OPTFORMER: BEYOND TRANSFORMER FOR BLACK-BOX OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We design a novel Transformer for continuous unconstrained black-box optimization, called Optformer. Inspired by the similarity between Vision Transformer and evolutionary algorithms (EAs), we modify Tansformer's multi-head self-attention layer, feed-forward network, and residual connection to implement the functions of crossover, mutation, and selection operators. Moreover, we devise an iterated mode to generate and survive potential solutions like EAs. Optformer establishes a mapping from the random population to the optimal population. Compared to baselines, such as EAs, Bayesian optimization, and the learning-to-optimize (L2O) method, Optformer shows the top performance in six black-box functions and one real-world application. We also find that untrained Optformer can also achieve good performance. We bring a new and efficient Transformer-based optimization framework for the L2O and evolutionary computation communities.

1 INTRODUCTION

Many tasks, such as neural architecture search Elsken et al. (2019) and hyperparameter optimization Hutter et al. (2019), can be abstracted as black-box optimization problems, which means that although we can evaluate $f(x)$ for any $x \in X$, we have no access to any other information about f, such as the Hessian and gradients. A series of hand-designed algorithms, such as evolutionary algorithms (EAs) Mitchell (1998); Khadka & Tumer (2018); Zhang & Li (2007), Bayesian optimization Snoek et al. (2012), and evolutionary strategies (ES) Wierstra et al. (2014); Hansen & Ostermeier (2001); Auger & Hansen (2005); Salimans et al. (2017), have been designed to solve black-box optimization.

Recently, the learning to optimize (L2O) framework Chen et al. (2021) gives an new insight on optimization by leveraging the recurrent neural network (RNN), long short-term memory architecture (LSTM) or multilayer perceptron (MLP) as the optimizer to develop optimization methods, aiming at reducing the laborious iterations of hand engineering Sun et al. (2018); Chen et al. (2020); Andrychowicz et al. (2016); Chen et al. (2017); Vicol et al. (2021); Flennerhag et al. (2021); Li & Malik (2016); Metz et al. (2019); Wichrowska et al. (2017); Bello et al. (2017); Sun et al. (2018). They rarely focus on black-box optimization problems. We think that the core of L2O is how to construct a strong mapping from the initial solutions to the optimal solution. Although several efforts like Cao et al. (2019); Chen et al. (2017) have coped with these problems, they have limited performance due to poor representation ability of RNN, LSTM, and MLP.

This paper designs a novel L2O framework based on the advantages of Vision Transformer Dosovitskiy et al. (2021) and EAs to overcome the above limitation, termed Optformer. In EAs, the hand-designed crossover, mutation, and selection operators make the initial population move near the optimal solution. This updated model has stood the test of time. Moreover, Transformer Han et al. (2022) owns a strong representation ability, and there is currently no work to use Transformer for optimization. Inspired by the similarity of EAs and Transformer Zhang et al. (2021; 2022), Optformer employs the critical part of Transformer to realize the mapping from the random and optimal populations. In order to generate potential individuals to approach the optimal solution, we first design an SA-based crossover module (SAC) to simulate the crossover operator of EA, and then the output of this module is input into the proposed FFN-based mutation module (FM) to perform mutation. Moreover, the residual and selection module (RSSM) is designed to survive the fittest individuals. RSSM is a pairwise comparison between the output of SAC, FM, and the input

population regarding their fitness. We design an Optformer Block (OB) consisting of SAC, FM, and RSSM. Finally, we construct Optformer by stacking OBs to simulate generations of EAs.

Moreover, to cope with black-box optimization, we establish a function set to train Optformer under an unsupervised mode. We construct a set of functions with similar properties to the targeted black-box optimization problems, and these functions are differentiable. This training set contains the pair of the initial population and the designed function. Thus, we can use gradient-based methods to optimize the parameters of Optformer.

We test the performance of Optformer on six standard functions and the problem of protein docking (Cao & Shen, 2020). The experimental results demonstrate the top rank of Optformer and the strong representation compared with three population-based baselines, Bayesian optimization, and one learning-to-optimize method Cao et al. (2019). Moreover, we also analyze the effect of weight sharing between SAC, FM, and RSSM in order to avoid overfitting. The highlights of this paper are summarized as follows:

1. For the L2O community, we propose a Transformer-based framework to focus on black-box optimization problems. We have shown its advantage with vanilla black-box optimization methods.
2. For the EA community, we design a new and efficient deep neural network framework to implement the mapping from the random population to the optimal solution, which is different from the hand-designed operators in traditional EAs.

2 Related Work

2.1 Transformer

Transformer structure achieves significant progress for machine translation task Vaswani et al. (2017), computer vision task Dosovitskiy et al. (2021), time series task Zhou et al. (2021), and so on. Many improved models are proposed and obtain great achievements Han et al. (2022). There are no Transformer-based efforts for handling optimization problems, which is crucial in the machine learning community. However, the Optformer proposed in this paper expands the application scope of Transformer and can effectively deal with this case.

2.2 Evolutionary Algorithm

Inspired by the evolution of species, EAs have provided surprising performance for black-box optimization Mitchell (1998). Many influential variants have been proposed to deal with different problems Das & Suganthan (2010); Wu & Liu (2019), but at their core they are: 1) recombination and mutation, how to produce the excellent solution; 2) selection, how to choose the best individuals between the parents and offspring. Thus, many algorithmic components have been designed for different tasks. The performance of algorithms varies towards various tasks, as different optimization strategies may be required given diverse landscapes. Current methods manually adjust genetic operators' hyperparameters and design the combination between them Kerschke et al. (2019); Tian et al. (2020) to map the random population to the optimal solution. The proposed Optformer replaces the hand-designed crossover, mutation, and selection operators with a Transformer framework. Then, we do not manually design the genetic operator but let the constructed Transformer learn to design and optimize the genetic operator.

3 Background

3.1 Problem Definition

A black-box optimization problem can be transformed as a minimization problem, as shown in Equation (1), and constraints may exist for corresponding solutions:

$$\min\ f(\boldsymbol{x}), s.t.\ x_i \in [l_i, u_i] \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_d)$ represents the solution of optimization problem f, the lower and upper bounds $\boldsymbol{l} = (l_1, l_2, \cdots, l_d)$ and $\boldsymbol{u} = (u_1, u_2, \cdots, u_d)$, and d is the dimension

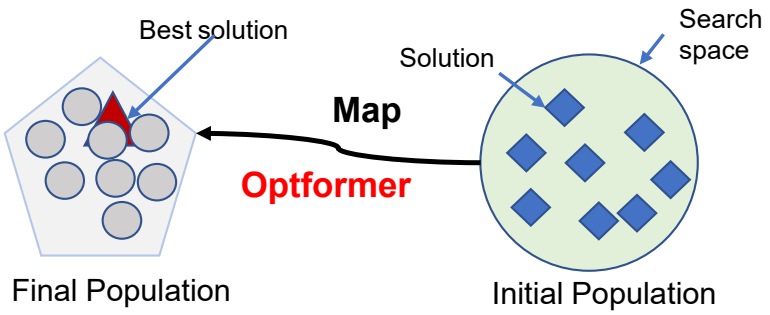

Figure 1: The goal of Optformer in solving two dimensional problem.

of $\boldsymbol{x}$. Suppose n individuals of one population be $\boldsymbol{X}_1 = (X_{1,1}, X_{1,2}, \cdots, X_{1,d}), \boldsymbol{X}_2 = (X_{2,1}, X_{2,2}, \cdots, X_{2,d}), \cdots, \boldsymbol{X}_n = (X_{n,1}, X_{n,2}, \cdots, X_{n,d})$, then EAs are required to find the population near the optimal solution $\hat{\boldsymbol{x}}$. Figure 1 shows the goal of human-designed EAs and the proposed Optformer in handling the two-dimensional problem.

3.2 Preliminary EAs

The crossover, mutation, and selection operators form the basic framework of EAs. EA starts with a randomly generated initial population. Then, genetic operations such as crossover and mutation will be carried out. After the fitness evaluation of all individuals in the population, a selection operation is performed to identify fitter individuals to undergo reproduction to generate offspring. Such an evolutionary process will be repeated until specific predefined stopping criteria are satisfied.

Crossover The crossover operator generates a new individual $\hat{\boldsymbol{X}}_i$ by Equation (2), and cr is the probability of the crossover operator.

$$\hat{\boldsymbol{X}}^c_{i,k} = \begin{cases} \boldsymbol{X}_{j,k} & rand(0,1) < cr \\ \boldsymbol{X}_{i,k} & otherwise \end{cases} \tag{2}$$

where $k \in [1, \cdots, d]$. This operator is commonly conducted on n individuals. After an expression expansion, we re-formulate Equation (2) as $\sum_{i=1}^{n} \boldsymbol{X}_i \boldsymbol{W}^c_i$ Zhang et al. (2021). $\boldsymbol{W}^c_i$ is the diagonal matrix. If $\boldsymbol{W}^c_i$ is full of zeros, the ith individual has no contribution.

Mutation The mutation operator brings random changes into the population. Specifically, an individual $\boldsymbol{X}_i$ in the population goes through the mutation operator to form the new individual $\hat{\boldsymbol{X}}_i$, formulated as follows:

$$\hat{\boldsymbol{X}}^m_{i,k} = \begin{cases} rand(l_k, u_k) & rand(0,1) < mr \\ \hat{\boldsymbol{X}}^c_{i,k} & otherwise \end{cases} \tag{3}$$

where mr is the probability of mutation operator and $k \in [1, \cdots, d]$. Similarly, Equation (3) can be re-formulated as $\sum_{i=1}^{n} \boldsymbol{X}_i \boldsymbol{W}^m_i$.

Selection We introduce the binary tournament mating selection operator in Equation (4). The selection operator survives individuals of higher quality for the next generation until the number of individuals is chosen.

$$p_i = \begin{cases} 1 & f(\boldsymbol{X}_i) < f(\boldsymbol{X}_k) \\ 0 & f(\boldsymbol{X}_i) > f(\boldsymbol{X}_k) \end{cases}, \quad (\boldsymbol{X}_i, \boldsymbol{X}_k) \in \boldsymbol{X}, \tag{4}$$

where p_i reflects the probability that $\boldsymbol{X}_i$ is selected for the next generation, and $(\boldsymbol{X}_i, \boldsymbol{X}_k)$ in Equation (4) are randomly selected from the population $\boldsymbol{X} \cup \hat{\boldsymbol{X}}^m$.

3.3 Vision Transformer

We mainly introduce the core part of Vision Transformer, such as the multi-head self-attention layer (MSA), feed-forward network (FFN), layer normalization (LN), and residual connection (RC).

MSA MSA fuses several SA operations to handle the queries ($\boldsymbol{Q}$), keys ($\boldsymbol{K}$), and values ($\boldsymbol{V}$) that jointly attend to information from different representation subspaces. MSA is formulated as follows: $MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Concat(\boldsymbol{H}_1, \boldsymbol{H}_2, \cdots \boldsymbol{H}_h)\boldsymbol{W}^O$. where $Concat$ means concatenation operation. The head feature $\boldsymbol{H}_i$ can be formulated as:

$$\begin{aligned}\boldsymbol{H}_i &= SA(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \\ &= Softmax\left(\boldsymbol{Q}\boldsymbol{W}_i^Q(\boldsymbol{K}\boldsymbol{W}_i^K)^T/sqrt(d_k)\right)\boldsymbol{V}\boldsymbol{W}_i^V = \boldsymbol{A}\boldsymbol{V}\boldsymbol{W}_i^V\end{aligned}$$

where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_m \times d_q}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_m \times d_k}$, and $\boldsymbol{W}_i^V \in \mathbb{R}^{d_m \times d_v}$ are parameter matrices for queries, keys, and values, respectively; $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d_m}$ maps each head feature $\boldsymbol{H}_i$ to the output. Moreover, d_m is the input dimension, while d_q, d_k, and d_v are hidden dimensions of the corresponding projection subspace; h is the head number. $\boldsymbol{A} \in \mathbb{R}^{l \times l}$ is the attention matrix of hth head, l is the sequence length.

FFN FFN employs two cascaded linear transformations with a ReLU activation to handle $\boldsymbol{X}$, which is shown as: $FFN(\boldsymbol{X}) = max(0, \boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$, where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are weights of two linear layers, and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are corresponding biases.

LN LN is applied before each layer of MSA and FFN, and the output of LN is calculated by $\boldsymbol{X} + [MSA|FFN](LN(\boldsymbol{X}))$.

4 OPTFORMER

The training procedure of Optformer can be formulated as $\boldsymbol{X}^t = E_\theta(\boldsymbol{X}^0, f_i(\boldsymbol{x}|\xi))$, where ξ is the known parameters of f_i, $\boldsymbol{X}^0$ is the initial population, and $\boldsymbol{X}^t$ is the output population. Optformer (E) determines the optimization strategy θ based on the training dataset $(\boldsymbol{X}^0, f_i(\boldsymbol{x}|\xi))$. Based on θ, Optformer optimizes the unseen funtion $f(\boldsymbol{x})$ by $\boldsymbol{X}^t = E_\theta(\boldsymbol{X}^0, f(\boldsymbol{x}))$.

4.1 SELF-ATTENTION CROSSOVER MODULE

We propose a new module based on SA to simulate a crossover operator. Suppose a population $\boldsymbol{X}$ is arranged in a non-descending order of fitness, and $\boldsymbol{F} \in \mathbb{R}^{n \times 1}$ be the fitness matrix of $\boldsymbol{X}$. Then, this module can be represented as follows:

$$\boldsymbol{X}^c = SAC(\boldsymbol{X}, \boldsymbol{F}) \tag{5}$$

where $\boldsymbol{X}^c$ is the output population of the proposed SAC module.

Since the object processed by Optformer is the population, and the order of individuals in the population does not affect the population distribution, SA does not require position coding. Standard SA projects the input sequence $\boldsymbol{X}$ into a d-dimensional space via QKV. These three mappings enable the SA module to capture better the characteristics of the problems encountered during training. In other words, these three mappings strengthen the ability of SA to focus on specific problems but do not necessarily make SA have good transferability between different problems. Therefore, we consider removing these three mappings for enhanced transferability, and $\boldsymbol{X}^c = \boldsymbol{A}\boldsymbol{X}$. $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a self-attention matrix that can be learned to maximize inter-individual information interaction based on individual ranking information. This is why the population needs to be sorted in non-descending order.

However, designing crossover operations based solely on population ranking information is a coarse-grained approach. Because this method only considers the location information of individuals in the population, but does not consider the fitness relationship between individuals. Therefore, we further introduce fitness information to assist in learning crossover operators:

$$\begin{aligned}\boldsymbol{A}^F &= SA(\boldsymbol{F}) \\ &= Softmax\left(\boldsymbol{F}\boldsymbol{W}^Q(\boldsymbol{F}\boldsymbol{W}^K)^T/sqrt(d_k)\right)\end{aligned}$$

Thus, $\boldsymbol{X}^c = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{A}^F\boldsymbol{X}$. To better balance the roles of $\boldsymbol{A}$ and $\boldsymbol{A}^F$, we introduce two learnable weights $\boldsymbol{W}_1^c \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{W}_2^c \in \mathbb{R}^{n \times 1}$. Therefore, the final crossover operation is shown as follows:

$$\boldsymbol{X}^c = tile(\boldsymbol{W}_1^c) \odot (\boldsymbol{A}\boldsymbol{X}) + tile(\boldsymbol{W}_2^c) \odot (\boldsymbol{A}^F\boldsymbol{X}) \tag{6}$$

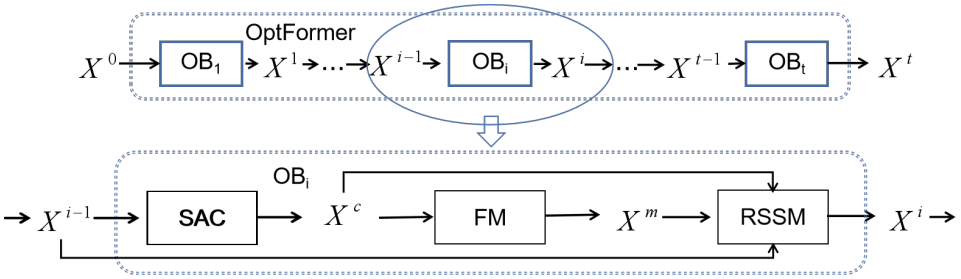

Figure 2: Overall architecture of Optformer and OB.

where $\boldsymbol{X}^c \in \mathbb{R}^{n \times d}$ is the population obtained by $\boldsymbol{X}$ through the SAC module; $\odot$ represents Hadamard product; the *tile* copy function extends the vector to a matrix.

4.2 FFN-BASED MUTATION MODULE

In Transformer, each patch embedding carries on directional feature transformation through the FFN module. We take one linear layer as an example: $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{W}^F$, where $\boldsymbol{W}^F$ is the weight of the linear layer, and it is applied to each embedding separately and identically. This equation and 3 have the same formula format, which inspires us to design a learnable mutation module FM based on FFN with $ReLU$ activation function:

$$\begin{aligned} \boldsymbol{X}^m &= FM(\boldsymbol{X}^c) \\ &= (ReLU(\boldsymbol{X}\boldsymbol{W}_1^F + \boldsymbol{b}_1))\boldsymbol{W}_2^F + \boldsymbol{b}_2 \end{aligned} \tag{7}$$

where $\boldsymbol{X}^m$ is the population after the mutation of $\boldsymbol{X}^c$. $\boldsymbol{W}_2^F$ and $\boldsymbol{W}_1^F$ represent the weight of the second layer of FFN and the weight of the first layer of FFN, respectively. $\boldsymbol{b}_2$ and $\boldsymbol{b}_1$ represent the bias of the second layer and the first layer of FFN, respectively.

4.3 SELECTION MODULE

The residual connection in the transformer can be analogized to the selection operation in EA Zhang et al. (2021). We combine the residual structure and SM to design a learnable selection module RSSM. The RSSM generates the offspring population according to the following equation:

$$\begin{aligned} \hat{\boldsymbol{X}} &= RSSM(\boldsymbol{X}, \boldsymbol{X}^c, \boldsymbol{X}^m) \\ &= Sort(SM(\boldsymbol{X}, tile(\boldsymbol{W}_1^s) \odot \boldsymbol{X} + tile(\boldsymbol{W}_2^s) \odot \boldsymbol{X}^c + tile(\boldsymbol{W}_3^s) \odot \boldsymbol{X}^m)) \end{aligned} \tag{8}$$

where $\hat{\boldsymbol{X}}$ is the fittest population for the next generation; the learnable weights $\boldsymbol{W}_1^s \in \mathbb{R}^{n \times 1}$, $\boldsymbol{W}_2^s \in \mathbb{R}^{n \times 1}$, and $\boldsymbol{W}_3^s \in \mathbb{R}^{n \times 1}$ are the weights for $\boldsymbol{X}$, $\boldsymbol{X}^c$, and $\boldsymbol{X}^m$, respectively. $Sort(\boldsymbol{X})$ represents that $\boldsymbol{X}$ is sorted in non-descending order of fitness. These three learnable weight matrices realize the weighted summation of residual connections, thereby simulating a learnable selection strategy. At the same time, the introduction of residual structure also enhances the model's representation ability, enabling Optformer to form a deep architecture.

SM updates individuals based on a pairwise comparison between the offspring and input population regarding their fitness. Suppose that $\boldsymbol{X}$ and $\boldsymbol{X}^{'}$ are the input populations of SM. We compare the quality of individuals from $\boldsymbol{X}$ and $\boldsymbol{X}^{'}$ pairwise based on fitness. A binary mask matrix indicating the selected individual can be obtained based on the indicator function $l_{x>0}(x)$, where $l_{x>0}(x) = 1$ if $x > 0$ and $l_{x>0}(x) = 0$ if $x < 0$. SM forms a new population $\hat{\boldsymbol{X}}$ by employing Equation (9).

$$\hat{\boldsymbol{X}} = tile(l_{x>0}(\boldsymbol{M}_{F'} - \boldsymbol{M}_F)) \odot \boldsymbol{X} + tile(1 - l_{x>0}(M_{F'} - M_F)) \odot \boldsymbol{X}^{'}, \tag{9}$$

where the *tile* copy function extends the indication vector to a matrix, $\boldsymbol{M}_F(\boldsymbol{M}_{F'})$ denotes the fitness matrix of $\boldsymbol{X}(\boldsymbol{X}^{'})$.

4.4 STRUCTURE OF OPTFORMER

Optformer comprises basic t Optformer blocks (OBs), and parameters can be shared among these t OBs or not. The overall architecture of Optformer and OB is shown in Figure 2. Each OB consists

of SAC, FM, and RSSM. $\boldsymbol{X}^0 \in R^{n\times d}$ represents the initial population input into Optformer, which needs to be sorted in non-descending order of fitness. In Equation 10, $\boldsymbol{X}^{i-1}$ is fed into OB_t to get $\boldsymbol{X}^i$, where $i \in [1, t]$. Optformer realizes the mapping from the random initial population to the target population by stacking t OBs.

$$\begin{aligned} \boldsymbol{X}^i &= OB(\boldsymbol{X}^{i-1}); \\ \boldsymbol{X}^c &= SAC(\boldsymbol{X}^{i-1}, \boldsymbol{F}); \\ \boldsymbol{X}^m &= FM(\boldsymbol{X}^c); \\ \boldsymbol{X}^i &= RSSM(\boldsymbol{X}^{i-1}, \boldsymbol{X}^c, \boldsymbol{X}^m) \end{aligned} \tag{10}$$

4.5 Training of Optformer

Training Dataset Training data is a crucial factor beyond the objective functions. Based on the theoretical basis of current machine learning: "independent and identical distribution of data," it is hard to adapt a trained model to different landscapes. This paper establishes the training set by constructing a set of differentiable functions related to the optimization objective. This training dataset only contains $(\boldsymbol{X}_0, f_i(\boldsymbol{x}|\xi))$, the initial population and objective function, respectively. The variance of ξ causes the shift in landscapes.

The training dataset is designed as follows: 1) Randomly initialize the input population $\boldsymbol{X}_0$; 2) Randomly produce a shifted objective function $f_i(\boldsymbol{x}|\xi)$ by adjusting the corresponding location of optima-namely, adjusting the parameter ξ; 3) Evaluate S_0 by $f_i(\boldsymbol{x}|\xi)$; 4) Repeat Steps 1)-3) to generate the corresponding dataset. We show the designed training and testing datasets as follows:

$$\begin{aligned} F^{train} &= \{f_1(\boldsymbol{x}|\xi_{1,i}^{train}), \cdots, f_m(\boldsymbol{x}|\xi_{m,i}^{train})\} \\ F^{test} &= \{F_1(\boldsymbol{x}|\xi_{1,i}^{test}), \cdots, F_k(\boldsymbol{x}|\xi_{k,i}^{test})\} \end{aligned} \tag{11}$$

where $\xi_{m,i}^{train}$ represents the ith different values of ξ in mth function f_m, which is true for any index pair. F_i is not employed in the training stage. F^{train} and F^{test} are comprised of different functions.

Loss Function Optformer attempts to search for individuals with high quality based on the available information. The loss function tells how to obtain the parameters of Optformer to generate individuals closer to the optimal solution by maximizing the difference between the initial population and the output population of Optformer. The following loss function is employed,

$$l_i(\boldsymbol{X}^0, f(\boldsymbol{x}|\xi)) = \frac{\frac{1}{|\boldsymbol{X}^0|}\sum\limits_{\boldsymbol{x}\in\boldsymbol{X}^0} f_i(\boldsymbol{x}|\xi) - \frac{1}{|E_\theta(\boldsymbol{X}^0)|}\sum\limits_{\boldsymbol{x}\in E_\theta(\boldsymbol{X}^0)} f_i(\boldsymbol{x}|\xi)}{\left|\frac{1}{|\boldsymbol{X}^0|}\sum\limits_{\boldsymbol{x}\in\boldsymbol{X}^0} f_i(\boldsymbol{x}|\xi)\right|} \tag{12}$$

where θ denotes parameters of Optformer. Equation (12) calculates the average fitness difference between the input and output, further normalized within $[0, 1]$. To encourage Optformer to explore the fitness landscape, for example, the constructed Bayesian posterior distribution over the global optimum Cao & Shen (2020) can be added to Equation (12). Since the derivatives of functions in the training dataset are available, we can obtain the gradient information of Equation (12) for the training process. Also, we can employ REINFORCE Williams (1992) to approximate these derivatives.

Training Optformer We then train Optformer under a supervised mode. Since the gradient is unnecessary during the test process, Optformer can solve black-box optimization problems. To prepare Optformer to learn a balanced performance upon different optimization problems, we design a loss function formulated as follows:

$$l_\Omega = -\frac{1}{K}\sum_{\boldsymbol{X}^0\in\Omega} l_i(\boldsymbol{X}^0, f_i(\boldsymbol{x}|\xi_i^{train})) \tag{13}$$

We employ Adam Kingma & Ba (2014) method with a minibatch Ω to train Optformer upon the constructed training dataset. Algorithm 1 gives a detailed process of Optformer, where m is the number of functions in F^{train}.

Algorithm 1 Training of Optformer

Input: Ω, F^{train}; **Output:** The parameters of Optformer.
Randomly initialize Optformer;
while The Condition is not satisfied **do**
  Randomly initialize a minibatch Ω comprised of K populations $\boldsymbol{X}^0$;
  for f_i **in** Fs^{train} **do**
    Update θ by l_Ω given training data $(\boldsymbol{X}^0, f_i)$;
  end for
  Update θ by minimizing $-1/m \sum_i l_\Omega$;
end while

Table 1: The compared results on six functions.

d	f	DE	ES	CMA-ES	L2O-swarm	Dragonfly	Optformer
10	F4	0.13(0.06)	0.22(0.30)	4.2e-4(3.5e-4)	16.92(2.10)	1.3e3(1.3e3)	**1.2e-4(5e-5)**
	F5	4.99(1.24)	0.55(0.37)	0.03(0.01)	2.97(0.01)	48.4(9.58)	**8e-3(2e-3)**
	F6	210.2(49.7)	60.02(48.28)	61.90(96.25)	26.83(21.48)	3.8e8(1.4e8)	**8.93(0.03)**
	F7	17.83(3.59)	51.53(8.05)	45.74(17.02)	4.88(3.55)	81.1(24.0)	**0.01(0.03)**
	F8	0.21(0.07)	0.26(0.18)	7.6e-3(0.01)	1.02(1.4e-3)	35.4(22.6)	**le-5(2e-5)**
	F9	1.90(0.32)	20.56(0.03)	0.04(0.02)	9.06(0.67)	16.2(3.64)	**0.01(3.4e-3)**
100	F4	8.2e3(3.8e2)	8.9e4(9.5e3)	7.8e3(1.2e3)	0.32(0.03)	–	**0.11(0.09)**
	F5	28.2(0.61)	80.2(2.10)	78.3(9.18)	0.28(1.3e-3)	–	**0.14(0.15)**
	F6	2.4e8(2.3e7)	2.5e10(4.5e9)	3.3e8(8.7e7)	692(108)	–	**129(346)**
	F7	9410(548)	8.9e4(1.1e4)	8050(775)	85.7(18.6)	–	**24.1(13)**
	F8	3.04(0.14)	23.0(3.00)	3(0.22)	0.16(2.4e-4)	–	**0.02(0.03)**
	F9	18.9(0.14)	21.4(0.02)	21.4(0.04)	2.49(2.5e-3)	–	**0.15(0.05)**

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

5.1.1 DATASETS

This paper employs nine commonly used functions and protein docking problems Cao & Shen (2020) to show the effectiveness of the proposed Optformer. The characteristics of these nine functions are shown in Tables 5 and 6 (Appendix). Here, Optformer is trained on F^{train} is generated based on functions in Table 5, and F^{test} is generated based on functions in Table 6 (Appendix). Here, $d = 10, 100$.

We also handle the problem of *Ab initio* protein docking Cao & Shen (2020), which optimizes a noisy and costly function in a high-dimensional conformational space. Mathematically, this problem is formulated as optimizing the Gibbs binding free energy $f(\boldsymbol{x})$ for conformation $\boldsymbol{x}$. We calculate the energy function in a CHARMM 19 force field as in Moal & Bates (2010) and shift it so that $f(\boldsymbol{x}) = 0$ at the origin of the search space. $f(\boldsymbol{x})$ is differentiable when we parameterize the search space as $\mathbb{R}^{12}$ Smith & Sternberg (2002). Here, only 100 interface atoms are considered.

Training dataset. 25 protein-protein complexes from the protein docking benchmark set 4.0 Hwang et al. (2010), each of which has 5 starting points (top-5 models from ZDOCK Pierce et al. (2014)).

Testing dataset. Three complexes (with one starting model each) of different levels of docking difficulty are selected, including 1ATN_7, 2JEL_1, and 7CEI_1.

5.1.2 BASELINES

Optformer is compared with standard EA baselines, such as DE Das & Suganthan (2010), ES, and CMA-ES. L2O-swarm Cao et al. (2019) is a representative L2O method for black-box optimization. Moreover, Dragonfly Kandasamy et al. (2020), a representative algorithm for Bayesian optimization, is employed as a reference.

5.1.3 PARAMETERS

For example, Optformer(ws30) contains 30 OBs, and each OB consists of 1 SAC, 1 FM, and 1 RSSM. In Optformer(ws30), these 30 OBs share parameters. Optformer(nws5) has 5 OBs, and no parameters are shared among them. During the training process, Optformer is iterated for 1000 epochs. The initial learning rate (lr) was set to 0.01 and $lr = lr \times 0.9$ each 100 cycles.The 2-norm of the gradient is clipped so that it is not larger than 10. The bias of the function is regenerated each epoch, and a new batch of random initial populations is generated. The number of generations of the reference algorithms is set to 100. The population size of ES, DE, and CMA-ES is set to 100. All experiments were performed on a Ubuntu20.04 PC with Intel(R) Core I7 (TM) I3-8100 CPU at 3.60GHz and NVIDIA GeForce GTX 1060.

Table 2: The results on the problem of protein docking.

Methods	1ATN_7	2JEL_1	7CEI_1
L2O-swarm	2090(25.08)	2765(24.80)	1689(23.64)
Optformer(ws30)	-6.03e3(145)	-6.03e3(127)	-5.97e3(125)
Optformer(nws5)	-6.01e3(146)	-6.02e3(125)	-5.98e3(131)

5.2 COMPARED RESULTS

The results on nine functions are provided in Table 1. Optformer outperforms three EA baselines, Dragonfly, and L2O-swarm in all cases. These cases also show the excellent generalization ability of Optformer on more tasks unseen during the training stage. We think the transferability of Optformer is proportional to the fitness landscape similarity between the training set and the problem. Although new problem attributes are not available in the training set, Optformer can still perform better. However, this conclusion only holds when the similarity between the problem and training dataset is high.

We also test the performance of Optformer on the problem of protein docking. The experimental results is shown in 2. The performance of Optformer exceeds that of L2O-swarm. During training, L2O-swarm does not converge. At the same time, we find that better solutions exist in the initial population than those found by L2O-swarm. However, during testing, L2O-swarm lost these good solutions.

5.3 PARAMETER ANALYSIS

We consider the performance of different Optformer architectures. We design three models, including Optformer(ws3), Optformer(nws5), and Optformer(ws30). Optformer(ws3) contains three layers. The experimental results are shown in Table 3.

We find that they were sorted from good to worst by their performance, and the result is Optformer(ws30)>Optformer(nws5)>Optformer(ws3). Deep architectures have better representation capabilities and also lead to better performance. However, it is challenging to train non-weight sharing Optformers with more layers due to the difficulty of training deep architectures.

Table 3: The performance of different Optformer structures. $d = 10$.

f	Optformer (nws5-r)	Optformer (nws5)	Optformer (ws30)	Optformer (ws3)
F4	0.28(0.09)	0.08(0.03)	**1.2e-4(5e-5)**	8.16(3.44)
F5	0.37(0.05)	0.15(0.03)	**0.008(0.002)**	1.47(0.40)
F6	45.8(16.9)	15.43(2.34)	**8.93(0.03)**	1891(1396)
F7	1.08(0.72)	4.43(1.82)	**0.01(0.03)**	35.72(8.52)
F8	0.69(0.09)	0.06(0.03)	**1e-5(2e-5)**	0.82(0.10)
F9	0.85(0.20)	0.29(0.07)	**0.01(0.003)**	3.28(1.00)

Table 4: The results of ablation study. $d = 10$.

f	Optformer-nc	Optformer-nm	Optformer-nr	Optformer-ns	Optformer
F4	52.7(15.0)	23.3(6.68)	50.5(6.81)	271(346)	3.03(5.82)
F5	5.08(0.98)	2.94(0.41)	3.75(0.09)	3.51(1.49)	1.02(0.20)
F6	1.25e5(9.26e4)	1.10e4(1.44e4)	5.22e4(1.28e4)	6.99e6(1.21e8)	4.03e3(6.94e4)
F7	47.2(5.11)	19.7(4.27)	63.4(8.60)	40.6(8.10)	44.8(8.34)
F8	1.18(0.04)	1.19(0.07)	0.87(0.07)	3.28(1.14)	0.60(0.17)
F9	4.49(1.09)	3.42(0.78)	8.36(0.43)	3.56(0.72)	3.03(0.70)

Optformer(nws5-r) represents that the parameters of Optformer(nws5) are randomly initialized. The results show that Optformer(nws5) outperforms Optformer(nws5-r), which demonstrates the effectiveness of the designed training process.

We also find an interesting phenomenon: Optformer(nws5-r) outperforms Optformer(ws3) in all cases. Our untrained deep architecture, Optformer (nws5-r), can achieve good results, which shows that Optformer retains the advantages of Transformer architecture and has strong generalization ability. Perhaps using ES is a good training strategy Vicol et al. (2021).

5.4 Ablation Study

This section considers the performance impact of different parts in Optformer. We take Optformer(ws3) as an example, which is trained on F1-F3 and tested on F4-F9. We remove SAC, FM, RSSM, and RC in Optformer, respectively, and denote them as Optformer-nc, Optformer, Optformer-nm, Optformer-ns, and Optformer-nr. The experimental results are shown in Table 4.

When their results were sorted from good to worst, the rank is Optformer $>$ Optformer-nm $>$ Optformer-nr $\approx$ Optformer-nc $\approx$ Optformer-ns. The role of FM is slightly weaker than that of the other three modules. Taken as a whole, the parts of SAC, RSSM, and RC are of equal importance. The absence of these core components can seriously affect the performance of Optformer. At the same time, it also shows the effectiveness of the proposed four modules. The removal of any one of the modules in the crossover, mutation, and selection of EAs will degrade the performance of EAs. This shows that Optformer implements a learnable EA framework that does not require human-designed parameters.

6 Conclusions

We successfully designed the Transformer-based L2O framework for black-box optimization, which does not need hand-designed operators. The better performance than that of EA baselines, Bayesian optimization, and the L2O method demonstrates the effectiveness of Optformer. Moreover, Optformer can be well adapted to unseen black-box optimization. Meanwhile, we experimentally demonstrate that the proposed three modules have positive effects. Optformer still has room for improvement.

1. Our scheme is not limited to black-box optimization. Similar to the LSTM architecture, our scheme can directly optimize differentiable functions. However, the architecture of Optformer does not directly involve the gradient information of the optimization target, which makes Optformer inferior to existing L2O schemes. In future work, we will design a new module that embeds the gradient information of the optimization target;

2. In the loss function, we did not effectively consider the diversity of the population, and the population can be regularized in the future;

3. The training set seriously affects the performance of Optformer. If the similarity between the training set and the optimization objective is low, it will cause the performance of Optformer to degrade drastically. Building the dataset as relevant to the target as possible is essential.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pp. 1769–1776, 2005.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pp. 459–468. PMLR, 2017.

Yue Cao and Yang Shen. Bayesian active learning for optimization and uncertainty quantification in protein docking. *Journal of Chemical Theory and Computation*, 16(8):5334–5347, 2020.

Yue Cao, Tianlong Chen, Zhangyang Wang, and Yang Shen. Learning to optimize in swarms. *Advances in Neural Information Processing Systems*, 32:15044–15054, 2019.

Tianlong Chen, Weiyi Zhang, Zhou Jingyang, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7332–7343. Curran Associates, Inc., 2020.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pp. 748–756. PMLR, 2017.

Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on Evolutionary Computation*, 15(1):4–31, 2010.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Sebastian Flennerhag, Yannick Schroecker, Tom Zahavy, Hado van Hasselt, David Silver, and Satinder Singh. Bootstrapped meta-learning. *arXiv preprint arXiv:2109.04504*, 2021.

Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 2022.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

Howook Hwang, Thom Vreven, Joël Janin, and Zhiping Weng. Protein–protein docking benchmark version 4.0. *Proteins: Structure, Function, and Bioinformatics*, 78(15):3111–3114, 2010.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabas Poczos, and Eric P. Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565. PMLR, 2019.

Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

Iain H Moal and Paul A Bates. Swarmdock and the use of normal modes in protein-protein docking. *International Journal of Molecular Sciences*, 11(10):3623–3648, 2010.

Brian G Pierce, Kevin Wiehe, Howook Hwang, Bong-Hyun Kim, Thom Vreven, and Zhiping Weng. Zdock server: interactive docking prediction of protein–protein complexes and symmetric multimers. *Bioinformatics*, 30(12):1771–1773, 2014.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Graham R Smith and Michael JE Sternberg. Prediction of protein–protein interactions by docking methods. *Current opinion in structural biology*, 12(1):28–35, 2002.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Haoran Sun, Xiangyi Chen, Qingjiang Shi, Mingyi Hong, Xiao Fu, and Nicholas D. Sidiropoulos. Learning to optimize: Training deep neural networks for interference management. *IEEE Transactions on Signal Processing*, 66(20):5438–5453, 2018. doi: 10.1109/TSP.2018.2866382.

Ye Tian, Shichen Peng, Xingyi Zhang, Tobias Rodemann, Kay Chen Tan, and Yaochu Jin. A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks. *IEEE Transactions on Artificial Intelligence*, 1(1):5–18, 2020. doi: 10.1109/TAI.2020.3022339.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pp. 10553–10563. PMLR, 2021.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pp. 3751–3760. PMLR, 2017.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Kai Wu and Jing Liu. Classification-based optimization with multi-fidelity evaluations. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1126–1131. IEEE, 2019.

Jiangning Zhang, Chao Xu, Jian Li, Wenzhou Chen, Yabiao Wang, Ying Tai, Shuo Chen, Chengjie Wang, Feiyue Huang, and Yong Liu. Analogous to evolutionary algorithm: Designing a unified sequence model. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 26674–26688. Curran Associates, Inc., 2021.

Jiangning Zhang, Xiangtai Li, Yabiao Wang, Chengjie Wang, Yibo Yang, Yong Liu, and Dacheng Tao. Eatformer: Improving vision transformer inspired by evolutionary algorithm. *arXiv preprint arXiv:2206.09325*, 2022.

Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11106–11115, 2021.

A APPENDIX

Table 5: Training functions.

ID	Functions	Range
F1	$\sum_i \lvert w_i sin(x_i - b_i)\rvert$	$x \in [-10, 10], b \in [-10, 10]$
F2	$\sum_i \lvert x_i - b_i\rvert$	$x \in [-10, 10], b \in [-10, 10]$
F3	$\sum_i \lvert(x_i - b_i) + (x_{i+1} - b_{i+1})\rvert + \sum_i \lvert x_i - b_i\rvert$	$x \in [-10, 10], b \in [-10, 10]$

Table 6: Testing Functions.

ID	Functions	Range
F4(Sphere)	$\sum_i z_i^2, z_i = x_i - b_i$	$x \in [-100, 100], b \in [-50, 50]$
F5	$\max\{\lvert z_i\rvert, 1 \le i \le D\}, z_i = x_i - b_i$	$x \in [-100, 100], b \in [-50, 50]$
F6(Rosenbrock)	$\sum_{i=1}^{D-1} (100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2), z_i = x_i - b_i$	$x \in [-100, 100], b \in [-50, 50]$
F7(Rastrigin)	$\sum_{i=1}^{D} (z_i^2 - 10\cos(2\pi z_i) + 10), z_i = x_i - b_i$	$x \in [-5, 5], b \in [-2.5, 2.5]$
F8(Griewank)	$\sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{z_i}{\sqrt{i}}) + 1, z_i = x_i - b_i$	$x \in [-600, 600], b \in [-300, 300]$
F9(Ackley)	$-20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)) + 20 + \exp(1), z_i = x_i - b_i$	$x \in [-32, 32], b \in [-16, 16]$