtain $\psi(\mathbf{z}_i) = [\phi(\mathbf{z}_i), \lambda \mathbf{ind}_i]^T$, where λ is a penalty constant. $\phi(\mathbf{z}_i)$ is employed to avoid the case when two events are close to each other so that they share the same index vector **ind**. Hierarchical clustering can be used to group sensors in the same area with similar behaviors. This means that there could be some relative far location with large change, however more sensors at the fixed event location will join the vote according to the hierarchy of overall system behavior.

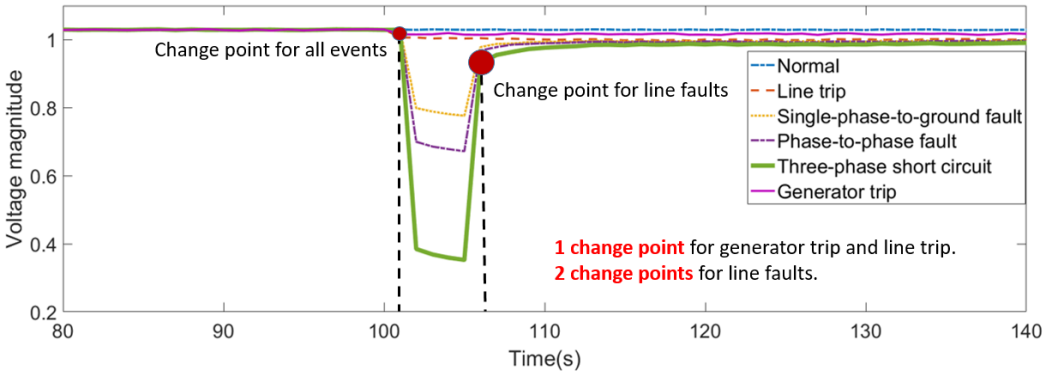


Figure 13. The results of change point detection.

In such a clustering method, one needs to choose linkage criterion. For example, Euclidean distance $d(i,k) = ||\psi(\mathbf{z}_i) - \psi(\mathbf{z}_k)||$ can be used among the i^{th} and k^{th} points. Such grouping individual members, one can start to connect groups, e.g., group F and group C: $d(F,C) = \frac{1}{N_F \cdot N_C} \sum_{f \in F} \sum_{c \in C} d(f,c)$, where N_F and N_C are the numbers of points in F and C. I utilize the following stopping criterion in hierarchical clustering: $d(F,C) > \epsilon_3$, where ϵ_3 is a constant. If the linkage criterion between F and C is large, the compactness helps to distinguish two groups.

Notably, the change-point detection and hierarchical clustering should be used even for historical sensor data. So, when a new event comes, I detect it and calculate the distance between this event to each cluster centers and label the new event via the minimal distance. If several distances have similar value, the new event is regarded as an unknown event location that needs further query from the utility.

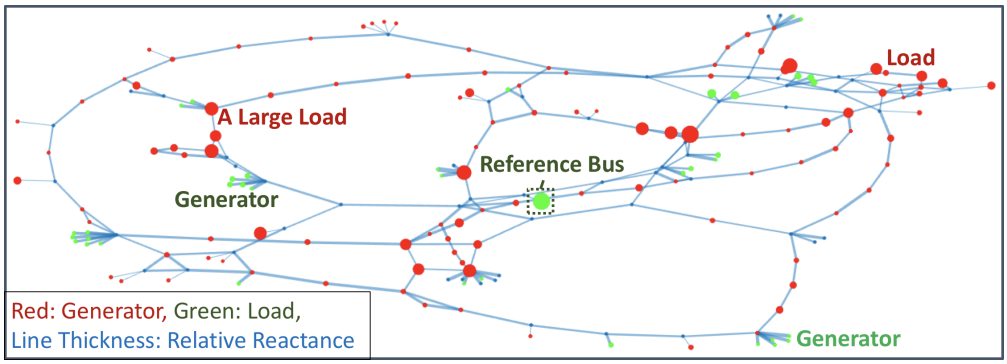


Figure 14. I display the topology of Illinois 200-bus system for PSLF.

3.4.3 Numerical Result

3.4.3.0.1 Settings

Numerical experiments are conducted using Positive Sequence Load Flow (PSLF) tool to generate synthetic measurements. The Illinois 200-bus system (Fig. 14), known as ACTIVSg200 case (Engineering Texas A&M University 2016a), is utilized to demonstrate the results. In the following experiments, I test 5 events: 1) line trip, 2) single-phase-to-ground fault, 3) phase-to-phase fault, 4) three-phase fault, and 5) generator trip.

3.4.3.0.2 Results of Event Type Differentiation

In this subsection, I conduct line trip, line faults and generator trip at 80 lines and 20 generators. Then, I test voltage magnitude data in stage 1 clustering to find the number of clusters. For each event type, I average the number of clusters with respect to different locations. Different PMU penetrations are considered and the average

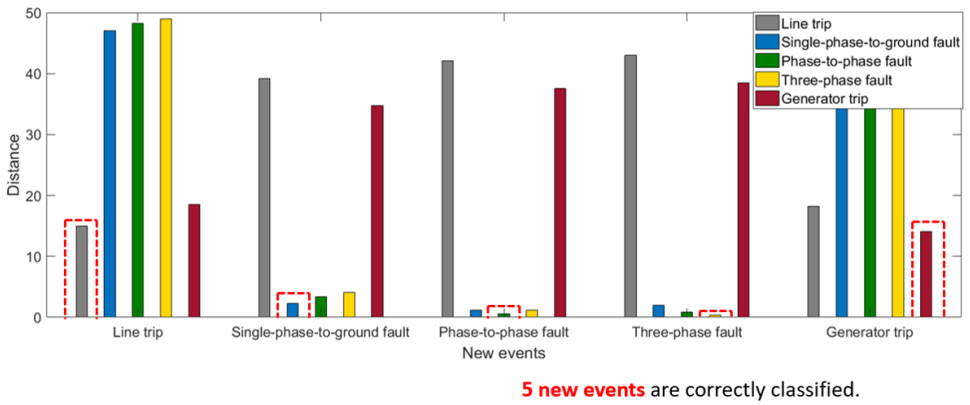


Figure 15. Distances between 5 new events and the cluster centers.

number of clusters is shown in Table 6. I find the proposed top-down clustering is robust for different PMU penetrations as well as different locations.

PMU penetration(%)	10	20	40	60	80	100
Line trip	2	2	2	2	2	2
Generator trip	2	2	2	2	2	2
Single-phase-to-ground fault	3	3	3	3	3	3
Phase-to-phase fault	3	3	3	3	3	3
Three-phase fault	3	3	3	3	3	3

Table 6. The average number of clusters after clustering.

Subsequently, I input the normalized event data to k-means clustering where $k = 5$ and obtain 5 clustering centers $c_1, \cdots, c_5$. Then, I randomly conduct 5 new events (1 for each event type) for 10 times, the general performance shows that I can classify each event type correctly. A typical case is shown in a bar graph in Fig. 15. The distance in the dotted red block is the smallest distance between new events and cluster centers. Thus, the 5 new events are correctly classified.

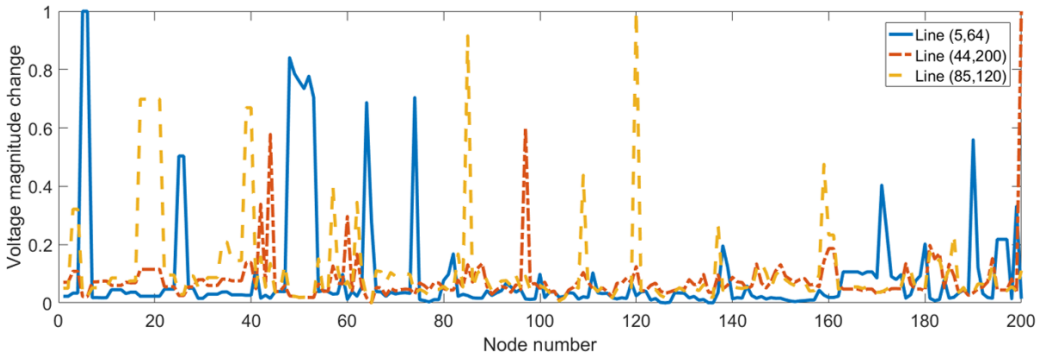


Figure 16. 200 buses' voltage magnitude change before and after a change point.

3.4.3.0.3 Results of Event Location

In this subsection, I first study the PMU measurement change (voltage magnitude) in (3.5). I consider single-phase-to-ground fault at line $(5, 64)$, $(44, 200)$ and $(85, 120)$. 200 buses' voltage magnitude change before and after a change point is calculated and normalized in Fig. 16. For these 3 events, they share different peaks representing selected PMUs.

Then, I utilize different PMU penetrations and voltage magnitude data of events in 80 lines to do the third phase bottom-up clustering. 80 clusters are correctly found. Then, I test 20 new data points that share the same location and type of the 80 clusters but have different loading conditions. Fig. 17 illustrates an example of the distance matrix. It is the heat map of the distance matrix. The gray sub-block represent values that are close to 0 and I can find that each new event is correctly classified into the corresponding cluster.

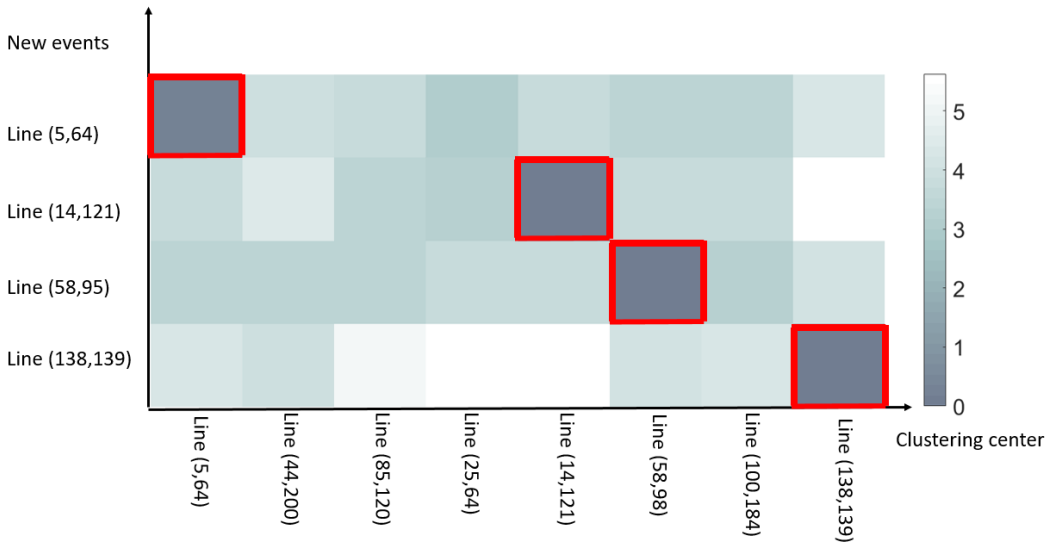


Figure 17. Heat map of distance matrix between new points and cluster centers.

3.5 Semi-supervised Event Identification with Fast Tensor Computation

In this section, I tackle Problem 3 in Section 3.2. To integrate event data compression and machine learning in one tensorized framework, I first introduce basics of tensor algebra (Kolda and Bader 2009) and the corresponding notations. To summarize, Table 7 presents the basic notations for different types of variables and operations.

3.5.1 Tensor Notations and Preliminaries

Multi-mode data can be stored in the so-called tensor (Lock 2018), the multi-dimensional arrays. The number of dimensions for a tensor is referred to as order. For example, scalar (0-order tensor), vector (1-order tensor) and matrix (2-order tensor). Then, for a D-order tensor $\mathcal{X}$, $I_1 \times I_2 \times \cdots \times I_D$ are denoted as the dimensions, i.e., $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ or $\mathcal{X}^{I_1 \times I_2 \times \cdots \times I_D}$, where I_i $(\forall 1 \leq i \leq D)$ is the dimensionality of

Table 7. Overview of tensor notations and operators.

Notation	Interpretation
α	scalar
$\mathbf{a}$	vector
$\mathbf{A}$	matrix
$\mathcal{X}$	tensor, set, or space
$\mathbf{X}_{(n)}$	unfolding of tensor $\mathcal{X}$ along mode n
$\circ$	outer product
$\times_n$	mode-n product
$\otimes$	Kronecker product
$\|\cdot\|_2$	l_2 norm of a vector
$\|\cdot\|_F$	l_2 Frobenius norm of a matrix or a high-order tensor

the i^{th} dimension of $\mathcal{X}$. There are many types of operations for a tensor like folding, unfolding, product, etc. This subsection provides some operations used in the proposed method.

mode-n unfolding of a tensor. A tensor can be unfolded to a matrix, a process that is also known as matrization. Specifically, for $\mathcal{X}^{I_1 \times I_2 \times \cdots \times I_D}$, one can unfold it along the n-dimension (mode) to obtain $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i=1, i \neq n} I_i}$. Mathematically, the result is:

$$\mathcal{X}(i_1, i_2, \cdots, i_D) = \boldsymbol{X}_{(n)}(i_n, j),$$

$$j = 1 + \sum_{k=1, k \neq n}^{D} (i_k - 1) J_k, \;\; J_k = \prod_{m-1, m \neq n}^{k-1} I_m,$$

where $\mathcal{X}(i_1, \cdots, i_n)$ is denoted as the $(i_1, \cdots, i_n)^{th}$ entry of tensor $\mathcal{X}$.

n-mode product. For a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ and a matrix $\boldsymbol{U} \in \mathbb{R}^{K \times I_n}$, the n-mode product is denoted as:

$$(\mathcal{X} \times_n \mathbf{U})(i_1, \cdots, i_{n-1}, k, i_{n+1}, \cdots, i_D)$$

$$= \sum_{i_n=1}^{I_n} \mathcal{X}(i_1, i_2, \cdots, i_D) \mathbf{U}(k, i_n),$$

where $\mathcal{X} \times_n \mathbf{U} \in \mathbb{R}^{I_1 \times \cdots I_{n-1} \times K \times I_{n+1} \cdots \times I_N}$ is a tensor.

Tensor Tucker Decomposition. For a D-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$, one key research topic is to find the approximation using a set of small tensors. For example, PMU data is of high volume and low rank (M. Liao et al. 2018). Thus, the low-rank approximation is preferred to efficiently represent the PMU data and remove the redundant information. The target can be achieved via tensor decomposition. Specifically, the so-called Tucker decomposition is (Kolda and Bader 2009):

$$\begin{aligned}\mathcal{X} &\approx \mathcal{G} \times_1 \boldsymbol{U}_1 \times_2 \boldsymbol{U}_2 \cdots \times_D \boldsymbol{U}_D \\ &\approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_J=1}^{R_D} \mathcal{G}(r_1, \cdots, r_D) \mathbf{u}_1^{r_1} \circ \cdots \circ \mathbf{u}_D^{r_D},\end{aligned}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$ is a core tensor in the factorization, and $\boldsymbol{U}_i \in \mathbb{R}^{I_i \times R_i}$ is a base matrix along mode i. $\mathbf{u}_i^{r_i}$ is the r_i^{th} column of $\boldsymbol{U}_i$ and $\circ$ is the outer product. Note $\mathcal{G}$ is different from the definitions in Chapter 2 with a slight abuse of notation.

Finally, the Tucker decomposition can be rewritten in a matrix format:

$$\boldsymbol{X}_{(n)} = \boldsymbol{U}_n \boldsymbol{G}_{(n)} (\boldsymbol{U}_D \otimes \cdots \otimes \boldsymbol{U}_{n+1} \otimes \boldsymbol{U}_{n-1} \otimes \cdots \otimes \boldsymbol{U}_1)^\top,$$

where $\otimes$ is the so-called Kronecker product and $\top$ represents the matrix transpose. In summary, the introduced tensor operations lay foundations for our integrated model with certain physical interpretations. Specifically, tensor decomposition provides efficient feature extraction while maintaining certain physical structures in the core tensor $\mathcal{G}$. Further, tensor unfolding converts $\mathcal{G}$ to vectors that can be input to a classifier, which enables an end-to-end model of decomposition and classification.

3.5.2 Proposed Model KTDC-Se: Kernelized Tensor Decomposition and Classification with Semi-supervision

The design of the classification model can be diversified. However, existing work suffers a key challenge of biased selections of data compression and event identification without proper integration. In this section, I design an end-to-end model that makes full use of tensor structure to achieve fast computations, physical interpretations, high capacity with non-linear feature extractions, and high accuracy under semi-supervision.

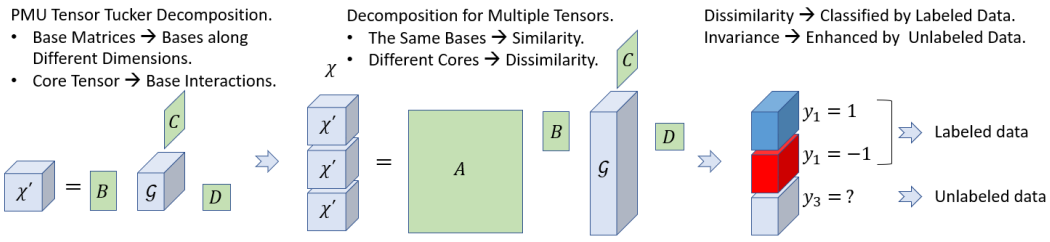


Figure 18. The motivation of the KTDC-Se model.

For an efficient model, I need to remove the redundancy in sensor measurements. Section 3.5.1 illustrates that tensor is a natural container of high-dimensional data and tensor Tucker decomposition is an excellent approach to uncover the cross-dimension correlations. However, it is still unclear how I can design an efficient model to remove redundancy and capture event information for different event tensors, and how I can guarantee the model robustness by tackling some labeled and rich unlabeled tensors.

For a detailed design, I start with the tensor decomposition. For each tensor $\mathcal{X}'$, the left part of Fig. 18 visualizes the process of a Tucker decomposition into base matrices $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ and a core tensor $\mathcal{G}$. $\mathcal{G}$ can maintain the structure as the tensor $\mathcal{X}'$, leading to specific physical interpretations. Specifically, the base matrices can be viewed as the bases along different dimensions, and the core tensor $\mathcal{G}$ represents the

interactions among these bases (Sidiropoulos et al. 2017). Thus, I can assume bases for different tensors are similar as long as the number of bases is sufficiently enough. In contrast, the interaction tensor $\mathcal{G}$ contains discriminative event information.

Then, to maximally remove the redundancy, I directly keep the same bases $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ for different tensors during the decomposition, shown in the middle part of Fig. 18. Namely, I utilize a direct Tucker decomposition for a 4-D total event tensor $\mathcal{X}$. Then, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are naturally kept to be the same. Further, I want the core tensor $\mathcal{G}$ to contain distinguished event information. Therefore, I employ the event labels to conduct a supervised learning-based classification for dissimilarity maximization. The above procedure is for labeled tensors. For unlabeled tensors, only the decomposition procedure is implemented to increase the model robustness to different loading conditions. The concrete mathematical model of joint optimization is formulated in the following description.

In the semi-supervised learning setting, the 4-D total event tensor $\mathcal{X}$ in Section 3.2 contains all labeled and unlabeled data. Mathematically, I implement the Tucker decomposition for $\mathcal{X}$ as follows:

$$\begin{aligned} \mathcal{X} &\approx \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C} \times_4 \boldsymbol{D} \\ &\approx \sum_{r_1=1}^{N} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \sum_{r_4=1}^{R_4} \mathcal{G}(r_1, r_2, r_3, r_4) \mathbf{a}^{r_1} \circ \mathbf{b}^{r_2} \circ \mathbf{c}^{r_3} \circ \mathbf{d}^{r_4}, \end{aligned} \tag{3.6}$$

where $\mathcal{G} \in \mathbb{R}^{N \times R_2 \times R_3 \times R_4}$ is the core tensor, i.e., the compressed tensor with small information redundancy. The matrices to scale the core tensors are $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, $\boldsymbol{B} \in \mathbb{R}^{T \times R_2}$, $\boldsymbol{C} \in \mathbb{R}^{L \times R_3}$, and $\boldsymbol{D} \in \mathbb{R}^{M \times R_4}$. $\boldsymbol{a}^{r_1}$, $\boldsymbol{b}^{r_2}$, $\boldsymbol{c}^{r_3}$, and $\boldsymbol{d}^{r_4}$ are the r_1^{th}, r_2^{th}, r_3^{th}, and r_4^{th} columns of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$, respectively. $R_2 < T$, $R_3 < L$, and $R_4 < M$ are the pre-defined dimensions of the reduced tensors to achieve the information compression.

In the decomposition of Equation (3.6), the first dimension is fixed of the core tensor $\mathcal{G}$ to be N so that the decomposition can still bring N features to represent different tensors. Furthermore, the decomposition can be rewritten as:

$$\boldsymbol{X}_{(1)} \approx \boldsymbol{A}\boldsymbol{G}_{(1)}(\boldsymbol{D} \otimes \boldsymbol{C} \otimes \boldsymbol{B})^{\top}, \tag{3.7}$$

where $\boldsymbol{X}_{(1)} \in \mathbb{R}^{N \times (T \cdot L \cdot M)}$ and $\boldsymbol{G}_{(1)} \in \mathbb{R}^{N \times (R_2 \cdot R_3 \cdot R_4)}$ represent the mode-1 unfolding matrix of tensors $\mathcal{X}$ and $\boldsymbol{G}$, respectively. Clearly, columns in $\boldsymbol{G}_{(1)}$ represent the compressed features that can be utilized for the classifier training. Under semi-supervision, I utilize the labeled tensors with labels for the classification. Then, I propose a joint decomposition-classification model.

$$\begin{aligned} \min_{\boldsymbol{G}_{(1)}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{w}, b} J = & \underbrace{||\mathcal{X} - \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C} \times_4 \boldsymbol{D}||_F^2}_{J_1,\ \text{Reconstruction Loss}} \\ & + \gamma_1 \underbrace{l(\boldsymbol{E}\boldsymbol{G}_{(1)} \cdot \boldsymbol{w} + b, \boldsymbol{y})}_{J_2,\ \text{Classification Loss}} + \gamma_2 \underbrace{||\boldsymbol{w}||_2^2}_{J_3,\ l_2\ \text{Norm}}, \end{aligned} \tag{3.8}$$

where J, J_1, J_2, and J_3 denote the total loss, reconstruction loss, classification loss and regularization terms, respectively. $\boldsymbol{E} = [\boldsymbol{I}^{H \times H}, \boldsymbol{0}^{H \times (N-H)}] \in \mathbb{R}^{H \times N}$ denotes a selection matrix to select the first H feature instances (i.e., the instances with labels) in $\boldsymbol{G}_{(1)}$ for the classification. $||\cdot||_F$ and $||\cdot||_2$ denote the Frobenius norm and the l_2 norm, respectively. $l(\cdot, \cdot)$ represents the classification loss function. The hinge loss of Support Vector Machine (SVM) is considered in this paper, i.e., $l(\boldsymbol{E}\boldsymbol{G}_{(1)} \cdot \boldsymbol{w}, \boldsymbol{y}) = \sum_{i=1}^{H}[1 - y_i(\boldsymbol{f}_i^{\top}\boldsymbol{w} + b)]_+$, where $\boldsymbol{f}_i \in \mathbb{R}^{(R_2 \cdot R_3 \cdot R_4) \times 1}$ is the i^{th} instance of the transformed features and y_i is the i^{th} label in $\boldsymbol{y}$. Namely, $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_H]^{\top} = \boldsymbol{E}\boldsymbol{G}_{(1)}$. Further, the hinge loss is defined as $[1 - t]_+ = \max(0, 1 - t)^p$. Usually, one can treat $p = 1$ or $p = 2$ for l_1- or l_2-SVM (L. He et al. 2017), respectively. γ_1 and γ_2 are positive hyper-parameters to reweight the three terms in Equation (3.8).

In above descriptions, I successfully merge the data reduction and machine learning model into one optimization under semi-supervision. However, many measurements have non-linear correlations. It is challenging to add non-linearity due to the computational cost. For example, adding sigmoid or polynomial functions to the loss function l in Equation (3.8) significantly increases the computations. Thus, I propose to utilize kernel function to lift the data to high-dimensional or even infinite-dimensional feature space, and the kernel trick can enable the calculation to happen in the original data space, which easily maintains efficient calculations (Kung 2014).

Specifically, due to the Representer theorem (Schölkopf, Herbrich, and Smola 2001), the inner product of the classification model can be rewritten as $\boldsymbol{f}^\top \boldsymbol{w} = \sum_{i=1}^{H} \alpha_i k(\boldsymbol{f}, \boldsymbol{f}_i)$, where $k(\cdot,\cdot)$ is the kernel function and $\boldsymbol{f} \in \mathbb{R}^{(R_2 \cdot R_3 \cdot R_4)\times 1}$ is a variable in the feature space. Based on this equation, the kernelized learning process is:

$$\begin{aligned} \min_{\boldsymbol{G}_{(1)},\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D},\boldsymbol{\alpha},b} J = & \underbrace{||\mathcal{X} - \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C} \times_4 \boldsymbol{D}||_F^2}_{J_1,\ \text{Reconstruction Loss}} \\ & + \gamma_1 \underbrace{\sum_{i=1}^{H} \left[1 - y_i(\sum_{j=1}^{H} \alpha_j k(\boldsymbol{f}_i, \boldsymbol{f}_j) + b)\right]_+}_{J_2,\ \text{Classification Loss}} \\ & + \gamma_2 \underbrace{\sum_{i=1}^{H}\sum_{j=1}^{H} \alpha_i \alpha_j k(\boldsymbol{f}_i, \boldsymbol{f}_j)}_{J_3,\ \text{Regularization}}, \end{aligned} \tag{3.9}$$

where $\boldsymbol{\alpha}$ is the vector of all α_is. Eventually, I re-emphasize the nice properties of KTDC-Se by (1) using tensors to capture multi-dimensional correlations, (2) proposing a joint model for decomposition and classification, (3) introducing kernels for non-linear features, and (4) conveniently tackling both labeled and unlabeled data.

3.5.3 Learning Algorithm of KTDC-Se

The optimization in Equation (3.9) is non-convex. Thus, an alternative optimization algorithm is proposed to update the individual variable in the optimization while fixing other variables, i.e., the so-called coordinate descent method. This method is prevailing in the domain of tensor learning due to its efficiency and good convergence property (Kolda and Bader 2009; L. He et al. 2017). Further, to calculate the gradient and avoid the non-differentiable scenario, the l_2 SVM (L. He et al. 2017) is utilized, i.e., $p = 2$ in the loss function $l(\cdot, \cdot)$. Then, to update each variable, KTDC-Se only needs to calculate the gradients with respect to every single variable and utilize the gradient descent for updating. Thus, the calculation of the gradients is shown as follows.

Gradient of $\boldsymbol{A}$. Based on matrix format of Tucker decomposition and for the convenience of later derivations, I define $\boldsymbol{H}_1 = \boldsymbol{G}_{(1)}(\boldsymbol{D} \otimes \boldsymbol{C} \otimes \boldsymbol{B})^\top$. Then, the gradient of the loss function in Equation (3.9) is calculated with respect to $\boldsymbol{A}$. Mathematically, the gradient is:

$$\begin{aligned} \nabla_{\boldsymbol{A}} J = \nabla_{\boldsymbol{A}} J_1 &= \nabla_{\boldsymbol{A}} \mathrm{tr}\big((\boldsymbol{X}_{(1)} - \boldsymbol{A}\boldsymbol{H}_1)^\top \cdot (\boldsymbol{X}_{(1)} - \boldsymbol{A}\boldsymbol{H}_1)\big) \\ &= 2(\boldsymbol{A}\boldsymbol{H}_1\boldsymbol{H}_1^\top - \boldsymbol{X}_{(1)}\boldsymbol{H}_1^\top), \end{aligned} \tag{3.10}$$

where $\mathrm{tr}(\cdot)$ represents the operation to obtain the matrix trace.

Gradients of $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$. By symmetry, $\boldsymbol{H}_2 = \boldsymbol{G}_{(2)}(\boldsymbol{D} \otimes \boldsymbol{C} \otimes \boldsymbol{A})^\top$, $\boldsymbol{H}_3 = \boldsymbol{G}_{(3)}(\boldsymbol{D} \otimes \boldsymbol{B} \otimes \boldsymbol{A})^\top$, and $\boldsymbol{H}_4 = \boldsymbol{G}_{(4)}(\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A})^\top$ can be defined. Then, $\nabla_{\boldsymbol{B}} J$, $\nabla_{\boldsymbol{C}} J$, and $\nabla_{\boldsymbol{D}} J$ are calculated as follows:

$$\begin{aligned} \nabla_{\boldsymbol{B}} J &= 2(\boldsymbol{B}\boldsymbol{H}_2\boldsymbol{H}_2^\top - \boldsymbol{X}_{(2)}\boldsymbol{H}_2^\top), \\ \nabla_{\boldsymbol{C}} J &= 2(\boldsymbol{C}\boldsymbol{H}_3\boldsymbol{H}_3^\top - \boldsymbol{X}_{(3)}\boldsymbol{H}_3^\top), \\ \nabla_{\boldsymbol{D}} J &= 2(\boldsymbol{D}\boldsymbol{H}_4\boldsymbol{H}_4^\top - \boldsymbol{X}_{(4)}\boldsymbol{H}_4^\top). \end{aligned} \tag{3.11}$$

To update $\boldsymbol{G}_{(1)}$, the following gradients are separately derived. For the reconstruction loss, $\tilde{\boldsymbol{H}} = (\boldsymbol{D} \otimes \boldsymbol{C} \otimes \boldsymbol{B})^\top$ is denoted. Then, the gradient is:

$$\begin{aligned} \nabla_{\boldsymbol{G}_{(1)}} J_1 &= \nabla_{\boldsymbol{G}_{(1)}} \mathrm{tr}\big((\boldsymbol{X}_{(1)} - \boldsymbol{A}\boldsymbol{G}_{(1)}\tilde{\boldsymbol{H}})^\top \cdot (\boldsymbol{X}_{(1)} - \boldsymbol{A}\boldsymbol{G}_{(1)}\tilde{\boldsymbol{H}})\big) \\ &= 2(\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{G}_{(1)}\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top - \boldsymbol{A}^\top \boldsymbol{X}_{(1)}\tilde{\boldsymbol{H}}^\top). \end{aligned} \tag{3.12}$$

For the classification loss, $\hat{y}_i = \sum_{j=1}^{H} \alpha_j k(\boldsymbol{f}_i, \boldsymbol{f}_j) + b$ is denoted for simplification. Based on the chain rule, the gradient is:

$$\nabla_{\boldsymbol{f}_i} J_2 = \begin{cases} 2(\hat{y}_i - y_i) \sum_{j=1}^{H} \alpha_j \frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_j)}{\partial \boldsymbol{f}_i} \\ \\ +2\alpha_i \sum_{j \neq i}^{H} (\hat{y}_j - y_j) \frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_j)}{\partial \boldsymbol{f}_i}, & \text{if } y_i \hat{y}_i < 1 \ , \\ \\ \boldsymbol{0}, & \text{if } y_i \hat{y}_i \geq 1 \ . \end{cases} \tag{3.13}$$

To elaborate on the above equation, polynomial and Radial Basis Function (RBF) kernels are utilized as examples. For the polynomial kernel $k(\boldsymbol{f}_i, \boldsymbol{f}_j) = (\boldsymbol{f}_i^\top \boldsymbol{f}_j + c)^d$, where c is a constant and d is the degree of the polynomial function, then

$$\frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_j)}{\partial \boldsymbol{f}_i} = \begin{cases} d(\boldsymbol{f}_i^\top \boldsymbol{f}_j + c)^{d-1} \boldsymbol{f}_j, & \text{if } i \neq j \ , \\ \\ 2d(\boldsymbol{f}_i^\top \boldsymbol{f}_j + c)^{d-1} \boldsymbol{f}_i, & \text{if } i = j \ . \end{cases} \tag{3.14}$$

For the RBF kernel $k(\boldsymbol{f}_i, \boldsymbol{f}_j) = \exp(-\lambda ||\boldsymbol{f}_i - \boldsymbol{f}_j||_2^2)$, where λ is a positive constant:

$$\frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_j)}{\partial \boldsymbol{f}_i} = 2\lambda k(\boldsymbol{f}_i, \boldsymbol{f}_j)(\boldsymbol{f}_j - \boldsymbol{f}_i). \tag{3.15}$$

Recall that $\boldsymbol{F} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_H]^\top = \boldsymbol{E}\boldsymbol{G}_{(1)}$, $\nabla_{\boldsymbol{F}} J_2 = [\nabla_{\boldsymbol{f}_1} J_2, \nabla_{\boldsymbol{f}_2} J_2, \cdots, \nabla_{\boldsymbol{f}_H} J_2]^\top$ can be obtained. Thus, $\nabla_{\boldsymbol{G}_{(1)}} J_2 = [\nabla_{\boldsymbol{f}_1} J_2, \nabla_{\boldsymbol{f}_2} J_2, \cdots, \nabla_{\boldsymbol{f}_H} J_2, \boldsymbol{0}, \cdots, \boldsymbol{0}]^\top$.

For the Regularization term, the result is:

$$\nabla_{\boldsymbol{f}_i} J_3 = \alpha_i^2 \frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_i)}{\partial \boldsymbol{f}_i} + 2\alpha_i \sum_{j \neq i} \alpha_j \frac{\partial k(\boldsymbol{f}_i, \boldsymbol{f}_j)}{\partial \boldsymbol{f}_i}. \tag{3.16}$$

Similarly, $\nabla_{\boldsymbol{G}_{(1)}} J_3 = [\nabla_{\boldsymbol{f}_1} J_3, \nabla_{\boldsymbol{f}_2} J_3, \cdots, \nabla_{\boldsymbol{f}_H} J_3, \mathbf{0}, \cdots, \mathbf{0}]^\top$ can be obtained. Summing the gradients of the three loss functions can bring the total gradient, i.e., $\nabla_{\boldsymbol{G}_{(1)}} J = \nabla_{\boldsymbol{G}_{(1)}} J_1 + \gamma_1 \nabla_{\boldsymbol{G}_{(1)}} J_2 + \gamma_2 \nabla_{\boldsymbol{G}_{(1)}} J_3$.

Gradient of $\boldsymbol{\alpha}$. The learning weight $\boldsymbol{\alpha}$ is coupled with the classification loss and the regularization. For the classification loss, then

$$\nabla_{\alpha_i} J_2 = \begin{cases} \sum_{j=1}^{H} 2(\hat{y}_j - y_j) k(\boldsymbol{f}_i, \boldsymbol{f}_j), & \text{if } y_i \hat{y}_i < 1 \ , \\ \\ 0, & \text{if } y_i \hat{y}_i \geq 1 \ . \end{cases} \tag{3.17}$$

Note that $\hat{y}_j$ can be explicitly expressed by $\boldsymbol{\alpha}$, i.e., $\hat{y}_j = \boldsymbol{k}_j^\top \boldsymbol{\alpha} + b$, where $\boldsymbol{k}_i$ is the i^{th} column vector of the kernel matrix $\boldsymbol{K}$, and the kernel matrix is defined as $\boldsymbol{K}(i,j) = k(\boldsymbol{f}_i, \boldsymbol{f}_j)$. Thus, the above equation can be written to a matrix format. Specifically, if $y_i \hat{y}_i < 1$, $\nabla_{\alpha_i} J_2 = 2\boldsymbol{k}_i^\top \cdot \boldsymbol{K}\boldsymbol{\alpha} + 2\boldsymbol{k}_i^\top \cdot (b - y_i)\mathbf{1}$ can be written, where $\mathbf{1}$ is an all-one column vector. Further, one can obtain a general format:

$$\nabla_{\boldsymbol{\alpha}} J_2 = 2\boldsymbol{K}\boldsymbol{I}^0(\boldsymbol{K}\boldsymbol{\alpha} + b\mathbf{1} - \boldsymbol{y}), \tag{3.18}$$

where $\boldsymbol{I}^0$ satisfies

$$\boldsymbol{I}^0(i,j) = \begin{cases} 1, & \text{if } i = j \text{ and } y_i \hat{y}_i < 1 \ , \\ \\ 0, & \text{otherwise.} \end{cases} \tag{3.19}$$

Further, for the regularization, it's easy to find that

$$\nabla_{\boldsymbol{\alpha}} J_3 = 2\boldsymbol{K}\boldsymbol{\alpha}. \tag{3.20}$$

Finally, the total gradient is $\nabla_{\boldsymbol{\alpha}} J = \gamma_1 \nabla_{\boldsymbol{\alpha}} J_2 + \gamma_2 \nabla_{\boldsymbol{\alpha}} J_3$.

Gradient of b. Similarly, I calculate the gradient with respect to b:

$$\nabla_b J = \mathbf{1}^\top \boldsymbol{I}^0 (\hat{\boldsymbol{y}} - \boldsymbol{y}), \tag{3.21}$$

where $\hat{\boldsymbol{y}}$ is the vector of all $\hat{y}_i$s. With the above derivations, the final learning algorithm is presented in Algorithm 4.

Algorithm 4 Train-KTDC-Se($\mathcal{X}, \boldsymbol{y}$).

Input: Training tensor $\mathcal{X}$ and labels $\boldsymbol{y}$.
Hyper-parameters: number of labeled data H, core tensor dimensions R_2, R_3, and R_4, regularization parameters γ_1 and γ_2, polynomial kernel parameters d and c, RBF kernel parameters λ, and learning rate lr.
Initialize $\boldsymbol{G}_{(1)}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\alpha}$, and b.
while Not converge **do**
  Calculate $\nabla_{\boldsymbol{A}} J$, $\nabla_{\boldsymbol{B}} J$, $\nabla_{\boldsymbol{C}} J$, and $\nabla_{\boldsymbol{D}} J$ by Equations (3.10) and (3.11).
  $\boldsymbol{A} = \boldsymbol{A} - lr \cdot \nabla_{\boldsymbol{A}} J$.
  $\boldsymbol{B} = \boldsymbol{B} - lr \cdot \nabla_{\boldsymbol{B}} J$.
  $\boldsymbol{C} = \boldsymbol{C} - lr \cdot \nabla_{\boldsymbol{C}} J$.
  $\boldsymbol{D} = \boldsymbol{D} - lr \cdot \nabla_{\boldsymbol{D}} J$.
  Calculate $\nabla_{\boldsymbol{G}_{(1)}} J_1$ by Equation (3.12).
  for $i = 1$ to H **do**
    Calculate $\nabla_{\boldsymbol{f}_i} J_2$ and $\nabla_{\boldsymbol{f}_i} J_3$ by Equations (3.13) and (3.16) while fixing other parameters.
  Formalize $\nabla_{\boldsymbol{G}_{(1)}} J_2$ and $\nabla_{\boldsymbol{G}_{(1)}} J_3$. Then, obtain $\nabla_{\boldsymbol{G}_{(1)}} J$.
  $\boldsymbol{G}_{(1)} = \boldsymbol{G}_{(1)} - lr \cdot \nabla_{\boldsymbol{G}_{(1)}} J$.
  Build matrix $\boldsymbol{I}^0$ by Equation (3.19).
  Calculate $\nabla_{\boldsymbol{\alpha}} J_2$ and $\nabla_{\boldsymbol{\alpha}} J_3$ by Equations (3.18) and (3.20). Then, obtain $\nabla_{\boldsymbol{\alpha}} J$.
  $\boldsymbol{\alpha} = \boldsymbol{\alpha} - lr \cdot \nabla_{\boldsymbol{\alpha}} J$.
  Calculate $\nabla_b J$ by Equation (3.21).
  $b = b - lr \cdot \nabla_b J$.
Output: Parameters $\boldsymbol{G}^k_{(1)}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\alpha}$, and b.

3.5.4 Training and Testing on Mini-batches

The above learning process may suffer storage issues when the number of training data N is large. Specifically, when updating $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ in Equations (3.11), the Kronecker product to calculate $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, and $\boldsymbol{H}_4$ requires a large cost of storage as $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ for a large N. To mitigate this issue, I modify Algorithm 4 to train on mini-batches to save the memory (Zhao et al. 2014).

Mathematically, I divide the training tensor $\mathcal{X}$ and labels $\boldsymbol{y}$ into K mini-batches $\{\mathcal{X}^i\}_{i=1}^K$ and $\{\boldsymbol{y}^i\}_{i=1}^K$, respectively. Each $\mathcal{X}^i$ contains some labeled data with labels to be $\boldsymbol{y}^i$ and many unlabeled data. Then, in each iteration, I update the mini-batch-independent weights $\tilde{\boldsymbol{A}}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ and b. $\tilde{\boldsymbol{A}} \in \mathbb{R}^{\tilde{N} \times \tilde{N}}$ is the matrix along the first dimension of each mini-batch tensor, where $\tilde{N} = \lfloor N/K \rfloor$ and $\lfloor \cdot \rfloor$ represents the floor function. Notably, keeping $\tilde{\boldsymbol{A}}$ the same for different mini-batches is an additional restriction to maintain similarity of $\boldsymbol{A}$, which doesn't appear in the direct training of Algorithm 4. However, this restriction further guarantees that the discriminative information is contained in core tensors.

Further, KTDC-Se uses mini-batch-dependent parameters $\boldsymbol{G}_{(1)}^i \in \mathbb{R}^{\tilde{N} \times (R_2 \cdot R_3 \cdot R_4)}$ and $\boldsymbol{\alpha}^i$ for the i^{th} mini-batch, where $\boldsymbol{\alpha}^i \in \mathbb{R}^{\tilde{H} \times 1}$ and $\tilde{H} = \lfloor H/K \rfloor$. Especially, $\boldsymbol{G}_{(1)}^i$ is a sub-block of $\boldsymbol{G}_{(1)}$, i.e., $\boldsymbol{G}_{(1)} = [(\boldsymbol{G}_{(1)}^1)^\top, \cdots, (\boldsymbol{G}_{(1)}^K)^\top]^\top$. Correspondingly, I have $\boldsymbol{F} = [(\boldsymbol{F}^1)^\top, \cdots, (\boldsymbol{F}^K)^\top]^\top$, where $\boldsymbol{F}^i = \boldsymbol{E}^i \boldsymbol{G}_{(1)}^i$ and $\boldsymbol{E}^i = [\boldsymbol{I}^{\tilde{H} \times \tilde{H}}, \boldsymbol{0}^{\tilde{H} \times (\tilde{N} - \tilde{H})}]$. Essentially, $\boldsymbol{\alpha}^i$ is a sub vector of the final weight vector $\boldsymbol{\alpha} = [(\boldsymbol{\alpha}^1)^\top, \cdots, (\boldsymbol{\alpha}^K)^\top]^\top$.

Then, for the i^{th} mini-batch, the training algorithm should obtain features $\boldsymbol{G}_{(1)}^i$, check the support vectors in these features, and update their corresponding weights in $\boldsymbol{\alpha}^i$. To maintain the coupling between $\boldsymbol{G}_{(1)}^i$(or $\boldsymbol{\alpha}^i$) and the rest $\boldsymbol{G}_{(1)}^j$s (or $\boldsymbol{\alpha}^j$s), where $j \neq i$, all the information in $\boldsymbol{F}$, $\boldsymbol{y}$ and $\boldsymbol{\alpha}$ should be utilized to obtain $\nabla_{\boldsymbol{G}_{(1)}^i} J_2$,

$\nabla_{\boldsymbol{G}^i_{(1)}} J_3$, $\nabla_{\boldsymbol{\alpha}^i} J_2$, and $\nabla_{\boldsymbol{\alpha}^i} J_3$ by Equations (3.13), (3.16), (3.18), and (3.20), respectively. Then, I can fix the correspondingly gradients with respect to support vectors in other mini-batches to $\mathbf{0}$s. Consequently, the complete algorithm can be seen in Algorithm 5.

Algorithm 5 Train-mini-batch-KTDC-Se($\mathcal{X}, \boldsymbol{y}$).

Input: Training tensor $\mathcal{X} = \{\mathcal{X}^i\}_{i=1}^K$ and labels $\boldsymbol{y} = \{\boldsymbol{y}^i\}_{i=1}^K$.
while Not converge **do**
  for $i = 1$ to K **do**
    Utilize the complete information in $\boldsymbol{G}$, $\boldsymbol{y}$, and $\boldsymbol{\alpha}$ and the mini-batch data to obtain: $\boldsymbol{G}^i_{(1)}, \tilde{\boldsymbol{A}}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\alpha}^i, b$ = Train-KTDC-Se($\mathcal{X}^i, \boldsymbol{y}^i | \boldsymbol{F}, \boldsymbol{y}, \boldsymbol{\alpha}$).
    $\boldsymbol{F} = [(\boldsymbol{F}^1)^\top, \cdots, (\boldsymbol{F}^K)^\top]^\top$.
    $\boldsymbol{\alpha} = [(\boldsymbol{\alpha}^1)^\top, \cdots, (\boldsymbol{\alpha}^K)^\top]^\top$.
Output: Parameters $\{\boldsymbol{G}^i_{(1)}\}_{i=1}^K, \tilde{\boldsymbol{A}}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{\alpha}$, and b.

For the cross-validation process or online testing, I have another total test tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{\tilde{N} \times T \times L \times M}$ that needs to experience the decomposition and classification to obtain the label $\tilde{\boldsymbol{y}} \in \mathbb{Z}^{\tilde{N} \times 1}$, where I fix $\tilde{N}$ to be the number of tensors in one mini-batch. The reason of fixing $\tilde{N}$ is that the mini-batch training yields a parameter matrix $\tilde{\boldsymbol{A}} \in \mathbb{R}^{\tilde{N} \times \tilde{N}}$ that must be utilized for the test tensor decomposition. Therefore, there should be $\tilde{N}$ tensors in the total test tensor. For real-time testing, if the testing tensor number is not sufficient, I can repeat the testing tensor or utilize some data from the historical dataset to complete the testing mini-batch. Thus the testing procedures are as follows.

Obtain Test Feature Matrix $\tilde{\boldsymbol{G}}_{(1)}$. To obtain $\tilde{\boldsymbol{G}}_{(1)}$, I utilize the learned parameters $\tilde{\boldsymbol{A}}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{D}$. By setting the gradient in Equation (3.12) to $\mathbf{0}$, I can obtain

$$\tilde{\boldsymbol{G}}_{(1)} = (\tilde{\boldsymbol{A}}^\top \tilde{\boldsymbol{A}})^{-1} \cdot (\tilde{\boldsymbol{A}}^\top \tilde{\boldsymbol{X}}_{(1)} \tilde{\boldsymbol{H}}^\top) \cdot (\tilde{\boldsymbol{H}} \tilde{\boldsymbol{H}}^\top)^{-1}, \tag{3.22}$$

where $\tilde{\boldsymbol{X}}_{(1)}$ represents mode-1 unfolding of tensor $\tilde{\mathcal{X}}$.

Predict label vector $\tilde{\boldsymbol{y}}$. Based on the Representer theorem, I need the learned weights $\boldsymbol{\alpha}$ and b, historical features in $\boldsymbol{F} = \boldsymbol{G}_{(1)}$, and test features in $\tilde{\boldsymbol{G}}_{(1)} = [\tilde{\boldsymbol{f}}_1^\top, \cdots, \tilde{\boldsymbol{f}}_{\tilde{N}}^\top]^\top$ to predict labels. Specifically, I can first calculate a test kernel matrix $\tilde{\boldsymbol{K}}(i,j) = k(\tilde{\boldsymbol{f}}_i, \boldsymbol{f}_j)$, where $\tilde{\boldsymbol{f}}_i \in \tilde{\boldsymbol{G}}_{(1)}$ and $\boldsymbol{f}_j \in \boldsymbol{F}$. Then, the predicted label can be obtained by:

$$\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{K}}\boldsymbol{\alpha} + b. \tag{3.23}$$

3.5.5 Numerical Result

3.5.5.0.1 Settings

Dataset. I utilize Illinois 200-bus system South Carolina 500-bus system (Engineering Texas A&M University 2016b) to generate event data. Five event types are considered, including line trip, generator trip, single-phase-to-ground fault, phase-to-phase fault, and three-phase fault. For each event type, I consider 2 different event locations. Thus, there are 10 unique combinations of event types and locations, i.e., 10 event labels.

Then, I vary the loading conditions for the simulation to generate diversified event files. Totally, I have 80 event files each of which has 10s event data. Further, I consider the data resolution to be 60 samples per second, yielding 600 samples for each event file. To extract tensors from these streams, I utilize the moving window with the length to be 0.5s (i.e., 30 samples) and the moving gap to be 0.083s (i.e., 5 samples) to cut the PMU streams. Therefore, I have 5840 PMU tensors in total. For each PMU tensor, I have $T = 30$ for the time dimension, $L = 200\eta_1$ or $L = 500\eta_1$ for the 200-bus and the 500-bus system, respectively, where $\eta_1 \in \{0.05, 0.1, 0.15, 0.2\}$ represents the

PMU penetrations for the grid. For each fixed η_1, PMU locations are randomly chosen for 5 times. Then, I set $M = 3$ for the measurement types, i.e., voltage magnitude, voltage angle, and frequency. To summarize, I have $\mathcal{X} \in \mathbb{R}^{5840\times30\times200\eta_1\times3}$ or $\mathcal{X} \in \mathbb{R}^{5840\times30\times500\eta_1\times3}$ for training and testing. To mimic a semi-supervised setting, I consider labeled data with the ratio of $\eta_2 = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, leading to a label vector $\boldsymbol{y} \in \mathbb{Z}^{5840\eta_2\times1}$.

Finally, I also test our proposed method using real-world PMU data from our partner in Arizona, USA. These files totally have 5 labels covering 3 types of line faults at 2 locations. After tensorization of data in 35 PMUs, I can obtain $\mathcal{X} \in \mathbb{R}^{511\times30\times35\times3}$ and $\boldsymbol{y} \in \mathbb{Z}^{511\times1}$.

Benchmark Methods. First, I train our KTDC-Se within the labeled data as a benchmark to demonstrate the impacts of the unlabeled data. Further, I employ state-of-the-art Semi-Supervised Learning (SSL) methods as benchmarks. The details of these methods are shown as follows.

- Deep Residual Network (Resnet) (K. He et al. 2016): Resnet is an efficient deep learning model for classification. For this supervised learning model, I utilize only labeled data as comparison. As PMU data have high dimensionality (e.g., 9000 for the 500-bus system with $\eta_1 = 0.2$), Principal Component Analysis (PCA) is utilized to pre-process data before training the Resnet.
- KTDC-Se-L: KTDC-Se-L is to train a KTDC-Se model with only labeled data by setting $N = H$ in the model, which demonstrates the effectiveness of employing unlabeled data for training a classifier.
- MixMatch (Berthelot et al. 2019): MixMatch can guess low-entropy labels for unlabeled instances with data augmentation. Then, MixMatch develops a probabilistic procedure to mix the labeled and unlabeled data to train a deep

learning classifier. Similarly, I employ PCA to reduce the dimensionality of the mixed dataset from MixMatch and input them to a Resnet (K. He et al. 2016) as the final classifier. For a fair comparison, the Resnet has the same architecture as the first benchmark.

- FixMatch (Sohn et al. 2020): FixMatch first generates pseudo labels for data with weak data augmentation. Then, FixMatch develops a criterion to decide the pseudo label is retained or not. Finally, data with retained pseudo labels experience a strong data augmentation for the classifier training. Similar to MixMatch, I utilize PCA and Resnet as the final classifier. For fair comparison, the Resnet has the same architecture as the first benchmark.
- Semi-supervised Ladder Network (SSLN) (Rasmus et al. 2015): SSLN combines supervised and unsupervised learning in deep neural networks with a joint loss function in a ladder network with an auto-encoder model structure. Similarly, I pre-process the data with the PCA method.

During the testing, the hyper-parameters for all models are fine-tuned in the 3-fold cross-validation to achieve the best accuracy. In general, by comparing the testing accuracy of KTDC-Se with Resnet, and KTDC-Se-L, I can illustrate the effectiveness of using unlabeled data. By comparing the testing accuracy of KTDC-Se and other methods, I can evaluate the performance of using an integrated model and two-stage models. Especially, Resnet, MixMatch, FixMatch, and SSLN have two separate steps of data pre-processing and learning, which have their biased selections. By comparing the label predicting time of KTDC-Se and other methods, I can evaluate the efficiency of the methods for real-time inference. Finally, by comparing the choice of kernel selection, I can understand how the non-linear kernels boost the performance.

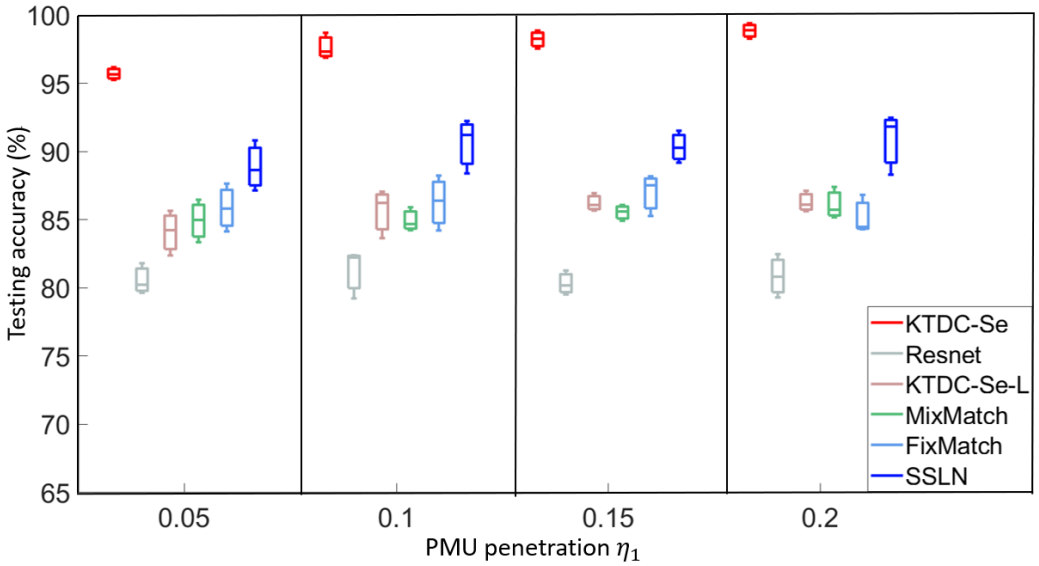
3.5.5.0.2 Joint Optimization of KTDC-Se is Better Than Two-stage Models

In this subsection, we evaluate the integration design by comparing our KTDC-Se and other two-stage models. Thanks to the integration without biased selection for the two-stage compression and classification, our model performs much better than benchmarks. Specifically, we report the results of simulated and real-world data as follows.

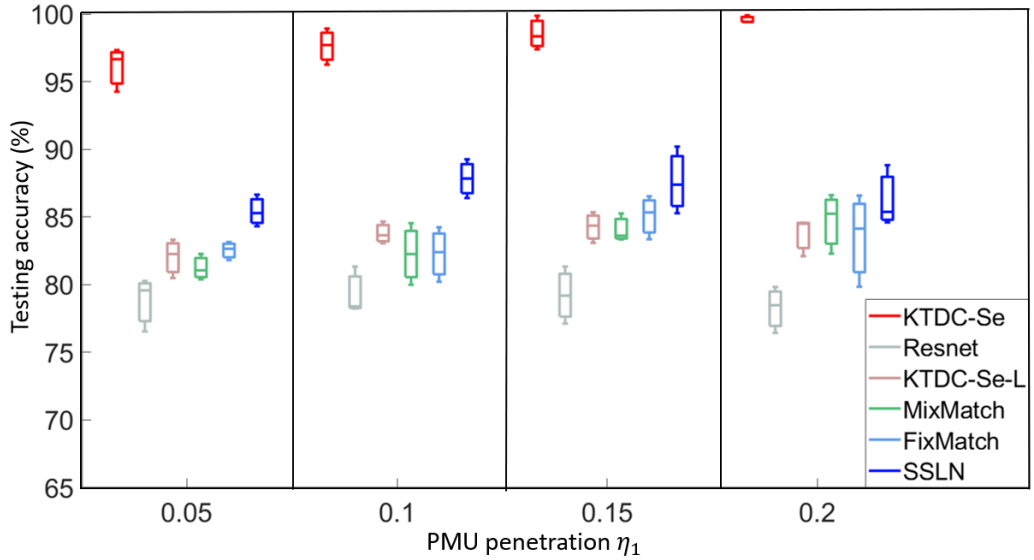
For the simulated data, we first fix $\eta_2 = 0.3$ to divide the labeled and unlabeled datasets for training and testing. Fig. 19a and 19b demonstrate the results for the two systems. Since for each η_1 of one system, we conduct 5 times randomization and 3-fold cross validation of the PMU location selection, there can be multiple different values of the testing accuracy for each testing scenario. Thus, we present the box plot in Fig. 19a and 19b to show the average and the variance.

We find that our KTDC-Se has an average accuracy promotion of 13.3%, 13.6%, and 9.3%, compared to MixMatch, FixMatch, and SSLN, respectively. Notably, the latter 3 benchmark models utilize PCA to pre-process data so that they can be trained with a reasonable cost. Even though these 4 methods utilize the same labeled and unlabeled data for training, the better performance of KTDC-Se shows that an integrated model can be better than the two-stage models. This is because that the integrated model avoids the biased selection of the two separate models. Further, the joint model enables the identification of discriminative core tensors that are sensitive to the event labels.

For the real-world data, we report the testing accuracy in Table 8. We observe that the average accuracy promotions of KTDC-Se are 9.7%, 9%, and 8.8%, compared



(a) Testing accuracy (%) for the 200-bus system.



(b) Testing accuracy (%) for the 500-bus system.

Figure 19. Performances of event identification using different methods.

Table 8. Testing accuracy (%) (mean ± standard deviation) for real-world PMU data.

	KTDC-Se	Resnet	KTDC-Se-l	MixMatch	FixMatch	SSLN
Accuracy	$\mathbf{92.3 \pm 0.8}$	77.1 ± 0.6	80.5 ± 1.1	82.6 ± 0.8	83.3 ± 1.6	83.5 ± 2.1

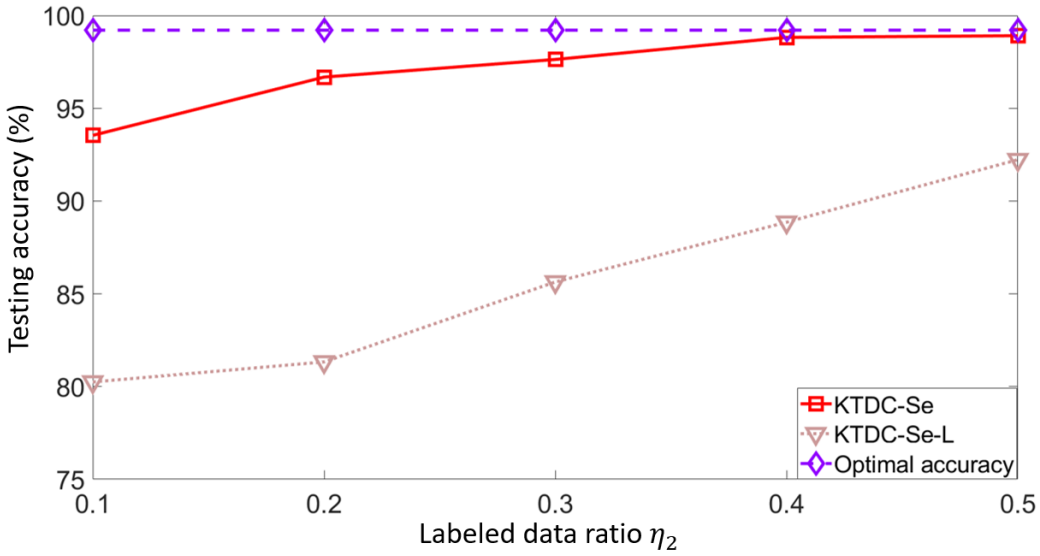
to MixMatch, FixMatch, and SSLN, respectively. This still supports the advantage of using an integrated model.

3.5.5.0.3 Semi-supervised Learning Boosts KTDC-Se Model Performance

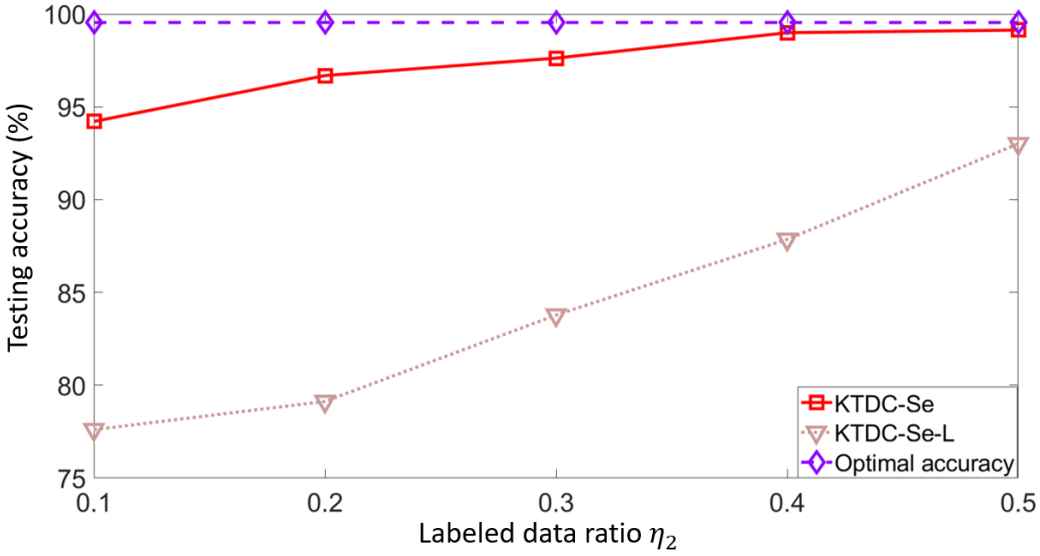
In this subsection, we compare KTDC-Se, Resnet, and KTDC-Se-L to illustrate the effectiveness of semi-supervised learning. Specifically, for synthetic data, the average accuracy promotions are 18% and 13.3%, compared to Resnet and KTDC-Se-L, respectively. For real-world data, the corresponding accuracy promotions are 15.2% and 11.8%. These results show that there is a significant improvement when using unlabeled data.

The performance can be explained as follows. First, we can compare KTDC-Se and KTDC-Se-L. When doing the tensor decomposition of the PMU tensors, the unlabeled tensors in KDTC-Se enable the core tensors to understand different loading conditions and maintain similarity for different conditions as long as the event label is the same. Then, the trained classifier can successfully tackle different operational scenarios. Second, we can compare KTDC-Se-L and Resnet. We find that Resnet has an even worse performance as Resnet also employs PCA to pre-process data. As illustrated in Section 3.5.5.0.2, the two-stage model performs worse.

To further understand how many labeled data are needed for a good performance of KTDC-Se, we utilize the simulated data to test and fix $\eta_1 = 0.1$ and vary $\eta_2 \in$



(a) Sensitivity analysis for the 200-bus system.



(b) Sensitivity analysis for the 500-bus system.

Figure 20. Results of the sensitivity analysis with respect to labeled data ratios.

$\{0.1, 0.2, 0.3, 0.4, 0.5\}$ for the number of labeled data. As a comparison, we also implement KTDC-Se-L for the labeled data. Further, we also present the optimal testing accuracy via employing labels for all the PMU tensors to train the model, i.e., $\eta_2 = 1$. The results of average testing accuracy are shown in Fig. 20a and 20b.

Based on the plots, we have the following observations. (1) Training with extra unlabeled data significantly improves the accuracy, which has been explained before. (2) KTDC-Se can efficiently utilize the limited labels. For example, when $\eta_2 = 0.1$, the testing accuracy is still higher than 94%. This implies that KTDC-Se can filter information and group data by event labels in the feature space. Such a good result comes from the joint learning process to generate compact and discriminative features. (3) As η_2 increases, the testing accuracy of KTDC-Se gradually increases up to the optimal accuracy. Further, with only 30% of the labeled data, KTDC-Se can bring an accuracy almost close to the optimal one. This again indicates the high efficiency of KTDC-Se to utilize limited labels.

3.5.5.0.4 Tensor-based Framework Enables Fast Inference

Table 9. The average predicting time (s) of the testing dataset for different methods.

	KTDC-Se	Resnet	KTDC-Se-l	MixMatch	FixMatch	SSLN
Time	1.3	3.8	0.4	3.8	3.7	6.4

The model efficiency can be evaluated via the predicting time of methods on testing datasets. We utilize the simulated data in Section 3.5.5.0.2 and report the computational time of all methods on the testing dataset in Table 9. We find that KTDC-Se has the second lower computational time, which demonstrates its efficiency.

Further, KTDC-Se-L has the lowest time because it uses less data (i.e., only labeled data) for training. Thus, the test kernel matrix has a much smaller size compared to that in KTDC-Se. According to the prediction function in Equation (3.23), KTDC-Se-L has a lower computational time compared to KTDC-Se.

However, if we utilize both labeled and unlabeled data, KTDC-Se is much more efficient than other methods. The reason is that KTDC-Se employs tensor decomposition to explore cross-dimension correlations and obtain a very compact core tensor for training, but other methods utilize PCA to compress data, which needs more features to achieve the best accuracy. Secondly, other methods utilize a deep model to extract non-linear features, requiring more computational time. Thirdly, Resnet, FixMatch, and MixMatch share the same DNN architecture and consume close time for testing. On the other hand, the ladder network in SSLN has a larger size and needs more time to compute.

3.5.5.0.5 Non-linear Kernelization Largely Increases the Accuracy

In this subsection, we study the effectiveness of kernelization to the capacity of the classifier. Specifically, we report the testing accuracy for the 200-bus system when $\eta_1 = 0.1$ and $\eta_2 = 0.3$. Other results are similar and ignored due to the space limit. Then, we study the case with a constant kernel (e.g., $d = 1$ for the polynomial kernel) and cases with different non-linear kernels. When kernel is a polynomial function, we vary $d \in \{1, 2, 3\}$. When kernel is a RBF function, we vary $\lambda \in \{0.05, 0.1, 0.15\}$.

The results are shown in Table 10. First, if we compare the constant kernels with other kernels, we find that adding non-linear kernels can significantly increase the accuracy. This is because the PMU measurements have high non-linear correlations.

Secondly, the polynomial kernel can lead to an accuracy 95.5% when $d \geq 2$. For the RBF kernel, the accuracy is around 94.7% when $\lambda \geq 0.1$. This shows that the polynomial kernel, especially when $d = 2$, is better than others. The partial reason is that the quadratic correlations largely lie in the power flow equations.

Table 10. The testing accuracy (%) (mean ± standard deviation) for different kernels.

Kernel name	d or λ	Accuracy
Polynomial	1	89.22 ± 0.6
Polynomial	2	$\mathbf{95.7 \pm 0.3}$
Polynomial	3	95.4 ± 0.2
RBF	0.05	93.3 ± 0.6
RBF	0.1	94.7 ± 0.3
RBF	0.15	94.6 ± 0.5

Chapter 4

TRANSFER SYSTEM KNOWLEDGE ACROSS DIFFERENT SYSTEMS

4.1 Introduction

Modern physical systems are experiencing tremendous changes. First, The system territory is expanding to better serve society. Second, to increase the efficiency and decrease the cost of the system, new components are continuously introduced. Some of them bring a lot of uncertainties due to intrinsic uncertain sources, imperfect component monitoring ability, etc. For example, power systems integrate a large portion of renewable energies like PhotoVoltaic (PV) power and wind power, the output of which depends on dynamic weather conditions. In general, modern physical systems have become more complex.

While conventional system analysis may fail for large-scale edge areas with high uncertainties, Data Mining (DM) and Machine Learning (ML) methods could play a game-changing role with accurate and cost-efficient implementations for new technologies of system monitoring, control, and protection. Especially, DM/ML models have been developed for physical system state estimation, cyber-attack detection, and event identification (Li et al. 2019a, 2019b), etc. Such wide applications are due to DM/ML's flexible modeling capability, robust performance, and high inference speed for real-time operations.

However, most of the existing DM/ML models on physical systems require a certain amount of training data. If the data are limited, the model training is likely to fail due to the curse of dimensionality. Unfortunately, data-limited scenarios often occur

for a completely new grid or an old grid with increased metering, not to mention the common upgrading process with new nodes/lines. Thus, it is urgent to employ new methods to transfer knowledge from the source grid with rich data to the target grid with limited data.

For this goal, Transfer Learning (TL) is defined conceptually as an efficient procedure to extract common knowledge from two different domains and boost the performance of the data-limited domain. Specifically, an efficient approach is Domain Adaptation (DA) (Pan, Kwok, Yang, et al. 2008; Long et al. 2013; Long et al. 2016) which minimizes the data distribution discrepancy of two domains, usually in a low-dimensional space for domain invariant features. Therefore, DA promises that joint training using common knowledge can significantly boost the learning process. While many efficient DA models have been applied to computer vision (Long et al. 2013; Ganin et al. 2016a) and natural language process (Wu and Huang 2016), relatively few work has been done for graph data (Shen et al. 2021).

For physical systems, the widely placed sensors bring numerous networked measurements for nodes and edges with complex spatial-temporal correlations. Can I fully explore the correlations to improve DA methods? Further, nodes or edges can have different physical characteristics, i.e., different labels. Is the label information beneficial to the distribution adaptation? Intuitively, the answer shall be yes for both questions as the above information represents different levels of graph similarity that further guides the extraction of common knowledge in DA.

However, limited work exists in this topic (Dai et al. 2022; Fang, Yin, and Zhu 2013; Lee et al. 2017; Fu et al. 2019; Shen et al. 2021). Specifically, (Fang, Yin, and Zhu 2013) projects networked data into a latent space that captures the common sub-structure. However, finding common sub-structures will cause the non-ignorable loss by removing

non-common parts. (Fu et al. 2019) can measure the similarity of different graphs with an index based on the node degree similarity. However, the index is not sufficient to tackle physical systems with measurements and labels for nodes and edges. In this paper, I propose a graph kernel to make use of all data to measure the graph similarity for two different graphs. (Dai et al. 2022; Lee et al. 2017; Shen et al. 2021) build CNNs and GNNs to process networked data to learn transferable features. However, their projected feature space, though containing network information, is restricted with the same dimensionality and causes information losses for systems of different sizes. Further, they generally suffer high computational costs for large-scale physical systems and lack theoretical guarantees. I tackle these issues by proposing an efficient optimization using graph kernels and MMD measures with rigorous proofs.

4.2 Problem Formulation

Physical systems can be modeled as spatial-temporal graphs (Wu et al. 2020). Specifically, I denote $\{\mathcal{G}^n = \{\mathcal{V}, \mathcal{E}, \mathcal{D}^n\}\}_{n=1}^{N}$ as the source graph, where $\mathcal{V}$ and $\mathcal{E}$ are the node and edge sets, respectively. D^n is the corresponding data at the time slot n and N is the number of time slots for the graphs. Similarly, the target graph is denoted as $\{\widetilde{\mathcal{G}}^n = \{\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}}^n\}\}_{n=1}^{\widetilde{N}}$. In the following derivations, if there are no special notices, quantities with tilde are by default linked to the target graph.

The physical graph contains data of different modality, which together can boosts the knowledge transfer and feature extraction for the final machine learning model. In this paper, I assume the source graph contains node measurements $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times N}$, node labels $\boldsymbol{L} \in \mathbb{Z}^{|\mathcal{V}| \times N}$, edge measurements $\boldsymbol{Y} \in \mathbb{R}^{|\mathcal{E}| \times N}$, and edge labels $\boldsymbol{M} \in \mathbb{Z}^{|\mathcal{E}| \times N}$, where $|\cdot|$ for a set represents the operation to obtain the set cardinality. Secondly, I focus

on the graph-level classification and assume the task label vector is $\boldsymbol{h} \in \mathbb{Z}^N$. Thirdly, I assume the node/edge/task label lies in the range of $\{0, 1, \cdots, K_1\}$, $\{0, 1, \cdots, K_2\}$, and $\{0, 1, \cdots, K_3\}$, respectively. Finally, the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is assumed to be known. Similarly, I denote the corresponding quantities for the target graph as $\widetilde{\boldsymbol{X}} \in \mathbb{R}^{|\widetilde{\mathcal{V}}|\times\widetilde{N}}$, $\widetilde{\boldsymbol{L}} \in \mathbb{Z}^{|\widetilde{\mathcal{V}}|\times\widetilde{N}}$, $\widetilde{\boldsymbol{Y}} \in \mathbb{R}^{|\widetilde{\mathcal{E}}|\times\widetilde{N}}$, $\widetilde{\boldsymbol{M}} \in \mathbb{Z}^{|\widetilde{\mathcal{E}}|\times\widetilde{N}}$, $\widetilde{\boldsymbol{h}} \in \mathbb{Z}^{\widetilde{N}}$, and $\widetilde{\boldsymbol{A}} \in \mathbb{R}^{|\widetilde{\mathcal{V}}|\times|\widetilde{\mathcal{V}}|}$.

Then, I can find that $\{\mathcal{D}^n\}_{n=1}^N = \{\boldsymbol{X}, \boldsymbol{L}, \boldsymbol{Y}, \boldsymbol{M}, \boldsymbol{h}\}$ and $\{\widetilde{\mathcal{D}}^n\}_{n=1}^{\widetilde{N}} = \{\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{L}}, \widetilde{\boldsymbol{Y}}, \widetilde{\boldsymbol{M}}, \widetilde{\boldsymbol{h}}\}$. For some realistic scenarios when there is no label information, one can utilize matrices with all ones to indicate that nodes (edges) share the same label. Further, the model can be easily generalized to the case when there exist multiple measurements/labels for one node/edge at each time slot.

In the transfer learning setting, I address the problem when distributions of $\{\mathcal{D}^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{D}}^n\}_{n=1}^{\widetilde{N}}$ have a certain distance, which causes troubles for utilizing knowledge from both graphs for ML model training. Secondly, I assume there are much more data in the source graph, i.e., $N \gg \widetilde{N}$. Therefore, it is essential to convert data or features from the source graph to the target graph to enhance the ML models. Therefore, the problem is defined as follows.

- Problem: Transfer Learning between physical systems.
- Given: a source system $\{\mathcal{G}^n = \{\mathcal{V}, \mathcal{E}, \mathcal{D}^n\}\}_{n=1}^N$ and a target system $\{\widetilde{\mathcal{G}}^n = \{\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}}^n\}\}_{n=1}^{\widetilde{N}}$. Next, I assume $N \gg \widetilde{N}$.
- Find: an ML model $f(\cdot)$ that takes in features learned from $\{\mathcal{G}^n = \{\mathcal{V}, \mathcal{E}, \mathcal{D}^n\}\}_{n=1}^N$ and $\{\widetilde{\mathcal{G}}^n = \{\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{D}}^n\}\}_{n=1}^{\widetilde{N}}$ to perform tasks in the target system.

4.3 Improve the Speed of Knowledge Transfer via Graph Coarsening

In this section, I propose a pre-processing method to obtain coarser graphs which are much smaller than source and target graphs, thus improving the model efficiency for the following computations. Mathematically, I aim to find spatial-temporal coarser graphs $\{\mathcal{G}_C^n = \{\mathcal{V}_C, \mathcal{E}_C, \mathcal{D}_C^n\}\}_{n=1}^N$ and $\{\widetilde{\mathcal{G}}_C^n = \{\widetilde{\mathcal{V}}_C, \widetilde{\mathcal{E}}_C, \widetilde{\mathcal{D}}_C^n\}\}_{n=1}^{\widetilde{N}}$ such that $|\mathcal{V}_C| < |\mathcal{V}|$ and $|\widetilde{\mathcal{V}}_C| < |\widetilde{\mathcal{V}}|$. To obtain coarser graphs, I have the following objectives.

- Propose surjective maps $\widetilde{\pi} : \mathcal{V} \to \mathcal{V}_C$ and $\widetilde{\pi} : \widetilde{\mathcal{V}} \to \widetilde{\mathcal{V}}_C$ to aggregate nodes. Then, design the corresponding maps that can project nodal measurements/labels from the source/target graphs to the coarser graphs. Namely, I need to find $\pi^M : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}_C|}$, $\pi^L : \mathbb{Z}^{|\mathcal{V}|} \to \mathbb{Z}^{|\mathcal{V}_C|}$, $\widetilde{\pi}^M : \mathbb{R}^{|\widetilde{\mathcal{V}}|} \to \mathbb{R}^{|\widetilde{\mathcal{V}}_C|}$, and $\widetilde{\pi}^L : \mathbb{Z}^{|\widetilde{\mathcal{V}}|} \to \mathbb{Z}^{|\widetilde{\mathcal{V}}_C|}$.
- Figure out the connections $\mathcal{E}_C$ and $\widetilde{\mathcal{E}}_C$ with the corresponding maps $\omega : \mathcal{E} \to \mathcal{E}_C$ and $\widetilde{\omega} : \widetilde{\mathcal{E}} \to \widetilde{\mathcal{E}}_C$. Then, find the maps of the edge measurements/labels from source/target graphs to the coarser graphs. Namely, I need to find $\omega^M : \mathbb{R}^{|\mathcal{E}|} \to \mathbb{R}^{|\mathcal{E}_C|}$, $\omega^L : \mathbb{Z}^{|\mathcal{E}|} \to \mathbb{Z}^{|\mathcal{E}_C|}$, $\widetilde{\omega}^M : \mathbb{R}^{|\widetilde{\mathcal{E}}|} \to \mathbb{R}^{|\widetilde{\mathcal{E}}_C|}$, and $\widetilde{\omega}^L : \mathbb{Z}^{|\widetilde{\mathcal{E}}|} \to \mathbb{Z}^{|\widetilde{\mathcal{E}}_C|}$.

There is much prior work on calculating V_C and $\widetilde{V}_C$ and the related maps according to different measures and targets (Cai, Wang, and Wang 2021; Loukas 2019). In this paper, I focus on utilizing the structural information and the knowledge of node/edge labels to maintain data structure and distribution similarity.

Structural Node Aggregation with Label Information. The neighborhood of nodes for a physical system usually indicates a group with similar physical behaviors. Thus, I can utilize one node to represent them in the coarser graph. Secondly, the label information of nodes can further improve the node grouping for the local areas. Thus, node maps π and $\widetilde{\pi}$ should take in both factors for a reasonable coarsening

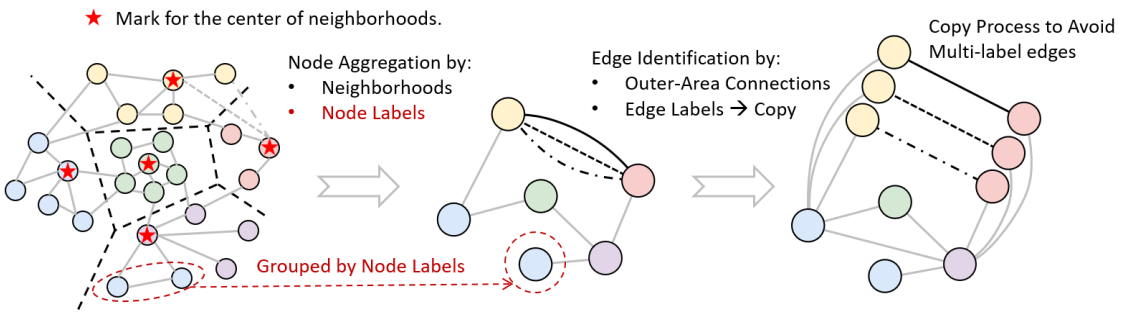


Figure 21. The procedure of graph coarsening. Different colors of nodes represent different labels. Different shape of lines represent different labels.

process. For the convenience of derivations, I only consider the coarsening of the source graph. Subsequently, I first initialize a set of neighborhoods based on the connectivity. Each neighborhood contains a center $i \in \mathcal{V}$ and the neighbouring nodes $\text{Neigh}(i)$ such that $j \in \text{Neigh}(i)$ if and only if $d(i,j) \leq d_{max}$, where $d(i,j)$ represent the pre-defined distance between node i and j, and d_{max} represents the maximum threshold. For example, I can define $d(i,j)$ to be the number of edges along the shortest path between i and j. As shown in Fig. 21, I consider $d_{max} = 1$ to aggregate nodes that are 1-hop away from the center nodes marked with a red star. Further, if there is no path between i and j, I define $d(i,j) = \infty$.

Subsequently, based on the previous discussion, the grouping of nodes in a neighborhood is too coarse and needs to be divided by the node labels. Thus, I divide $\text{Neigh}(i) = \{\text{Neigh}_k(i)\}_k^{K_1}$ such that $\forall j_1, j_2 \in \text{Neigh}_k(i)$, $\boldsymbol{L}(j_1, n) = \boldsymbol{L}(j_2, n) = k$ (recall that n is the index for the time slot for the discussed graph and data). For example, in Fig. 21, the blue and purple nodes have different labels. Thus, even though they share one neighborhood, they should be divided into two groups. Then, I have

$$\pi\big(\text{Neigh}_k(i)\big) = h \in \mathcal{V}_C. \tag{4.1}$$

Based on the above node aggregation rules, since nodes in $\text{Neigh}_k(i)$ share one label k, the projected node $h \in \mathcal{V}_C$ also has a label k. Namely,

$$\pi^L\Big(\boldsymbol{L}\big(\text{Neigh}_k(i), n\big)\Big) = k. \tag{4.2}$$

For the measurements, as I assume there is no prior on the importance of nodes, I directly average the measurements in the related nodes. Specifically, I have

$$\pi^M\Big(\boldsymbol{X}\big(\text{Neigh}_k(i), n\big)\Big) = \frac{1}{|\text{Neigh}_k(i)|}\sum_{j=1}^{|\text{Neigh}_k(i)|}\boldsymbol{X}(j, n). \tag{4.3}$$

Edge Identification with Label Information. After obtaining $\mathcal{V}_C$, I identify $\mathcal{E}_C$ with a simple and popular $\mathcal{V}_C$-induced approach (Cai, Wang, and Wang 2021). Namely, $(h_1, h_2) \in \mathcal{E}_C$ if and only if there is an edge $(i_1, i_2) \in \mathcal{E}$ and labels $1 \leq k_1, k_2 \leq K_2$ such that $\pi\big(\text{Neigh}_{k_1}(i_1)\big) = h_1$ and $\pi\big(\text{Neigh}_{k_2}(i_2)\big) = h_2$. In other words, the connection between two neighborhoods in the original graph implies a connection in the coarser graph.

However, the identified edge may have multiple labels as in the original graph, there may be many connections with multiple labels between two neighborhoods. This causes issues when formalizing edge measurement and label maps. For example, in the left and middle part of Fig. 21, the connections between yellow and pink nodes have multiple labels (i.e., different shapes of lines). To tackle this problem, for nodes in the coarser graph with multi-labeled edges, I propose to copy them according to the number of labels and assign each copied pair a uni-labeled edge. In the meantime, the outer connections to other nodes remain unchanged for each copied pair, shown in the right part of Fig. 21. Finally, the coarser graph only contains uni-labeled edges. For simplicity of later derivations, I denote $\text{Conne}_k(h_1, h_2) \subset \mathcal{E}$ to be the set of edges in the original graph that has a label k ($1 \leq k \leq K_2$) and connects nodes in $\pi^{-1}(h_1)$

and $\pi^{-1}(h_2)$, where π^{-1} represents the inverse map of π. Then, the edge map is

$$\omega\big(\text{Conne}_k(h_1, h_2)\big) = h \in \mathcal{E}_C. \tag{4.4}$$

Further, the edge label map can be defined as

$$\omega^L\Big(\boldsymbol{M}\big(\text{Conne}_k(h_1, h_2), n\big)\Big) = k. \tag{4.5}$$

Similarly, I can average the edge measurements in $\text{Conne}_k(h_1, h_2)$ to obtain the edge measurement in the coarser graph:

$$\omega^M\Big(\boldsymbol{Y}\big(\text{Conne}_k(h_1, h_2), n\big)\Big) = \frac{1}{|\text{Conne}_k(h_1, h_2)|} \sum_{j=1}^{|\text{Conne}_k(h_1,h_2)|} \boldsymbol{Y}(j, n). \tag{4.6}$$

Finally, I can follow the above procedures and generate the coarser graph for the target grid. For the two coarser graphs, the number of nodes can vary due to different numbers of neighborhoods and the node copy process in this subsection. Further, the connectivity can be different due to the varying topology of the two original graphs. Thus, the obtained spatial-temporal coarser graphs $\{\mathcal{G}_C^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{G}_C^n}\}_{n=1}^{\widetilde{N}}$ are usually different.

4.4 Conduct Cross-system Knowledge Transfer Using GNA: Graph Kernel-based Domain Adaptation

Distribution Adaptation for Two Sample Test. The two coarser graphs bring two sets $\{\mathcal{D}_C^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{D}}_C^n\}_{n=1}^{\widetilde{N}}$ of samples (measurements and labels) converted from source and target physical systems, which are generated via Equations (4.2), (4.3), (4.5), and (4.6). Thus, I target measuring and minimizing the distribution difference from the two sample sets and propose the Graph kerNel-based distribution Adaptation (GNA).

A prevalent measure for the distribution difference with two sample sets is the so-called Maximum Mean Discrepancy (MMD) (Long et al. 2013; Pan, Kwok, Yang, et al. 2008). MMD can compare distributions based on reproducing Hilbert Space (RKHS). Then, MMD-based domain adaptation finds a low-dimensional latent space via a linear transformation of the kernels of data samples, where the linear transformation is learnable to achieve minimal MMD measure. Specifically, the Joint Domain Adaptation (JDA) in (Long et al. 2013) tries to minimize:

$$\begin{aligned} &\min_{\boldsymbol{W}^\top \boldsymbol{K}\boldsymbol{H}\boldsymbol{K}^\top \boldsymbol{W}=\boldsymbol{I}} MMD(\{\mathcal{D}_C^n\}_{n=1}^N, \{\widetilde{\mathcal{D}}_C^n\}_{n=1}^{\widetilde{N}}) \\ &= \sum_{k=0}^{K_3} \mathrm{tr}(\boldsymbol{W}^\top \boldsymbol{K}\boldsymbol{N}_k\boldsymbol{K}^\top \boldsymbol{W}) + \lambda||\boldsymbol{W}||_F^2, \end{aligned} \tag{4.7}$$

where $\boldsymbol{H} = \boldsymbol{I} - \frac{1}{N+\widetilde{N}}\mathbf{1}$, $\boldsymbol{I}$ is the identity matrix, $\mathbf{1} \in \mathbb{R}^{(N+\widetilde{N})\times(N+\widetilde{N})}$ is a matrix of ones, and $\boldsymbol{W} \in \mathbb{R}^{(N+\widetilde{N})\times n_0}$ ($n_0 < N + \widetilde{N}$) is the learnable linear transformation to project kernel features to a low-dimensional space. $\boldsymbol{K}$ is the kernel matrix obtained from samples in sets $\{\mathcal{D}_C^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{D}}_C^n\}_{n=1}^{\widetilde{N}}$. λ is a positive penalty term and $||\cdot||_F$ is the Frobenius norm. $\boldsymbol{N}_k$ is a weight matrix. When $k = 0$, I have:

$$\boldsymbol{N}_0(i,j) = \begin{cases} \frac{1}{N^2}, & \text{if } 1 \le i,j \le N \ , \\ \frac{1}{\widetilde{N}^2}, & \text{if } N+1 \le i,j \le N+\widetilde{N} \ , \\ \frac{-1}{N\widetilde{N}}, & \text{otherwise,} \end{cases} \tag{4.8}$$

When $k \ge 1$, I have:

$$\boldsymbol{N}_k(i,j) = \begin{cases} \frac{1}{N_k^2}, & \text{if } 1 \le i,j \le N \text{ and } \boldsymbol{h}(i) = \boldsymbol{h}(j) = k \ , \\ \frac{1}{\widetilde{N}_k^2}, & \text{if } N+1 \le i,j \le N+\widetilde{N} \text{ and } \widetilde{\boldsymbol{h}}(i) = \widetilde{\boldsymbol{h}}(j) = k \ , \\ \frac{-1}{N_k\widetilde{N}_k}, & \text{if } 1 \le i \le N, N+1 \le j \le N+\widetilde{N}, \boldsymbol{h}(i) = \widetilde{\boldsymbol{h}}(j) = k \\ \frac{-1}{N_k\widetilde{N}_k}, & \text{if } 1 \le j \le N, N+1 \le i \le N+\widetilde{N}, \boldsymbol{h}(j) = \widetilde{\boldsymbol{h}}(i) = k \\ 0, & \text{otherwise,} \end{cases} \tag{4.9}$$

where N_k and $\widetilde{N}_k$ represent the samples with task label k for the source and the target coarser graph, respectively.

To calculate the kernel matrix, traditional kernels like the polynomial kernel (k_{poly}) or the Radial Basis Function (RBF) kernel (k_{rbf}) are largely utilized for image, WiFi, and text datasets. However, they are not suitable for datasets from physical systems, i.e., $\{\mathcal{D}_C^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{D}}_C^n\}_{n=1}^{\widetilde{N}}$, with structural knowledge, label information, and measurements with temporal correlations. To tackle this issue, I propose to leverage the graph kernel to incorporate all the above information.

The designed kernel is a modified version of the random walk kernel (Borgwardt et al. 2005; Vishwanathan et al. 2010) to process both the measurements and labels from nodes and edges. Further, the design can consider complex spatial-temporal correlations for better feature extractions. Random walk kernels (Vishwanathan et al. 2010) perform random walks on two graphs and calculate the number of matching walks to describe the graph similarity. The random walk within two graphs can be counted under a direct product graph. The reason to utilize random walk kernel is that (1) the computation is efficient and (2) the kernel can incorporate both label and measurement information.

Spatial-Temporal Graph Kernel with Measurements and Labels. In this subsection, I propose a graph kernel design for $\{\mathcal{G}_C^n\}_{n=1}^N$ and $\{\widetilde{\mathcal{G}_C^n}\}_{n=1}^{\widetilde{N}}$ to formalize Equation (4.7).

To propose the graph kernel, I consider two graphs $\mathcal{G}_C^{n_1}$ and $\widetilde{\mathcal{G}}_C^{n_2}$. For simplification, I eliminate the unnecessary time slot index n_1 and n_2 in the later derivations. Subsequently, the direct product graph, denoted as $\mathcal{G}_C \times \widetilde{\mathcal{G}}_C$, contains the node, edge set, and the adjacency matrix: $\mathcal{V}_\times = \{(i,j) : i \in V_C, j \in \widetilde{\mathcal{V}_C}\}$, $\mathcal{E}_\times = \{\big((i_1,j_1),(i_2,j_2)\big) : (i_1,i_2) \in \mathcal{E}_C \cap (j_1,j_2) \in \widetilde{\mathcal{E}_C}\}$, and $\boldsymbol{A}_\times = \boldsymbol{A}_C \otimes \widetilde{\boldsymbol{A}_C}$, where

$\boldsymbol{A}_C$ and $\widetilde{\boldsymbol{A}_C}$ are the adjacency matrices of the $\mathcal{G}_C$ and $\widetilde{\mathcal{G}_C}$, respectively. $\otimes$ is the kronecker product. Then, the classical random walk kernel considers the weighted sum of random walks on the direct product graph:

$$\boldsymbol{K}(n_1, n_2) = \sum_{i,j=1}^{|V_\times|} \sum_{l=0}^{\infty} \alpha(\boldsymbol{A}_\times)^l(i,j), \tag{4.10}$$

where α is a positive weight factor to guarantee convergence.

To better process nodal/edge measurement and label data, (Borgwardt et al. 2005) proposes to develop a modified random walk by replacing $\boldsymbol{A}_\times$ in Equation (4.10) with a new weighted adjacency matrix $\boldsymbol{B}_\times$ such that:

$$\begin{aligned} &\boldsymbol{B}_\times(i_1 + |\widetilde{\mathcal{V}_C}| \times (j_1 - 1), i_2 + |\widetilde{\mathcal{V}_C}| \times (j_2 - 1)) \\ &= \begin{cases} k_{step}\big((i_1, j_1), (i_2, j_2)\big), & \text{if } (i_1, j_1), (i_2, j_2) \in \mathcal{E}_\times \ , \\ \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.11}$$

where $k_{step}\big((i_1, j_1), (i_2, j_2)\big)$ is the step kernel to measure the similarity between steps $(i_1, i_2) \in \mathcal{E}_C$ and $(j_1, j_2) \in \widetilde{\mathcal{E}_C}$. However, the defined step kernel in (Borgwardt et al. 2005) can hardly capture the temporal correlations. To include the temporal correlations, I denote $\boldsymbol{l}_1 = [n_1 - \delta_n, n_1 - \delta_n + 1, \cdots, n_1, n_1 + 1, \cdots, n_1 + \delta_n]^\top$ and $\boldsymbol{l}_2 = [n_2 - \delta_n, n_2 - \delta_n + 1, \cdots, n_2, n_2 + 1, \cdots, n_2 + \delta_n]^\top$ to be index vectors for the neighbors of time slot n_1 and n_2 in the temporal dimension, respectively. Then, I

propose a new step kernel definition:

$$
\begin{aligned}
&k_{step}\big((i_1,j_1),(i_2,j_2)\big) = k^L\big((i_1,j_1),(i_2,j_2)\big) \times k^M\big((i_1,j_1),(i_2,j_2)\big),\\
&k^L\big((i_1,j_1),(i_2,j_2)\big) = \mathbf{1}\Big(\pi^L\Big(\boldsymbol{L}\big(\pi^{-1}(i_1),n_1\big)\Big) = \widetilde{\pi}^L\Big(\widetilde{\boldsymbol{L}}\big(\widetilde{\pi}^{-1}(j_1),n_2\big)\Big)\Big)\\
&\times \mathbf{1}\Big(\pi^L\Big(\boldsymbol{L}\big(\pi^{-1}(i_2),n_1\big)\Big) = \widetilde{\pi}^L\Big(\widetilde{\boldsymbol{L}}\big(\widetilde{\pi}^{-1}(j_2),n_2\big)\Big)\Big)\\
&\times \mathbf{1}\Big(\omega^L\Big(\boldsymbol{M}\big(\omega^{-1}(i_1,i_2),n_1\big)\Big) = \widetilde{\omega}^L\Big(\widetilde{\boldsymbol{M}}\big(\widetilde{\omega}^{-1}(j_1,j_2),n_2\big)\Big)\Big),\\
&k^M\big((i_1,j_1),(i_2,j_2)\big) = k_{rbf}\Big(\pi^M\Big(\boldsymbol{X}\big(\pi^{-1}(i_1),\boldsymbol{l}_1\big)\Big),\widetilde{\pi}^M\Big(\widetilde{\boldsymbol{X}}\big(\widetilde{\pi}^{-1}(j_1),\boldsymbol{l}_2\big)\Big)\Big)\\
&\times k_{rbf}\Big(\pi^M\Big(\boldsymbol{X}\big(\pi^{-1}(i_2),\boldsymbol{l}_1\big)\Big),\widetilde{\pi}^M\Big(\widetilde{\boldsymbol{X}}\big(\widetilde{\pi}^{-1}(j_2),\boldsymbol{l}_2\big)\Big)\Big)\\
&\times k_{rbf}\Big(\omega^M\Big(\boldsymbol{Y}\big(\omega^{-1}(i_1,i_2),\boldsymbol{l}_1\big)\Big),\widetilde{\omega}^M\Big(\widetilde{\boldsymbol{Y}}\big(\widetilde{\omega}^{-1}(j_1,j_2),\boldsymbol{l}_2\big)\big)\Big)\Big),
\end{aligned}
\tag{4.12}
$$

where $\mathbf{1}(\cdot)$ is the indicator function that takes 1 if the condition inside is true and 0 otherwise. One can replace the RBF kernel k_{rbf} with other kernels to evaluate the measurement similarity. Notably, to calculate the measurement kernel $k^M\big((i_1,j_1),(i_2,j_2)\big)$, I include the neighboring time slots in $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$ to extract features with temporal correlations. Then, I can use above equations to calculate $\boldsymbol{K}$. Specifically, (Vishwanathan et al. 2010) provides an equivalent equation to convert Equations (4.10) to:

$$\boldsymbol{K}(n_1,n_2) = \boldsymbol{q}_\times^\top(\boldsymbol{I} - \alpha\boldsymbol{B}_\times)^{-1}\boldsymbol{q}_\times, \tag{4.13}$$

where $\forall 1 \le i \le |\mathcal{V}_C|, \boldsymbol{q}_\times(i) = \frac{1}{|\mathcal{V}_C|}$ and $\forall |\mathcal{V}_C|+1 \le i \le |\mathcal{V}_C| + |\widetilde{\mathcal{V}}_C|, \boldsymbol{q}_\times(i) = \frac{1}{|\widetilde{\mathcal{V}}_C|}$. To calculate the inverse matrix in Equation (4.13), (Vishwanathan et al. 2010) provides an efficient way by solving Sylvester equations. One can refer to (Vishwanathan et al. 2010) for the procedure and Section 4.5 provides the time complexity of the calculation. Notice that the kernel calculation can be directly implemented on the source and the target graphs. However, the large size in the original graphs causes inefficiency for the computation, which prevents the realistic prediction for the classifier

trained with kernel features. Thus, graph coarsening in Section 4.3 is essential to guarantee model efficiency.

Finally, after the calculation of $\boldsymbol{K}$, I can construct the optimization in Equation (4.7), which can be efficiently solved as a generalized eigendecomposition problem (Long et al. 2013; Pan, Kwok, Yang, et al. 2008). Then, I summarize the complete learning algorithm in Algorithm 6.

Algorithm 6 Cross-Graph Domain Adaptation

Input: Spatial-temporal graphs $\{G^n\}_{n=1}^{N}$ and $\{\widetilde{G}^n\}_{n=1}^{\widetilde{N}}$
Hyper-parameters: Distance threshold d_{max} for node aggregation, temporal interval δ_n for temporal correlation integration, parameters of RBF kernel k_{rbf} (or polynomial kernel k_{poly}), and penalty term λ for the Frobenius norm in Equation (4.7).
Graph coarsening: Utilize Equations (4.1) to (4.6) to conduct graph coarsening with d_{max}. Then, obtain the coarser graphs $\{G_C^n\}_{n=1}^{N}$ and $\{\widetilde{G_C^n}\}_{n=1}^{\widetilde{N}}$.
Compute graph kernels: Utilize data of $\{G_C^n\}_{n=1}^{N}$ and $\{\widetilde{G_C^n}\}_{n=1}^{\widetilde{N}}$ to calculate weighted adjacency matrix $\boldsymbol{B}_{\times}$ under Equations (4.11) and (4.12). Then, solve Sylvester equations to compute kernel matrix for Equation (4.13).
Solve GNA: Construct GNA model in Equation (4.7) and solve the model as a generalized eigendecomposition problem.
Train classifier: Train a classifier f based on the obtained common features $\boldsymbol{W}^\top \boldsymbol{K}$ in Equation (4.7) and graph labels $\boldsymbol{h}$ and $\widetilde{\boldsymbol{h}}$.
Output: Kernel matrix $\boldsymbol{K}$, linear transformation matrix $\boldsymbol{W}$, and classifier f.

4.5 Theoretical Analysis

According to (Pan, Kwok, Yang, et al. 2008; Gretton et al. 2012), a universal kernel is required to guarantee MMD to be a correct statistic to measure distribution distance. Further, theorems in (Song 2008; Pan, Kwok, Yang, et al. 2008) show that the positive definite kernel matrix can guarantee the kernel is universal. Thus, I have the following theorem.

Theorem 7. *The proposed random walk kernel from Equations* (4.10) *to* (4.12) *is positive definite and universal.*

The proof can be seen in Section A.7. Next, I evaluate the computational cost of the proposed graph kernel. (Vishwanathan et al. 2010) obtain results of Equation (4.13) by solving Sylvester equations. Due to the space limit, I eliminate the derivations, and one can refer to (Vishwanathan et al. 2010) for more details. Based on the Sylvester equation methods, I propose the following theorem.

Theorem 8. *If* $|V|_C \approx |\widetilde{V}_C| \approx M$*, the computational complexity to calculate the proposed graph kernel matrix* $\boldsymbol{K}$ *is* $O\big((N+\widetilde{N})^2(M^3+\delta_n M^4)\big)$ *for Sylvester equation-based method.*

The proof can be seen in Section A.8. Finally, for the complete training algorithm, I follow the evaluation methods in (Long et al. 2013) to report the time complexity. Specifically, I have the following theorem.

Theorem 9. *If* $|\mathcal{V}|_C \approx |\widetilde{\mathcal{V}}_C| \approx M$*, the computational complexity of the training algorithm for Optimization* (4.7) *is* $O\big((n_0+K_3+1)(N+\widetilde{N})^2\big)$.

I note that it is possible to further reduce this complexity, by leveraging recent advance on scalable Sylvester equation solvers, such as (Du and Tong 2018).

4.6 Numerical Results

4.6.1 Settings

4.6.1.0.1 Dataset

In this experiment, I utilize power systems, mass-damper systems, and human activity sensing systems for testing. They are described as follows.

Power Systems. Power systems transmit electric power from generation sides to load sides. In this paper, I conduct a business-level simulation using Positive Sequence Load Flow (PSLF) (General Electric Energy Consulting 2018) from General Electric (GE) company. For system profiles, I employ data from Illinois 200-node system and South Carolina 500-node system (Engineering Texas A&M University 2016b). The profiles provide the label information for nodes and edges. Specifically, nodes can be categorized into generators, loads, and the slack bus. Edges can be divided into transformers and lines. After simulation, I can obtain nodal measurements of voltage magnitude, angle, and frequency and edge measurements of current magnitude and angle. Finally, the system labels include line trip, generator trip, single-phase fault, phase-to-phase fault, three-phase fault, load shedding, and transformer failure. For simplification, I denote the two systems as $P200$ and $P500$.

Mass-damper Systems. Mass damper systems study the mechanical functions of a structure to reduce the dynamic responses. Using MATLAB, I simulate the dynamic process of two mass-damper systems with 5 nodes and 10 nodes for transfer learning. Then, the force and the speed measurements of nodes are utilized for DA.

The labels of the system include edge trip and node trip. For simplification, I denote the two systems as $M5$ and $M10$.

Human Activity Sensing Systems. They are action measuring systems to measure the acceleration and the angular acceleration when a person is conducting an action (Zhang and Sawchuk 2012). In general, the researchers employ 14 subjects with 6 nodes for measuring. The node measurements include 3-axis acceleration and 3-axis angular acceleration data measurements for each time slot, with a total of around 10s to 30s. The labels include acceleration and angular acceleration. The system labels include 12 actions. I study the knowledge transfer between sensing systems of two subjects. For simplification, I denote the two systems as $H_1 6$ and $H_2 6$.

For power systems, graph coarsening is utilized to reduce the system size and increase the model efficiency. For other systems, I directly utilize the raw system data.

4.6.1.0.2 Benchmark Methods and Evaluations

GNA model can learn features in the kernelized feature space. Then, I utilize a deep Residual network (Resnet) (K. He et al. 2016) to conduct the classification task. Further, I use the following benchmark methods for comparison.

- Resnet + Principal Component Analysis (PCA): I utilize PCA to project data into a low-dimensional feature space that has the same dimensionality as features of GNA. Then, the features are input to Resnet *without transfer learning* as a benchmark method. The same Resnet is used for the classification.
- Transfer Component Analysis (TCA) (Pan, Kwok, Yang, et al. 2008) + Resnet: TCA minimizes the MMD of marginal distributions. The same Resnet is used for the classification.

- Joint Distribution Adaptation (JDA) (Long et al. 2013) + Resnet: JDA jointly minimizes the MMD of the marginal and conditional distributions. The same Resnet is used for the classification.
- Domain Adversarial Neural Network (DANN) (Ganin et al. 2016b): DANN employs adversarial training with Deep Neural Networks (DNNs) to learn domain-invariant features and do classification.
- Cross-network Deep Network Embedding (CDNE) (Shen et al. 2021): CDNE utilizes DNNs to learn label-discriminative and network-invariant representations. The network structure and networked data are employed.
- Graph Convolutional Adversarial Network (GCAN) (Ma, Zhang, and Xu 2019): GCAN employs Data Structure Analyzer to project source and target samples into instance graph. The graph captures the similarity between different samples. Then, GNN-based adversarial training is employed to learn invariant features over the graphs.

These methods are inclusive and representative according to the major DA categorizations, covering distribution metric-based (TCA, JDA, and CDNE) and adversarial learning-based (DANN, GCAN) methods. Further, traditional optimizations (TCA and JDA) and deep learning models (DANN, CDNE, and GCAN) are considered. In addition, CDNE considers networked data with structure information and GCAN considers graph structures between samples.

Note that for TCA, JDA, and DANN models, the dimensionality of the source and the target datasets should be the same. Thus, I copy some nodal measurements of the smaller system to align the dimensionality. For model evaluations, I conduct 5-fold cross-validation for the physical datasets and report the average test accuracy as the final score for each method.

4.6.2 Result of Test Accuracy

In this subsection, I evaluate the average test accuracy for different methods. I utilize the arrow to show the transferring process. For example, $P500 \rightarrow P200$ shows that 500-node power system is the source grid and 200-node power system is the target grid. Then, Tables 11 and 12 demonstrate the results. In general, the proposed GNA performs the best over other methods, with improvements of 7.24%, 10.63%, 6.99%, 6.58%, 7.78%, and 12.74% compared to JDA, TCA, DANN, CDNE, GCAN, and Resnet without transfer learning. This demonstrates that the proposed design can obtain better common knowledge over graphs to help train the classifier.

Table 11. Average test accuracy (%) of cross-system DA for different methods, table 1.

	$P500 \rightarrow P200$	$P200 \rightarrow P500$	$M10 \rightarrow M5$	$M5 \rightarrow M10$
GNA + RESNET	**93.28**	**94.34**	**95.53**	**98.87**
JDA + RESNET	86.71	86.19	91.15	90.24
TCA + RESNET	81.25	73.37	89.64	90.08
DANN	86.65	86.53	91.13	89.35
CDNE	88.25	87.93	90.34	91.02
GCAN	83.19	85.57	89.08	90.36
PCA + RESNET	78.75	75.21	87.85	88.96

Second, by comparing GNA, JDA, and TCA, I find that the graph label information can help improve DA. Specifically, TCA does not include the graph label information, thus inducing over-compressed features with different classes. On the other hand, graph labels re-weight the feature space and encourage adapting distributions within one class, leading to better performances of GNA and JDA. Third, GNA has an even better improvement since the node/edge labels enable a second layer of re-weighting

Table 12. Average test accuracy (%) of cross-system DA for different methods, table 2.

	$H_1 6 \rightarrow H_2 6$	$H_2 6 \rightarrow H_1 6$	AVERAGE
GNA + RESNET	**90.41**	**91.23**	**93.93**
JDA + RESNET	84.82	81.03	86.69
TCA + RESNET	82.32	83.15	83.30
DANN	84.51	83.47	86.94
CDNE	84.33	82.25	87.35
GCAN	83.46	85.25	86.15
PCA + RESNET	78.79	77.56	81.19

using graph kernels. Namely, GNA can encourage the minimization over nodes/edges from two graphs with the same label.

Fourth, GNA also has a stable improvement compared to deep learning models DANN, CDNE, and GCAN. Though the deep models have a high capacity of extracting domain-invariant features with discriminability, they may not include the structural, temporal, and node/edge label information properly. For example, DANN uses a convolutional kernel to integrate spatial correlations, which can not sufficiently process graph structures. Further, DANN can not incorporate the label information. Thus, DANN lacks the graph information as regularization, and the learned common feature representation could suffer from overfitting. GCAN considers the graph structure over instances. In this setting, it studies the temporal correlations between samples. However, the continuous change of system states encourage us to consider an interval of samples rather than the structures among samples. Thus, GCAN does not perform well. CDNE projects networked data into a common feature space under network embedding, but the embedding vector space is restricted to be the same for minimizing the distribution discrepancy. Thus, there is an information loss for DA between

systems of different sizes. In contrast, I design graph kernels to maximally preserve the original information and achieve better performances.

4.6.3 Ablation Study

In Section 4.6.2, for power systems, I utilize GNA with graph coarsening to process datasets of node/edge measurements and labels. In this subsection, I conduct an ablation study to understand the effects of different factors. Specifically, I test the proposed models by independently removing the following factors as comparisons: (1) graph coarsening, (2) node measurements, (3) node labels, (4) edge measurements, and (5) edge labels.

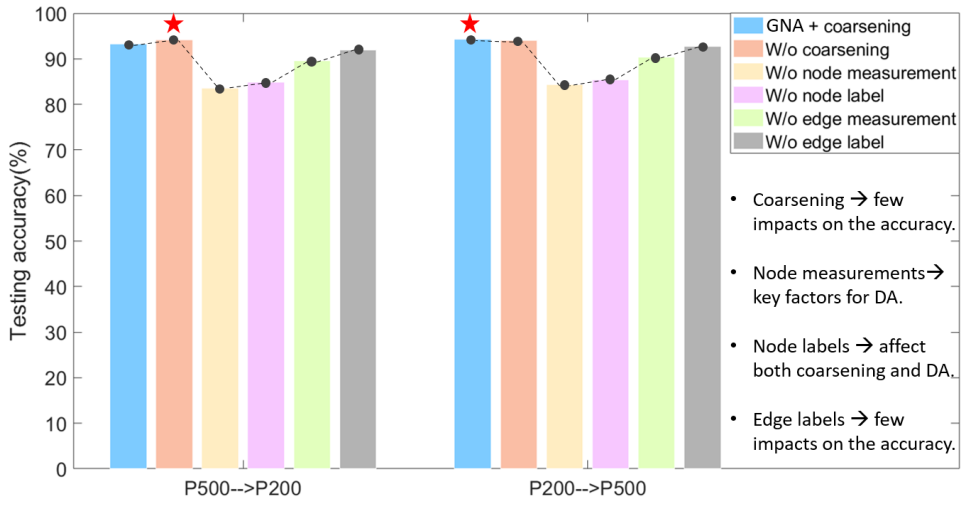
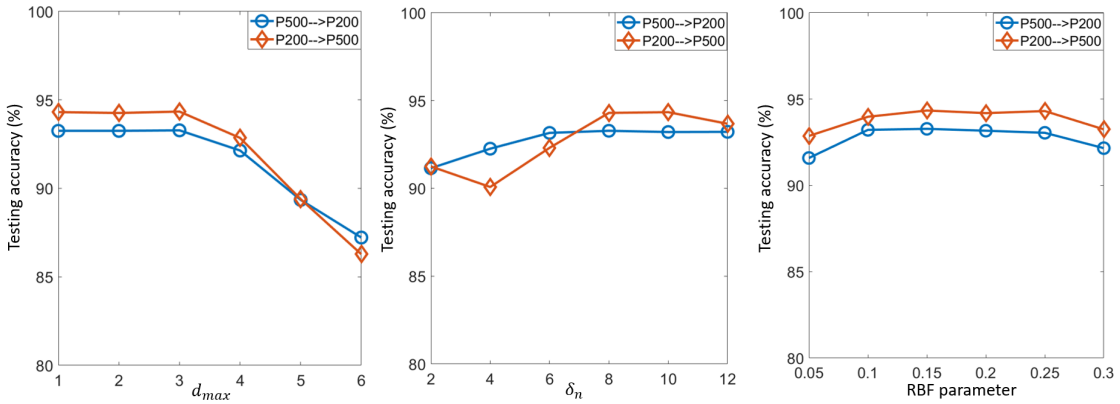


Figure 22. The ablation study for GNA and coarsening.

The result is shown in Fig. 22. I have the following observations. Firstly, if I compare the model with and without coarsening, I find the accuracy does not have a significant change. This is because the proposed coarsening process can correctly concentrate graphs based on the local structures and the label information, largely

saving the graph properties. For the scenario of $P200 \rightarrow P500$, the coarsening even helps to refine the raw data and improves the performance slightly. However, the running time of computing the complete kernel matrix for the original graphs and the coarser graphs is 19.98min and **0.23**min, respectively. This implies that the coarsening process significantly reduces the computational time.

4.6.4 Sensitivity Analysis



(a) Accuracy w.r.t. d_{max}. (b) Accuracy w.r.t. δ_n. (c) Accuracy w.r.t. k_{rbf}.

Figure 23. Results of the sensitivity analysis with respect to different hyper-parameters.

In this subsection, I study the model sensitivity with respect to different hyper-parameters. In particular, I investigate the threshold $d_{max} \in \{1, 2, 3, 4, 5, 6\}$ for spatial correlations, the interval range $\delta_n \in \{2, 4, 6, 8, 10, 12\}$ for temporal correlations, and the parameter of the RBF kernel k_{rbf} in the range of $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$. It is noteworthy that the polynomial kernel generally performs worse than the RBF kernel. Thus, I only report the results of RBF kernel to save space. Fig. 23 demonstrates the final results. Specifically, Fig. 23a shows that when $d_{max} \leq 3$, the accuracy has a small oscillation. Under this distance, nodes can be well grouped. Especially, the node

label information can effectively avoid over-grouping and keep the node separate by similarity. When $d_{max} \geq 4$, the accuracy decreases as d_{max} increases. This is because when d_{max} is too large, many non-similar and non-local nodes are aggregated together, which deteriorates the coarsening process.

Fig. 23b shows that when $6 \leq \delta_n \leq 10$, the temporal correlations are well-captured, and GNA model can achieves the best performance. However, when $\delta_n < 6$, the calculated RBF kernels can not consider the correct system changes to identify system labels. Further, they may be non-robust with respect to measurement noises or anomalies. When $\delta_n > 10$, the incorporation of state change may be too long and prevent the model from understanding system dynamics. Thus, δ_n should be set in a proper range to obtain high performances. Fig. 23c implies that GNA is robust to the change of the RBF kernel parameter: a wide range of parameter values (e.g., $0.1 \sim 0.25$) can lead to a relatively good performance.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis, I propose a systematic framework to extract system knowledge for a normal system, a system with dynamic events, and a new system with limited data. For each of the scenario, I provide novel and principled solutions with theoretical proofs and numerical validations. The results demonstrates that my proposed models gain better performances than cutting-edge methods. More specifically, the solution for each scenario is summarized as follows.

In Chapter 2, I develop new approaches to learn system equations of a normal system for fully or partially observable grids. The general goal is to learn a model that has certain physical consistency to the system. To achieve the consistency, I find that the convexification of the original NP-hard problem is a possible way. To learn the ground-truth physical equation in fully-observable systems, I show how to convexify two sub problems and yield global optimal results that correspond to the true physics. To gain a physics-consistent Machine Learning (ML) model in partially-observable systems, I show how to enforce convexified initialization to encourage consistency.

In Chapter 3, I investigate the problem of identify system event information under different data settings. Correspondingly, frameworks based on Supervised Learning, Semi-Supervised Learning, and Unsupervised Learning are proposed. The essential points lie in (1) the ensemble idea to employ benefits of advanced ML models for a better prediction, (2) physics-guided labeling processes for unlabeled data, and (3)

tensor-based framework to enable fast computations and inference for high-volume datasets.

In Chapter 4, I develop Transfer Learning (TL) techniques to transfer knowledge across different physical grids. Particularly, I find 3 key properties of physical systems prevent the efficiency of traditional TL methods: network structure, complex spatial-temporal correlations, and physical characters of different components. Thus, I build a unified model to incorporate above information and achieve significant improvements.

5.2 Future Work

In the future, the following aspects should be further investigated to make more contributions to the engineering and academic domains.

1. To learn symbolic equations of physical systems, the coherency of input data may cause the issue of not able to identify the system theoretically. Thus, further investigations must be made to study the identifiability of the physical system.
2. For partially-observable systems, current work assumes the connections and parameters within observed nodes are unchanged for a reduced grid. More studies should be made to investigate the performance when the inner connections and parameters can change.
3. Obtaining the inverse of the physics-consistent ML model is a promising direction with various engineering applications.
4. With the learned physics-consistent ML model, the next step is to achieve better planning, economic dispatch, and control of the system.
5. The proposed ensemble framework for event identification requires more theoretical validations.

6. More types of tensor decomposition techniques should be investigated to improve the tensor-based event identification.
7. The knowledge transfer in the proposed GNA model is currently based on an optimization model. However, Deep Learning techniques like Graph Neural Network (GNN) may bring more capacities to find better domain-invariant features. Thus, the next step is to combine GNN and our GNA model for the next generation's graph-level knowledge transfer.

REFERENCES

Amos, Brandon, Lei Xu, and J Zico Kolter. 2017. "Input convex neural networks." In *International Conference on Machine Learning,* 146–155. PMLR.

Baker, Bowen, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. "Designing neural network architectures using reinforcement learning." *arXiv preprint arXiv:1611.02167.*

Berthelot, David, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. 2019. "Mixmatch: A holistic approach to semi-supervised learning." *Advances in Neural Information Processing Systems.*

Bertsekas, Dimitri. 2015. *Convex optimization algorithms.* Athena Scientific.

Bhela, S., V. Kekatos, and S. Veeramachaneni. 2018. "Enhancing Observability in Distribution Grids Using Smart Meter Data." *IEEE Transactions on Smart Grid* 9, no. 6 (November): 5953–5961. https://doi.org/10.1109/TSG.2017.2699939.

Biggio, Luca, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. "Neural Symbolic Regression that Scales." In *International Conference on Machine Learning,* 936–945. PMLR.

Bordel, Borja, Diego Sánchez De Rivera, and Ramón Alcarria. 2016. "Plug-and-play transducers in cyber-physical systems for device-driven applications." In *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS),* 316–321. IEEE.

Borgwardt, Karsten M, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. "Protein function prediction via graph kernels." *Bioinformatics* 21 (suppl_1): i47–i56.

Brahma, S., R. Kavasseri, H. Cao, N. R. Chaudhuri, T. Alexopoulos, and Y. Cui. 2017. "Real-Time Identification of Dynamic Events in Power Systems Using PMU Data, and Potential Applications, Models, Promises, and Challenges." *IEEE Trans. on Power Delivery* 32, no. 1 (February): 294–301. https://doi.org/10.1109/TPWRD.2016.2590961.

Brunton, Steven L, Joshua L Proctor, and J Nathan Kutz. 2016. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems." *Proceedings of the national academy of sciences* 113 (15): 3932–3937.

Cai, Chen, Dingkang Wang, and Yusu Wang. 2021. "Graph Coarsening with Neural Networks." *arXiv preprint arXiv:2102.01350.*

Champion, Kathleen, Bethany Lusch, J Nathan Kutz, and Steven L Brunton. 2019. "Data-driven discovery of coordinates and governing equations." *Proceedings of the National Academy of Sciences* 116 (45): 22445–22451.

Chen, Zhao, Yang Liu, and Hao Sun. 2021. "Physics-informed learning of governing equations from scarce data." *Nature communications* 12 (1): 1–13.

Cui, M., J. Wang, J. Tan, A. Florita, and Y. Zhang. 2018. "A Novel Event Detection Method Using PMU Data with High Precision." *IEEE Trans. on Power Systems,* https://doi.org/10.1109/TPWRS.2018.2859323.

Dai, Quanyu, Xiao-Ming Wu, Jiaren Xiao, Xiao Shen, and Dan Wang. 2022. "Graph Transfer Learning via Adversarial Domain Adaptation with Graph Convolution." *IEEE Transactions on Knowledge and Data Engineering.*

Davis, Timothy A, and Yifan Hu. 2011. "The University of Florida sparse matrix collection." *ACM Transactions on Mathematical Software (TOMS)* 38 (1): 1–25.

De Yong, David, Sudipto Bhowmik, and Fernando Magnago. 2015. "An effective power quality classifier using wavelet transform and support vector machines." *Expert Systems with Applications.*

Du, Boxin, and Hanghang Tong. 2018. "Fasten: Fast sylvester equation solver for graph mining." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 1339–1347.

Engineering Texas A&M University. 2016a. "Illinois 200-Bus System: ACTIVSg200," https://electricgrids.engr.tamu.edu/electric-grid-test-cases/activsg200/.

———. 2016b. "SouthCarolina 500-Bus System: ACTIVSg500," https://electricgrids.engr.tamu.edu/electric-grid-test-cases/activsg500/.

Fajardo, O. F., and A. Vargas. 2008. "Reconfiguration of MV Distribution Networks With Multicost and Multipoint Alternative Supply, Part II: Reconfiguration Plan." *IEEE Transactions on Power Systems* 23, no. 3 (August): 1401–1407. https://doi.org/10.1109/TPWRS.2008.926702.

Fan, Jianqing, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. 2020. "A theoretical analysis of deep Q-learning." In *Learning for Dynamics and Control,* 486–489. PMLR.

Fang, Meng, Jie Yin, and Xingquan Zhu. 2013. "Transfer learning across networks for collective classification." In *2013 IEEE 13th International Conference on Data Mining,* 161–170. IEEE.

Fletcher, Roger. 2013. *Practical methods of optimization.* John Wiley & Sons.

Fu, Chenbo, Yongli Zheng, Yi Liu, Qi Xuan, and Guanrong Chen. 2019. "Nes-tl: Network embedding similarity-based transfer learning." *IEEE Transactions on Network Science and Engineering* 7 (3): 1607–1618.

Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016a. "Domain-adversarial training of neural networks." *The journal of machine learning research* 17 (1): 2096–2030.

———. 2016b. "Domain-adversarial training of neural networks." *The journal of machine learning research* 17 (1): 2096–2030.

General Electric Energy Consulting. 2018. "General Electric Concorda PSLF," https://www.geenergyconsulting.com/practice-area/software-products/pslf.

Gretton, Arthur, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. "A kernel two-sample test." *Journal of Machine Learning Research.*

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 770–778.

He, Lifang, Chun-Ta Lu, Guixiang Ma, Shen Wang, Linlin Shen, S Yu Philip, and Ann B Ragin. 2017. "Kernelized support tensor machines." In *International Conference on Machine Learning.*

Hester, Todd, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. "Deep q-learning from demonstrations." In *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 32. 1.

Hu, Xinyue, Haoji Hu, Saurabh Verma, and Zhi-Li Zhang. 2020. "Physics-Guided Deep Neural Networks for PowerFlow Analysis." *arXiv preprint arXiv:2002.00097.*

Iqbal, Kashif, Muhammad Adnan Khan, Sagheer Abbas, Zahid Hasan, and Areej Fatima. 2018. "Intelligent transportation system (ITS) for smart-cities using

Mamdani fuzzy inference system." *International journal of advanced computer science and applications* 9 (2).

Jabr, R. A. 2014. "Minimum loss operation of distribution networks with photovoltaic generation." *IET Renewable Power Generation* 8, no. 1 (January): 33–44. https://doi.org/10.1049/iet-rpg.2012.0213.

Kim, Do-In, Tae Yoon Chun, Sung-Hwa Yoon, Gyul Lee, and Yong-June Shin. 2015. "Wavelet-based event detection method using PMU data." *IEEE Transactions on Smart Grid.*

Kolda, T., and B. Bader. 2009. "Tensor Decompositions and Applications." *Journal of Applied Mathematics.*

Kung, Sun Yuan. 2014. *Kernel methods and machine learning.* Cambridge University Press.

Lee, Jaekoo, Hyunjae Kim, Jongsun Lee, and Sungroh Yoon. 2017. "Transfer learning for deep learning on graph-structured data." In *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 31. 1.

Lew, D., M. Asano, J. Boemer, C. Ching, U. Focken, R. Hydzik, M. Lange, and A. Motley. 2017. "The Power of Small: The Effects of Distributed Energy Resources on System Reliability." *IEEE Power and Energy Magazine* 15, no. 6 (November): 50–60. https://doi.org/10.1109/MPE.2017.2729104.

Li, Haoran, Zhihao Ma, and Yang Weng. 2022. "A Transfer Learning Framework for Power System Event Identification." *IEEE Transactions on Power Systems,* 1–1. https://doi.org/10.1109/TPWRS.2022.3153445.

Li, Haoran, and Yang Weng. 2021. "Physical Equation Discovery Using Physics-Consistent Neural Network (PCNN) Under Incomplete Observability." In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining,* 925–933.

Li, Haoran, Yang Weng, Evangelos Farantatos, and Mahendra Patel. 2019a. "A hybrid machine learning framework for enhancing PMU-based event identification with limited labels." In *IEEE International Conference on Smart Grid Synchronized Measurements and Analytics.*

———. 2019b. "An unsupervised learning framework for event detection, type identification and localization using PMUs without any historical labels." In *IEEE Power & Energy Society General Meeting.*

Li, Haoran, Yang Weng, Yizheng Liao, Brian Keel, and Kenneth E Brown. 2021. "Distribution grid impedance & topology estimation with limited or no micro-PMUs." *International Journal of Electrical Power & Energy Systems* 129:106794.

Li, Haoran, Yang Weng, and Hanghang Tong. 2020. "Heterogeneous Transfer Learning on Power Systems: A Merged Multi-modal Gaussian Graphical Model." In *2020 IEEE International Conference on Data Mining (ICDM),* 1088–1093. https://doi.org/10.1109/ICDM50108.2020.00130.

Liao, Mang, Di Shi, Zhe Yu, Zhehan Yi, Zhiwei Wang, and Yingmeng Xiang. 2018. "An alternating direction method of multipliers based approach for PMU data recovery." *IEEE Transactions on Smart Grid.*

Liao, Yizheng, Yang Weng, Guangyi Liu, and Ram Rajagopal. 2018. "Urban mv and lv distribution grid topology estimation via group lasso." *IEEE Transactions on Power Systems* 34 (1): 12–27. https://doi.org/10.1109/TPWRS.2018.2868877.

Liu, Yin, and Vincent Chen. 2018. "On the Generalization Effects of DenseNet Model Structures."

Lock, Eric F. 2018. "Tensor-on-Tensor Regression." *Journal of Computational and Graphical Statistics.*

Long, Mingsheng, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. 2013. "Transfer feature learning with joint distribution adaptation." In *Proceedings of the IEEE international conference on computer vision,* 2200–2207.

Long, Mingsheng, Han Zhu, Jianmin Wang, and Michael I Jordan. 2016. "Unsupervised Domain Adaptation with Residual Transfer Networks." In *NIPS.*

Lopes, João A Peças, Filipe Joel Soares, and Pedro M Rocha Almeida. 2010. "Integration of electric vehicles in the electric power system." *Proceedings of the IEEE* 99 (1): 168–183.

Loukas, Andreas. 2019. "Graph Reduction with Spectral and Cut Guarantees." *J. Mach. Learn. Res.* 20 (116): 1–42.

Lu, Qiang, Jun Ren, and Zhiguang Wang. 2016. "Using genetic programming with prior formula knowledge to solve symbolic regression problem." *Computational intelligence and neuroscience* 2016.

Lundberg, Scott M, and Su-In Lee. 2017. "A unified approach to interpreting model predictions." In *Advances in neural information processing systems,* 4765–4774.

Ma, Xinhong, Tianzhu Zhang, and Changsheng Xu. 2019. "Gcan: Graph convolutional adversarial network for unsupervised domain adaptation." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 8266–8276.

Martius, Georg, and Christoph H Lampert. 2016. "Extrapolation and learning equations." *arXiv preprint arXiv:1610.02995.*

MATPOWER community. 2020. "MATPOWER." Https://matpower.org/.

Milne, Tristan. 2019. "Piecewise Strong Convexity of Neural Networks." In *Advances in Neural Information Processing Systems,* edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/b33128cb0089003ddfb5199e1b679652-Paper.pdf.

Miraz, Mahdi H, Maaruf Ali, Peter S Excell, and Rich Picking. 2015. "A review on Internet of Things (IoT), Internet of everything (IoE) and Internet of nano things (IoNT)." *2015 Internet Technologies and Applications (ITA),* 219–224.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. "Human-level control through deep reinforcement learning." *nature* 518 (7540): 529–533.

Mundhenk, T Nathan, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. 2021. "Symbolic Regression via Neural-Guided Genetic Programming Population Seeding." *arXiv preprint arXiv:2111.00053.*

Orzechowski, Patryk, William La Cava, and Jason H Moore. 2018. "Where are we now? A large benchmark study of recent symbolic regression methods." In *Proceedings of the Genetic and Evolutionary Computation Conference,* 1183–1190.

Pan, Sinno Jialin, James T Kwok, Qiang Yang, et al. 2008. "Transfer learning via dimensionality reduction." In *AAAI,* 8:677–682.

Pan, Xiang, Minghua Chen, Tianyu Zhao, and Steven H Low. 2020. "Deepopf: A feasibility-optimized deep neural network approach for ac optimal power flow problems." *arXiv preprint arXiv:2007.01002.*

Pérez, Enrique, and Julio Barros. 2008. "A proposal for on-line detection and classification of voltage events in power systems." *IEEE Transactions on Power Delivery.*

Petersen, Brenden K, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. 2019. "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients." *arXiv preprint arXiv:1912.04871.*

PJM Interconnection LLC. 2018. "Metered Load Data." Https://dataminer2.pjm.com/feed/hrl_load_metered/definition.

Rasmus, Antti, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. 2015. "Semi-supervised learning with ladder networks." *arXiv.*

Sahoo, Subham, Christoph Lampert, and Georg Martius. 2018. "Learning Equations for Extrapolation and Control." In *Proceedings of the 35th International Conference on Machine Learning,* edited by Jennifer Dy and Andreas Krause, 80:4442–4450. Proceedings of Machine Learning Research. PMLR, October. https://proceedings.mlr.press/v80/sahoo18a.html.

Salih, Rania, Elmustafa Sayed Ali Ahmed, and Rashid A Saeed. 2021. "Machine learning in cyber-physical systems in industry 4.0." In *Artificial Intelligence Paradigms for Smart Cyber-Physical Systems,* 20–41. IGI Global.

Schaft, Arjan van der. 2017. "Modeling of physical network systems." *Systems & Control Letters* 101:21–27.

Schölkopf, Bernhard, Ralf Herbrich, and Alex J Smola. 2001. "A generalized representer theorem." In *International conference on computational learning theory.* Springer.

Shen, Xiao, Quanyu Dai, Sitong Mao, Fu-Lai Chung, and Kup-Sze Choi. 2021. "Network Together: Node Classification via Cross-Network Deep Network Embedding." *IEEE Transactions on Neural Networks and Learning Systems* 32 (5): 1935–1948. https://doi.org/10.1109/TNNLS.2020.2995483.

Shi, Di. 2012. *Power system network reduction for engineering and economic analysis.* Arizona State University.

Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje. 2017. "Learning important features through propagating activation differences." *arXiv preprint arXiv:1704.02685.*

Sidiropoulos, Nicholas D, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. 2017. "Tensor decomposition for signal processing and machine learning." *IEEE Transactions on Signal Processing.*

Smola, Alex J, and Bernhard Schölkopf. 1998. *Learning with kernels.* Vol. 4. Citeseer.

Sohn, Kihyuk, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. 2020. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in Neural Information Processing Systems.*

Song, Le. 2008. "Learning via Hilbert space embedding of distributions."

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15 (1): 1929–1958.

Udrescu, Silviu-Marian, and Max Tegmark. 2020. "AI Feynman: A physics-inspired method for symbolic regression." *Science Advances* 6 (16): eaay2631.

Vishwanathan, S Vichy N, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. "Graph kernels." *Journal of Machine Learning Research* 11:1201–1242.

Wang, Dawei, Kedi Zheng, Qixin Chen, Xuan Zhang, and Gang Luo. 2020. "A data-driven probabilistic power flow method based on convolutional neural networks." *International Transactions on Electrical Energy Systems* 30 (7): e12367.

Weng, Y., Y. Liao, and R. Rajagopal. 2017. "Distributed Energy Resources Topology Identification via Graphical Modeling." *IEEE Transactions on Power Systems* 32, no. 4 (July): 2682–2694. https://doi.org/10.1109/TPWRS.2016.2628876.

Werner, Matthias, Andrej Junginger, Philipp Hennig, and Georg Martius. 2021. "Informed Equation Learning." *arXiv preprint arXiv:2105.06331.*

Wu, Fangzhao, and Yongfeng Huang. 2016. "Sentiment domain adaptation with multiple sources." In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 301–310.

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. "A comprehensive survey on graph neural networks." *IEEE transactions on neural networks and learning systems* 32 (1): 4–24.

Yu, J., Y. Weng, and R. Rajagopal. 2017. "Robust mapping rule estimation for power flow analysis in distribution grids." In *2017 North American Power Symposium (NAPS),* 1–6.

Yu, Jiafan, Yang Weng, and Ram Rajagopal. 2017. "Mapping rule estimation for power flow analysis in distribution grids." *arXiv preprint arXiv:1702.07948.*

Yuan, Yuxuan, Yifei Guo, Kaveh Dehghanpour, Zhaoyu Wang, and Yanchao Wang. 2021. "Learning-Based real-time event identification using rich real PMU data." *IEEE Transactions on Power Systems.*

Yuan, Yuxuan, Zhaoyu Wang, and Yanchao Wang. 2020. "Learning Latent Interactions for Event Identification via Graph Neural Networks and PMU Data." *arXiv preprint arXiv:2010.01616.*

Zhang, Mi, and Alexander A. Sawchuk. 2012. "USC-HAD: A Daily Activity Dataset for Ubiquitous Activity Recognition Using Wearable Sensors." In *ACM International Conference on Ubiquitous Computing (Ubicomp) Workshop on Situation, Activity and Goal Awareness (SAGAware).* Pittsburgh, Pennsylvania, USA, September.

Zhang, S., Y. Wang, M. Liu, and Z. Bao. 2018. "Data-Based Line Trip Fault Prediction in Power Systems Using LSTM Networks and SVM." *IEEE Access* 6:7675–7686. https://doi.org/10.1109/ACCESS.2017.2785763.

Zhao, Tuo, Mo Yu, Yiming Wang, Raman Arora, and Han Liu. 2014. "Accelerated mini-batch randomized block coordinate descent method." *Advances in Neural Information Processing Systems.*

APPENDIX A

PROOFS OF PROPOSED THEOREMS

A.1 Proofs of Theorem 1

Proof. The Bellman Equation of $Q^*(\cdot)$ is:

$$-Q^*(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) = -\mathbb{E}[R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) + \gamma \max_{\boldsymbol{a}} Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})] = -R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k) - \gamma \max_{\tilde{\boldsymbol{a}}} Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}}), \tag{A.1}$$

where the second equality holds since our state transitions are deterministic by Equation (2.2). I prove the convexity from the induction method. When $k = K-1$, the $(k+1)^{th}$ state is the terminal state without action selections. Thus, I have

$$-Q^*(\boldsymbol{s}_{K-1}, \tilde{\boldsymbol{a}}_{K-1}) = -R(\boldsymbol{s}_{K-1}, \tilde{\boldsymbol{a}}_{K-1}).$$

Since $-R(\cdot)$ is an ICNN and is convex in input, $-Q^*(\boldsymbol{s}_{K-1}, \tilde{\boldsymbol{a}}_{K-1})$ is convex in $\boldsymbol{s}_{K-1}$ and $\tilde{\boldsymbol{a}}_{K-1}$.

When $0 \leq k < K-1$ and assume $-Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}}_{k+1})$ is convex in $\boldsymbol{s}_{k+1}$ and $\tilde{\boldsymbol{a}}_{k+1}$, I have $-\max_{\tilde{\boldsymbol{a}}} Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}}) = \min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ is convex in $\boldsymbol{s}_{k+1}$ given the fixed optimal action. Let $\boldsymbol{H}$ denote the Hessian matrix of $\min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ with respect to $\boldsymbol{s}_{k+1}$. Due to the convexity, $\boldsymbol{H}$ is positive semi-definite. Thus, by Equation (2.2) and the chain rule, the Hessian matrix of $\min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ with respect to $\boldsymbol{s}_k$ can be written as:

$$\boldsymbol{H}' = (\boldsymbol{Z}_k')^\top \boldsymbol{H} \boldsymbol{Z}_k'.$$

$\boldsymbol{H}'$ is also positive semi-definite. Therefore, $\min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ is convex in $\boldsymbol{s}_k$. Since $-R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\boldsymbol{s}_k$, $-Q^*(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\boldsymbol{s}_k$.

Similarly, vectorizing the state transition equation can give:

$$\boldsymbol{s}_{k+1} = (\boldsymbol{s}_k^\top \bigotimes \boldsymbol{I}_{n_s}) \boldsymbol{a}_k',$$

where $\boldsymbol{I}_{n_s}$ is the $n_s \times n_s$ identity matrix and $\bigotimes$ is the Kronecker product. $\boldsymbol{a}_k' = [(\boldsymbol{a}_k)^\top, \boldsymbol{0}]^\top$ is the concatenation of the discrete action $\boldsymbol{a}_k$ and a zero vector to maintain the fixed dimensionality of action vectors. With similar proofs based on the Hessian matrix and the fact that $-Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}}_k)$ is convex in $\boldsymbol{s}_{k+1}$, I have $\min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ is convex in $\boldsymbol{a}_k'$ and also $\boldsymbol{a}_k$. Subsequently, arbitrary $\tilde{\boldsymbol{a}}_k \in \text{conv}(\{0,1\}^{n_a})$ can be written as a convex combination of the discrete actions $\boldsymbol{a}_k$. Thus, $\min_{\tilde{\boldsymbol{a}}} -Q^*(\boldsymbol{s}_{k+1}, \tilde{\boldsymbol{a}})$ is convex in $\tilde{\boldsymbol{a}}_k$. Since $-R(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\tilde{\boldsymbol{a}}_k$, $-Q^*(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\tilde{\boldsymbol{a}}_k$. Eventually, $-Q^*(\boldsymbol{s}_k, \tilde{\boldsymbol{a}}_k)$ is convex in $\boldsymbol{s}_k$ and $\tilde{\boldsymbol{a}}_k$, which concludes the proof. □

A.2 Proofs of Theorem 2

Proof. If $f^*(\cdot; W)$ can't represent the exact equations, there are two cases: (1) the structure of $f^*(\cdot; W)$ is correct to represent the equations, but the learned weights

W^* don't represent the symbol coefficients, and (2) the structure of $f^*(\cdot; W)$ can't represent the equations. Case (1) doesn't hold since I assume W^* is the global optimal weights for noiseless data. If case (2) holds, $\exists 0 \leq j \leq K-1$, $\boldsymbol{b}_j^* = \min_{\tilde{\boldsymbol{a}}_j} -Q^*(\boldsymbol{s}_j, \tilde{\boldsymbol{a}}_j)$ and $\boldsymbol{b}_j^*$ doesn't represent the symbol connections in the underlying equations. Further, I assume $\forall 0 \leq i < j$, $\boldsymbol{a}_i^* = \min_{\tilde{\boldsymbol{a}}_i} -Q(\boldsymbol{s}_i, \tilde{\boldsymbol{a}}_i)$ and $\boldsymbol{a}_i^*$ represents the true connections.

If $j = K-1$, Equation (A.1) implies that $\tilde{\boldsymbol{a}}_j^* = \min_{\tilde{\boldsymbol{a}}_j} -Q^*(\boldsymbol{s}_j, \tilde{\boldsymbol{a}}_j) = \arg\min_{\tilde{\boldsymbol{a}}} -R(\boldsymbol{s}_j, \tilde{\boldsymbol{a}})$. Since $-R(\boldsymbol{s}_j, \tilde{\boldsymbol{a}})$ is convex in $\tilde{\boldsymbol{a}}$, I know the discrete version of $\tilde{\boldsymbol{a}}_j^*$, namely $\boldsymbol{a}_j^*$, represents the true connection of the last layer for the underlying equations. Otherwise, the reward is not maximized. However, by definition of $\boldsymbol{b}_j^*$, $\boldsymbol{b}_j^* \neq \boldsymbol{a}_j^*$.

If $j < K-1$, Equation (A.1) implies:

$$\begin{aligned}\min_{\tilde{\boldsymbol{a}}_j} -Q^*(\boldsymbol{s}_j, \tilde{\boldsymbol{a}}_j) = \min_{\tilde{\boldsymbol{a}}_j} -R(\boldsymbol{s}_j, \tilde{\boldsymbol{a}}_j) + \gamma \min_{\tilde{\boldsymbol{a}}_j} \min_{\tilde{\boldsymbol{a}}_{j+1}} -R\big(\boldsymbol{s}_{j+1}(\tilde{\boldsymbol{a}}_j), \tilde{\boldsymbol{a}}_{j+1}\big) \\ + \cdots + \gamma^{K-1-j} \min_{\tilde{\boldsymbol{a}}_j} \cdots \min_{\tilde{\boldsymbol{a}}_{K-1}} -R(\boldsymbol{s}_{K-1}(\tilde{\boldsymbol{a}}_j, \cdots, \tilde{\boldsymbol{a}}_{K-2}), \tilde{\boldsymbol{a}}_{K-1}).\end{aligned} \tag{A.2}$$

By definition of $\boldsymbol{b}_j^*$, $\boldsymbol{b}_j^*$ is not the solution of Equation (A.2). This is because $\boldsymbol{b}_j^*$ can't achieve the minimum value for each summation term on the right hand side of Equation (A.2), according to the convexity of the reward function. In general, $\boldsymbol{b}_j^* \neq \min_{\tilde{\boldsymbol{a}}_j} -Q^*(\boldsymbol{s}_j, \tilde{\boldsymbol{a}}_j)$, which contradicts the definition of $\boldsymbol{b}_j^*$. Thus, $\boldsymbol{b}_j^*$ doesn't exist. Therefore, case (2) doesn't hold and $f^*(\cdot; W^*)$ represents the exact equations. □

A.3 Proofs of Theorem 3

Proof. To simplify the proof, I consider scalar output of the LoCaL, i.e., one equation, and the proof can be easily extended to the multi-output case. I follow the idea of (Milne 2019) to study the second derivative of LoCaL with perturbations. Let $\hat{y}(\boldsymbol{x}, W)$ denote the LoCaL with input to be $\boldsymbol{x}$ and the weight set to be W. Let X be a perturbation direction of W and t be a small step size. For the i^{th} noiseless instance $(\boldsymbol{x}^i, y^i)$, I denote $e(\boldsymbol{x}^i, W + tX) = \hat{y}(\boldsymbol{x}^i, W + tX) - y^i$. Obviously, the loss function can be written as $L(W + tX) = \frac{1}{2N} \sum_{i=1}^{N} (e(\boldsymbol{x}^i, W + tX))^2$. Then, I can calculate the second-order derivative based on the chain rule:

$$\begin{aligned}\frac{d^2}{dt^2}\big|_{t=0} L(W + tX) =& \frac{1}{N} \frac{d}{dt}\big|_{t=0} \sum_{i=1}^{N} e(\boldsymbol{x}^i, W + tX) \frac{d}{dt} \hat{y}(\boldsymbol{x}^i, W + tX), \\ =& \frac{1}{N} \sum_{i=1}^{N} \big(\frac{d}{dt}\big|_{t=0} \hat{y}(\boldsymbol{x}^i, W + tX)\big)^2 \\ &+ e(\boldsymbol{x}^i, W) \frac{d^2}{dt^2}\big|_{t=0} \hat{y}(\boldsymbol{x}^i, W + tX).\end{aligned} \tag{A.3}$$

Next, I denote the global optimal solution to be W^*. Based on the Assumptions (3) and (4), $\forall i, \hat{y}(\boldsymbol{x}^i, W^*) = g(\boldsymbol{x}^i) = y^i$. Therefore, I have $\frac{d^2}{dt^2}\big|_{t=0} L(W^* + tX) =$

$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W^*)\right)^2 > 0$, where the inequality strictly holds. This is because by Assumptions (3), $\hat{y}(\boldsymbol{x}, W^*)$ can be mathematically simplified to obtain $g(\boldsymbol{x})$. Then, by Assumption (2), $\frac{1}{N}\sum_{i=1}^{N}\left(\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W^*)\right)^2 > 0$. Finally, by Assumption (1) and (3), $\frac{d^2}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W + tX)$ is bounded and there is a local region around W^* such that $\frac{d^2}{dt^2}\big|_{t=0}L(W + tX) > 0$, which concludes the proof. □

A.4 Proofs of Theorem 4

Proof. For the target LocAL, I similarly consider the scalar output and write the function analytically:

$$\hat{y}(\boldsymbol{x}, W) = \boldsymbol{W}_1^\top \Psi\big(\Phi(\boldsymbol{W}_0^\top \boldsymbol{x})\big), \tag{A.4}$$

where $\boldsymbol{W}_0 \in \mathbb{R}^{n_0 \times n_1}$ is the weight matrix for activation, $\Phi : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1}$ represents the activation with symbol functions like x^2, $cos(x)$, and $\log(x)$, etc. $\Psi : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ is the function to select some activated neurons for multiplications, and $\boldsymbol{W}_1 \in \mathbb{R}^{n_2 \times n_3}$ ($n_3 = 1$) represents the weight for summation. I rewrite Equation (A.4) with the help of exponential and logarithm mappings.

$$\hat{y}(\boldsymbol{x}, W) = \boldsymbol{W}_1^\top \exp\left(\boldsymbol{S}^\top \log(\Phi(\boldsymbol{W}_0^\top \boldsymbol{x}))\right), \tag{A.5}$$

where $\boldsymbol{S} \in \mathbb{R}^{n_1 \times n_2}$ represents a selection matrix such that $\boldsymbol{S}[i, j] = 1$ if and only if the i^{th} neuron is selected as the multiplicative factor for the j^{th} neuron in the multiplication layer. Given the fixed structure of $\hat{y}(\cdot)$ from the deep Q-learning, $\boldsymbol{S}$ is a known matrix. $\log(\cdot)$ and $\exp(\cdot)$ represent the element-wise logarithm and exponential functions. Notably, the corresponding element in $\Phi(\boldsymbol{W}_0^\top \boldsymbol{x})$ should be positive in Equation (A.5). If there are negative entries, one can utilize $\boldsymbol{W}_1^\top \boldsymbol{s} \circ \exp\left(\boldsymbol{S}^\top \log(|\Phi(\boldsymbol{W}_0^\top \boldsymbol{x})|)\right)$ to take place of the right hand side term in Equation (A.5), where $\boldsymbol{s}[i] = (-1)^{n_-^i}$ and $0 \leq n_-^i \leq n_1$ represents the number of negative entries selected for the i^{th} neuron of the multiplication layer. $\circ$ represents the Hadamard product. However, both expressions have the same values and gradients. Thus, I utilize Equation (A.5) in later derivations.

Then, let X be a perturbation direction such that $X = \{\boldsymbol{X}_0, \boldsymbol{X}_1\}$. Thus, for a small step t, I have:

$$\hat{y}(\boldsymbol{x}, W + tX) = (\boldsymbol{W}_1 + t\boldsymbol{X}_1)^\top \exp\left(\boldsymbol{S}^\top \log(\Phi((\boldsymbol{W}_0 + t\boldsymbol{X}_0)^\top \boldsymbol{x}))\right). \tag{A.6}$$

Based on Equation (A.6), I can compute:

$$\begin{aligned}\frac{d}{dt}\hat{y}(\boldsymbol{x}^i, W+tX) = \boldsymbol{X}_1^\top \exp\Big(\boldsymbol{S}^\top \log(\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i))\Big) \\ + (\boldsymbol{W}_1+t\boldsymbol{X}_1)^\top \Bigg[\exp\Big(\boldsymbol{S}^\top \log(\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i))\Big) \\ \circ \boldsymbol{S}^\top \frac{1}{\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i)} \circ \Phi^{'}((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i) \circ \boldsymbol{X}_0^\top \boldsymbol{x}^i\Bigg],\end{aligned} \tag{A.7}$$

where $\frac{1}{\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i)} \in \mathbb{R}^{n_1}$ is the element-wise division and $\Phi^{'}$ is the element-wise first derivative of $\Phi^{'}$. Without special notifications, I assume all the division for vectors is element-wise in the following derivations. Then, I denote

$$\begin{aligned}\boldsymbol{u}(\boldsymbol{x}^i, W+tX) &= \exp\Big(\boldsymbol{S}^\top \log(\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i))\Big), \\ \boldsymbol{v}(\boldsymbol{x}^i, W+tX) &= \boldsymbol{S}^\top \frac{1}{\Phi((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i)} \circ \Phi^{'}((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i) \circ \boldsymbol{X}_0^\top \boldsymbol{x}^i, \\ \boldsymbol{w}(\boldsymbol{x}^i, W+tX) &= \boldsymbol{S}^\top \frac{1}{\Phi'((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i)} \circ \Phi^{''}((\boldsymbol{W}_0+t\boldsymbol{X}_0)^\top \boldsymbol{x}^i) \circ \boldsymbol{X}_0^\top \boldsymbol{x}^i.\end{aligned} \tag{A.8}$$

With above definitions, I can calculate:

$$\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX) = \boldsymbol{X}_1^\top \boldsymbol{u}(\boldsymbol{x}^i, W) + \boldsymbol{W}_1^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W) \circ \boldsymbol{v}(\boldsymbol{x}^i, W)\big]. \tag{A.9}$$

Further, I calculate the second derivative based on Equation (A.7) and the fact that element-wise operations for vectors are commutative:

$$\begin{aligned}&\frac{d}{dt^2}\hat{y}(\boldsymbol{x}^i, W+tX) = \boldsymbol{X}_1^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX)\big] \\ &+ \boldsymbol{X}_1^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX)\big] \\ &+ (\boldsymbol{W}_1+t\boldsymbol{X}_1)^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX)\big] \\ &- (\boldsymbol{W}_1+t\boldsymbol{X}_1)^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX)\big] \\ &+ (\boldsymbol{W}_1+t\boldsymbol{X}_1)^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{v}(\boldsymbol{x}^i, W+tX) \circ \boldsymbol{w}(\boldsymbol{x}^i, W+tX)\big],\end{aligned} \tag{A.10}$$

When $t \to 0$, I have:

$$\begin{aligned}&\frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX) = \qquad\qquad 2\boldsymbol{X}_1^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W) \circ \boldsymbol{v}(\boldsymbol{x}^i, W)\big] + \\ &\boldsymbol{W}_1^\top \big[\boldsymbol{u}(\boldsymbol{x}^i, W) \circ \boldsymbol{v}(\boldsymbol{x}^i, W) \circ \boldsymbol{w}(\boldsymbol{x}^i, W)\big]\end{aligned} \tag{A.11}$$

The above equation can reflect the relationship between the second and the first derivative. However, I first identify the inequality between these two derivatives to enable a strictly convex region.

Let $\hat{\boldsymbol{y}}^{'} = [\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}_1, W+tX), \cdots, \frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}_N, W+tX)]^\top$, $\hat{\boldsymbol{y}}^{''} = [\frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}_1, W+tX), \cdots, \frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}_N, W+tX)]^\top$, and $\boldsymbol{e} = [e(\boldsymbol{x}_1, W), \cdots, e(\boldsymbol{x}_N, W)]^\top$. Equation (A.3) implies that:

$$\begin{aligned}\frac{d^2}{dt^2}\big|_{t=0}L(W+tX) &= \frac{1}{N}(||\hat{\boldsymbol{y}}^{'}||_2^2 + \boldsymbol{e}^\top\hat{\boldsymbol{y}}^{''}) \\ &\geq \frac{1}{N}(||\hat{\boldsymbol{y}}^{'}||_2^2 - ||\boldsymbol{e}||_2||\hat{\boldsymbol{y}}^{''}||_2)\end{aligned} \tag{A.12}$$

To find a region to restrict the convexity, I restrict the lower bound of the second derivative to be positive and compute:

$$||\boldsymbol{e}||_2 < \frac{||\hat{\boldsymbol{y}}^{'}||_2^2}{||\hat{\boldsymbol{y}}^{''}||_2} \tag{A.13}$$

The right hand side of Equation (A.13) can be easily bounded by:

$$\frac{||\hat{\boldsymbol{y}}^{'}||_2^2}{||\hat{\boldsymbol{y}}^{''}||_2} \geq \frac{\sqrt{N}\min(|\hat{\boldsymbol{y}}^{'}|)^2}{\max(|\hat{\boldsymbol{y}}^{''}|)} = \frac{\sqrt{N}\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX)\big|^2}{\big|\frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\big|}, \tag{A.14}$$

where $|\cdot|$ for a vector is to calculate the absolute value for each element of the vector, $i = \arg\min(|\hat{\boldsymbol{y}}^{'}|)$ and $j = \arg\max(|\hat{\boldsymbol{y}}^{''}|)$. Namely, I consider a sufficient condition for convexity.

$$\frac{\sqrt{N}\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX)\big|^2}{\big|\frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\big|} > ||\boldsymbol{e}||_2 \tag{A.15}$$

Next, Equation (A.11) indicates that:

$$\begin{aligned}\big|\frac{d}{dt^2}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\big| =& \Big|\boldsymbol{X}_1^\top\big[\boldsymbol{u}(\boldsymbol{x}^j, W)\circ 2\boldsymbol{v}(\boldsymbol{x}^j, W)\big] \\ &+ \boldsymbol{W}_1^\top\big[\boldsymbol{u}(\boldsymbol{x}^j, W)\circ\boldsymbol{v}(\boldsymbol{x}^j, W)\circ\boldsymbol{w}(\boldsymbol{x}^j, W)\big]\Big| \\ \leq& \eta\Big(\Big|\boldsymbol{X}_1^\top\boldsymbol{u}(\boldsymbol{x}^j, W) + \boldsymbol{W}_1^\top\big[\boldsymbol{u}(\boldsymbol{x}^j, W)\circ\boldsymbol{v}(\boldsymbol{x}^j, W)\big]\Big|\Big) \\ =& \eta\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\big|,\end{aligned} \tag{A.16}$$

where η is a positive constant. Note that $\eta < \infty$ by Assumptions (1) and (2) in Theorem 3. Therefore, I have the following sufficient condition to make $\frac{d^2}{dt^2}\big|_{t=0}L(W+tX) > 0$ always hold.

$$\frac{\sqrt{N}\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W+tX)\big|^2}{\eta\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W+tX)\big|} > \sqrt{N}|\hat{y}(\boldsymbol{x}^k, W) - y^k| \geq ||\boldsymbol{e}||_2, \tag{A.17}$$

where $k = \arg\max(|\boldsymbol{e}|)$. The above equation leads to a set U of local regions that have strong convexity. Namely,

$$U = \{W \Big| \frac{\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^i, W + tX)\big|^2}{\eta\big|\frac{d}{dt}\big|_{t=0}\hat{y}(\boldsymbol{x}^j, W + tX)\big|} > |\hat{y}(\boldsymbol{x}^k, W) - y^k|\}. \tag{A.18}$$

Clearly, the global optimal solution $W^* \in U$ since $\hat{y}(\boldsymbol{x}^k, W^*) - y^k = 0$. Note that there may be multiple global optimal solutions of the loss minimization in LoCaL. Thus, U is the set of local convex regions that contain global optima. This implies that for each $W^* \in U$, I can find a locally and strictly convex region $U^* = U \cap B(r)$, where $B(r) = ||\boldsymbol{w} - \boldsymbol{w}^*||_2 \leq r$ is a norm ball and I vectorize W and W^* to obtain $\boldsymbol{w}$ and $\boldsymbol{w}^*$, respectively. Subsequently, range r can be set relatively large such that $U^* \subset B(r)$ and $U^{**} \cap B(r) = \emptyset$, where U^{**} is the local region for another global optimal point W^{**} if it exists. Then, the range for U^* still depends on the inequality in Equation (A.18).

□

A.5 Proofs of Theorem 5

Proof. F-Graph Layer has the feasible solution of the ground-truth physical grid parameters.

As for P-Graph Layer and N-Approximation Layer, I denote the PCS as $\{\bar{w}_k, \{\bar{x}_k^n\}_{n=1}^N\}_{k=1}^K$ in equation (2.12). For node j, I assume there are M number of true hidden nodes connecting j with line jm parameter as $b_m, 1 \leq m \leq M$ and true input nodal measurements as $x_m^n, \forall 1 \leq n \leq N$ for the n^{th} sampling time. Since I assume the PCS produces 0 loss, I have the following equations:

$$\begin{aligned} &\sum_{k=1}^{K} \bar{w}_k(x^n - \bar{x}_k^n) = \sum_{m=1}^{M} b_m(x^n - x_m^n), \forall 1 \leq n \leq N \\ &s.t. \;\; \{\bar{w}_k, \{\bar{x}_k^n\}_{n=1}^N\}_{k=1}^K \in \mathcal{C}_K. \end{aligned} \tag{A.19}$$

It's obvious that when $M = K$, $\{b_m, \{x_m^n\}_{n=1}^N\}_{m=1}^M$ is a PCS. However, since equation (A.19) is under-determined, multiple PCSs exist within $\mathcal{C}_K$, even when $K \neq M$. To find one of these solutions, I show in the next subsection that the problem $\mathbb{P}_K^j$ is convex under certain assumptions, and I can iteratively increase K and solve $\mathbb{P}_K^j$ to obtain one PCS. □

A.6 Proofs of Theorem 6

Proof. For F-Graph Layer, the pre-training in Equation (2.7) is an LASSO-based regression, which is convex.

For P-Graph Layer, the optimization $\mathbb{P}_K^j$ is in Equation (2.12). Since $\mathcal{C}_K$ is convex, I only need to consider the convexity of the loss function. Thus, I construct the Hessian matrix of the loss function with the following elements:

$$\frac{\partial L}{\partial^2 w_k} = \sum_{n=1}^{N} 2(x_k^n - x^n)^2,\ \frac{\partial L}{\partial w_k \partial w_h} = \sum_{n=1}^{N} 2(x_h^n - x^n)(x_k^n - x^n),$$
$$\frac{\partial L}{\partial w_k \partial x_k^n} = 2(p_n - \sum_{l=1}^{K} w_l(x^n - x_l^n) - w_k(x^n - x_k^n)),\ \frac{\partial L}{\partial^2 x_k^n} = 2w_k^2,$$
$$\frac{\partial L}{\partial w_k \partial x_h^n} = 2(x_k^n - x^n)w_h,\ \frac{\partial L}{\partial x_k^n x_h^n} = 2w_k w_h,\ \frac{\partial L}{\partial x_k^n x_h^m} = 0.$$

I study the positive-definiteness of the Hessian matrix $\boldsymbol{H}_0$ with respect to the variable vector $[w_1, \cdots, w_K, x_1^1, x_2^1 \cdots, x_K^1, x_1^2, \cdots, x_K^N]^\top$. It's clear that $\boldsymbol{H}_0[1:K, 1:K]$ is positive semi-definite, since this Hessian matrix $\boldsymbol{H}_0[1:K, 1:K]$ represents a linear least square loss. On the other hand, if I conduct a Gaussian elimination process to iteratively prove the positive semi-definiteness, I need to iteratively prove the first entry of each eliminated matrix is positive. Due to the positive semi-definiteness of $\boldsymbol{H}_0[1:K, 1:K]$, it's obvious that during the first $K-1$ eliminations, all the first entries of the eliminated matrices are positive, i.e., $\boldsymbol{H}_1(1,1), \cdots, \boldsymbol{H}_{K-1}(1,1) > 0$. Thus, I focus on the impacts of eliminations on diagonal entries after the previous K numbers.

Specifically, for the i^{th} Gaussian elimination, I can evaluate the diagonal element of the eliminated matrix as:

$$\boldsymbol{H}_i(l,h) = \boldsymbol{H}_{i-1}(l+1, h+1) - \frac{\big(\boldsymbol{H}_{i-1}(1,h+1)\big)\big(\boldsymbol{H}_{i-1}(1,l+1)\big)}{\boldsymbol{H}_{i-1}(1,1)}, \tag{A.20}$$

where $1 \le l \le (N+1)K - i$.

If I assume N is sufficiently large, $\mathbf{H}_0(k,k) = \sum_{n=1}^{N} 2(x_k^n - x^n)^2, \forall 1 \le k \le K$. The non-zero flow of line jk (recall j is the index of the center node of $\mathcal{G}_j$) implies $x_k^n - x^n \neq 0$. Namely, $\mathbf{H}_0(k,k)$ is a sufficiently large positive number. More specifically, equation (A.6) implies that $\forall b \ge K, \boldsymbol{H}_0(1, 1+b) \ll \boldsymbol{H}_0(1,1)$. Therefore, the elimination process in equation (A.20) indicates that $\boldsymbol{H}_1(b,b) > 0$. However, I need to consider the values of $\boldsymbol{H}_1(1,b)$ and $\boldsymbol{H}_1(1,1)$ to continue the iteration. Due to the triangle inequality, I know that $\boldsymbol{H}_1(1,1) > 0$ given $x_1^n \neq x_2^n$ for any $1 \le n \le N$. If N is sufficiently large, I can claim that $\boldsymbol{H}_1(1,1)$ has a sufficiently large positive accumulation value, compared to a fixed value of $\boldsymbol{H}_1(1,b)$. Thus, I have $\boldsymbol{H}_1(1,1) \gg \boldsymbol{H}_1(1,b)$. Repeating the above eliminations for K times and I have $\boldsymbol{H}_K(b+1-K, b+1-K) > 0$.

Then, for the rest of $b-K$ eliminations, the diagonal element $\boldsymbol{H}_{K+a}(1,1), \forall 1 \leq a \leq b-K$ is not sufficiently large. However, since the off-diagonal $\boldsymbol{H}_{K+a}(1, b+1-(K+a)) = 0$ always hold during the Gaussian eliminations, I can still guarantee $\boldsymbol{H}_{K+a}(b+1-(K+a), b+1-(K+a)) > 0$. Finally, the elimination will end up with $\boldsymbol{H}_b(1,1) > 0$.

In general, the above iteration process proves the positivity of each first entry of the eliminated matrices, indicating the positive semi-definiteness of the Hessian matrix and the convexity of our problem $\mathbb{P}_K^j$. □

A.7 Proofs of Theorem 7

Proof. First, I prove the matrix of step kernel k_{step} is positive definite. Equation (4.12) shows that the step kernel is a multiplication of Dirac kernels and RBF kernels. Since the Dirac and RBF kernels are positive definite (Smola and Schölkopf 1998), and the multiplication preserves the positive definiteness, the step kernel matrix is positive definite. Second, according to (Borgwardt et al. 2005), the positive definiteness of step kernels leads to the positive definiteness of the proposed random walk kernel, which further implies that the proposed random walk kernel is universal. □

A.8 Proofs of Theorem 8

Proof. Equation (4.12) shows that the calculation of the step kernel has a computational time complexity $O(\delta_n)$. Then, it takes $O(\delta_n M^4)$ to construct $M^2 \times M^2$ weighted adjacency matrix $\boldsymbol{B}_\times$. Further, (Vishwanathan et al. 2010) shows that it takes $O(M^3)$ for the Sylvester equation-based method to calculate one entry of $\boldsymbol{K}$ using the obtained matrix $\boldsymbol{B}_\times$. Thus, for $(N+\widetilde{N}) \times (N+\widetilde{N})$ kernel matrix $\boldsymbol{K}$, the total time complexity is $O\big((N+\widetilde{N})^2(M^3+\delta_n M^4)\big)$. □