INPUT CONVEX GRAPH NEURAL NETWORKS: AN APPLICATION TO OPTIMAL CONTROL AND DESIGN OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the success of modeling networked systems via graph neural networks (GNN), applying GNN for the model-based control is pessimistic since the non-convexity of GNN models hinders solving model-based control problems. In this regard, we propose the input convex graph neural networks (ICGNN) whose inputs and outputs are related via convex functions. When ICGNN is used to model the target objective function, the decision-making problem becomes a convex optimization problem due to the convexity of ICGNN and the corresponding solution can be obtained efficiently. We assess the prediction and control performance of ICGNN on several benchmarks and physical heat diffusion problems, respectively. On the physical heat diffusion, we further apply ICGNN to solve a design optimization problem, which seeks to find the optimal heater allocations while considering the optimal operation of the heaters, by using a gradient-based method. We cast the design optimization problem as a bi-level optimization problem. In there, the input convexity of ICGNN allows us to compute the gradient of the lower level problem (i.e., control problem with a given heater allocation) without bias. We confirm that ICGNN significantly outperforms non-input convex GNN to solve the design optimization problem.

1 INTRODUCTION

Decision-making problems are often written in a form of mathematical optimization, where each part of the optimization problem is required to be *modeled* to well represent the nature of problem while encouraging the solvability of the problem. This is also true when applying machine learning (ML) models as a component (or whole) of those decision-making problems. If one focuses only on accurately modeling a target system, finding the (optimal) solution of the problem becomes challenging. On the other hand, if one focuses only on effective solution finding by restricting the representability of the model, the found solution may not be appropriate as the model cannot well represent the nature of the problem. Thus, it is crucial to balance an expressive representation for the problem and the mathematical tractability to solve the formulated problem.

The inductive biases for representability Incorporating the knowledge about the target systems into ML models often leads the models to have higher generalization performances (Battaglia et al., 2018). One famous approach is using graph representation to represent the state of graph-structured target systems and employing graph neural networks (GNN) to learn the relationships among entities composing the target system. (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018; Park & Park, 2019). These GNN approaches learn the interactions among the graph entities (e.g., nodes, edges) and apply the learned interactions to perform predictions. Notably, the approaches often show outstanding generalization capabilities compared to the different types of network models. Such property of GNN models becomes more important when the model is used to formulate the decision-making problem that needs to produce the optimal decision given the conditions that have not been considered during training the model.

The inductive biases for solvability Imposing structural assumptions to the optimization can make the problem to be solved effectively. In various ML pipe-lined optimization problems,

valid structural assumptions enhance the performance while decreasing computational burdens (Rashid et al., 2018; Sunehag et al., 2017; Chen et al., 2018b). The exemplary approach is imposing convexity to the ML models so that entire decision-making can be done by solving convex optimization problems. The input convex neural network (ICNN) (Amos et al., 2017) is a general method for reformulating NN models as they become convex function w.r.t the inputs. The convexity of ICNN helps to solve optimal control problems by employing recurrent extensions of ICNN (Chen et al., 2018b; 2020; Yang & Bequette, 2021).

Balancing between representability and solvability In order to solve the decision-making problem with ML models, both of higher representability (generalizability) of model and solvability of problem are essential. In this perspective, the marriage of the exceptional generalization capability of GNN and solvability of ICNN enable us to construct a decision-making problem to represent the target system well and be solved effectively. In this paper, we propose input convex GNNs (ICGNN), a class of GNN whose inputs and outputs are related via convex functions so that it can be used to solve the various decision-making, i.e., optimal control, and bi-level design optimization problems. We provide a general-yet-simple recipe that transforms well-known GNN architectures (e.g., GCN (Kipf & Welling, 2016), GAT Veličković et al. (2017), GIN Xu et al. (2018), GN blocks Battaglia et al. (2018)) into ICGNN. We also propose recurrent extensions of ICGNN so that it can be used to predict the multi-step ahead responses of network systems.

Training ICGNN We achieve the convexity of ICGNN by restricting some parameters to be non-negative and utilizing a convex and non-decreasing activation functions (e.g. ReLU, LeakyReLU). Thus training ICGNN is conducted by solving a constrained optimization. This constrained training problem is often iteratively solved by solving unconstrained training problem and then projecting the parameters into the feasible region (Amos et al., 2017; Chen et al., 2018b) at each step. We found that such training scheme can deteriorate the predictive performance of the trained model. To circumvent that issue, we employ a reparameterization scheme which reformulates the constrained training problem into an unconstrained training problem. We found that such optimization scheme is more effective than constrained optimization with projections.

Validation To validate the efficacy of the proposed ICGNN by solving the following two types of decision-making problems.

- **Optimal control problem** We employ ICGNN to model the state transition of PDE systems (physical heat diffusion) and use this dynamic model to control the PDE systems using the model predictive control (MPC) scheme. From our numerical experiments, we confirm the proposed ICGNN excels its non-input convex counterpart in predicting the target system's future trajectories and controlling the system.
- **Bi-level design optimization** We also apply the proposed ICGNN to solve a design optimization problem, seeking to find the optimal controller allocations that can maximize control performance. We compute the gradient of the control objective with respect to the design parameters using implicit differentiation and use the gradient to optimize the optimal controller layout via a gradient-based method. This result opens an opportunity to utilize data-driven models in solving a long-standing engineering problem efficiently.

2 Related Works

2.1 Relational inductive biases: Graph neural networks

GNN is a type of neural network that operates on graph-structured data. Majority of GNN methods aims to learn the pairwise interaction patterns of the edges from various graph domains ranging from social network, combinatorial optimizations and physics domains (Kipf & Welling, 2016; Park et al., 2021a; Sanchez-Gonzalez et al., 2018; Park & Park, 2019). Such learned pairwise interactions allows GNN to predict the results from the graphs that are distinct from the training graphs. This property is especially effective for modeling physics systems such as particle simulators and FEM methods (Alet et al., 2019; Sanchez-Gonzalez et al., 2020). We utilize the proposed ICGNN to model one of the physical systems; diffusion of heat. The trained ICGNN shows better predictive results than GNN. Furthermore, we confirmed that the input convexity of ICGNN improves the control performance of the simulated heat system compared to the plain GNN.

2.2 FUNCTIONAL INDUCTIVE BIASES: INPUT CONVEX NEURAL NETWORKS

Imposing some mathematical properties (e.g. homogeneity, positivity, monotonicity, and convexity) on neural networks has been investigated from various contexts (Sill, 1998; Tang et al., 2020; Park et al., 2021b; Amos et al., 2017). Such mathematical properties helps the generalization capabilities of the networks when the mathematical properties are well aligned with target problems (Tang et al., 2020). Among the approaches, input convexity posits an attractive property when the network serves as a component of optimization problems as the optimization problem becomes convex. input convex neural network (ICNN) (Amos et al., 2017) proposes a general recipe, which limits the weight parameters of MLP to be positive and non-linearities are monotone, to construct a neural network whose inputs and outputs are related via convex functions. Based on the ICNN formula and input convexity, optimal control methods (Chen et al., 2018b), optimal transportation methods (Makkuva et al., 2020), and norm-learning methods (Pitis et al., 2019) are has been proposed. ICGNN is a graph-extension of ICNN so that ICNN framework still valid in graph GNN. We investigate input convex reformulations of famous GNNs, and also simple optimizations tricks which makes entire training results of ICNNs much better.

2.3 BEHAVIOURAL INDUCTIVE BIASES: IMPLICIT NNS

Implicit neural networks (also referred as infinite depth models) impose "behavioural" inductive biases, that are represented as a form of a mathematical (optimization) problems, to the neural networks and the gradient of the problem can be computed in a computationally efficient manner. The graident is then used to optimize the parameters of neural networks. For instance, Neural ODE (NODE) Chen et al. (2018a) applies the adjoint method to estimate the gradient of ODE problems. The other well-known members of implicit neural networks contains neural fixed point methods (Bai et al., 2019; Park et al., 2021b) and differentiable optimization layers (Amos & Kolter, 2017; Agrawal et al., 2019). We found the efficacy of the proposed ICGNN when it is used as a part of differentiable convex optimization layers. Since the optimization problem becomes convex, we can find optimal solutions in theory. Additionally, we show ICGNN can provide the exact gradient of differentiable optimization layer without introducing bias due to its convexity. Based on this property, we cast the design optimization problem as a bi-level optimization whose inner loop is for optimizing the control inputs of the heat simulation, which convexity improves the optimization performance, and the outer loop is for optimizing the position of controllers given the optimal control, which graph representation of input and GNN provides high-fidelity predictions.

3 PRELIMINARIES

Before we discuss ICGNN, we provide a brief introduction for the building blocks of ICGNN. We first present a Lemma about the composition of convex functions:

Lemma 1. *If $f(\cdot)$ is convex and $g(\cdot)$ is non-decreasing and convex, then $h(\cdot) = (g \circ f)(\cdot) = g(f(\cdot))$ is convex.*

The proof of the Lemma 1 is given in Boyd et al. (2004, Ch.3.2). All propositions that will appear in this paper can be proved by the Lemma 1.

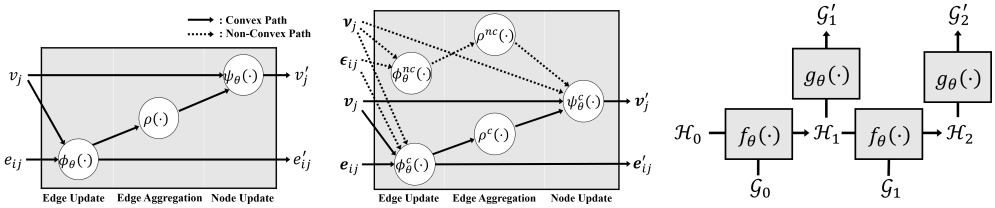
The input convex neural network (ICNN) $f_\theta(\cdot)$ is a neural network whose input and output are related with convex functions. The general expression of k-layers fully input convex neural network (FICNN) is as follows: For $i = 0, ..., k-1$,

$$\boldsymbol{z}_0 = \boldsymbol{x}, \boldsymbol{z}_{i+1} = \sigma_i(\boldsymbol{W}_i^{(\boldsymbol{z})}\boldsymbol{z}_i + \boldsymbol{W}_i^{(\boldsymbol{x})}\boldsymbol{x} + \boldsymbol{b}_i), f_\theta(\boldsymbol{x}) = \boldsymbol{z}_k \tag{1}$$

where $\boldsymbol{z}_i$ is the hidden unit of i-th layer, $\sigma_i(\cdot)$ is an activation function of i-th layer and $\theta = \left\{\boldsymbol{W}_{0:k-1}^{(\boldsymbol{z})}, \boldsymbol{W}_{0:k-1}^{(\boldsymbol{x})}, \boldsymbol{b}_{0:k-1}\right\}$ are parameters. Then the following proposition holds:

Proposition 1. *FICNN $f_\theta(\cdot)$ is convex if $\boldsymbol{W}_{0:k-1}^{(\boldsymbol{z})}$ are non-negative and $\sigma_{0:k-1}(\cdot)$ are convex and non-decreasing functions.*

The proof of Proposition 1 is straight-forward due to Lemma 1. Depending on the applications, some part of $\boldsymbol{x}$ may not require to be convex to $\boldsymbol{z}_k$. In such cases, the use of partially input convex



(a) ICGNN (b) PICGNN (c) ICGRNN

Figure 1: An overview of ICGNN architectures

neural network (PICNN) can be considered. For clear explanation, we overload $\boldsymbol{x}$ so that it is corresponding to the convex features and $\boldsymbol{y}$ denotes the features that is not required to be convex. PICNN is defined as follows: For $i = 0, ..., k-1$,

$$\boldsymbol{u}_0 = \boldsymbol{y}, \boldsymbol{z}_0 = \boldsymbol{x} \tag{2}$$

$$\boldsymbol{u}_{i+1} = \xi_i(\boldsymbol{V}_i^{(\boldsymbol{u})}\boldsymbol{u}_i + \boldsymbol{V}_i^{(\boldsymbol{y})}\boldsymbol{y} + \boldsymbol{c}_i) \tag{3}$$

$$\boldsymbol{z}_{i+1} = \sigma_i(\boldsymbol{W}_i^{(\boldsymbol{z})}\boldsymbol{z}_i + \boldsymbol{W}_i^{(\boldsymbol{u})}\boldsymbol{u}_i + \boldsymbol{W}_i^{(\boldsymbol{x})}\boldsymbol{x} + \boldsymbol{W}_i^{(\boldsymbol{y})}\boldsymbol{y} + \boldsymbol{b}_i) \tag{4}$$

$$f_\theta(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{z}_k \tag{5}$$

where $\boldsymbol{z}_i$ and $\boldsymbol{u}_i$ are the hidden units for convex and non convex features respectively. Those are called namely "convex path" and "non-convex path", respectively. Then the following proposition holds:

Proposition 2. *PICNN $f_\theta(\cdot)$ is convex in $\boldsymbol{x}$ if $\boldsymbol{W}_{0:k-1}^{(\boldsymbol{z})}$ are non-negative and $\sigma_{0:k-1}(\cdot)$ are convex and non-decreasing functions.*

A recurrent extension of ICNN, the input convex recurrent neural network (ICRNN), is investigated (Chen et al., 2018b). ICRNN takes $\boldsymbol{x}_{0:T-1}$ and an initial hidden state $\boldsymbol{h}_0$ as inputs and predicts a sequence of outputs $\boldsymbol{y}_{1:T}$ as follows: For $t = 0, ..., T-1$,

$$\boldsymbol{h}_{t+1} = f_\theta(\boldsymbol{h}_t, \boldsymbol{x}_t) \tag{6}$$

$$\boldsymbol{y}_{t+1} = g_\theta(\boldsymbol{h}_{t+1}) \tag{7}$$

where $\boldsymbol{h}_t$ is the hidden state at t, $f_\theta(\cdot)$ is a hidden update function and $g_\theta(\cdot)$ is a decoder function. Then the following proposition holds.

Proposition 3. *ICRNN is convex and non-decreasing function if $f_\theta(\cdot)$ and $g_\theta(\cdot)$ are non-decreasing ICNN.*

For further details of ICNN, PICNN, and ICRNN, please refer to the following papers (Amos et al., 2017; Chen et al., 2018b).

4 Input Convex Graph Neural Networks

In this section, we discuss the input convex formulation of the general GNN and its recurrent extension. We first introduce the notations and a general formulation of GNN layer. Then we provide a general recipe for transforming the GNN to input convex models and their partial and recurrent extensions.

In this paper, we consider a directed graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = \{\boldsymbol{v}_i\}$, $\mathbb{E} = \{\boldsymbol{e}_{ij}\} \subset \mathbb{V} \times \mathbb{V}$, v_i is i^{th} node, and $\boldsymbol{e}_{ij}$ is the edge from v_i to v_j, as inputs of GNN models. A generalized GNN layer utilizes $\mathcal{G}$ as inputs and produces the updated graph $\mathcal{G}' = (\mathbb{V}', \mathbb{E}')$ via the following steps:

$$\boldsymbol{e}'_{ij} = \phi_\theta(\boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij}) \qquad \forall \boldsymbol{e}_{ij} \in \mathbb{E} \tag{8}$$

$$\boldsymbol{v}'_j = \psi_\theta\big(\boldsymbol{v}_j, \rho(\{\boldsymbol{e}'_{ij}\}_{i \in \mathcal{N}_j})\big) \qquad \forall \boldsymbol{v}_j \in \mathbb{V} \tag{9}$$

where $\phi_\theta(\cdot)$ is an edge update function, $\rho(\cdot)$ is a permutation-invariant aggregation function (e.g. sum, mean, max), $\psi_\theta(\cdot)$ is a node update function and $\mathcal{N}_j$ is the neighborhood set of $\boldsymbol{v}_j$. Notice that

this formulation is a generalization of famous GNN layers including GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018), GN (Battaglia et al., 2018). Based on the generalized GNN layer, we propose the input convex graph neural network (ICGNN).

Proposition 4. *ICGNN is convex if $\phi_\theta(\cdot)$ is convex and $\psi_\theta(\cdot)$ and $\rho(\cdot)$ are convex and non-decreasing functions.*

The conditions of ICGNN are attained by employing FICNN for $\phi_\theta(\cdot)$ and $\psi_\theta(\cdot)$, and commonly-used aggregation functions (e.g. sum, mean, max) as $\rho(\cdot)$. Furthermore, as similar to PICNN, we can extend ICGNN to the partially convex variants called the partially ICGNN (PICGNN). A generalized PICGNN utilizes $\mathcal{G} = (\mathcal{G}^c, \mathcal{G}^{nc})$ where $\mathcal{G}^c = (\mathbb{V}^c, \mathbb{E}^c)$ and $\mathcal{G}^{nc} = (\mathbb{V}^{nc}, \mathbb{E}^{nc})$ as inputs and produces the updated graph $\mathcal{G}' = (\mathbb{V}', \mathbb{E}')$ via the following steps:

$$\boldsymbol{\epsilon}'_{ij} = \phi^{nc}_\theta(\boldsymbol{\nu}_i, \boldsymbol{\nu}_j, \boldsymbol{\epsilon}_{ij}) \qquad \forall \boldsymbol{\epsilon}_{ij} \in \mathbb{E}^{nc} \tag{10}$$

$$\boldsymbol{e}'_{ij} = \phi^{c}_\theta(\boldsymbol{v}_i, \boldsymbol{v}_j, \boldsymbol{e}_{ij}, \boldsymbol{\nu}_i, \boldsymbol{\nu}_j, \boldsymbol{\epsilon}_{ij}) \qquad \forall \boldsymbol{e}_{ij} \in \mathbb{E}^{c} \tag{11}$$

$$\boldsymbol{v}'_j = \psi^{c}_\theta\big(\boldsymbol{v}_j, \rho^c(\{\boldsymbol{e}'_{ij}\}_{i\in\mathcal{N}_j}), \boldsymbol{\nu}_j, \rho^{nc}(\{\boldsymbol{\epsilon}'_{ij}\}_{i\in\mathcal{N}_j})\big) \qquad \forall \boldsymbol{v}_j \in \mathbb{V}^{c} \tag{12}$$

where $\mathcal{G}^c$ and $\mathcal{G}^{nc}$ are inputs for convex path and non-convex path. Then the following proposition holds:

Proposition 5. *PICGNN is convex in $\mathcal{G}$ if $\phi^c_\theta(\cdot)$ is convex and $\psi^c_\theta(\cdot)$ and $\rho^c(\cdot)$ are convex and non-decreasing through convex path.*

We can satisfy the condition of Proposition 5 by applying PICNN for $\phi^c_\theta(\cdot)$, non-decreasing PICNN for $\psi^c_\theta(\cdot)$ and non-decreasing convex aggregation function for $\rho^c(\cdot)$. We provide the GNN architectures that can be modified into ICGNN and PICGNN in the Appendix A.1.

We also introduce a recurrent extension of ICGNN, called the input convex graph recurrent neural network (ICGRNN). ICGRNN takes a sequence of input graphs $\mathcal{G}_{0:T-1}$ and an initial hidden embedding graph $\mathcal{H}_0$ to produce a sequence of graphs $\mathcal{G}^{'}_{1:T}$ as follows: For $t = 0, ..., T-1$,

$$\mathcal{H}_{t+1} = f_\theta(\mathcal{H}_t, \mathcal{G}_t) \tag{13}$$

$$\mathcal{G}^{'}_{t+1} = g_\theta(\mathcal{H}_{t+1}) \tag{14}$$

where $\mathcal{H}_t$ is the hidden embedding graph at t, $f_\theta(\cdot)$ is a hidden graph update function and $g_\theta(\cdot)$ is a graph decoder function. Then the following proposition holds.

Proposition 6. *ICGRNN is a convex and non-decreasing function if $f_\theta(\cdot)$ and $g_\theta(\cdot)$ are non-decreasing ICGNN.*

Figure 1(a), 1(b) and 1(c) show the architectures of ICGNN, PICGNN and ICGRNN. We omit the architecture of the partially ICGRNN which is the partially convex variant of ICGRNN.

5 Training ICGNN

Training ICGNN requires to solve a constraint optimization problem where the constraints are imposing the non-negativity on $\boldsymbol{W}$ of ICNN. Solving a constrained optimization problem is more challenging than an unconstrained optimization especially when the number of variables (e.g., the number of training parameters) becomes larger. Therefore, a simple heuristic, which projecting the parameter values to the non-negative region after the gradient update, is often used (Amos et al., 2017; Chen et al., 2018a). We observe that such heuristic deteriorates the predictive performance of ICNN. To circumvent such issue, we propose to use a variable reparameterization as follows:

$$\boldsymbol{W}_{ij} = \sigma(\omega_{ij}) \tag{15}$$

where $\boldsymbol{W}_{ij}$ is the (i, j)-th component of $\boldsymbol{W}$, $\sigma(\cdot)$ is a non-negative function (e.g. ReLU, absolute value function), and ω_{ij} is the reparameterized variable.

6 Experiments

We investigate the proposed ICGNN in three different domains: (1) benchmark graph problems, (2) dynamic control problems on the physical heat diffusion environment with model predictive control (MPC), and (3) design optimization problems where the input convexity and graph property of ICGNN show distinct advantages.

	Cora	Citeseer	Pubmed		MUTAG	COLLAB	IMDB-B.	IMDB-M.
GCN	**0.81**	0.70	**0.79**	GIN	0.85	**0.67**	**0.65**	**0.43**
ICGCN	**0.81**	**0.73**	**0.79**	ICGIN	**0.89**	0.60	0.52	0.39

Table 1: GNN benchmark results (accuracy)

6.1 ICGNN ON THE PUBLIC BENCHMARKS

We investigate the predictive performance of input convex reformulation of the famous GNN on the public benchmark domains. As the IC reformulations restrict the parameter space, it may harms the predictive performances of the GNN. However, from our experimental results, the performance drop may not be severe and, surprisingly, for some cases, the IC reformulation shows better predictive performance than original GNNs.

We evaluate GCN, GIN and thier convex reformulation on cora, citeseer, pubmed, and MUTAG, COLLAB, IMDBBINARY, IMDBMULTI, respectively. For implementing the GNN models, we uses the hyperparmeters of the open-source implementations[1]. Table 1 shows the classification accuracy of the GNN models and their input convex counterparts. As shown in Table 1, the IC reformulations do not severely decrease the classification performances. For some cases (e.g., Citeseer, MUTAG), they show improved classification performances.

6.2 ICGNN ON THE CONTROL PROBLEMS

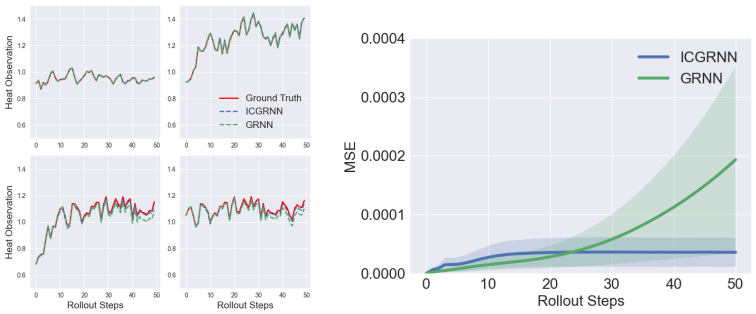
One of prominent applications of convex predictive model is optimal control. We evaluate the predictive and control performance of ICGNN on a partially observable heat diffusion environment. On the domain of the heat diffusion environment, a number of sensors $\mathbb{V}^x$ which observe the heat value and controllers $\mathbb{V}^u$ which generate the heat are spatially distributed. Note that the number and location of sensors and controllers are chosen at random. The heat is evolved from the controllers, diffused through the entire domain and observed at the sensors. The observation and control input of the heat diffusion environment at time-step t are denoted as $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ respectively. Please refer Appendix B.1 for the details of the partially observable heat diffusion environment.

We represent the environment at time-step t as a directed graph $\mathcal{G}_t = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = \mathbb{V}^u \cup \mathbb{V}^x$ and $\mathbb{E} = (\mathbb{V} \times \mathbb{V}) \setminus (\mathbb{V}^u \times \mathbb{V}^u)$ (i.e., complete but controller to controller edges). The i^{th} sensor has the feature $\boldsymbol{v}_i^x$ which contains the location and the heat observation of the i^{th} sensor. The j^{th} controller has $\boldsymbol{v}_j^u$ which contains the location and heat input of the j^{th} controller. The edge between two nodes has the Euclidean distance between two nodes as the edge feature $\boldsymbol{e}_{ij}$.

We model the dynamics of the environment by using the partially ICGRNN. The partially ICGRNN utilizes the location and distance features as the inputs of non-convex path and heat observations of sensors and heat inputs of controllers as the inputs of convex path. ICGRNN predicts a sequence of heat observation $\hat{\boldsymbol{x}}_{1:T}$ from an initial hidden embedding graph $\mathcal{H}_0$ and a sequence of heat inputs $\boldsymbol{u}_{0:T-1}$ by recursively updating the hidden embedding graph $\mathcal{H}_t$. The model utilizes four past and current observations to generate an initial hidden embedding graph $\mathcal{H}_0$. We use three-layer partially ICGNN for $f_\theta(\cdot)$ of equation 13 and four-layer FICNN for $g_\theta(\cdot)$ of equation 14. We utilize the same architecture of ICGRNN for GRNN model as a baseline. To obtain training and test data, we randomly initialize 30 environments which has different sensor and heater allocations and gather state-control trajectories by applying random heat inputs. Both models are trained by minimizing the mean squared error (MSE) between the rollout predicted heat values of 10 future steps and ground truth observations. Please refer Appendix B.2 for the details of the predictive models and training.

Evaluating predictive performance We evaluate the rollout prediction performance of ICGRNN in the heat diffusion environment. Figure 2(a) illustrates the rollout predictions of ICGRNN and GRNN model. As shown in Figure 2(a), both of the ICGRNN and GRNN models show accurate predictions when the rollout step is short. However, when the rollout step becomes longer, ICGRNN model shows more reliable predictions than GRNN model. To further understand the generalization performances of the models, we build a test dataset consist of the 10 sensor/heater layouts and the

[1] https://github.com/dmlc/dgl



(a) 4 rollout results (b) Average rollout prediction errors

Figure 2: Heat diffusion prediction results

action trajectories length of 100 whose actions are sampled from $\mathcal{U}(0.0, 50.0)$. Figure 2(b) visualizes the averages prediction errors of the ICGRNN and GRNN models on the test dataset. As shown in Figure 2(a), both of the ICGRNN and GRNN can well predict the 10 future states as they are trained to predict until 10 future steps. However, after the 10 steps, the prediction errors of GRNN starts to diverge while ICGRNN shows relatively stable prediction errors.

Evaluating control performance Now we study the control performance of ICGRNN in the heat diffusion environment. In our experiments, we use model predictive control (MPC) framework to control the environment. In the MPC framework, at each time-step t, we solve an optimization problem to find the optimal control input $\boldsymbol{u}^*_{t:t+K-1}$ of the future K steps, which minimizes the control objective while satisfying the feasible condition and the predictive model. After solving the optimization problem, we execute the first optimized controls to the target environment and repeat the process. The optimization problem is given as follows:

$$\underset{\boldsymbol{u}_{t:t+K-1}}{\arg\min} \quad \sum_{k=0}^{K-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+k+1}, \bar{\boldsymbol{x}}_{t+k+1}, \boldsymbol{u}_{t+k}) \tag{16}$$

$$\text{s.t.} \quad \hat{\boldsymbol{x}}_{t+1:t+K} = F_\theta(\mathcal{H}_t, \boldsymbol{u}_{t:t+K-1}) \tag{17}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_{t:t+K-1} \leq \bar{\boldsymbol{u}} \tag{18}$$

where $\mathcal{J}(\cdot)$ is the control objective, $F_\theta(\cdot)$ is the predictive model, $\hat{\boldsymbol{x}}_{t+k}$ is predicted heat value at time-step $t+k$, $\bar{\boldsymbol{x}}_{0:T-1}$ is a reference heat trajectory (i.e., target to track), $\mathcal{H}_t$ is an initial hidden embedding graph at time-step t and $\underline{\boldsymbol{u}}$ and $\bar{\boldsymbol{u}}$ are the lower and upper bound of $\boldsymbol{u}$.

In the following experiments, we investigate the control performance of ICGRNN model for two widely-used $\mathcal{J}(\cdot)$: (1) reference tracking problem (i.e., deriving the heats x to the reference $\bar{x}$) and (2) input minimization problem (i.e., minimizing the control inputs with heat-level constraints). To evaluate the control performances, we run MPC on five randomly initialized environments and report the average of the control objectives. We consider the ground truth controller and GRNN as baselines. The detail of the control problem is given in Appendix B.3.

Figure 3[top] illustrates the results of the MPC experiments whose objectives are the reference tracking. The red line shows $\bar{x}$ and the blue line shows the observed state values when applying $\boldsymbol{u}^*_t$ from each controller. The green lines shows the optimized action sequences of each controller. From Figure 3[top], we can confirm that the MPC controller which utilizes ICGRNN as $F_\theta(\cdot)$ produces the control results that are closed to the controller with ground truth model. On the other hand, the control which utilizes GRNN tends to underperform than the control with ICGRNN. Table 2 summarizes the average control performances on the test data set. The numerical results highlights that ICGRNN model shows better control performance (i.e. providing higher solvability) than GRNN model.

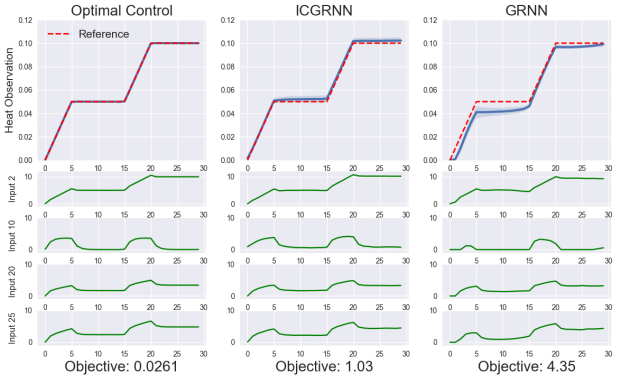


Figure 3: Sample optimal control and MPC result of ICGRNN and GRNN and their control objective values ($\times 10^{-5}$) on reference tracking problem.

Table 2: Average control objective value. ($\times 10^{-5}$)

	Reference Tracking	Input Minimization
Optimal	0.026	1.62
GRNN	7.31	2.04
ICGRNN	2.46	1.72

6.3 ICGNN ON THE DESIGN OPTIMIZATION PROBLEM

From the previous section, we confirm that ICGNN model provides better solvability than GRNN models. we now apply ICGNN to solve more practically demanding decision-making problem.

Design optimization, which aims to find the (optimal) design parameter $\boldsymbol{p}$ that optimizes the system's performance metric $\mathcal{J}(\boldsymbol{p})$, has numerous real-world applications. A few of ML researches tackle such problems by employing the differentiable learned model $f_\theta(\boldsymbol{p}) = \mathcal{J}(\boldsymbol{p})$ and gradient-based optimizations. However, when the performance metric is related with not only $\boldsymbol{p}$ but also the operations $\boldsymbol{u}(\boldsymbol{p})$ that do not have explicit expressions, it is less straightforward to solve the design optimization problem as did in previous researches. For instance, we aim to find the optimal controller allocations that minimize the control objective functions of Section 6.2.

We cast the design optimization as a bi-level optimization problem whose lower-level optimization is seeking for optimal controls $\boldsymbol{u}^*(\boldsymbol{p})$ with the given $\boldsymbol{p}$ and upper-level optimization is for finding the optimal heater allocations $\boldsymbol{p}^*$. The bi-level optimization is written as follows:

$$\arg\min_{\boldsymbol{p}} \quad \sum_{t=0}^{T-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{u}_t) \tag{19}$$

$$\text{s.t.} \quad \boldsymbol{u}_{0:T-1} = \arg\min_{\boldsymbol{v}_{0:T-1}} \sum_{t=0}^{T-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{v}_t) \tag{20}$$

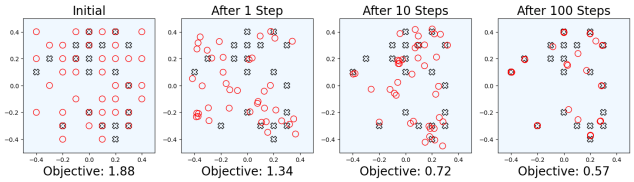
$$\text{s.t. equation } (17), (18)$$

$$\hat{\boldsymbol{x}}_{1:T} = F_\theta(\mathcal{H}_0, \boldsymbol{u}_{0:T-1}; \boldsymbol{p}) \tag{21}$$

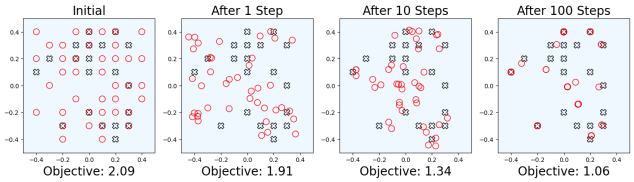
$$\underline{\boldsymbol{p}} \le \boldsymbol{p} \le \bar{\boldsymbol{p}} \tag{22}$$

where $\mathcal{J}(\cdot)$ is the control objective function, $F_\theta(\cdot; \boldsymbol{p})$ is the predictive model when the controller position $\boldsymbol{p}$ is given, and $\underline{\boldsymbol{p}}$, $\bar{\boldsymbol{p}}$ is the upper and lower bound of $\boldsymbol{p}$.

To solve the proposed bi-level optimization via a gradient-based method, it is required to compute the gradient of $\boldsymbol{u}$ with respect to $\boldsymbol{p}$; $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$. In general, computing $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$ is challenging as it has no explicit expression. However, the convexity of ICGRNN makes the lower-level problem as convex

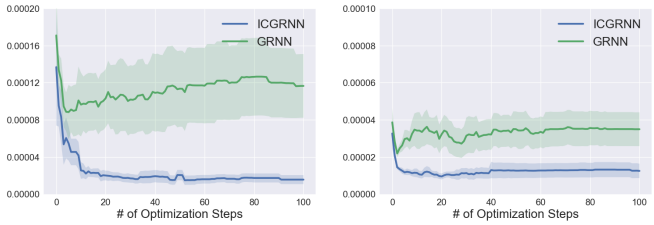


(a) ICGRNN with control objective values ($\times 10^{-5}$)



(b) GRNN with control objective values ($\times 10^{-5}$)

Figure 4: Sample design optimization result on the input minimization problem.



(a) Reference Tracking Problem (b) Input Minimization Problem

Figure 5: Design optimization performance for two control problems.

optimization. As a result, solving the lower-level optimization and finding a root of its Karush-Kuhn-Tucker (KKT) conditions are equivalent. Based on this, we apply the implicit function theorem to the KKT conditions to efficiently compute the gradient $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$ without introducing biases. Once we attain $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$, we can solve the design optimization problem via a gradient-based method as follows:

$$\boldsymbol{p}_{t+1} \leftarrow \boldsymbol{p}_t + \alpha \times \frac{\partial \mathcal{J}(\boldsymbol{p}_t)}{\partial \boldsymbol{p}_t} \tag{23}$$

Please refer Appendix B.4 for full derivation of the gradient $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{p}}$.

From the initial layout $\boldsymbol{p}_0$, we employ the ICGRNN and GRNN model as $F_\theta(\cdot)$ to solve the design optimization problem. As shown in Figure 4(a), the ICGRNN model can be used for successful design optimization. On contrary, the design optimization results with GRNN convereges to the solution worse than the ICGRNN solutions (see Figure 4(b). To verify this observation is generally established, we repeat the similar experiments with the different initial layouts and different control problems. From the Figure 5, we can observe that ICGRNN consistently provides better optimized design than GRNN.

7 Conclusion

In this work, we proposed ICGNN that balances the representability (generalizability) of GNN models and solvability of ICNNs in ML pipe-lined decision-making problems. We verify the representability and solvability of ICGNN on the public benchmark domains and dynamic control on the physical heat diffusion environment. We also employ ICGNN to solve the design optimization problem via a gradient-based method. Experimental results support the representability and solvability of ICGNN on various predicting and decision-making problems.

REFERENCES

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. *arXiv preprint arXiv:1910.12430*, 2019.

Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pp. 212–222. PMLR, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

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

Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *arXiv preprint arXiv:1810.13400*, 2018.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018a.

Yize Chen, Yuanyuan Shi, and Baosen Zhang. Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*, 2018b.

Yize Chen, Yuanyuan Shi, and Baosen Zhang. Input convex neural networks for optimal voltage regulation. *arXiv preprint arXiv:2002.08684*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Ashok Makkuva, Amirhossein Taghvaei, Sewoong Oh, and Jason Lee. Optimal transport mapping via input convex neural networks. In *International Conference on Machine Learning*, pp. 6672–6681. PMLR, 2020.

Junyoung Park and Jinkyoo Park. Physics-induced graph neural network: An application to wind-farm power estimation. *Energy*, 187:115883, 2019.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021a.

Junyoung Park, Jinhyun Choo, and Jinkyoo Park. Convergent graph solvers. *arXiv preprint arXiv:2106.01680*, 2021b.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Silviu Pitis, Harris Chan, Kiarash Jamali, and Jimmy Ba. An inductive bias for distances: Neural nets that respect the triangle inequality. In *International Conference on Learning Representations*, 2019.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Joseph Sill. Monotonic networks. 1998.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Shu Yang and B Wayne Bequette. Optimization-based control using input convex neural networks. *Computers & Chemical Engineering*, 144:107143, 2021.

A DETAIL INFORMATION OF ICGNN

A.1 ICGNN FORMULATION FOR FAMOUS GNN ARCHITECTURES

Here, we provide a list of famous GNN architectures that can be transformed into input-convex GNN.

A.1.1 INPUT CONVEX GNN

The input-convex formulations for GCN, GIN and GN block are straight-forward so we omit the detail formulations.

A.1.2 PARTIALLY INPUT CONVEX GNN

Graph Attention Network (GAT) Since the operator softmax$(\cdot)$ is not a convex function, it should be formulated as the partially input convex GAT. The partially input convex GAT layer takes two node features $\{\boldsymbol{h}_i\}$ and $\{\boldsymbol{v}_i\}$ as inputs for convex path and non-convex path and produces an updated node feature $\{\boldsymbol{h}'_i\}$ by the following steps:

$$e_{ij} = \xi(W^{(\boldsymbol{v})}\boldsymbol{v}_i, W^{(\boldsymbol{v})}\boldsymbol{v}_j), \qquad \forall (i,j) \in \mathbb{E} \tag{24}$$

$$\alpha_{ij} = \text{softmax}_i(e_{ij}), \qquad \forall j \in \mathbb{V} \tag{25}$$

$$\boldsymbol{h}'_j = \sigma\left(\sum_{i \in \mathcal{N}_j} \alpha_{ij} W^{(\boldsymbol{h})}\boldsymbol{h}_i\right), \qquad \forall j \in \mathbb{V} \tag{26}$$

where $\xi(\cdot,\cdot)$ is a shared attention function, α_{ij} is the attention score between node i and node j, $\sigma(\cdot)$ is an activation function and $W^{(\boldsymbol{v})}$ and $W^{(\boldsymbol{h})}$ are parameters. Then the output $\boldsymbol{h}'_j$ is convex in $\{\boldsymbol{h}_i\}$ if $W^{(h)}$ is non-negative and $\sigma(\cdot)$ is a non-decreasing convex function.

B EXPERIMENTAL DETAILS

B.1 DETAILS OF HEAT DIFFUSION ENVIRONMENT

B.1.1 HEAT EQUATION

The heat equation or heat diffusion equation on domain $\mathcal{D} \in \mathbb{R}^2$ is given as:

$$\frac{\partial u}{\partial t} = \kappa \Delta u + f = \kappa(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x^2}) + f, \qquad \forall (x,y) \in \mathcal{D}, t \geq 0 \tag{27}$$

$$u(x,y,0) = u_0(x,y), \qquad \forall (x,y) \in \mathcal{D} \tag{28}$$

$$u(x,y,t) = v(x,y,t), \qquad \forall t \geq 0 \tag{29}$$

where $u(\cdot)$ is the heat value, $f(\cdot)$ is a heat source function, $u_0(\cdot)$ is an initial condition, $v(\cdot)$ is a boundary condition and κ is the thermal diffusivity. Here, we observe the values of u at sensors as observation and change the value of f at controllers as control input. For simplicity, we choose $u_0(x,y) = 0, v(x,y,t) = 0$ and $\kappa = 1$ in our experiment.

B.1.2 HEAT DIFFUSION ENVIRONMENT

To simulate the heat equation, we use finite element method to discretize the time and the domain $\mathcal{D}$ into time-space grid mesh and perform the difference version of dynamic of heat equation. Let Δt and Δx be the interval of time mesh and space mesh. The one-step computation to advance from the time-step t to the time-step $t+1$ is the following:

$$u^{t+1}_{i,j} = s(u^t_{i+1,j+1} + u^t_{i+1,j-1} + u^t_{i-1,j+1} + u^t_{i-1,j-1}) + (1-4s)u^t_{i,j} + (\Delta t) f^t_{i,j} \tag{30}$$

where $s = \frac{\Delta t}{(\Delta x)^2}$ and $u^t_{i,j}$ and $f^t_{i,j}$ imply the heat value and heat source on the domain $(i\Delta x, j\Delta x)$ at time $t\Delta t$, respectively. In the heat diffusion environment, once the heat source f comes to the environment, we perform the equation **??** T times.

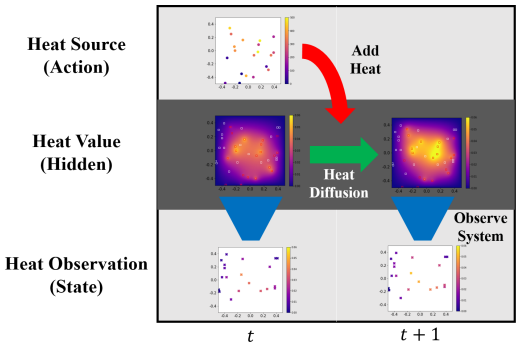


Figure 6: Dynamic of the partially observable heat diffusion simulator.

B.1.3 Partially Observable Heat Diffusion Environment

In our experiment, we use the partially observable heat diffusion environment. The reason why we called "partially" is because we are only able to observe $u^t_{i,j}$ at some specific values of (i, j), not the whole area. Figure 6 describes how the action affects to the partially observable heat diffusion environment, how the dynamic undergoes and how the observation is made.

B.1.4 Hyperparameters

In our experiment, we choose the domain $\mathcal{D} = [-0.5, 0.5]^2, \Delta x = 0.1, \Delta t = 0.000025, T = 400$.

B.2 Training Predictive Models

B.2.1 Model Architecture

The role of predictive model is to predict the future heat observation of the heat diffusion environment given past observation-action trajectory and current and future heat input. We construct the partially ICGRNN model with three different parts: 1) An initial hidden embedding function, 2) a hidden graph update function and 3) a graph decoder function. The partially ICGRNN model is convex with the current and future heat input trajectory and not convex with the other features such as the past observation-action trajectory, position of controllers and sensors.

Initial hidden embedding function Inputs of initial hidden embedding function are past heat observation trajectory $\{\boldsymbol{x}^{(\tau)}\}_{0:t}$ and past heat input trajectory $\{\boldsymbol{u}^{(\tau)}\}_{0:t-1}$ from time-step 0 to t. To deal with two different temporal data efficiently, we use two different GNN architectures. To obtain an initial hidden embedding node feature $\mathcal{H}_t = \{\boldsymbol{h}^{(t)}\}$, for each time $\tau = 0, ..., t-1$,

$$\boldsymbol{y}_j^{(\tau)} = GAT(\boldsymbol{x}^{(\tau+1)}, \boldsymbol{h}^{(\tau)}, \boldsymbol{h}_j^{(\tau)}, \boldsymbol{p}^{\boldsymbol{x}}, \boldsymbol{p}_j^{\boldsymbol{x}})) \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{31}$$

$$\mathbf{z}_j^{(\tau)} = GCN(\boldsymbol{u}^{(\tau)}, \boldsymbol{h}_j^{(\tau)}, \boldsymbol{p}^{\boldsymbol{u}}, \boldsymbol{p}_j^{\boldsymbol{x}}), \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{32}$$

$$\boldsymbol{h}_j^{(\tau+1)} = NN(\boldsymbol{h}_j^{(\tau)}, \boldsymbol{y}_j^{(\tau)}, \mathbf{z}_j^{(\tau)}, \boldsymbol{p}_j^{\boldsymbol{x}}), \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{33}$$

where $\boldsymbol{h}_j^{(0)} = NN(\boldsymbol{x}_j^{(0)}), \forall j \in \mathbb{V}^{\boldsymbol{x}}$, $\boldsymbol{y}_j^{(\tau)}$ and $\mathbf{z}_j^{(\tau)}$ are aggregated sensor and controller messages at node $j \in \mathbb{V}^{\boldsymbol{x}}$. For $GAT(\cdot)$ and $GCN(\cdot)$, we additionally use the controller positions $\boldsymbol{p}^u$ and sensor positions $\boldsymbol{p}^x$ as additional features.

Hidden graph update function To update the hidden graph, the update function aggregates the message from controller nodes and other sensor nodes. Similar to initial hidden embedding function, we construct two different GNN architectures as follows: Given the initial hidden graph $\mathcal{H}_t$ and the

heat input $\boldsymbol{u}^{(t)}$,

$$\boldsymbol{y}_j^{(t)} = PICGAT(\boldsymbol{h}^{(t)}, \boldsymbol{h}_j^{(t)}, \boldsymbol{p}^{\boldsymbol{x}}, \boldsymbol{p}_j^{\boldsymbol{x}}), \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{34}$$

$$\mathbf{z}_j^{(t)} = PICGCN(\boldsymbol{u}^{(t)}, \boldsymbol{h}_j^{(t)}, \boldsymbol{p}^{\boldsymbol{u}}, \boldsymbol{p}_j^{\boldsymbol{x}}), \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{35}$$

$$\boldsymbol{h}_j^{(t+1)} = PICNN(\boldsymbol{h}_j^{(t)}, \boldsymbol{y}_j^{(t)}, \mathbf{z}_j^{(t)}, \boldsymbol{p}_j^{x}) \qquad \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{36}$$

Here, all architectures are convex in all but the sensor positions $\boldsymbol{p}^{(\boldsymbol{x})}$ and controller positions $\boldsymbol{p}^{(\boldsymbol{u})}$.

Graph decoder function We use 4-layer FICNN to obtain predicted heat observations $\{\hat{\boldsymbol{x}}_j^{(t)}\}$ from the embedding graph $\mathcal{H}_t$:

$$\hat{\boldsymbol{x}}_j^{(t)} = FICNN(\boldsymbol{h}_j^{(t)}), \forall j \in \mathbb{V}^{\boldsymbol{x}} \tag{37}$$

We use absolute value function for reparameterization trick of the parameters that should be non-negative. We use a parametric ReLU, called PReLU, for activation function, which is defined as:

$$PReLU(x; a) = \begin{cases} x & \text{if} \quad x \geq 0 \\ ax & \text{else} \end{cases} \tag{38}$$

with a parameter a.

To make a fair comparison, we build the exact same GRNN architecture without any constraint about input convexity and use it as a baseline model.

B.2.2 Data Generation and Training Hyperparameters

We randomly initialize 120 target environments from the number of sensors and controllers from $\mathcal{U}(20, 81)$. For each episode, we choose the control input from $\mathcal{U}(0, 50)$ and collect the state-action trajectory with trajectory length 100. We divide 100, 10, 10 state-action trajectories for training, validation and test data. We implement on the Python by using PyTorch (Paszke et al., 2019) and DGL (Wang et al., 2019) library. We us the Adam (Kingma & Ba, 2014) optimizer with decaying learning rate from 0.001 to 0.0001 with decaying rate 0.5 for every 500 epochs.

B.3 MPC on Heat Diffusion Environment

B.3.1 Optimization Problem Setup

At time-step t, MPC solves the following optimization problem:

$$\min_{\boldsymbol{u}_{t:t+K-1}} \quad \sum_{k=0}^{K-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+k+1}, \bar{\boldsymbol{x}}_{t+k+1}, \boldsymbol{u}_{t+k}) \tag{39}$$

$$\text{s.t.} \quad \hat{\boldsymbol{x}}_{t+1:t+K} = F_\theta(\mathcal{H}_t, \boldsymbol{u}_{t:t+K-1}) \tag{40}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_{t:t+K-1} \leq \bar{\boldsymbol{u}} \tag{41}$$

where $\mathcal{J}(\cdot)$ is the control objective function, $F_\theta(\cdot)$ is the predictive model, $\hat{\boldsymbol{x}}_{t+k}$ is predicted heat value at time-step $t+k$, $\bar{\boldsymbol{x}}_{0:T-1}$ is a reference heat trajectory, $\mathcal{H}_t$ is an initial hidden embedding graph at time-step t and $\underline{\boldsymbol{u}} = 0$ and $\bar{\boldsymbol{u}} = 50$ are the lower and upper bound of $\boldsymbol{u}$. For both control problems, we choose $K = 10$ and the values of $\underline{\boldsymbol{u}}$ and $\bar{\boldsymbol{u}}$ are 0 and 50, respectively.

For reference tracking problem, we use $\mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{u}_t) = \|\hat{\boldsymbol{x}}_{t+1} - \bar{\boldsymbol{x}}_{t+1}\|^2$ and run the projected gradient-descent algorithm 3000 times with the Adam optimizer. We reduce the learning rate from 0.005 to 0.0001 with decaying factor 0.5 when the validation score does not decrease for 5 consecutive steps.

For input minimization problem, we use $\mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{u}_t) = (\bar{\boldsymbol{x}}_{t+1} - \hat{\boldsymbol{x}}_{t+1})^+ + \alpha\|\boldsymbol{u}_t\|^2$ with $\alpha = 0.001$ and run the projected gradient-descent algorithm 1000 times with the Adam optimizer. We choose the same learning rate scheduler of reference tracking problem, start the learning rate from 0.001.

B.4 Design Optimization on Heat Diffusion Environment

B.4.1 Full derivation of Implicit Gradients

Problem definition We build the design optimization problem as a bi-level optimization. The lower-level optimization problem is formulated as:

$$\boldsymbol{u}^*_{0:T-1} = u^*(\boldsymbol{p}) = \underset{\boldsymbol{u}_{0:T-1}}{\arg\min} \quad \sum_{t=0}^{T-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{u}_t) \tag{42}$$

$$\text{s.t.} \quad \hat{\boldsymbol{x}}_{1:T} = F_\theta(\mathcal{H}_t, \boldsymbol{u}_{0:T-1}; \boldsymbol{p}) \tag{43}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_{0:T-1} \leq \bar{\boldsymbol{u}} \tag{44}$$

where $\mathcal{J}(\cdot)$ is the control objective function, $F_\theta(\cdot; \boldsymbol{p})$ is the predictive model when the controller position $\boldsymbol{p}$ is given. By putting 17 into $\hat{\boldsymbol{x}}_{1:T}$ on the control objective function, we can simply write the lower-level problem as $\boldsymbol{u}^*_{0:T-1} = u^*(\boldsymbol{p}) = \arg\min_{\boldsymbol{u}} \{\mathcal{L}(\boldsymbol{u}, \boldsymbol{p}) : \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \bar{\boldsymbol{u}}\}$.

Now, the upper-level optimization problem is formulated as:

$$\min_{\boldsymbol{p}} \quad \sum_{t=0}^{T-1} \mathcal{J}(\hat{\boldsymbol{x}}_{t+1}, \bar{\boldsymbol{x}}_{t+1}, \boldsymbol{u}_t) \tag{45}$$

$$\text{s.t.} \quad \boldsymbol{u}_{0:T-1} = u^*(\boldsymbol{p}) \tag{46}$$

$$\hat{\boldsymbol{x}}_{1:T} = F_\theta(\mathcal{H}_0, \boldsymbol{u}_{0:T-1}; \boldsymbol{p}) \tag{47}$$

$$\underline{\boldsymbol{p}} \leq \boldsymbol{p} \leq \bar{\boldsymbol{p}} \tag{48}$$

where $\underline{\boldsymbol{p}} = -0.5$, $\bar{\boldsymbol{p}} = 0.5$ is the lower and upper bound of $\boldsymbol{p}$. We can simplify the upper-level problem as:

$$\min_{\boldsymbol{p}} \quad \mathcal{L}^*(\boldsymbol{p}) = \mathcal{L}(u^*(\boldsymbol{p}), \boldsymbol{p}) \tag{49}$$

$$\text{s.t.} \quad \underline{\boldsymbol{p}} \leq \boldsymbol{p} \leq \bar{\boldsymbol{p}} \tag{50}$$

From the fact that the gradient of $\mathcal{L}^*(\boldsymbol{p})$ w.r.t. $\boldsymbol{p}$ is given as

$$\nabla_{\boldsymbol{p}} \mathcal{L}^*(\boldsymbol{p}) = \nabla_{\boldsymbol{p}} \mathcal{L}(u^*(\boldsymbol{p}), \boldsymbol{p}) = (\nabla_{\boldsymbol{u}} \mathcal{L}(u^*(\boldsymbol{p}), \boldsymbol{p}))(\nabla_{\boldsymbol{p}} u^*(\boldsymbol{p})) + \nabla_{\boldsymbol{p}} \mathcal{L}(u^*(\boldsymbol{p}), \boldsymbol{p}), \tag{51}$$

we can compute the gradient of control objective function when the gradient $\nabla_{\boldsymbol{p}} u^*(\boldsymbol{p})$ is computed.

KKT conditions In convex optimization problem, solving the optimization problem is equivalent to finding a root of KKT conditions. We state the KKT conditions of the lower-level problem:

$$\nabla_{\boldsymbol{u}} \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{u}) + \lambda_1 \nabla_{\boldsymbol{u}}(\underline{\boldsymbol{u}} - \boldsymbol{u}) + \lambda_2 \nabla_{\boldsymbol{u}}(\boldsymbol{u} - \bar{\boldsymbol{u}}) = \nabla_{\boldsymbol{u}} \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{u}) - \lambda_1 + \lambda_2 = 0 \tag{52}$$

$$\lambda_1(\underline{\boldsymbol{u}} - \boldsymbol{u}) = \boldsymbol{0} \tag{53}$$

$$\lambda_2(\boldsymbol{u} - \bar{\boldsymbol{u}}) = \boldsymbol{0} \tag{54}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \bar{\boldsymbol{u}} \tag{55}$$

$$\lambda_1, \lambda_2 \geq 0 \tag{56}$$

where λ_1 and λ_2 are Lagrange multipliers of inequality constraints ($\underline{\boldsymbol{u}} \leq \boldsymbol{u}$) and ($\boldsymbol{u} \leq \bar{\boldsymbol{u}}$), respectively. However, the implicit function theorem can only handle the equations, not inequalities. Thus, we select only active inequalities of the lower-level optimization problem at the optimal value $u^*(\boldsymbol{p})$ and change the active inequality constraints as equality constraints, denoted as $G\boldsymbol{u} - \boldsymbol{h} = \boldsymbol{0}$ (Amos et al., 2018). With a new Lagrange multiplier ν for the equation ($G\boldsymbol{u} - \boldsymbol{h} = \boldsymbol{0}$), we state a transformed KKT conditions:

$$\nabla_{\boldsymbol{u}} \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{u}) + G^T \nu = \boldsymbol{0} \tag{57}$$

$$G\boldsymbol{u} - \boldsymbol{h} = \boldsymbol{0} \tag{58}$$

which simply denoted as $F(\boldsymbol{w}, \boldsymbol{p}) = 0$ where $\boldsymbol{w} = [\boldsymbol{u}, \nu]$.

Applying the implicit function theorem We employ the implicit function theorem on the KKT conditions described on the equation 57. Then we can derive $\nabla_{\boldsymbol{p}} u^*(\boldsymbol{p})$ by:

$$\nabla_{\boldsymbol{p}} \boldsymbol{w}^*(\boldsymbol{p}) = \begin{bmatrix} \nabla_{\boldsymbol{p}} u^*(\boldsymbol{p}) \\ \nabla_{\boldsymbol{p}} \nu^*(\boldsymbol{p}) \end{bmatrix} = -(\nabla_{\boldsymbol{w}} F(\boldsymbol{w}^*(\boldsymbol{p}), \boldsymbol{p}))^{-1} (\nabla_{\boldsymbol{p}} F(\boldsymbol{w}^*(\boldsymbol{p}), \boldsymbol{p})) \tag{59}$$

$$\nabla_{\boldsymbol{w}} F(\boldsymbol{w}^*(\boldsymbol{p}), \boldsymbol{p}) = \begin{bmatrix} \mathbf{H}_{\boldsymbol{u}} \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{u}) & G^T \\ G & \mathbf{0} \end{bmatrix} \tag{60}$$

$$\nabla_{\boldsymbol{p}} F(\boldsymbol{w}^*(\boldsymbol{p}), \boldsymbol{p}) = \begin{bmatrix} \nabla_{\boldsymbol{p}} (\nabla_{\boldsymbol{u}} \mathcal{L}_{\boldsymbol{p}}(\boldsymbol{u})) \\ \mathbf{0} \end{bmatrix} \tag{61}$$

B.4.2 HYPERPARAMETERS

We use the projected gradient-descent algorithm 100 times with the Adam optimizer for upper-level optimization problem on the both control problems. We reduce the upper-level learning rate from 0.05 to 0.001 with decaying factor 0.5 when the validation score does not decrease for 5 consecutive steps. For lower level problem, we use the same optimizer and learning rate scheduler described in Section B.3.