Parametric Pareto Set Learning: Amortizing Multi-Objective Optimization with Parameters

Ji Cheng, Xi Lin, Bo Xue, Qingfu Zhang, *Fellow, IEEE*

***Abstract*—Parametric multiobjective optimization (PMO) addresses the challenge of solving an infinite number of multiobjective optimization problems (MOPs), where optimal solutions must adapt to varying parameters. Traditional methods, such as parameterized multiobjective evolutionary algorithms (MOEAs), generate only a finite set of solutions and are unable to capture the continuous structure of the Pareto set across the entire parameter space, thereby leaving critical trade-offs unexplored. To overcome these challenges, we propose parametric Pareto set learning (PPSL), a novel framework that leverages amortized optimization to learn a unified mapping from parameters to the Pareto set with respect to the parameter. The framework employs a hypernetwork conditioned on input parameters, enabling real-time inference of optimal solutions for any parameter setting without the need for repeated optimization. By integrating low-rank adaptation (LoRA), PPSL achieves computational efficiency and scalability for PMO problems. To verify its effectiveness, we apply PPSL to dynamic multiobjective optimization problems (DMOPs) and MOPs with shared component design, where the parameters encode time-dependent or modular constraints. The experimental results demonstrate that PPSL significantly outperforms existing methods in terms of solution quality and adaptability. Importantly, we emphasize that PPSL is not limited to the two problems tested in this work; it is broadly applicable to any real-world problem that can be modeled as a PMO problem, offering a versatile and powerful solution for a wide range of applications. Code is available at: https://github.com/jicheng9617/PPSL.**

***Index Terms*—Parametric Multiobjective Optimization, Amortized Optimization, Pareto Set Learning, Dynamic Multiobjective Optimization, Multiobjective Optimization with Shared Modular.**

I. Introduction

PARAMETRIC optimization (PO), a pivotal methodology in mathematical programming [1], [2], examines how optimal solutions evolve in response to exogenous parameters. In this context, *parameters* are the variables that influence the objectives or constraints of the system but remain uncertain or beyond the control of the decision maker [3]. Unlike conventional optimization, which seeks a static solution, PO characterizes the optimal response as a function of these parameters, enabling adaptive decision-making across dynamic environments [4]. A variety of single-objective PO methods have been proposed, primarily rooted in linear or nonlinear programming frameworks [5], [6]. These techniques have been utilized across diverse fields, including industrial process optimization [7], automotive engine development [8], and medical applications such as diabetes management [9]. Despite their well-established development, most of these methods are capable of yielding only a limited set of solutions, which may not fully capture the complexity or diversity of potential optimal outcomes in real-world applications.

In contrast to these approximate numerical methods, amortized optimization has emerged as a powerful paradigm that leverages machine learning techniques to efficiently solve PO problems [10]. Unlike conventional methods that solve each problem instance from scratch, amortized optimization trains a model, such as a neural network, to predict optimal solutions directly from input parameters [11]. This approach capitalizes on recent advances in function approximation and computational infrastructure, enabling rapid inference times even for high-dimensional problems. By learning a mapping from problem parameters to solutions, amortized optimization shifts the computational burden from online solving to offline training, making it particularly suitable for real-time applications or scenarios requiring repeated evaluations [12], [13], [14], [15].

Although amortized optimization has proven to be effective in resolving PO problems, most existing methods have been developed for single-objective scenarios. However, real-world problems often involve multiple conflicting metrics, necessitating a shift to multiobjective formulations. In this paper, we consider the following parametric multiobjective optimization (PMO) problem:

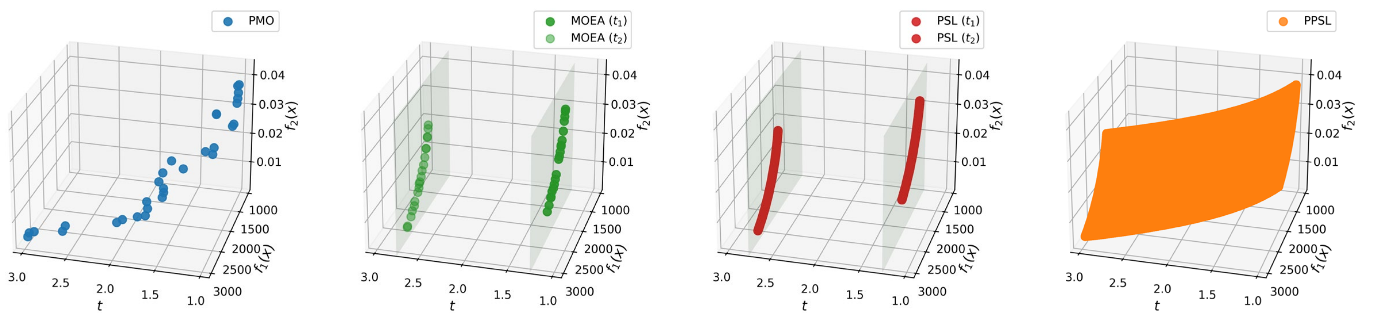
$$\mathcal{F}^*(\boldsymbol{t}) \subset \arg\min_{\boldsymbol{x}\in\mathcal{X}} \left(f_1(\boldsymbol{x},\boldsymbol{t}), f_2(\boldsymbol{x},\boldsymbol{t}), \ldots, f_m(\boldsymbol{x},\boldsymbol{t})\right), \qquad (1)$$

where $\boldsymbol{x}$ represents the decision variable vector in the decision space $\mathcal{X} \subset \mathbb{R}^n$, with n representing the dimensionality of the decision vector; $\boldsymbol{t}$ denotes parameters in the space $\Theta \subset \mathbb{R}^p$, where p is the dimensionality of the parameter space. $\mathcal{F}^*(\boldsymbol{t})$ defines the optimal solutions that simultaneously minimize the m objectives. For each specific parameter value $\boldsymbol{t}$, the Eq. (1) reduces to a conventional multiobjective optimization problem (MOP). Consequently, solving Eq. (1) involves addressing an infinite number of such MOPs, each corresponding to a different parameter value $\boldsymbol{t}$. In general, the involved m objectives conflict with each other, in which case the optimal solutions constitute a $(m-1)$-dimensional piecewise continuous manifold in the decision space, called Pareto set (PS), and its image in objective space forms the Pareto front (PF).

To solve PMO problems, an intuitive approach is to restrict the parameter to a specific value and then obtain a set of finite solutions with general-purpose multiobjective evolutionary algorithms (MOEAs), such as NSGA-II [16] and MOEA/D

This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China under Grant CityU11212524 and the Natural Science Foundation of China under Grant 62276223. *(Corresponding author: Qingfu Zhang)*

All the authors are with Department of Computer Science, City University of Hong Kong, Hong Kong, China. Ji Cheng and Qingfu Zhang are also with the City University of Hong Kong Shenzhen Research Institute, Shenzhen, China. Email: {J.Cheng, xi.lin, boxue4-c}@my.cityu.edu.hk, qingfu.zhang@cityu.edu.hk.



(a) PMO (b) MOEA (c) PSL (d) PPSL

Fig. 1. Comparison between **(a)** solutions found by a dedicated PMO method (10,000 evaluations), **(b)** approximate PFs for two specific parameter values found by MOEA (10,000 evaluations for each parameter value), **(c)** approximate PFs for two specific parameter values learned by PSL (10,000 evaluations for each parameter value), and **(d)** approximate PFs for all parameter values found by our PPSL method (26,000 evaluations to obtain the manifold for all parameter values).

[17], as shown in Fig. 1(b). Similarly, dedicated PMO methods [18], [19] have been developed to handle parameterized scenarios by incorporating mechanisms like parameterized Pareto dominance [20]. While these approaches adapt to parameter variations, they still generate a finite set of solutions distributed across both objective and parameter spaces. As illustrated in Fig. 1(a), such discrete solutions may fail to cover critical regions of interest for specific parameter values, leaving gaps in the decision-maker's ability to explore optimal trade-offs comprehensively.

Traditional MOEAs are fundamentally limited by their reliance on fixed populations, producing solutions that often inadequately represent the intricate structure of the PS. In high-dimensional or complex decision spaces, the PS may form continuous manifolds or exhibit nonlinear patterns that cannot be captured by a sparse set of points. This limitation hinders the exploration of objective trade-offs and risks excluding solutions critical to real-world applications. To address this gap, Pareto set learning (PSL) was recently proposed to approximate the entire PS as a continuous mapping via neural networks [21], [22], [23], [24], [25]. By unifying the PS into a single model, PSL enables instant retrieval of optimal solutions for arbitrary trade-off preferences through a single forward pass, bypassing the need for iterative optimization. This paradigm shift enhances flexibility and efficiency, particularly in scenarios requiring rapid decision-making or dynamic adjustments. However, adopting PSL to PMO problems remains confined to fixed parameter values, as demonstrated in Fig. 1(c), necessitating retraining for each new instance of $\boldsymbol{t}$. This restriction undermines their applicability to dynamic systems where parameters evolve continuously, highlighting an unresolved challenge in bridging parameterized optimization with continuous PS approximation.

To fill in the gap between amortized optimization and PMO, this work proposes a parametric Pareto set learning (PPSL) method. PPSL leverages the strengths of amortized optimization to learn a mapping from parameters to the entire PS, enabling efficient and adaptive solutions for PMO problems. Specifically, PPSL employs a hypernetwork to generate the parameters of a PS model conditioned on the input parameters, allowing the model to predict approximate PS for any given parameter setting without re-optimization. By integrating low-rank adaptation (LoRA) [26] into the framework, PPSL further reduces computational complexity and enhances scalability, making it suitable for high-dimensional and large-scale problems. This approach not only bridges the gap between traditional PMO and machine learning but also provides a practical and efficient solution for real-world applications requiring parameter-dependent decision-making.

Our main contributions can be concluded as follows:

- We make a first attempt to resolve PMO problems using amortized optimization, introducing a novel framework that leverages parameterized models to learn the mapping from parameters to the family of Pareto sets. This approach significantly reduces the computational burden compared to traditional methods that require re-optimization for each parameter setting.
- We apply the PPSL method to effectively solve dynamic multiobjective optimization problems (DMOPs). By learning the PS as a function of time-varying parameters, PPSL achieves substantial improvements in efficiency and adaptability. We conducted experiments on 14 commonly tested DMOP cases, and the results verify that the DMOPs can be efficiently solved by our proposed PMO method.
- We extend the application of PPSL to multiobjective optimization with shared components, demonstrating its effectiveness in scenarios where certain design variables must share identical settings. Our experiments show that PPSL enables more efficient and practical designs, particularly in modular design and manufacturing contexts, where shared components are critical for cost reduction and production efficiency.

The remainder of this paper is organized as follows. Section II provides the necessary preliminaries, including definitions and discussions related to PMO and PSL. Section III presents the proposed PPSL framework, detailing its model structure, training process, and algorithmic implementation. Section IV demonstrates the application of PPSL to DMOPs. Section V explores the application of PPSL to MOPs with shared components, highlighting its practical utility in real-world scenarios. Finally, Section VI concludes this work and discusses it with future work.

II. Preliminaries: Parametric multiobjective Optimization and Pareto Set Learning

A. Multiobjective Optimization

A continuous MOP can be defined as:

$$\min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_m(\boldsymbol{x})), \tag{2}$$

where $\boldsymbol{x}$ is an element in the decision space $\mathcal{X} \subset \mathbb{R}^n$, and $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^m$ is the mapping from decision space to objective space. Without loss of generality, the m objectives conflict with each other, which means a single solution can not optimize them simultaneously. To describe the optimality, we have following definitions:

Definition 1 (Pareto Dominance). *Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$, $\boldsymbol{x}_1$ is said to dominate $\boldsymbol{x}_2$ (denoted as $\boldsymbol{x}_1 \prec \boldsymbol{x}_2$), if and only if $f_i(\boldsymbol{x}_1) \leq f_i(\boldsymbol{x}_2), \forall i \in \{1, 2, \dots, m\}$ and $\exists j \in \{1, 2, \dots, m\}, f_j(\boldsymbol{x}_1) < f_j(\boldsymbol{x}_2)$.*

Definition 2 (Pareto Optimality). *A solution $\boldsymbol{x}^*$ is Pareto optimal if there is no $\boldsymbol{x} \in \mathcal{X}$ such that $\boldsymbol{x} \prec \boldsymbol{x}^*$. The set consists of all Pareto optimal solutions is called as Pareto set, and its image in objective space is called Pareto front.*

Definition 3 (Strict Pareto Dominance). *Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$. $\boldsymbol{x}_1$ is said to strictly dominate $\boldsymbol{x}_2$, denoted by $\boldsymbol{x}_1 \prec_{\text{strict}} \boldsymbol{x}_2$, if and only if $f_i(\boldsymbol{x}_1) < f_i(\boldsymbol{x}_2), \forall i \in \{1, \dots, m\}$.*

Definition 4 (Weakly Pareto Optimality). *A solution $\boldsymbol{x}_1 \in \mathcal{X}$ is weakly Pareto optimal if there does not exist $\boldsymbol{x}_2 \in \mathcal{X}$ such that $\boldsymbol{x}_2 \prec_{\text{strict}} \boldsymbol{x}_1$. The set of all weakly Pareto optimal solutions is called the weakly Pareto set, and its image in the objective space is called the weakly Pareto front.*

For a continuous MOP, the PS is a $(m-1)$-dimensional manifold under mild conditions [27], [28].

B. Parametric Multiobjective Optimization

PMO extends traditional multiobjective optimization by introducing parameters that influence the problem's structure. Formally, a PMO problem can be defined as in Eq. (1). These parameters induce a family of MOPs indexed by $\boldsymbol{t}$, leading to dynamic Pareto sets and fronts parameterized by $\boldsymbol{t}$. Unlike traditional MOPs, where the PS is a $(m-1)$-dimensional manifold, the PS in PMO problems often forms a higher-dimensional manifold (dimension $m + p - 1$) due to the additional dependency on $\boldsymbol{t}$. Moreover, small changes in $\boldsymbol{t}$ can significantly alter the trade-off relationships between objectives, requiring adaptive optimization strategies.

We would like to make the following remarks on PMO problems:

- If $\boldsymbol{t}$ is fixed, the PMO problem reduces to a conventional MOP, and existing MOEAs like MOEA/D or NSGA-II can be applied directly; however, solving each $\boldsymbol{t}$-specific MOP independently (e.g., via MOEAs) is computationally prohibitive, especially for real-time applications.
- The goal of PMO is to learn the mapping from $\boldsymbol{t}$ to the PS $\mathcal{F}^*(\boldsymbol{t})$, enabling efficient adaptation to parameter changes without re-optimization. In addition, ensuring smooth transitions in PS/PF as $\boldsymbol{t}$ varies is critical for practical decision-making but difficult to achieve with disjoint optimizations.
- Existing methods lack mechanisms to explicitly model the relationship between $\boldsymbol{t}$ and the PS, limiting their ability to generalize for unseen parameters, resulting in the challenge of capturing the complex dependency of the PS on $\boldsymbol{t}$ while ensuring smooth and continuous transitions in the solution space.

These challenges motivate the need for the method that directly learns the *parameter-to-PS* mapping, enabling efficient adaptation to parameter changes while preserving optimality and smoothness.

C. Pareto Set Learning

PSL was developed in [21], [22], [23], [24] to represent the entire PS by a single model. For a MOP, the PS is modeled as:

$$\boldsymbol{x} = h_{\boldsymbol{\theta}}(\boldsymbol{\lambda}), \tag{3}$$

where $\boldsymbol{\lambda} \in \Delta^{m-1}$ is the preference vector on the simplex $\Delta^{m-1} = \{\boldsymbol{\lambda} \in \mathbb{R}^m_+ \mid \sum_{i=1}^m \lambda_i = 1\}$, and $h_{\boldsymbol{\theta}}(\cdot)$ is a model with learnable parameters $\boldsymbol{\theta}$. The goal of PSL is to find the optimal model with $\boldsymbol{\theta}^*$ such that $\boldsymbol{x}^* = h_{\boldsymbol{\theta}^*}(\boldsymbol{\lambda})$ is the optimal solution for:

$$\min_{\boldsymbol{x}\in\mathcal{X}} l_{\text{agg}}(\boldsymbol{x}|\boldsymbol{\lambda}), \quad \forall \boldsymbol{\lambda} \in \Delta^{m-1}, \tag{4}$$

where $l_{\text{agg}}(\cdot)$ is the aggregation function. PSL naturally extends the decomposition-based MOEA (MOEA/D) [17] by generalizing the finite population to a parameterized model. Using the learned PS model, practitioners can readily obtain suboptimal solutions for their preferences. Let $P_{\boldsymbol{\lambda}}$ be distribution of the preference assigned by decision maker; the task of PSL can be formally defined as:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\lambda}\sim P_{\boldsymbol{\lambda}}} \left[l_{\text{agg}}(h_{\boldsymbol{\theta}}(\boldsymbol{\lambda})|\boldsymbol{\lambda}) \right]. \tag{5}$$

The optimization quality directly depends on the aggregation function $l_{\text{agg}}(\cdot)$. Aggregation methods transform m objective metrics into a single function value, and the optimal value of the aggregation function corresponds to a solution on the PS [29]. A commonly used and straightforward method is the weighted sum, formulated as:

$$l_{\text{ws}}(\boldsymbol{x}|\boldsymbol{\lambda}) = \sum_{i=1}^m \lambda_i f_i(\boldsymbol{x}). \tag{6}$$

Despite its simplicity, the weighted sum method fails to capture solutions on the non-convex parts of the PF, resulting in solutions only on the convex hull [30], [31]. Another widely used scalarization method, the Tchebycheff (TCH) scalarization, is well-known for its theoretical guarantees [27]. The TCH scalarization is defined as:

$$l_{\text{tch}}(\boldsymbol{x}|\boldsymbol{\lambda}) = \max_{i\in[m]} \left\{ \lambda_i \left(f_i(\boldsymbol{x}) - (z_i^* - \varepsilon) \right) \right\}, \tag{7}$$

where $z_i^* \leq \min_{\boldsymbol{x}\in\mathcal{X}} f_i(\boldsymbol{x})$ is the ide al value for the i-th objective, and $\varepsilon \geq 0$ is a small positive value. Under mild

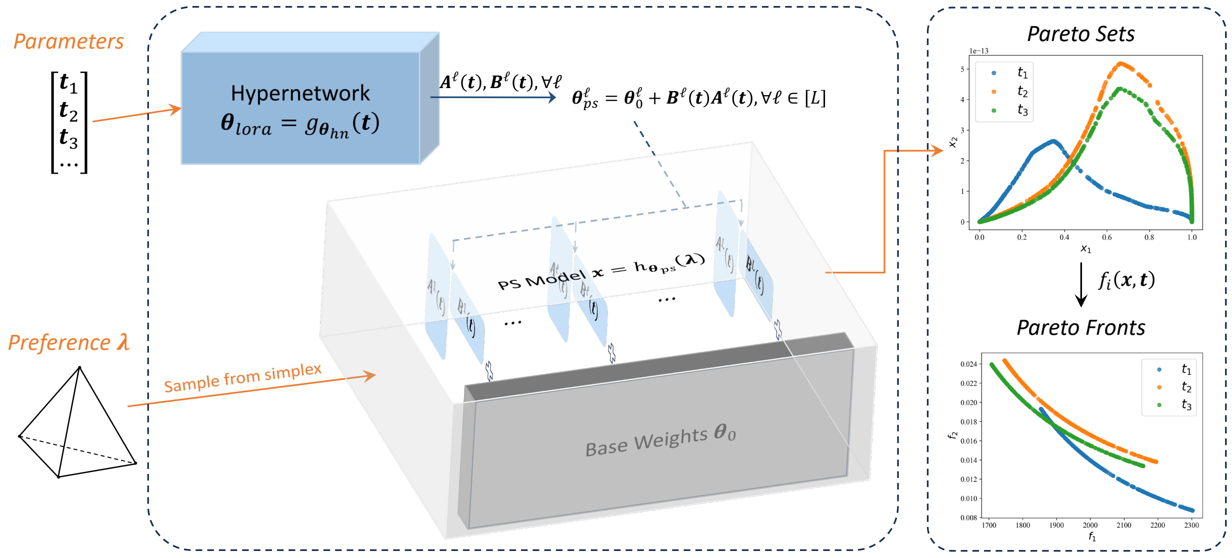

Parameters
Hypernetwork
$\theta_{lora} = g_{\theta_{hn}}(t)$
$A^\ell(t), B^\ell(t), \forall\ell$
$\theta_{ps}^\ell = \theta_0^\ell + B^\ell(t)A^\ell(t), \forall\ell \in [L]$
PS Model $x = h_{\theta_{ps}}(\lambda)$
Preference λ
Sample from simplex
Base Weights θ_0
Pareto Sets
$f_i(x, t)$
Pareto Fronts

Fig. 2. **Parametric Pareto set learning**. Three examples t_1, t_2, and t_3 are demonstrated for the process predicting the PS for each parameter.

conditions, the optimal solution x^*_λ of the TCH scalarization exhibits a desirable property [31]:

$$\lambda_i(f_i(x^*_\lambda) - z_i^*) = \lambda_j(f_j(x^*_\lambda) - z_j^*), \quad \forall 1 \le i, j \le m. \tag{8}$$

This property ensures that the solution x^*_λ satisfies the exact preferred trade-off requirement [32]. Furthermore, the TCH scalarization provides a necessary and sufficient condition for identifying all weakly Pareto optimal solutions [29]:

Theorem 1 ([29])**.** *A feasible solution $x \in \mathcal{X}$ is weakly Pareto optimal for the original MOP if and only if there exists a valid preference vector λ such that x is an optimal solution of minimizing Eq.* (7).

This theoretical foundation makes the TCH scalarization a powerful tool for solving MOPs, as it guarantees the discovery of all relevant Pareto solutions under appropriate conditions.

Recently, an improved version of the TCH method, known as the smooth Tchebycheff (STCH) aggregation [33], has been proposed to address the limitations of the classical TCH scalarization. The STCH aggregation is defined as:

$$l_{\text{stch}}(x \mid \lambda, \mu) = \mu \log\left(\sum_{i=1}^{m} e^{\frac{\lambda_i(f_i(x) - (z_i^* - \varepsilon))}{\mu}}\right), \tag{9}$$

where μ is the smoothing parameter. With a constant smoothing value, we denote the STCH scalarization as $l_{\text{stch}}(x \mid \lambda)$ for simplicity. The STCH aggregation serves as a smooth approximation to the classical TCH method, offering several advantages, including faster convergence rates, lower per-iteration complexity, and strong theoretical guarantees. Under mild conditions, the STCH aggregation can identify the entire PS, as stated in the following theorem:

Theorem 2 ([33])**.** *There exists a smoothing parameter μ^* such that, for any $0 < \mu < \mu^*$, every Pareto solution of the MOP corresponds to an optimal solution of minimizing Eq.* (9) *with some valid preference vector λ.*

The smoothness of the STCH aggregation not only enhances its computational efficiency but also makes it particularly suitable for applications requiring fast and robust optimization. In this work, we use the STCH method for the proposed and comparative methods. Furthermore, the proposed framework can be easily extended to other aggregation approaches, making it a valuable tool for a wide range of MOPs.

III. Parametric Pareto Set Learning

A. Model Structure

We consider the combination of amortized optimization and PSL to learn all the PS models for all parameters. The PS model can be represented as:

$$x = h_{\theta_{\text{ps}}}(\lambda), \tag{10}$$

where λ is the preference vector, and $h(\cdot)$ denotes the model with learnable parameter θ_{ps} mapping the preference to optimal solutions. To learn the PS model for every parameter setting, we propose using a hypernetwork to construct the mapping from the parameter t to the learnable parameter θ_{ps}, where the hypernetwork can be formulated as:

$$\theta_{\text{ps}} = g_{\theta_{\text{hn}}}(t) \tag{11}$$

where θ_{hn} is the learnable parameter of the model $g(\cdot)$.

The parameter t is typically low-dimensional, often being one-dimensional or having only a few dimensions, whereas the output of the hypernetwork, θ_{ps}, can be extremely high-dimensional, ranging from hundreds of thousands to millions or more. This significant dimensionality gap may lead to potential issues such as overfitting or inefficient model training. To address these challenges, we propose to employ the LoRA technique for the PS model. Furthermore, since the PS for different parameters may share a common structure, the LoRA framework is particularly well-suited for capturing such concurrent characteristics. The whole model structure is illustrated in Fig. 2.

Algorithm 1 Parametric Pareto set learning

```
1:  Input: Hypernetwork model $\boldsymbol{\theta}_{lora} = g_{\boldsymbol{\theta}_{hn}}(\boldsymbol{t})$, PS model $\boldsymbol{x} = h_{\boldsymbol{\theta}_{ps}}(\boldsymbol{\lambda})$, number of epochs $T$, batch size of sampled parameters $B_t$, batch size of sampled preference $B_l$, learning rate for hypernetwork $\eta_{hn}$, learning rate for base PS model $\eta_b$.
2:  for $i = 1, \dots, T$ do
3:      Sample $B_t$ parameters $\{\boldsymbol{t}_1, \dots, \boldsymbol{t}_{B_t}\}$ based on the distribution $P_{\boldsymbol{t}}$ for specific task
4:      for $\boldsymbol{t} = \boldsymbol{t}_1, \dots, \boldsymbol{t}_{B_t}$ do
5:          Sample $B_l$ weight vectors $\{\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_{B_l}\}$ with the distribution $P_{\boldsymbol{\lambda}}$
6:          Obtain the weights of the PS model by Eq. (13) and Eq. (12)
7:          Generate the solutions with respect to the weight vectors by Eq. (10)
8:          Compute the loss $\mathcal{L}$ and evaluate the gradient by back-propagation Eq. (19) and Eq. (20) (combined with evolution strategy Eq. (21) for black-box scenario)
9:          Update the learnable parameters of the base PS model $\boldsymbol{\theta}_0$ by Eq. (22)
10:     end for
11:     Update the learnable parameters of the hypernetwork model $\boldsymbol{\theta}_{\mathrm{hn}}$ as in Eq. (23)
12: end for
13: Output: Hypernetwork model $\boldsymbol{\theta}_{lora} = g_{\boldsymbol{\theta}_{hn}}(\boldsymbol{t})$ and PS model $\boldsymbol{x} = h_{\boldsymbol{\theta}_{ps}}(\boldsymbol{\lambda})$
```

Let $\boldsymbol{\theta}_{\mathrm{ps}}^{\ell}$ denote the learnable parameter of the ℓth layer in the PS model. We employ a base network with the learnable parameter $\boldsymbol{\theta}_0^{\ell}$ for the ℓth layer, which shares the same structure as $\boldsymbol{\theta}_{\mathrm{ps}}^{\ell}$. Integrating a low-rank matrix $\boldsymbol{B}^{\ell}(\boldsymbol{t})\boldsymbol{A}^{\ell}(\boldsymbol{t})$, the learnable parameter $\boldsymbol{\theta}_{\mathrm{ps}}^{\ell}$ can be formally expresses as:

$$\boldsymbol{\theta}_{\mathrm{ps}}^{\ell} = \boldsymbol{\theta}_0^{\ell} + \boldsymbol{B}^{\ell}(\boldsymbol{t})\boldsymbol{A}^{\ell}(\boldsymbol{t}). \tag{12}$$

The collection of the items of these LoRA matrices across all layers constitutes the output of the hypernetwork, denoted as $\boldsymbol{\theta}_{lora}$; then the hypernetwork in Eq. (11) can be adapted to:

$$\boldsymbol{\theta}_{\mathrm{lora}} = g_{\boldsymbol{\theta}_{\mathrm{hn}}}(\boldsymbol{t}). \tag{13}$$

Let $\boldsymbol{B}^{\ell}(\boldsymbol{t}) \in \mathbb{R}^{d^{\ell} \times r}$ and $\boldsymbol{A}^{\ell}(\boldsymbol{t}) \in \mathbb{R}^{r \times k^{\ell}}$ denote the dimensionality of the low-rank matrices. Then, the number of predictable parameters for the ℓth layer is $r(d^{\ell} + k^{\ell})$. Compared to the original framework, which requires $d^{\ell}k^{\ell}$ parameters, the LoRA framework significantly reduces the number of parameters due to the low value of rank r. This advantage becomes particularly pronounced as the size of the PS model increases to accommodate more complex PS manifolds.

B. Training Framework

To solve PMO problems, our task is to optimize the learnable parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_{\mathrm{hn}}$ such that $\boldsymbol{x} = h_{\boldsymbol{\theta}_{\mathrm{ps}}}(\boldsymbol{\lambda})$ is the optimal solution for the following problem.

$$\min_{\boldsymbol{x} \in \mathcal{X}} \; l_{\mathrm{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t}), \quad \forall \boldsymbol{\lambda} \in \Delta^{m-1}, \boldsymbol{t} \in \Theta, \tag{14}$$

where l_{stch} represents the STCH aggregation function.

Let $P_{\boldsymbol{t}}$ be the task-specific distribution for the parameter $\boldsymbol{t}$, where the distribution is highly correlated with the characteristics of the task. It can be increased sequentially or uniformly distributed over the parameter space $\mathcal{T}$, where concrete applications are studied in the experimental sections.

Let $P_{\boldsymbol{\lambda}}$ denote the probability distribution of the preference specified by users. Note that the user's priority with respect to different objectives can be fitted in this distribution. In this context, our task can be expressed as:

$$\min_{\boldsymbol{\theta}_{\mathrm{hn}}, \boldsymbol{\theta}_0} \mathcal{L}(\boldsymbol{\theta}_{\mathrm{hn}}, \boldsymbol{\theta}_0) = \mathbb{E}_{\boldsymbol{t} \sim P_{\boldsymbol{t}}, \boldsymbol{\lambda} \sim P_{\boldsymbol{\lambda}}} \left[l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t})}(\boldsymbol{\lambda}) | \boldsymbol{\lambda}, \boldsymbol{t} \right) \right], \tag{15}$$

where we use $\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t})$ to represent $\boldsymbol{\theta}_{\mathrm{ps}}$ with respect to parameter $\boldsymbol{t}$ for simplicity.

Algorithm 1 presents the training framework of the proposed PPSL method. The algorithm initiates with the hypernetwork and PS models with randomly initialized or trained parameters. Then each epoch of the training procedure can be divided into three steps:

1) Generate Solutions for Parameter and Preference Samples: To begin with, we use Monte Carlo sampling to approximate the training objective in Eq. (15), since the expectation can not be directly calculated in most cases. In this way, we first sample B_t parameters from its probability distribution $P_{\boldsymbol{t}}$, then for each parameter we subsequently sample B_l preference vectors from the specified distribution $P_{\boldsymbol{\lambda}}$. The PPSL model captures the common characteristics of Pareto sets between different parameter settings through the base PS model, and generates the corresponding sets of solutions $\{h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t}_i)}(\boldsymbol{\lambda}_j)\}_{j=1}^{B_l}$ for each parameter $\boldsymbol{t}_i$.

2) Evaluate Gradients for Learnable Parameters: With the generated solutions, we can approximate the task objective $\mathcal{L}(\boldsymbol{\theta}_{\mathrm{hn}}, \boldsymbol{\theta}_0)$ by:

$$\tilde{\mathcal{L}} = \frac{1}{B_t} \sum_{i=1}^{B_t} \frac{1}{B_l} \sum_{j=1}^{B_l} l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t}_i)}(\boldsymbol{\lambda}_j) | \boldsymbol{\lambda}_j, \boldsymbol{t}_i \right), \tag{16}$$

where we use $\mathcal{L}$ to represent $\mathcal{L}(\boldsymbol{\theta}_{\mathrm{hn}}, \boldsymbol{\theta}_0)$ with no specification.

If $l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t}_i)}(\boldsymbol{\lambda}_j) | \boldsymbol{\lambda}_j, \boldsymbol{t}_i \right)$ is well-behaved [34] for all $\boldsymbol{t} \sim P_{\boldsymbol{t}}$ and $\boldsymbol{\lambda} \sim P_{\boldsymbol{\lambda}}$, we can evaluate the gradient with respect to the learnable parameters of the base PS model as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_0} \mathcal{L} &= \nabla_{\boldsymbol{\theta}_0} \mathbb{E}_{\boldsymbol{t} \sim P_{\boldsymbol{t}}, \boldsymbol{\lambda} \sim P_{\boldsymbol{\lambda}}} \left[l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t})}(\boldsymbol{\lambda}) | \boldsymbol{\lambda}, \boldsymbol{t} \right) \right] \\ &= \mathbb{E}_{\boldsymbol{t} \sim P_{\boldsymbol{t}}, \boldsymbol{\lambda} \sim P_{\boldsymbol{\lambda}}} \nabla_{\boldsymbol{\theta}_0} \left[l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t})}(\boldsymbol{\lambda}) | \boldsymbol{\lambda}, \boldsymbol{t} \right) \right], \end{aligned} \tag{17}$$

where the expectation and differentiation are swapped. Based on this, we have

$$\nabla_{\boldsymbol{\theta}_0} \tilde{\mathcal{L}} = \frac{1}{B_t B_l} \sum_{i=1}^{B_t} \sum_{j=1}^{B_l} \nabla_{\boldsymbol{\theta}_0} l_{\mathrm{stch}} \left(h_{\boldsymbol{\theta}_{\mathrm{ps}}(\boldsymbol{t}_i)}(\boldsymbol{\lambda}_j) | \boldsymbol{\lambda}_j, \boldsymbol{t}_i \right). \tag{18}$$

With a specific $\boldsymbol{\lambda}$, we can evaluate the gradient by the chain rule:

$$\nabla_{\boldsymbol{\theta}_0} l_{\mathrm{stch}} = \frac{\partial \boldsymbol{\theta}_{\mathrm{ps}}}{\partial \boldsymbol{\theta}_0} \cdot \frac{\partial h_{\boldsymbol{\theta}_{\mathrm{ps}}}(\boldsymbol{\lambda})}{\partial \boldsymbol{\theta}_{\mathrm{ps}}} \cdot \nabla_{\boldsymbol{x}} l_{\mathrm{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t}). \tag{19}$$

Similarly, for the gradient with respect to $\boldsymbol{\theta}_{\text{hn}}$, we have

$$\nabla_{\boldsymbol{\theta}_{\text{hn}}} l_{\text{stch}} = \frac{\partial \boldsymbol{\theta}_{\text{lora}}}{\partial \boldsymbol{\theta}_{\text{hn}}} \cdot \frac{\partial \boldsymbol{\theta}_{\text{ps}}}{\partial \boldsymbol{\theta}_{\text{lora}}} \cdot \frac{\partial h_{\boldsymbol{\theta}_{\text{ps}}}(\boldsymbol{\lambda})}{\partial \boldsymbol{\theta}_{\text{ps}}} \cdot \nabla_{\boldsymbol{x}} l_{\text{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t}). \quad (20)$$

In Eq. (19) and Eq. (20), the Jacobian matrices $\frac{\partial h_{\boldsymbol{\theta}_{\text{ps}}}(\boldsymbol{\lambda})}{\partial \boldsymbol{\theta}_{\text{ps}}}$ and $\frac{\partial \boldsymbol{\theta}_{\text{lora}}}{\partial \boldsymbol{\theta}_{\text{hn}}}$ can be directly obtained by the backpropagation algorithm, as well as the term $\frac{\partial \boldsymbol{\theta}_{\text{ps}}}{\partial \boldsymbol{\theta}_{\text{lora}}}$ due to the matrix operator in Eq. (12).

Real-world applications often lack access to gradient information, meaning we can only evaluate the scalarized aggregation function $l_{\text{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t})$ for a given solution $\boldsymbol{x}$, preference vector $\boldsymbol{\lambda}$, and parameter $\boldsymbol{t}$, but cannot directly compute its gradient $\nabla_{\boldsymbol{x}} l_{\text{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t})$. To address this limitation, we employ a simple yet effective evolution strategy (ES) method [35], [36] to enable gradient-based training within our PPSL framework. This approach estimates the gradient by sampling K independent n-dimensional random vectors $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K\}$, where each $\boldsymbol{u}_k$ is drawn from a carefully designed distribution. Specifically, we follow the method in [37], sampling each $\boldsymbol{u}_k$ as $\boldsymbol{u}_k \sim (B_{0.5} - 0.5)/0.5$, where $B_{0.5}$ follows a Bernoulli distribution with a success probability of 0.5. This sampling strategy ensures diversity in exploration while maintaining computational efficiency.

To further enhance the optimization process, we adopt a rank-based fitness shaping approach [38], [25], which normalizes the scalarized function values $l_{\text{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t})$ across different preference vectors $\boldsymbol{\lambda}$ and parameters $\boldsymbol{t}$. By sorting the sampled vectors based on their aggregation function values and assigning them evenly spaced rank-based fitness values, we approximate the gradient as follows:

$$\overline{\nabla}_{\boldsymbol{x}} l_{\text{stch}}(\boldsymbol{x}|\boldsymbol{\lambda}, \boldsymbol{t}) = \frac{1}{\sigma K} \sum_{k'=1}^{K} r_{k'} \cdot \boldsymbol{u}_{k'}, \quad (21)$$

where $r_{k'}$ represents the rank-based fitness value assigned to each sampled vector $\boldsymbol{u}_{k'}$, which is uniformly distributed on $[-0.5, 0.5]$. To determine $r_{k'}$, we first sort the sampled vectors $\{\boldsymbol{u}_k\}_{k=1}^K$ in ascending order based on their corresponding values $l_{\text{stch}}(\boldsymbol{x} + \sigma \boldsymbol{u}_k|\boldsymbol{\lambda}, \boldsymbol{t})$. Then, we assign predefined normalized values $r_1 \leq r_2 \leq \ldots \leq r_K$ to these vectors according to their ranks. The specific choice of fitness values is nonessential; thus, we use a simple transformation to ensure consistency [25]. With the estimated gradient, we can directly compute the gradients in Eq. (19) and Eq. (20) using the backpropagation method.

3) Update Model Parameters: The learnable parameters of the base PS model are trained based on all the parameter samples, which indicates that it is used to capture the common feature of the Pareto sets with respect to different parameters. By stochastic gradient descent method, the update process can be expressed as:

$$\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}_0 - \eta_b \nabla_{\boldsymbol{\theta}_0} \tilde{\mathcal{L}}, \quad (22)$$

where η_b denotes the learning rate of the base PS model. This suggests that the learnable parameters of the LoRA structure are fine-tuned on the discrepancy between the base PS model and the PS for the specific parameters. The parameters of the hypernetwork are optimized by:

$$\boldsymbol{\theta}_{hn} \leftarrow \boldsymbol{\theta}_{hn} - \eta_{hn} \nabla_{\boldsymbol{\theta}_{hn}} \tilde{\mathcal{L}}, \quad (23)$$

where η_{hn} is the rate for hypernetwork.

C. Evaluation Protocol

PPSL is evaluated with respect to the *task–parameter distribution* $P_{\boldsymbol{t}}$, and the evaluation strategy is tailored to the specific characteristics of each application. For any given parameter value $\boldsymbol{t}$, the hypernetwork generates the task–specific weights $\boldsymbol{\theta}_{\text{ps}}(\boldsymbol{t})$, which are used by the PS model to compute solutions as $\boldsymbol{x} = h_{\boldsymbol{\theta}_{\text{ps}}(\boldsymbol{t})}(\boldsymbol{\lambda})$.

In this work, evaluation is performed differently for the two application domains: *(i) DMOPs:* Evaluation is performed *online*, which means that in each generation, we evaluate the network only for the current time parameter. *(ii) MOPs with Shared Components:* Evaluation is conducted *offline*, as the admissible values of the shared component are known beforehand. After training on the complete parameter distribution P_t, the model is fixed. We then evaluate it by querying arbitrary parameter values.

IV. Application Study: Dynamic multiobjective Optimization

In this section, we apply the proposed PPSL method to a widely studied and complex problem: dynamic multiobjective optimization.

A. Problem Introduction

DMOPs involve multiple conflicting objectives that evolve over time, leading to changes of the PS and PF as the problem landscape shifts. These time-varying characteristics make DMOPs particularly challenging, as algorithms must not only optimize for multiple objectives but also adapt to dynamic changes in the environment. DMOPs are prevalent in real-world applications, such as dynamic portfolio optimization [39], industrial process control [40], and machine learning [41], where parameters or conditions frequently change during operation.

Traditional dynamic multiobjective optimization algorithms typically follow a two-step process: first, detect environmental changes, and if changes occur, re-initialize the population and

TABLE I
Hyperparameters setting for PPSL.

Hyperparameter	Value
# layers for hypernetwork	4
# neurons for hypernetwork	1024
# layers for PS model	2 for $m = 2$ and 3 for $m = 3$
# neurons for PS model	512 for $m = 2$ and 256 for $m = 3$
Rank r	3
Learning rate η_{hn}	10^{-5}
Learning rate η_{b}	10^{-3}
# samples for parameter B_t	20
# samples for weight vector B_l	10
Smoothing parameter in STCH	0.01

apply static multiobjective optimization techniques. However, this approach overlooks the inherent correlations between solutions at different time indices t, treating each time step as an independent problem. In contrast, using the proposed PPSL method in DMOPs explicitly leverages the sequential nature of t as a parameter, enabling the discovery of relationships between solutions across different time steps. This section explores the effectiveness and efficiency of PPSL in solving DMOPs, particularly in capturing and utilizing the temporal correlations inherent in such problems.

B. Experimental Setups

1) Optimization Problems: To systematically evaluate the performance of the proposed PPSL method in solving DMOPs, we employ the DF test suite [42], an advanced benchmark that extends earlier test suites (e.g., FDA [43] and dMOP [44]). The DF test suite is designed to encompass a wide range of challenging characteristics, such as mixed convexity-concavity and variable-linkage, providing a comprehensive benchmark for assessing algorithm robustness and adaptability. The detailed characteristics for DF problems can refer to [42]. In these test instances, the dynamic characteristic is obtained by:

$$t = \frac{1}{n_t} \lfloor \frac{\tau}{\tau_t} \rfloor, \tag{24}$$

where τ denotes the generation counter, and n_t, τ_t represent the change severity and frequency, respectively.

To adapt the PPSL method to the online nature of DMOPs, the task distribution $P_{\boldsymbol{t}}$ in step 3 of Algorithm 1 is set to a degenerate distribution. This means that at each time slot, all B_t sampled parameters $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{B_t}$ are identical, corresponding to the current normalized time index. Specifically, the normalized time index $\tau/T_{\max}$ is treated as a *discrete and sequentially increasing parameter*, where $T_{\max}$ is the predefined maximum number of generations.

For instance, consider a scenario where the true problem dynamics change in discrete steps; the environment might remain static for a certain duration (τ_t) and then abruptly change by degree (n_t). The algorithm, however, cannot observe this underlying pattern. Instead, it uses the normalized time index as a proxy for the changing environment. Assuming $T_{\max} = 10$, the sequence of parameters fed to the hypernetwork over successive generations would be $\{0.0, 0.1, 0.2, 0.3, \ldots\}$. Therefore, at the first generation, the model is trained exclusively using the time parameter $\boldsymbol{t} = 0.0$; at the second generation, it uses $\boldsymbol{t} = 0.1$, and so on. The hypernetwork is thus tasked with learning a mapping from this smooth, linear progression of time to the potentially complex and abrupt changes of the true Pareto set.

For all benchmark problems, the number of decision variables is set to 10, and the population size is fixed at 200. The environment undergoes 50 changes, corresponding to 50 τ_t generations.

2) Performance Indicator: To quantify performance, we utilize two widely adopted metrics: the mean inverted generational distance (MIGD) and the mean hypervolume (MHV), which are the average metrics over time steps. These metrics offer a compact and intuitive evaluation of both convergence and diversity. Detailed definitions of these measures can be found in [45], [46], [47].

It is important to emphasize that for the DMOP application, both the training and evaluation of PPSL are performed in an online fashion. At each generation τ, the model is updated using only the information from the current time index $\tau/T_{\max}$. The performance is then immediately evaluated on the solutions generated for that specific generation. This protocol aligns with the fundamental nature of dynamic optimization, where an algorithm is not required to perfectly recall past states but must adapt effectively to the present environment. This evaluation methodology is consistent with that used for all baseline DMOEAs. If needed, practitioners can save the model parameters at each time step, making it possible to inspect the PS manifold at any time even after the training process is complete.

3) Algorithm Setting: We compare the proposed PPSL method against several baseline dynamic multiobjective evolutionary algorithms (DMOEAs), including DNSGA-II-A [48], DNSGA-II-B [48], and KGB [49]. The static optimizers within these algorithms are implemented as described in their original papers to preserve their design integrity. All baselines are implemented through the pymoo platform [1], and our proposed PPSL method [2]. The hyperparameter setting for PPSL is shown in Table I. We evaluated the metrics of PPSL by generating 200 solutions at each time step. In addition, we also test two variants of PPSL: PPSL (Large Set) which generates 2,000 solutions for each time, and PPSL (Black Box) which needs to estimate the gradient by the ES method introduced.

Each experiment was carried out in ten independent runs. The MIGD and MHV metrics are reported in Table II and Table I in the Supplementary, respectively. In the results, the best values among the four algorithms are highlighted in bold and gray background, while "+", "=" and "-" indicate that the proposed algorithm is significantly better, similar or worse, respectively, based on the Wilcoxon rank sum test at a significance level of 0.05.

C. Experimental Results and Analysis

Experimental analysis is discussed from the following perspectives.

1) Overall Analysis: The PPSL method demonstrates superior performance across most benchmark problems compared to baseline DMOEAs. As shown in Table II and Table I in the Supplementary, PPSL achieves the best MIGD and MHV values in the majority of test cases, highlighting its effectiveness in balancing convergence and diversity. The consistent performance across varying problem complexities and dynamic scenarios validates PPSL as a promising approach for DMOPs. Furthermore, to position PPSL against the very latest machine learning-based DMOEAs, a comprehensive comparison with five state-of-the-art algorithms from recent literature [50], [51], [52], [53], [54] has been conducted. These

[1]https://pymoo.org/index.html
[2]https://github.com/jicheng9617/PPSL

TABLE II
THE AVERAGE AND STANDARD DEVIATION MIGD VALUES OF TEN INDEPENDENT RUNS ON DF PROBLEMS WITH FOUR DIFFERENT DYNAMIC ENVIRONMENT. THE +/=/- SYMBOLS DENOTE PPSL IS SIGNIFICANTLY BETTER THAN/EQUAL TO/WORSE THAN THE COMPARED METHOD. THE BEST RESULTS IN EACH TEST ARE HIGHLIGHTED IN BOLD AND GRAY BACKGROUND.

Problems	(n_t, τ_t)	DNSGA-II-A [48]	DNSGA-II-B [48]	KGB [49]	PPSL	PPSL (Large Set)	PPSL (Black Box)
DF1	$(5,5)$	1.38e-01 (4.89e-03) +	5.51e-01 (8.13e-03) +	8.09e-02 (1.54e-02) +	**4.29e-03 (6.62e-05)**	7.67e-04 (2.89e-05)	1.11e-01 (9.31e-03)
	$(5,10)$	6.68e-02 (3.71e-03) +	3.24e-01 (8.43e-03) +	1.79e-02 (1.20e-03) +	**4.46e-03 (1.62e-04)**	1.04e-03 (1.52e-04)	9.65e-02 (1.08e-01)
	$(10,5)$	1.02e-01 (3.57e-03) +	3.62e-01 (6.60e-03) +	6.95e-02 (3.44e-03) +	**4.24e-03 (8.11e-05)**	7.80e-04 (6.58e-05)	8.85e-02 (1.24e-02)
	$(10,10)$	4.34e-02 (2.12e-03) +	8.42e-02 (3.73e-03) +	2.28e-02 (1.70e-03) +	**4.30e-03 (1.37e-04)**	9.03e-04 (1.68e-04)	5.25e-02 (9.30e-03)
DF2	$(5,5)$	1.00e-01 (4.22e-03) +	3.81e-01 (1.27e-02) +	6.37e-02 (1.04e-02) +	**6.39e-03 (2.55e-04)**	3.49e-03 (2.47e-04)	8.55e-02 (6.88e-03)
	$(5,10)$	4.50e-02 (1.91e-03) +	2.47e-01 (8.46e-03) +	1.72e-02 (1.35e-03) +	**4.89e-03 (8.52e-05)**	1.66e-03 (1.05e-04)	3.70e-02 (2.98e-03)
	$(10,5)$	7.06e-02 (2.33e-03) +	2.88e-01 (1.03e-02) +	6.99e-02 (1.01e-02) +	**5.97e-03 (1.75e-04)**	2.96e-03 (2.00e-04)	7.20e-02 (8.88e-03)
	$(10,10)$	2.90e-02 (1.97e-03) +	1.22e-01 (4.25e-03) +	2.01e-02 (1.56e-03) +	**4.91e-03 (9.68e-05)**	1.68e-03 (1.49e-04)	3.18e-02 (3.33e-03)
DF3	$(5,5)$	5.28e-01 (2.90e-02) +	6.56e-01 (6.06e-02) +	3.91e-01 (1.35e-02) +	**6.73e-03 (4.76e-04)**	3.98e-03 (5.42e-04)	9.07e-02 (5.68e-03)
	$(5,10)$	3.25e-01 (1.60e-02) +	3.17e-01 (7.11e-03) +	2.76e-01 (2.02e-02) +	**7.71e-03 (9.13e-04)**	5.13e-03 (1.07e-03)	4.91e-02 (3.24e-03)
	$(10,5)$	3.94e-01 (1.21e-02) +	3.61e-01 (1.09e-02) +	3.67e-01 (2.61e-02) +	**7.45e-03 (8.41e-04)**	4.85e-03 (9.43e-04)	7.90e-02 (7.25e-03)
	$(10,10)$	2.87e-01 (3.11e-02) +	2.66e-01 (2.79e-02) +	2.66e-01 (2.56e-02) +	**8.00e-03 (1.95e-03)**	5.39e-03 (2.09e-03)	4.18e-02 (2.41e-03)
DF4	$(5,5)$	1.88e-01 (2.27e-02) +	1.67e-01 (2.40e-02) +	3.37e-01 (7.25e-02) +	**8.41e-02 (2.11e-03)**	7.59e-02 (1.81e-03)	1.63e-01 (7.08e-03)
	$(5,10)$	1.01e-01 (9.81e-03) +	9.52e-02 (7.24e-03) +	1.12e-01 (9.23e-03) +	**7.88e-02 (3.87e-03)**	6.97e-02 (3.62e-03)	1.37e-01 (6.21e-03)
	$(10,5)$	1.88e-01 (1.59e-02) +	1.97e-01 (2.05e-02) +	3.29e-01 (8.07e-02) +	**8.06e-02 (1.49e-03)**	7.18e-02 (1.45e-03)	1.58e-01 (1.01e-02)
	$(10,10)$	9.49e-02 (7.60e-03) +	9.84e-02 (4.48e-03) +	1.05e-01 (6.31e-03) +	**7.50e-02 (1.89e-03)**	6.57e-02 (2.23e-03)	1.33e-01 (5.92e-03)
DF5	$(5,5)$	3.09e-01 (1.99e-02) +	1.58e+00 (5.83e-02) +	2.79e-01 (3.94e-02) +	**5.81e-03 (9.88e-05)**	2.44e-03 (1.18e-04)	4.05e-02 (4.31e-03)
	$(5,10)$	1.24e-01 (6.97e-03) +	3.37e-01 (1.69e-02) +	8.97e-02 (3.13e-02) +	**5.98e-03 (1.67e-04)**	2.62e-03 (1.73e-04)	2.45e-02 (2.31e-03)
	$(10,5)$	1.91e-01 (6.96e-03) +	4.48e-01 (2.81e-02) +	2.75e-01 (8.21e-02) +	**5.30e-03 (9.93e-05)**	1.85e-03 (1.10e-04)	3.39e-02 (2.83e-03)
	$(10,10)$	5.12e-02 (1.46e-03) +	4.62e-02 (1.84e-03) +	9.23e-02 (3.20e-02) +	**5.38e-03 (1.13e-04)**	1.90e-03 (1.34e-04)	1.91e-02 (1.78e-03)
DF6	$(5,5)$	7.50e+00 (3.95e-01) +	1.42e+01 (4.36e-01) +	4.31e+00 (4.72e-01) +	**7.99e-02 (6.29e-03)**	7.37e-02 (6.39e-03)	1.64e+00 (6.95e-01)
	$(5,10)$	2.85e+00 (1.43e-01) +	7.54e+00 (3.57e-01) +	1.65e+00 (1.06e-01) +	**4.88e-02 (1.47e-02)**	3.95e-02 (1.49e-02)	1.65e+00 (3.81e-01)
	$(10,5)$	5.93e+00 (3.28e-01) +	1.03e+01 (4.78e-01) +	4.36e+00 (3.79e-01) +	**7.76e-02 (1.92e-02)**	7.07e-02 (2.01e-02)	1.80e+00 (4.81e-01)
	$(10,10)$	2.09e+00 (7.71e-02) +	3.20e+00 (2.99e-01) +	1.76e+00 (8.15e-02) +	**4.40e-02 (7.39e-03)**	3.49e-02 (7.52e-03)	2.21e+00 (5.85e-01)
DF7	$(5,5)$	6.01e-01 (5.87e-02) +	7.55e-01 (2.66e-02) +	3.72e-01 (2.67e-02) +	**9.68e-02 (1.77e-02)**	4.33e-02 (1.69e-02)	4.12e-01 (4.77e-02)
	$(5,10)$	3.60e-01 (4.91e-02) +	6.98e-01 (3.16e-02) +	2.81e-01 (5.48e-02) +	**8.79e-02 (7.34e-03)**	2.76e-02 (6.07e-03)	3.44e-01 (9.31e-02)
	$(10,5)$	2.24e-01 (8.46e-03) +	3.01e-01 (7.43e-02) +	3.01e-01 (3.50e-02) +	**5.13e-02 (1.01e-02)**	3.54e-02 (1.01e-02)	1.19e-01 (1.28e-02)
	$(10,10)$	2.14e-01 (1.48e-02) +	1.79e-01 (1.08e-02) +	1.76e-01 (2.07e-02) +	**3.27e-02 (2.96e-03)**	1.59e-02 (3.08e-03)	9.28e-02 (2.10e-02)
DF8	$(5,5)$	8.55e-02 (1.63e-03) +	8.32e-02 (1.44e-03) +	1.42e-01 (2.24e-02) +	**5.62e-02 (5.20e-03)**	5.41e-02 (5.41e-03)	6.96e-02 (3.25e-03)
	$(5,10)$	7.54e-02 (1.58e-03) +	7.36e-02 (1.26e-03) +	7.92e-02 (1.90e-03) +	**3.61e-02 (3.99e-03)**	3.20e-02 (3.98e-03)	6.26e-02 (2.00e-03)
	$(10,5)$	7.61e-02 (3.07e-03) +	7.34e-02 (2.24e-03) +	1.50e-01 (1.71e-02) +	**4.70e-02 (2.92e-03)**	4.44e-02 (3.10e-03)	6.22e-02 (3.10e-03)
	$(10,10)$	6.27e-02 (1.71e-03) +	5.84e-02 (1.48e-03) +	8.05e-02 (5.89e-03) +	**3.40e-02 (5.00e-03)**	2.84e-02 (6.64e-03)	5.75e-02 (2.71e-03)
DF9	$(5,5)$	6.27e-01 (3.73e-02) +	1.57e+00 (6.56e-02) +	4.77e-01 (1.14e-01) +	**1.66e-01 (9.79e-02)**	1.64e-01 (9.82e-02)	4.48e-01 (1.63e-01)
	$(5,10)$	3.31e-01 (9.58e-03) +	1.24e+00 (4.81e-02) +	2.24e-01 (3.25e-02) +	**5.28e-02 (9.00e-03)**	4.96e-02 (9.11e-03)	2.63e-01 (8.70e-02)
	$(10,5)$	4.47e-01 (2.28e-02) +	1.47e+00 (1.34e-01) +	4.24e-01 (6.08e-02) +	**6.25e-02 (8.86e-03)**	6.02e-02 (9.01e-03)	2.92e-01 (4.70e-02)
	$(10,10)$	2.38e-01 (1.38e-02) +	9.46e-01 (4.16e-02) +	1.68e-01 (1.69e-02) +	**3.09e-02 (2.10e-03)**	2.75e-02 (2.25e-03)	2.28e-01 (5.93e-02)
DF10	$(5,5)$	1.04e-01 (6.79e-03) +	8.86e-02 (5.90e-03) =	2.11e-01 (1.72e-02) +	**8.53e-02 (3.37e-03)**	6.89e-02 (3.73e-03)	1.94e-01 (8.41e-03)
	$(5,10)$	6.79e-02 (4.37e-03) -	**6.34e-02 (3.27e-03)** -	1.57e-01 (9.93e-03) +	7.78e-02 (3.50e-03)	5.82e-02 (3.21e-03)	1.58e-01 (7.53e-03)
	$(10,5)$	9.55e-02 (9.57e-03) +	**7.81e-02 (3.67e-03)** -	1.99e-01 (1.22e-02) +	8.50e-02 (2.30e-03)	6.88e-02 (3.06e-03)	1.87e-01 (8.76e-03)
	$(10,10)$	5.82e-02 (1.90e-03) -	**5.54e-02 (1.22e-03)** -	1.53e-01 (1.73e-02) +	9.35e-02 (2.82e-02)	6.64e-02 (1.70e-02)	1.58e-01 (6.04e-03)
DF11	$(5,5)$	3.59e-01 (2.21e-03) +	3.55e-01 (2.70e-03) +	4.15e-01 (1.62e-02) +	**3.25e-01 (5.57e-04)**	3.12e-01 (2.73e-04)	3.40e-01 (4.95e-03)
	$(5,10)$	3.44e-01 (1.53e-03) +	3.43e-01 (1.05e-03) +	3.67e-01 (3.61e-03) +	**3.23e-01 (1.97e-04)**	3.11e-01 (1.14e-04)	3.22e-01 (1.40e-03)
	$(10,5)$	3.55e-01 (1.47e-03) +	3.53e-01 (4.16e-03) +	4.01e-01 (1.24e-02) +	**3.20e-01 (4.35e-04)**	3.09e-01 (3.09e-04)	3.36e-01 (3.09e-03)
	$(10,10)$	3.40e-01 (1.82e-03) +	3.38e-01 (1.04e-03) +	3.61e-01 (2.74e-03) +	**3.19e-01 (4.03e-04)**	3.07e-01 (1.43e-04)	3.18e-01 (1.15e-03)
DF12	$(5,5)$	2.76e-01 (1.69e-02) +	2.68e-01 (9.49e-03) +	3.46e-01 (1.49e-02) +	**1.46e-01 (5.07e-03)**	1.17e-01 (5.21e-03)	3.79e-01 (3.85e-02)
	$(5,10)$	2.04e-01 (1.53e-02) +	1.86e-01 (1.04e-02) +	2.71e-01 (1.65e-02) +	**1.47e-01 (1.18e-02)**	1.15e-01 (1.21e-02)	3.36e-01 (1.95e-02)
	$(10,5)$	2.46e-01 (4.44e-02) +	1.94e-01 (2.02e-02) +	3.33e-01 (1.05e-02) +	**9.57e-02 (6.59e-03)**	6.20e-02 (7.22e-03)	1.66e-01 (8.25e-03)
	$(10,10)$	1.26e-01 (2.88e-03) +	1.18e-01 (3.35e-03) +	2.46e-01 (8.43e-03) +	**1.01e-01 (1.11e-02)**	6.62e-02 (1.12e-02)	1.47e-01 (7.44e-03)
DF13	$(5,5)$	6.04e-01 (2.22e-02) +	1.76e+00 (7.57e-02) +	4.25e-01 (3.58e-02) +	**1.66e-01 (3.41e-03)**	9.51e-02 (3.52e-03)	2.01e-01 (2.46e-02)
	$(5,10)$	3.08e-01 (8.54e-03) +	9.54e-01 (2.73e-02) +	2.33e-01 (1.38e-02) +	**1.64e-01 (4.84e-03)**	9.15e-02 (3.73e-03)	1.57e-01 (1.05e-02)
	$(10,5)$	4.18e-01 (2.11e-02) +	1.14e+00 (5.93e-02) +	4.44e-01 (5.28e-02) +	**1.69e-01 (1.51e-02)**	9.82e-02 (1.75e-02)	2.23e-01 (3.32e-02)
	$(10,10)$	1.89e-01 (5.89e-03) +	2.38e-01 (7.75e-03) +	2.28e-01 (2.58e-02) +	**1.65e-01 (6.29e-03)**	8.90e-02 (3.98e-03)	1.45e-01 (1.15e-02)
DF14	$(5,5)$	4.07e-01 (3.26e-02) +	1.01e+00 (9.52e-02) +	4.37e-01 (7.60e-02) +	**1.78e-01 (1.69e-02)**	1.64e-01 (1.77e-02)	1.89e-01 (1.45e-02)
	$(5,10)$	3.03e-01 (2.65e-02) +	5.06e-01 (7.27e-02) +	3.63e-01 (1.27e-01) +	**9.98e-02 (2.34e-02)**	8.06e-02 (2.47e-02)	1.84e-01 (1.80e-02)
	$(10,5)$	2.66e-01 (5.03e-02) +	5.60e-01 (8.57e-02) +	3.92e-01 (1.18e-01) +	**1.91e-01 (2.85e-02)**	1.78e-01 (3.02e-02)	1.95e-01 (2.29e-02)
	$(10,10)$	1.77e-01 (2.16e-02) +	3.27e-01 (8.10e-02) +	2.62e-01 (7.73e-02) +	**6.80e-02 (3.26e-02)**	4.55e-02 (3.47e-02)	1.53e-01 (3.81e-02)

results, which demonstrate the superior performance of PPSL, are detailed in Table II of the Supplementary.

2) Influence of Change Severity and Frequency: The performance of PPSL remains robust under varying degrees of change severity and frequency, as evidenced by its consistent superiority across different (n_t, τ_t) settings. For instance, in DF1 and DF2, PPSL achieves significantly lower MIGD values compared to baseline algorithms, even with high change severity ($n_t = 5$) and frequent changes ($\tau_t = 5$). This robustness stems from the hypernetwork's ability to adaptively generate PS models for each parameter setting, ensuring smooth transitions between dynamic states. However, in problems with extremely high severity (e.g., DF10), the performance slightly degrades, indicating a potential limitation in handling abrupt and large-scale changes. Nonetheless, its ability to maintain competitive performance across diverse dynamic scenarios highlights its adaptability and reliability.

3) Influence of Large Set Prediction: The PPSL (Large Set) variant, which generates 2,000 solutions per time step, consistently achieves better MIGD and MHV values compared to 200 solutions. This improvement is particularly evident in problems with complex PFs, such as DF10 and DF12. Note that the computational cost of predicting the PS through model inference remains minimal, as it can be executed at a millisecond level, ensuring that the increased solution set does not significantly impact real-time applicability. This suggests that PPSL (Large Set) is a practical and effective approach for real-time application, offering improved solution quality without compromising computational efficiency.

4) Influence of Black-Box Scenario: In the black-box scenario, we empirically use two samples to estimate the gradient ($K = 2$ in Eq. (21)). In this way, the number of samples for parameter B_t in Table I changes to 10 for a fair comparison. From the results, PPSL (Black Box) exhibits a noticeable

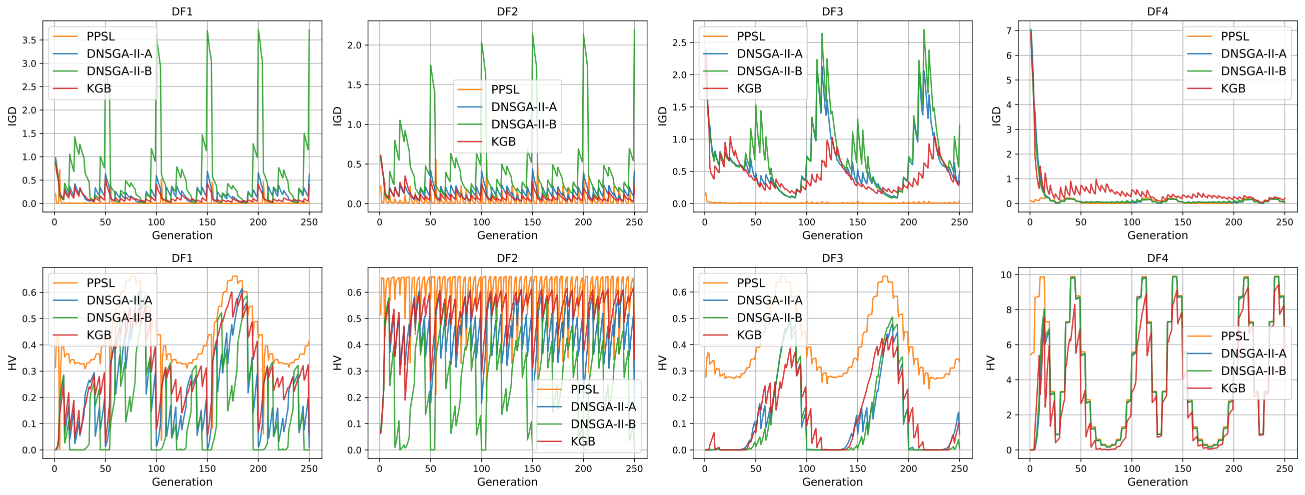


Fig. 3. Evolution curves of IGD and HV values (average over 10 runs) by PPSL on DF1 to DF4 with $n_t = 10$, $\tau_t = 5$.

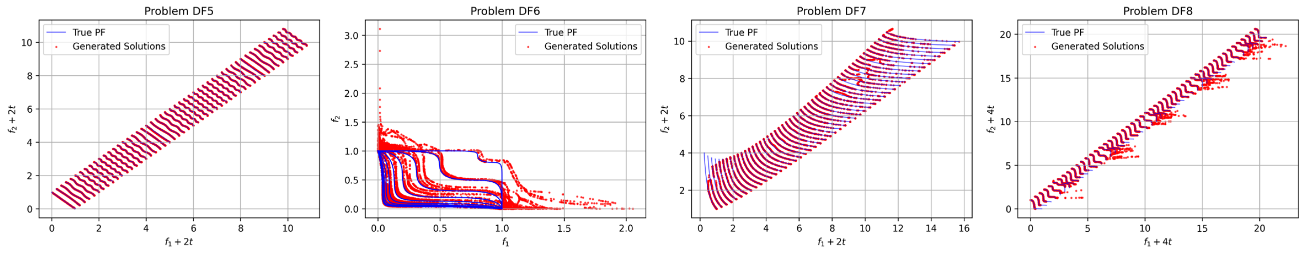


Fig. 4. The evolution of approximate Pareto fronts generated by PPSL, where 200 solutions are shown at each time step t. Each approximate Pareto front was generated and saved online at its corresponding time step during the optimization process, reflecting the model's real-time adaptive performance.

performance drop compared to the standard PPSL, primarily due to the inherent noise and approximation errors in gradient estimation, which hinder the optimization process. Nevertheless, PPSL (Black Box) outperforms DMOEAs in many cases, demonstrating its potential for applications where gradient information is unavailable. This highlights the versatility of PPSL: with explicit formulations, it achieves outstanding performance, while in black-box scenarios, it still delivers competitive results.

5) Analysis of Evolutionary Curves and Pareto Front Along the Evolutionary Process: Fig. 3 demonstrates the evolutionary curves of IGD and HV values by all compared methods on the first four problems. For the shown instances, PPSL achieves significantly lower IGD values compared to DNSGA-II variants and KGB, maintaining a steady decline over generations. This indicates efficient adaptation to environmental changes and robust optimization capabilities. The sharp convergence of the curves suggests rapid adaptation to dynamic shifts, while the gradual improvement of DMOEAs highlights their reliance on population re-initialization, which disrupts solution continuity.

Fig. 4 displays the generated solutions at each time. The scatter plots for DF5, DF7, and DF8 reveal dense clusters of solutions near the true PF, confirming the effectiveness of PPSL in approximating complex PFs. Minor deviations in DF6 at higher objective values suggest occasional challenges in maintaining uniformity under severe dynamic changes, yet the overall distribution remains competitive.

V. Application Study: Multiobjective Optimization with Shared Components

In this section, we adapt PPSL to another design problem: multiobjective optimization with structural constraints.

A. Problem Introduction and Formulation

In many real-world multiobjective applications, structural constraints are imposed on design variables, often requiring that a subset of components be shared across multiple solutions [56], [57]. This is a central challenge in fields like modular product design, where sharing components is critical for reducing manufacturing costs and complexity [58], [59], and in personalized manufacturing, where customized products must leverage common platforms to remain economically feasible.

Formally, let the design vector $\boldsymbol{x}$ be partitioned into a subvector of shared components, $\boldsymbol{x_s}$ (where $\boldsymbol{s} \subset \{1,\ldots,n\}$), and a subvector of unique, optimizable components, $\boldsymbol{x_p}$ (where $\boldsymbol{p} = \{1,\ldots,n\} \setminus \boldsymbol{s}$). For a fixed shared component value $\boldsymbol{x_s} = \boldsymbol{\beta}$, the corresponding MOP is:

$$\min_{\boldsymbol{x_p}} \; \boldsymbol{F}(\boldsymbol{x_p} | \boldsymbol{x_s} = \boldsymbol{\beta}). \tag{25}$$

Two distinct paradigms have emerged for addressing this class of problems. The conventional approach treats the shared component $\boldsymbol{\beta}$ as a decision variable to be optimized jointly

TABLE III
HV VALUES OF PPSL AND THE OTHER BASELINE APPROACHES ON MOPS WITH SHARED COMPONENTS.

Problem	Shared Comp.	NSGA-II [16]	MOEA/D [17]	NSGA-III [55]	PSL [25]	PPSL
RE21	(x_1)	7.24e-01 (2.48e-05)	5.32e-01 (2.92e-03)	7.25e-01 (5.53e-06)	**7.26e-01 (6.25e-05)**	**7.26e-01 (1.19e-04)**
	(x_2)	7.82e-01 (5.77e-05)	4.84e-01 (6.57e-03)	7.83e-01 (1.68e-05)	**7.85e-01 (7.42e-05)**	**7.85e-01 (5.25e-05)**
	(x_3)	6.21e-01 (8.44e-05)	2.06e-01 (1.07e-02)	6.24e-01 (2.81e-05)	**6.28e-01 (2.61e-06)**	**6.28e-01 (1.12e-05)**
	(x_4)	7.86e-01 (1.32e-05)	5.26e-01 (1.69e-02)	7.87e-01 (5.67e-06)	**7.88e-01 (5.87e-06)**	**7.88e-01 (4.32e-05)**
	(x_1, x_2)	**5.84e-01 (3.23e-06)**	5.26e-01 (3.84e-06)	**5.84e-01 (2.20e-09)**	**5.84e-01 (3.02e-06)**	**5.84e-01 (9.70e-06)**
	(x_2, x_3)	6.27e-01 (9.69e-06)	2.91e-01 (1.97e-03)	6.28e-01 (5.07e-06)	**6.30e-01 (1.69e-05)**	**6.30e-01 (4.92e-05)**
	(x_3, x_4)	5.76e-01 (2.71e-05)	2.89e-01 (1.57e-02)	5.77e-01 (2.10e-06)	**5.79e-01 (1.06e-05)**	5.78e-01 (6.41e-05)
	(x_1, x_2, x_3)	**4.13e-01 (5.32e-06)**	3.55e-01 (1.85e-06)	**4.13e-01 (1.39e-09)**	**4.13e-01 (2.67e-06)**	**4.13e-01 (1.30e-05)**
	(x_2, x_3, x_4)	**5.06e-01 (2.42e-05)**	3.90e-01 (7.94e-05)	**5.06e-01 (2.16e-09)**	**5.06e-01 (1.15e-05)**	**5.06e-01 (1.03e-05)**
RE33	(x_1)	8.37e-01 (2.15e-03)	7.87e-01 (8.19e-04)	8.65e-01 (2.27e-03)	8.77e-01 (2.37e-02)	**8.95e-01 (7.96e-04)**
	(x_2)	7.83e-01 (1.29e-03)	7.46e-01 (1.08e-04)	8.03e-01 (1.66e-04)	**8.15e-01 (6.71e-05)**	8.13e-01 (4.66e-04)
	(x_3)	5.06e-01 (2.29e-03)	4.66e-01 (2.52e-03)	5.26e-01 (1.70e-04)	2.41e-01 (1.51e-02)	**5.59e-01 (1.40e-03)**
	(x_4)	8.02e-01 (4.78e-03)	7.31e-01 (1.31e-03)	**8.49e-01 (3.11e-04)**	4.18e-01 (9.31e-02)	8.44e-01 (3.11e-03)
	(x_1, x_2)	5.37e-01 (4.08e-05)	5.36e-01 (1.65e-05)	5.27e-01 (3.68e-05)	**5.40e-01 (1.18e-05)**	5.34e-01 (7.31e-06)
	(x_2, x_3)	5.59e-01 (5.96e-04)	5.10e-01 (3.45e-04)	5.61e-01 (4.72e-05)	5.30e-01 (8.37e-03)	**5.63e-01 (2.56e-03)**
	(x_3, x_4)	4.59e-01 (6.18e-04)	3.92e-01 (6.73e-05)	4.57e-01 (1.63e-05)	1.78e-01 (1.49e-02)	**4.60e-01 (7.73e-04)**
	(x_1, x_2, x_3)	**1.49e-01 (1.38e-05)**	1.48e-01 (5.65e-07)	1.45e-01 (1.33e-08)	1.48e-01 (8.06e-06)	**1.49e-01 (9.36e-05)**
	(x_2, x_3, x_4)	**2.46e-01 (7.85e-06)**	1.93e-01 (4.26e-05)	2.42e-01 (8.96e-07)	2.24e-01 (4.16e-03)	2.45e-01 (3.94e-04)
RE37	(x_1)	5.68e-01 (7.29e-04)	5.60e-01 (4.31e-04)	5.65e-01 (1.85e-05)	**5.76e-01 (1.54e-03)**	5.69e-01 (1.57e-04)
	(x_2)	5.27e-01 (3.05e-04)	5.28e-01 (1.14e-04)	5.24e-01 (2.22e-05)	5.40e-01 (7.84e-04)	**5.42e-01 (1.35e-04)**
	(x_3)	6.29e-01 (6.25e-04)	6.28e-01 (4.26e-05)	6.15e-01 (9.66e-04)	6.39e-01 (4.63e-04)	**6.40e-01 (7.97e-05)**
	(x_4)	4.84e-01 (3.68e-04)	4.95e-01 (1.59e-04)	4.83e-01 (1.14e-04)	5.12e-01 (7.98e-05)	**5.14e-01 (3.96e-04)**
	(x_1, x_2)	3.39e-01 (2.22e-04)	3.26e-01 (8.82e-05)	3.36e-01 (8.05e-05)	3.39e-01 (6.60e-04)	**3.42e-01 (1.84e-04)**
	(x_2, x_3)	4.18e-01 (1.07e-04)	4.16e-01 (5.17e-05)	4.13e-01 (1.02e-04)	4.20e-01 (2.10e-04)	**4.21e-01 (2.85e-05)**
	(x_3, x_4)	**3.95e-01 (1.36e-04)**	3.90e-01 (6.06e-06)	3.83e-01 (6.25e-04)	**3.95e-01 (1.83e-03)**	3.85e-01 (1.96e-02)
	(x_1, x_2, x_3)	**2.78e-01 (2.89e-06)**	2.68e-01 (4.90e-06)	2.77e-01 (7.80e-07)	2.66e-01 (1.66e-03)	2.67e-01 (7.68e-03)
	(x_2, x_3, x_4)	**2.56e-01 (6.44e-06)**	**2.56e-01 (2.95e-07)**	2.52e-01 (1.07e-05)	**2.56e-01 (1.03e-03)**	**2.56e-01 (1.52e-05)**

TABLE IV
RUN TIME COMPARISON (IN SECONDS) OF PPSL AND BASELINE METHODS. WE CHOOSE TEN UNIFORMLY DISTRIBUTED PARAMETER VALUES TO EVALUATE THE PERFORMANCE. THE 'PPSL TRAINING' COLUMN INCLUDES THE TIME REQUIRED TO TRAIN THE HYPERNETWORK AND PS MODEL, WHILE THE 'PPSL INFERENCE' COLUMN REPRESENTS THE TIME REQUIRED TO OBTAIN SOLUTIONS FOR THE TEN PARAMETER VALUES AFTER TRAINING. PPSL AND OTHER MOEAS ARE ALL RUN ON THE SAME CPU.

Problem	NSGA-II [16]	MOEA/D [17]	NSGA-III [55]	PSL [25]	PPSL Training	PPSL Inference
RE21	12.6s	297.5s	17.2s	66.2s	34.1s	0.028s
RE33	23.8s	703.3s	32.4s	102.5s	67.3s	0.014s
RE37	26.1s	915.1s	35.0s	134.6s	83.7s	0.011s

with $\boldsymbol{x_p}$, aiming to find the single 'best' shared component that yields PF with the highest quality, often measured by hypervolumn (HV) [25], [60]. This can be formulated as:

$$\min_{\boldsymbol{\beta}, \boldsymbol{x_p}} \boldsymbol{F}(\boldsymbol{x_p} | \boldsymbol{x_s} = \boldsymbol{\beta}). \tag{26}$$

While valuable, this formulation assumes that a single, globally optimal shared component is always desirable and practically achievable.

Our work, however, proposes a paradigm shift: from finding a single optimal solution to learning a complete map of optimal solutions across a spectrum of problem contexts. This adaptation is motivated by two key principles:

1) *Amortized Optimization for Real-Time Decision Support:* From a machine learning perspective, our framework is a form of amortized optimization [61], [13], [62], [63]. We invest a significant one-time computational cost during training to learn a direct mapping from the problem's parameters (the shared component $\boldsymbol{x_s}$) to its optimal solution (the Pareto set). Once trained, our model can generate the entire Pareto set for any new, unseen component value $\boldsymbol{\beta}$ with a single, near-instantaneous forward pass. This amortizes the expensive optimization process and makes the framework exceptionally efficient for real-time decision support and rapid exploration of the design space.
2) *Decoupling for Strategic Management:* From an engineering management perspective, the choice of a shared component is a high-level, strategic decision often dictated by external constraints like cost, legacy systems, or supply chain logistics. As established in [57], component sharing is a critical strategy for navigating the trade-off between product variety and operational cost. Our framework directly addresses this managerial challenge by decoupling the shared component. Instead of prescribing a single optimal component, it generates a map of achievable performance trade-offs for any given choice of $\boldsymbol{\beta}$. The model enables practitioners to quantitatively assess how strategic choices about shared components, driven by factors like cost or existing inventory, directly impact the system's achievable performance trade-offs.

To realize this vision, PPSL models the Pareto set as a function of both the preference vector $\boldsymbol{\lambda}$ and the shared component $\boldsymbol{x_s}$. This is achieved using a hypernetwork that generates the parameters $\boldsymbol{\theta}_{\text{ps}}$ of PS model $h_{\boldsymbol{\theta}_{\text{ps}}}(\cdot)$ based on

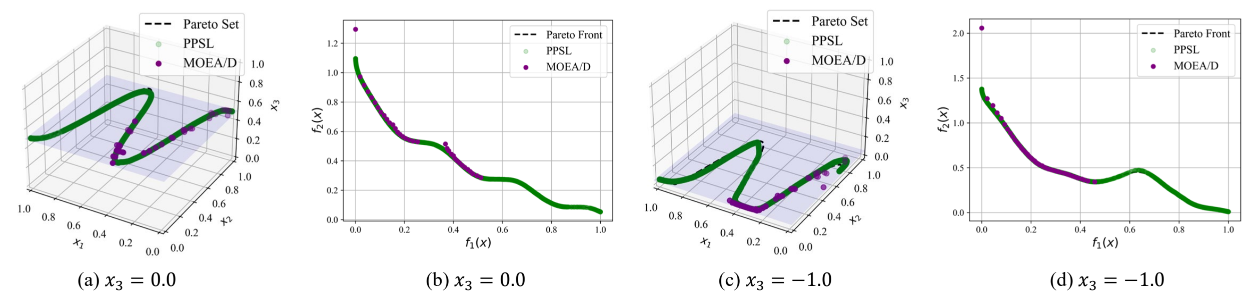


Fig. 5. **Learned Pareto sets and fronts under different parameter values in the demonstrative example. (a)** Learned PS and finite solutions by MOEA/D for $x_3 = 0$. **(b)** The corresponding PFs at the objective space for $x_3 = 0$. **(c)** Learned PS and finite solutions by MOEA/D for $x_3 = -1$. **(d)** The corresponding PFs for $x_3 = -1$. Decision makers can directly obtain the PS model for any given parameter value without optimization step, and further choose preferred solutions by exploring the PS model in real time.

$\boldsymbol{x_s}$:

$$\boldsymbol{x_p} = h_{\boldsymbol{\theta}_{\text{ps}}(\boldsymbol{x_s})}(\boldsymbol{\lambda}). \tag{27}$$

This formulation allows us to learn the continuous relationship between the shared component and the entire landscape of Pareto optimal solutions, transforming the optimization tool into a comprehensive decision-support system.

B. Experimental Setups

1) Optimization Problems: The real-world multiobjective test cases (RE problems [64]) are employed to evaluate the performance of the method. The problems in the test suite consist of PFs with different properties and shapes. The reader can refer to [64] for implementation[3] and a detailed summary.

2) Performance Indicator: HV [65] is adopted to measure the performance of the solutions found. The HV values are calculated by first normalizing all obtained solutions into the unit space $[0, 1]^m$ using the approximated ideal point z^{ideal} and nadir point z^{nadir} provided in [64]. The HV is then computed with a fixed reference vector $(1.1, \dots, 1.1)^\top$. Since no existing method is specially designed for the PMO problem, we use conventional MOEAs with the intuitive skill of re-execution for each parameter value. We randomly sample ten parameter values for each PMO problem and run each algorithm three times independently. Finally, the average HV values over ten parameters and three runs are reported for comparison.

3) Algorithm Setting: To verify the efficiency and efficacy of our proposed PPSL method, we compare it with three well-known MOEAs, MOEA/D [17], NSGA-II [16], and NSGA-III [55], and a learning-based method, PSL [25], [33]. The experiments are implemented through the pymoo platform, the PSL code[4], and our proposed PPSL method.

PPSL is trained for 1,000 epochs with each epoch randomly sampling 10 preference vectors, 5 parameters for two-objective problems, and 8 for problems with three objectives, resulting in 50,000 function evaluations for two-objective problems and 80,000 for problems with three or more objectives. The settings of the compared methods are kept consistent as that in [25], i.e., 26,000 and 41,000 function evaluations for problems with two and three, respectively. We train the PPSL model with the assigned budget and then evaluate the performance through inference on the ten selected parameter values.

[3]https://github.com/ryojitanabe/reproblems
[4]https://github.com/Xi-L/STCH

C. Illustrative Example

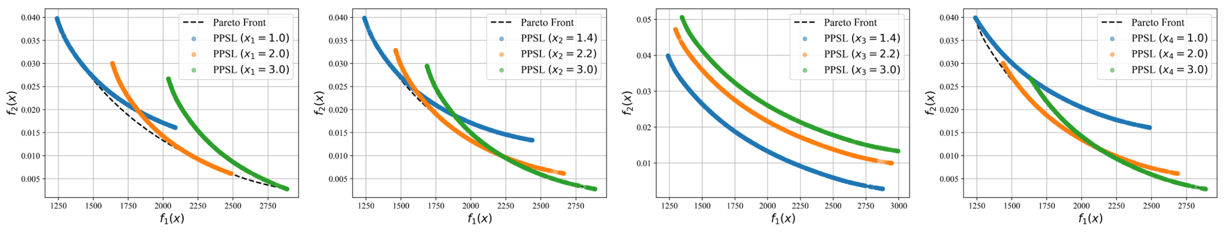
We first use a synthetic two-objective optimization problem to illustrate the function of the method:

$$\begin{aligned}
&\min_{x_1\in[0,1], x_{[2:n]}\in[-1,1]^{n-1}} \left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\right), \text{where} \\
&f_1(\boldsymbol{x}) = x_1, \\
&f_2(\boldsymbol{x}) = (1+g)\left(1 - \sqrt{x_1/(1+g)}\right), \\
&g = \frac{g_1 + g_2}{n-1}, \\
&g_1 = \left(x_2 - \sin(10(x_1 - 0.5)) - x_3\right)^2, \\
&g_2 = \sum_{i=3}^{n} \left(x_i - \sin(10(x_1 - 0.5))\right)^2.
\end{aligned} \tag{28}$$

Given the problem with $n = 10$, we set the requirement that all solutions share the same value in the component x_3. In this way, the problem can be viewed as the PMO problem with the parameter x_3, and the first three components of the solutions found by our proposed PPSL can be found in Fig. 5. We simply show the solutions for two specific parameter values; however, PPSL can directly obtain the PS model for any given parameter value. It is clear that the learned Pareto sets approximate the whole Pareto sets while MOEA/D can only obtain finite solutions; moreover, MOEA/D has to re-execute for every parameter value.

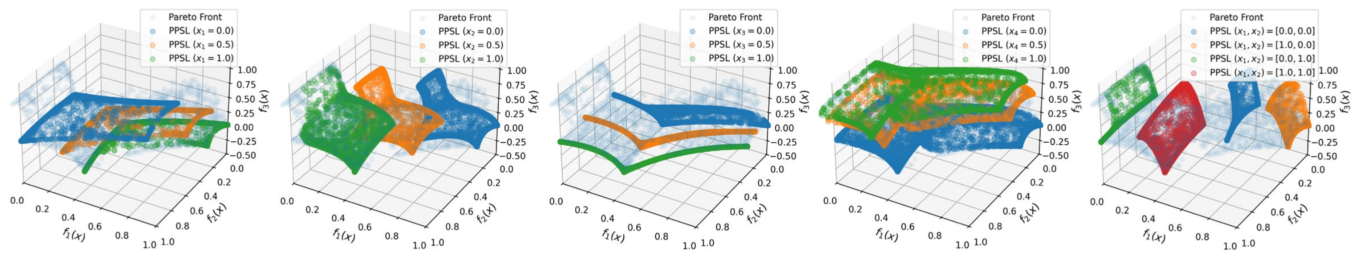
D. Comparison with Baseline methods

Table III presents the results of PPSL and the other baseline methods on the RE problems with different share combinations of the variable components. It is evident that PPSL consistently achieves competitive or superior performance across most test cases. For problem RE21, PPSL performs similarly to the learning-based PSL method. However, it should be noted that for every test parameter, PSL must re-execute the entire process to obtain the approximate PS, resulting in approximately 260,000 for the ten parameters, whereas PPSL costs only 50,000 evaluations and is able to obtain the PS for each parameter within inference time. This highlights the capability to handle multiobjective optimization with shared



(a) Shared x_1 (b) Shared x_2 (c) Shared x_3 (d) Shared x_4

Fig. 6. **Learned PFs for the Four Bar Truss Design Problem (RE21) with Shared Components.** We demonstrate three uniform distributed values for each shared component. PF (dashed line) denotes the optimal solutions of the original RE21 problem (without shared component).



(a) Shared x_1 (b) Shared x_2 (c) Shared x_3 (d) Shared x_4 (e) Shared x_1, x_2

Fig. 7. **Learned PFs for the Rocket Injector Design Problem (RE37) with Shared Components.** The transparent points denote the PFs for the problem without shared component. The shared components correspond to the decision variable x_1 (hydrogen flow angle), x_2 (hydrogen area), x_3 (oxygen area), and x_4 (oxidizer post tip thickness).

TABLE V
PERFORMANCE OF PPSL AND PPSL WITHOUT LORA STRUCTURE ON RE21 AND RE37 WITH ONE SHARED COMPONENT. THE VALUES IN BRACKETS REPRESENT THE TRAINING TIME.

Problem	Shared Comp.	PPSL	PPSL *w/o* LoRA
RE21	(x_1)	**7.254e-01** (29.2s)	7.218e-01 (152.9s)
	(x_2)	**7.847e-01** (26.9s)	7.725e-01 (152.8s)
	(x_3)	**6.271e-01** (28.1s)	5.974e-01 (152.3s)
	(x_4)	**7.883e-01** (28.1s)	7.774e-01 (152.2s)
RE37	(x_1)	**5.687e-01** (69.6s)	5.662e-01 (243.2s)
	(x_2)	**5.411e-01** (70.9s)	5.406e-01 (241.9s)
	(x_3)	**6.399e-01** (70.9s)	6.384e-01 (239.1s)
	(x_4)	**5.121e-01** (72.7s)	5.115e-01 (239.9s)

components effectively, offering a promising solution for real-world applications with structural constraints.

Table IV analyses the runtime of each comparative method. For the baseline methods, the runtime is the summation over ten independent runs. It is evident that the training time of PPSL is competitive with the other baseline methods, while its inference time for a single parameter value is significantly faster, averaging approximately 2 milliseconds. This demonstrates the efficiency of PPSL in both training and real-time inference, making it highly suitable for scenarios requiring a large number of decisions.

Fig. 6 and 7 illustrate the approximate PS for each component with three demonstrative values. The PS with different shared components may have quite different structures, as shown in Fig. 7 (e). Moreover, the value of the shared component also plays a critical role in its performance. For instance, the PS of the shared component with $x_3 = 1.4$ in problem RE21 has the same form as that of the unconstrained problem, and different values of the shared single component in problem RE37 have similar forms with a certain degree of offset.

E. Ablation Studies

In this subsection, we conduct experiments to verify the effectiveness of the LoRA structure with varied sizes of the PS model, and study the effect of different model structures as well as rank sizes (in Supplementary). We use the problems RE21 and RE37 with each variable treated as a parameter, resulting in four PMO problems.

Effects of LoRA structure. The LoRA structure significantly enhances both performance and efficiency in PPSL, as demonstrated in Table V. For all shared components in RE21 and RE37, PPSL with LoRA achieves higher HV values compared to PPSL without LoRA. This improvement is attributed to LoRA's ability to adaptively refine the model parameters, leading to a better approximation of the PS. Additionally, LoRA drastically reduces training time; for example, training with LoRA takes averagely 28 seconds for RE21, compared to 152.5 seconds without LoRA. This efficiency gain is due to the reduced complexity of gradient back-propagation in LoRA, enabling faster convergence. Overall, LoRA not only improves performance but also makes PPSL more computationally efficient, particularly beneficial for large-scale optimization problems.

VI. CONCLUSIONS

This paper has proposed Parametric Pareto set learning, a novel framework that leverages amortized optimization to address PMO problems. The proposed method generalizes PSL through two-aspect consideration: at the higher level, the computational budget is amortized across multiple sampled parameters, enabling direct prediction of the PS with respect to the parameter without iterative optimization. At the lower level, PPSL integrates scalarization methods to generalize the entire PS from a finite set of preference evaluations, aligning with the core idea of decomposition-based optimization. The method allows decision-makers to explore the entire PS for any parameter value, providing a comprehensive understanding of trade-offs across the parameter space. Experimental results demonstrate that PPSL can well resolve PMO problems, particularly in dynamic multiobjective optimization and MOPs with shared component constraints.

In terms of computational overhead, PPSL only involves a single run for a PMO problem, while it allows practitioners to identify any trade-off solution on any parameter value. We believe that PPSL has boarder application in any problem that can be modeled as a PMO problem. In addition, there are many improvements that can be achieved in the future:

- *Broader Applications:* Extending PPSL to tackle more specific optimization problems, particularly those with complex parameter distributions.
- *Model Enhancements:* Developing more powerful models to approximate complex Pareto sets, including those with discrete decision variables, disconnected structures, or intricate patterns.
- *Optimization Improvements:* Designing more efficient methods for training the PPSL model, such as advanced Monte-Carlo sampling techniques, refined gradient estimation, and adaptive optimization algorithms.
- *Expensive Optimization:* Considering the optimization scenario where the evaluation is computationally expensive. The gradient information can be obtained by surrogate models.

References

[1] A. V. Fiacco, "Sensitivity analysis for nonlinear programming using penalty methods," *Mathematical Programming*, vol. 10, no. 1, pp. 287–311, 1976.

[2] B. Bank, J. Guddat, D. Klatte, B. Kummer, and K. Tammer, *Non-linear parametric optimization*. Walter de Gruyter GmbH & Co KG, 1982, vol. 58.

[3] P. Milgrom and C. Shannon, "Monotone comparative statics," *Econometrica: Journal of the Econometric Society*, pp. 157–180, 1994.

[4] Y. Wang, H. Seki, S. Ohyama, M. Ogawa, M. Ohshima *et al.*, "Optimal grade transition control for polymerization reactors," *Computers & Chemical Engineering*, vol. 24, no. 2-7, pp. 1555–1561, 2000.

[5] T. Gal, *Linear parametric programming—a brief survey*. Springer, 1984.

[6] I. Pappas, D. Kenefake, B. Burnak, S. Avraamidou, H. S. Ganesh, J. Katz, N. A. Diangelakis, and E. N. Pistikopoulos, "Multiparametric programming in process systems engineering: Recent developments and path forward," *Frontiers in Chemical Engineering*, vol. 2, p. 620168, 2021.

[7] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[8] K. Al Handawi, P. Andersson, M. Panarotto, O. Isaksson, and M. Kokkolaras, "Scalable set-based design optimization and remanufacturing for meeting changing requirements," *Journal of Mechanical Design*, vol. 143, no. 2, p. 021702, 2021.

[9] P. Dua, F. J. Doyle, and E. N. Pistikopoulos, "Model-based blood glucose control for type 1 diabetes via parametric programming," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 8, pp. 1478–1491, 2006.

[10] B. Amos *et al.*, "Tutorial on amortized optimization," *Foundations and Trends® in Machine Learning*, vol. 16, no. 5, pp. 592–732, 2023.

[11] J. Marino, A. Piché, A. D. Ialongo, and Y. Yue, "Iterative amortized policy optimization," *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 15 667–15 681, 2021.

[12] J. Sjölund, "A tutorial on parametric variational inference," *arXiv preprint arXiv:2301.01236*, 2023.

[13] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. International Conference on Machine Learning (ICML)*, 2010, pp. 399–406.

[14] S. Venkataraman and B. Amos, "Neural fixed-point acceleration for convex optimization," in *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

[15] X. Lin, Z. Yang, X. Zhang, and Q. Zhang, "Continuation path learning for homotopy optimization," in *Proc. International Conference on Machine Learning (ICML)*, 2023, pp. 21 288–21 311.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[18] E. Galvan and R. J. Malak, "P3GA: An algorithm for technology characterization," *Journal of Mechanical Design*, vol. 137, no. 1, p. 011401, 2015.

[19] J. M. Weaver-Rosen and R. J. Malak, "An algorithm for multi-objective efficient parametric optimization," *Journal of Mechanical Design*, vol. 145, no. 3, p. 031709, 2023.

[20] J. Malak, Richard J. and C. J. J. Paredis, "Using parameterized pareto sets to model design concepts," *Journal of Mechanical Design*, vol. 132, no. 4, p. 041007, 2010.

[21] X. Lin, Z. Yang, Q. Zhang, and S. Kwong, "Controllable Pareto multi-task learning," *arXiv preprint arXiv:2010.06313*, 2020.

[22] A. Navon, A. Shamsian, G. Chechik, and E. Fetaya, "Learning the pareto front with hypernetworks," in *Proc. International Conference on Learning Representations (ICLR)*, 2021.

[23] X. Lin, Z. Yang, and Q. Zhang, "Pareto set learning for neural multi-objective combinatorial optimization," *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[24] X. Lin, Z. Yang, X. Zhang, and Q. Zhang, "Pareto set learning for expensive multi-objective optimization," *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 19 231–19 247, 2022.

[25] X. Lin, X. Zhang, Z. Yang, and Q. Zhang, "Dealing with structure constraints in evolutionary pareto set learning," *IEEE Transactions on Evolutionary Computation*, vol. 29, no. 3, pp. 616–630, 2025.

[26] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[27] K. Miettinen, *Nonlinear multiobjective optimization*. Springer Science & Business Media, 1999, vol. 12.

[28] Q. Zhang, A. Zhou, and Y. Jin, "RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 41–63, 2008.

[29] E. U. Choo and D. R. Atkins, "Proper efficiency in nonconvex multicriteria programming," *Mathematics of Operations Research*, vol. 8, no. 3, pp. 467–470, 1983.

[30] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[31] M. Ehrgott, *Multicriteria optimization*. Springer Science & Business Media, 2005, vol. 491.

[32] D. Mahapatra and V. Rajan, "Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization," in *Proc. International Conference on Machine Learning (ICML)*, 2020, pp. 6597–6607.

[33] X. Lin, X. Zhang, Z. Yang, F. Liu, Z. Wang, and Q. Zhang, "Smooth tchebycheff scalarization for multi-objective optimization," in *Proc. International Conference on Machine Learning (ICML)*, 2024.

[34] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on Optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[35] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, 2002.

[36] Z. Li, X. Lin, Q. Zhang, and H. Liu, "Evolution strategies for continuous optimization: A survey of the state-of-the-art," *Swarm and Evolutionary Computation*, vol. 56, p. 100694, 2020.

[37] K. Gao and O. Sener, "Generalizing gaussian smoothing for random search," in *Proc. International Conference on Machine Learning (ICML)*. PMLR, 2022, pp. 7077–7101.

[38] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949–980, 2014.

[39] J. Xu, P. B. Luh, F. B. White, E. Ni, and K. Kasiviswanathan, "Power portfolio optimization in deregulated electricity markets with risk management," *IEEE Transactions on Power Systems*, vol. 21, no. 4, pp. 1653–1662, 2006.

[40] L. Tang and Y. Meng, "Data analytics and optimization for smart industry," *Frontiers of Engineering Management*, vol. 8, no. 2, pp. 157–171, 2021.

[41] K. Kim, R. I. McKay, and B.-R. Moon, "Multiobjective evolutionary algorithms for dynamic social network clustering," in *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 2010, pp. 1179–1186.

[42] S. Jiang, S. Yang, X. Yao, K. C. Tan, M. Kaiser, and N. Krasnogor, "Benchmark functions for the CEC'2018 competition on dynamic multiobjective optimization," Newcastle University, Tech. Rep., 2018.

[43] M. Farina, K. Deb, and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 5, pp. 425–442, 2004.

[44] C.-K. Goh and K. C. Tan, "A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 1, pp. 103–127, 2008.

[45] A. Zhou, Y. Jin, and Q. Zhang, "A population prediction strategy for evolutionary dynamic multiobjective optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 40–53, 2013.

[46] S. Jiang, J. Zou, S. Yang, and X. Yao, "Evolutionary dynamic multiobjective optimisation: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–47, 2022.

[47] R. Azzouz, S. Bechikh, and L. Ben Said, "Dynamic multi-objective optimization using evolutionary algorithms: a survey," in *Recent Advances in Evolutionary Multi-objective Optimization*. Springer, 2016, pp. 31–70.

[48] K. Deb, U. B. Rao N, and S. Karthik, "Dynamic multi-objective optimization and decision-making using modified NSGA-II: a case study on hydro-thermal power scheduling," in *Evolutionary Multi-Criterion Optimization (EMO)*. Springer, 2007, pp. 803–817.

[49] Y. Ye, L. Li, Q. Lin, K.-C. Wong, J. Li, and Z. Ming, "Knowledge guided bayesian classification for dynamic multi-objective optimization," *Knowledge-Based Systems*, vol. 250, p. 109173, 2022.

[50] Y. Guo, G. Chen, M. Jiang, D. Gong, and J. Liang, "A knowledge guided transfer strategy for evolutionary dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1750–1764, 2022.

[51] Q. Lin, Y. Ye, L. Ma, M. Jiang, and K. C. Tan, "Dynamic multiobjective evolutionary optimization via knowledge transfer and maintenance," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 54, no. 2, pp. 936–949, 2023.

[52] Y. Ye, S. Liu, J. Zhou, Q. Lin, M. Jiang, and K. C. Tan, "Learning-based directional improvement prediction for dynamic multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, 2024.

[53] Y. Hu, J. Ou, P. N. Suganthan, W. Pedrycz, R. Wang, J. Zheng, J. Zou, and Y. Song, "Dynamic multi-objective optimization algorithm guided by recurrent neural network," *IEEE Transactions on Evolutionary Computation*, 2024.

[54] F. Wang, J. Xie, A. Zhou, and K. Tang, "A new prediction strategy for dynamic multi-objective optimization using diffusion model," *IEEE Transactions on Evolutionary Computation*, 2025.

[55] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2013.

[56] I. Oshri and S. Newell, "Component sharing in complex products and systems: challenges, solutions, and practical implications," *IEEE Transactions On Engineering Management*, vol. 52, no. 4, pp. 509–521, 2005.

[57] M. Fisher, K. Ramdas, and K. Ulrich, "Component sharing in the management of product variety: a study of automotive braking systems," *Management Science*, vol. 45, no. 3, pp. 297–315, 1999.

[58] S. Garcia and C. T. Trinh, "Modular design: implementing proven engineering principles in biotechnology," *Biotechnology Advances*, vol. 37, no. 7, p. 107403, 2019.

[59] P. Eremeev, A. De Cock, H. Devriendt, and F. Naets, "Product family design optimization considering manufacturing and assembly process costs," *Structural and Multidisciplinary Optimization*, vol. 67, no. 7, p. 115, 2024.

[60] L. Zhao, P. Wang, J. Shen, B. Song, and Q. Zhang, "Component-sharing preference in expensive multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, 2025.

[61] J. Marino, M. Cvitkovic, and Y. Yue, "A general method for amortizing variational filtering," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 7868–7879.

[62] B. Amos, "On amortizing convex conjugates for optimal transport," in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[63] A. Byravan, L. Hasenclever, P. Trochim, M. Mirza, A. D. Ialongo, Y. Tassa, J. T. Springenberg, A. Abdolmaleki, N. Heess, J. Merel, and M. Riedmiller, "Evaluating model-based planning and planner amortization for continuous control," in *Proc. International Conference on Learning Representations (ICLR)*, 2022.

[64] R. Tanabe and H. Ishibuchi, "An easy-to-use real-world multi-objective optimization problem suite," *Applied Soft Computing*, vol. 89, p. 106078, 2020.

[65] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.