Self-adaptive PSRO: Towards an Automatic Population-based Game Solver

Pengdeng Li[1], **Shuxin Li**[1*], **Chang Yang**[2*], **Xinrun Wang**[1†], **Xiao Huang**[2], **Hau Chan**[3] and **Bo An**[1]
[1]Nanyang Technological University, Singapore
[2]The Hong Kong Polytechnic University, Hong Kong SAR, China
[3]University of Nebraska-Lincoln, Lincoln, Nebraska, United States
{pengdeng.li, shuxin.li, xinrun.wang, boan}@ntu.edu.sg, chang.yang@connect.polyu.hk
xiaohuang@comp.polyu.edu.hk, hchan3@unl.edu

Abstract

Policy-Space Response Oracles (PSRO) as a general algorithmic framework has achieved state-of-the-art performance in learning equilibrium policies of two-player zero-sum games. However, the hand-crafted hyperparameter value selection in most of the existing works requires extensive domain knowledge, forming the main barrier to applying PSRO to different games. In this work, we make the first attempt to investigate the possibility of self-adaptively determining the optimal hyperparameter values in the PSRO framework. Our contributions are three-fold: (1) Using several hyperparameters, we propose a parametric PSRO that unifies the gradient descent ascent (GDA) and different PSRO variants. (2) We propose the self-adaptive PSRO (SPSRO) by casting the hyperparameter value selection of the parametric PSRO as a hyperparameter optimization (HPO) problem where our objective is to learn an HPO policy that can self-adaptively determine the optimal hyperparameter values during the running of the parametric PSRO. (3) To overcome the poor performance of online HPO methods, we propose a novel offline HPO approach to optimize the HPO policy based on the Transformer architecture. Experiments on various two-player zero-sum games demonstrate the superiority of SPSRO over different baselines.

1 Introduction

Policy-Space Response Oracles (PSRO) [Lanctot *et al.*, 2017] since proposed has been the mainstream algorithmic framework for solving two-player zero-sum games. At each epoch, PSRO constructs a meta-game by simulating outcomes of all match-ups of policies of all players and computes the meta-strategies for all players via a meta-solver. It then trains new policies for each player against the opponent's meta-strategy through an oracle and appends the new policies to the player's policy space. The two components – meta-solver and oracle – determine the nature of PSRO and various PSRO variants have been proposed [Balduzzi *et al.*, 2019; Muller *et al.*, 2020; Marris *et al.*, 2021]. Despite the advancements, determining the hyperparameter values in PSRO is non-trivial [Smith *et al.*, 2021] and typically involves extensive domain knowledge, which impedes it from broader real-world applications.

Precisely, one needs to determine the meta-solver and the best response (BR) oracle when instantiating PSRO. On one hand, existing works have suggested various meta-solvers such as uniform [Heinrich and Silver, 2016], Nash equilibrium [Lanctot *et al.*, 2017], α-Rank [Muller *et al.*, 2020], and correlated equilibrium [Marris *et al.*, 2021]. However, we observe that none of the meta-solvers can consistently beat all the others in terms of learning performance during game solving. On the other hand, the BR policies of a player are typically obtained via a deep reinforcement learning (RL) oracle, e.g., DQN [Mnih *et al.*, 2015], which involves the initialization and the number of updates of the BR policies. Unfortunately, the determination of these hyperparameter values in most of the existing works is often domain-specific (e.g., poker, soccer). Therefore, an important question is: *Can we automatically determine the optimal hyperparameter values in PSRO?*

In this work, we make the first attempt to answer this question. Specifically, we first propose a parametric PSRO (PPSRO) by introducing two types of hyperparameters: i) *game-free* hyperparameters are the weights of different meta-solvers considered during game solving, and ii) *game-based* hyperparameters are the initialization and the number of updates of a player's BR policies. PPSRO provides a general framework to unify the gradient descent ascent (GDA) [Fiez and Ratliff, 2021] and various PSRO variants [Ho *et al.*, 1998; Balduzzi *et al.*, 2019; Muller *et al.*, 2020; Marris *et al.*, 2021]. Then, a natural problem is how to determine the hyperparameter values of PPSRO. To solve this problem, we propose a novel framework, self-adaptive PSRO (SPSRO), by casting the hyperparameter value selection of PPSRO as a hyperparameter optimization (HPO) problem where our objective is to learn an HPO policy that can self-adaptively select the optimal hyperparameter values of PPSRO during game solving. A straightforward method to optimize the HPO policy is to use online approaches such as Optuna [Akiba *et al.*, 2019]. Unfortunately, online HPO methods only use online generated data (past epochs of SPSRO), typically constraining the training objectives to be cheaply computable [Chen *et al.*, 2022] and the performance could be poor. To overcome these limitations, we propose an offline HPO approach to optimize the HPO policy

[*]Equal contribution.
[†]Corresponding author.

based on the Transformer architecture [Vaswani *et al.*, 2017; Chen *et al.*, 2021a]. Specifically, we formulate the HPO policy optimization as a sequence modeling problem where a Transformer model is trained by using an offline dataset and then used to predict the hyperparameter values conditioned on past epochs of SPSRO. Intuitively, a well-trained HPO policy has the potential to transfer to different games, reducing the effort needed for researchers to conduct the costly hyperparameter tuning when applying PSRO to various games.

In summary, the contributions of this work are three-fold: (1) By introducing several hyperparameters, we propose a parametric version of PSRO (PPSRO) which unifies GDA and various PSRO variants. (2) We propose a novel framework, self-adaptive PSRO (SPSRO), by formulating an optimization problem where our objective is to learn an HPO policy that can self-adaptively determine the optimal hyperparameter values of PPSRO. (3) To overcome the poor performance of classic online HPO methods, we propose an offline HPO approach to optimize the HPO policy based on the Transformer architecture. We evaluate the effectiveness of our approach through extensive experiments on a set of two-player zero-sum normal-form and extensive-form games, and the results demonstrate that SPSRO with Transformer can achieve significantly better learning performance than different baselines.

2 Related Works

PSRO [Lanctot *et al.*, 2017] generalizes the double oracle algorithm [McMahan *et al.*, 2003] and unifies various multi-agent learning methods including the fictitious play (FP) [Robinson, 1951; Brown, 1951], neural fictitious self-play (NFSP) [Heinrich and Silver, 2016] (an extension of FP in the context of deep RL), iterated best response (IBR) [Ho *et al.*, 1998], and independent reinforcement learning (InRL) [Matignon *et al.*, 2012]. Recently, many works have been done toward improving PSRO, including the scalability [McAleer *et al.*, 2020; Smith *et al.*, 2021], diversity of BRs [Perez-Nieves *et al.*, 2021; Liu *et al.*, 2021a], the introduction of novel meta-solvers, e.g., α-Rank [Muller *et al.*, 2020], correlated equilibrium [Marris *et al.*, 2021], and neural meta-solver [Feng *et al.*, 2021], and application to mean-field games [Muller *et al.*, 2022a]. Moreover, the challenging strategy exploration problem has also been extensively investigated [Wellman, 2006; Schvartzman and Wellman, 2009b; Schvartzman and Wellman, 2009a; Jordan *et al.*, 2010; Wang *et al.*, 2022]. Despite the advancements, a critical observation we obtained is that given a set of meta-solvers, none of them can consistently beat (dominate) all the others in terms of learning performance during game solving (in the sense that we just evaluate PSRO as an online algorithm). On the other hand, the BRs of a player are typically obtained via a deep RL oracle such as DQN [Mnih *et al.*, 2015], where the hyperparameters (e.g., the initialization and the number of updates) are often domain-specific (e.g., poker, soccer) and most of the existing works manually select the hyperparameter values based on domain knowledge. In this work, we develop a novel framework to self-adaptively determine the optimal hyperparameter values in PSRO, which can be transferred to different games without fine-tuning.

Another line of related work is hyperparameter optimization (HPO). Existing works on HPO can be roughly categorized into online and offline HPO. The classic online HPO methods include Bayesian optimization [Snoek *et al.*, 2012] and its variants [Krause and Ong, 2011; Bardenet *et al.*, 2013; Swersky *et al.*, 2013; Feurer *et al.*, 2015; Volpp *et al.*, 2019; Wistuba and Grabocka, 2020; Rothfuss *et al.*, 2021], and recurrent neural networks (RNNs) [Duan *et al.*, 2016; Wang *et al.*, 2016; Chen *et al.*, 2017]. However, online HPO methods only use online generated data, typically constraining the training objectives to be cheaply computable and the performance could be poor. Our work is closely related to [Chen *et al.*, 2022] which proposes the first offline Transformer-based HPO method. Nevertheless, it is non-trivial to optimize the HPO policy as it involves several critical challenges such as how to generate an offline dataset for training. To our knowledge, this work is the first attempt to explore and develop a self-adaptive hyperparameter value selector in game theory.

3 Preliminaries

In this section, we first present the game definition and then the procedure of the PSRO algorithm.

3.1 Games

Consider a two-player zero-sum game represented by a tuple $G = (\mathcal{N}, \mathcal{S}, \mathcal{A}, p, \{r^i\}_{i\in\mathcal{N}}, \mathcal{T})$, where players are indexed by $\mathcal{N} = \{1, 2\}$. Let $N = |\mathcal{N}| = 2$. $\mathcal{S}$ and $\mathcal{A}$ denote the players' state and action spaces, respectively. $\mathcal{T} = \{0, 1, \cdots, T\}$ is time index set. At $t \in \mathcal{T}$, player i in state $s_t^i \in \mathcal{S}$ takes an action $a_t^i \in \mathcal{A}$ and then changes to new state $s_{t+1}^i \sim p(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and receives a reward $r^i(\boldsymbol{s}_t, \boldsymbol{a}_t)$, where $\boldsymbol{s}_t = (s_t^i)_{i\in\mathcal{N}}$ and $\boldsymbol{a}_t = (a_t^i)_{i\in\mathcal{N}}$ are respectively joint state and joint action of all players, $p : \mathcal{S}^N \times \mathcal{A}^N \to \Delta(\mathcal{S})$[1] is the transition function and $r^i : \mathcal{S}^N \times \mathcal{A}^N \to \mathbb{R}$ is the reward function with $\sum_{i\in\mathcal{N}} r^i(\boldsymbol{s}_t, \boldsymbol{a}_t) = 0$. Let $\pi^i : \mathcal{S} \to \Delta(\mathcal{A})$ denote the player i's policy (strategy)[2] with $\pi^i \in \Pi^i$ where Π^i is the policy space. Given the joint policy of all players $\boldsymbol{\pi} = (\pi^i)_{i\in\mathcal{N}} \in \Pi = \times_{i\in\mathcal{N}}\Pi^i$, each player i aims to maximize his own value function $V^i(\boldsymbol{\pi}, \boldsymbol{s}_0) = \mathbb{E}\big[\sum_{t=0}^{T} r^i(\boldsymbol{s}_t, \boldsymbol{a}_t)|\boldsymbol{a}_t \sim \boldsymbol{\pi}, \boldsymbol{s}_{t+1} \sim p\big]$, where $\boldsymbol{s}_0$ is the players' initial states.

A mixed strategy $\sigma^i \in \Delta(\Pi^i)$[3] is called a *meta-strategy* which is the probability distribution over the player i's policy space Π^i. More precisely, suppose that there are $c \geq 1$ policies in player i's policy space, then the meta-strategy of i is $\sigma^i = (\sigma^{i,1}, \cdots, \sigma^{i,c})$ with $\sigma^{i,j} \geq 0$ and $\sum_{j=1}^{c} \sigma^{i,j} = 1$. Accordingly, $\boldsymbol{\sigma} = (\sigma^i)_{i\in\mathcal{N}} \in \Delta(\Pi)$ is the joint meta-strategy of all players. Given the joint meta-strategy of all players except i, $\boldsymbol{\sigma}^{-i}$, the expected payoff to player i's policy $\pi^i \in \Pi^i$ is given by $R^i(\pi^i, \boldsymbol{\sigma}^{-i}) = \sum_{\boldsymbol{\pi}^{-i}\in\Pi^{-i}} \boldsymbol{\sigma}^{-i}(\boldsymbol{\pi}^{-i}) V^i(\pi^i, \boldsymbol{\pi}^{-i})$ and the set of *best responses (BRs)* of player i is defined as $\mathrm{BR}^i(\boldsymbol{\sigma}^{-i}) = \arg\max_{\pi^i\in\Pi^i} R^i(\pi^i, \boldsymbol{\sigma}^{-i})$. The quality of $\boldsymbol{\sigma}$ can be measured by the NashConv [Lanctot *et al.*, 2017]. For player $i \in \mathcal{N}$, the NashConv is defined as $\mathcal{R}^i(\boldsymbol{\sigma}) =$

[1] $\Delta(\mathcal{X})$ denotes the probability distribution over the space $\mathcal{X}$.

[2] We interchangeably use *policy* and *strategy* in this work.

[3] In principle Π^i could be an infinite set. However, Π^i is typically iteratively expanded by learning algorithms such as PSRO and hence, is considered finite in this work.

$R^i(\text{BR}^i(\boldsymbol{\sigma}^{-i}), \boldsymbol{\sigma}^{-i}) - \sum_{\pi^i \in \Pi^i} \sigma^i(\pi^i) R^i(\pi^i, \boldsymbol{\sigma}^{-i})$. That is, the NashConv of player i is the gain he can obtain when he unilaterally deviates from the current joint meta-strategy to a (pure) BR strategy. Therefore, the quality of $\boldsymbol{\sigma}$ is measured by the total NashConv of all players $\mathcal{R}(\boldsymbol{\sigma}) = \sum_{i \in \mathcal{N}} \mathcal{R}^i(\boldsymbol{\sigma})$.

3.2 Policy-Space Response Oracles

Given a game, PSRO first initializes the policy space of each player Π^i using randomly generated policies and then expands the policy space in three iterated phases[4] (as shown in Figure 2): (1) Synthesize a meta-game with all match-ups of policies of all players and compute the missing payoff entries in the payoff tensor M via simulation. (2) Compute the joint meta-strategy $\boldsymbol{\sigma}$ using a meta-solver $\mathcal{M}$ on the synthesized meta-game M. Different meta-solvers can be used during training, e.g., Nash equilibrium [Lanctot *et al.*, 2017], correlated equilibrium [Marris *et al.*, 2021], α-Rank [Muller *et al.*, 2020], and uniform distribution [Heinrich and Silver, 2016]. (3) Compute each player i's BR $\pi^{i,\text{BR}}$ using an oracle $\mathcal{O}^i$ given the joint meta-strategy $\boldsymbol{\sigma}$ and add the BR $\pi^{i,\text{BR}}$ to player i's policy space Π^i. As the other player's policy spaces Π^{-i} and joint meta-strategy $\boldsymbol{\sigma}^{-i}$ are fixed, the computation of BR is a single-player optimization problem from player i's perspective. In practice, the best response is typically approximated by using deep RL algorithms, e.g., DQN [Mnih *et al.*, 2015]. Specifically, the best response $\pi^{i,\text{BR}}$ is trained for K updates, given that the other player uses the policy $\boldsymbol{\pi}^{-i} \sim \boldsymbol{\sigma}^{-i}$, i.e., $\boldsymbol{\pi}^{-i}$ is sampled according to $\boldsymbol{\sigma}^{-i}$.

4 Motivating Example

In this section, we provide some examples to better illustrate the motivation of this work. Consider a two-player zero-sum normal-form game (NFG) of size $|\mathcal{A}_1| \times |\mathcal{A}_2|$. Let $\boldsymbol{\mathcal{M}}$ denote the set of meta-solvers of interest. In this example (as well as this work), we consider the most commonly used three meta-solvers: Uniform [Heinrich and Silver, 2016], α-Rank [Muller *et al.*, 2020], and PRD [Lanctot *et al.*, 2017]. We conduct two types of experiments: i) consistently using a single meta-solver during the PSRO procedure, and ii) switching the meta-solver from one to another at some intermediate iteration of the PSRO. The results are shown in Figure 1.

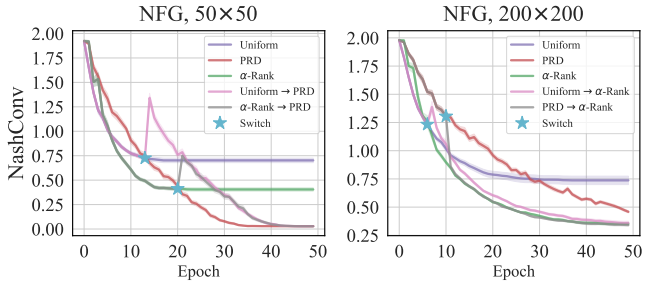


Figure 1: NashConv of different PSRO runs.

From the results, we can obtain the following observations. (1) For the three meta-solvers, none of them can consistently beat (dominate) all the others in terms of NashConv during the PSRO procedure. For instance, at the early stage of the PSRO procedure, Uniform performs better than the other two meta-solvers in terms of NashConv. However, it only converges to a high NashConv value, which is also observed in previous works [Muller *et al.*, 2020]. (2) By switching from one meta-solver to another during the PSRO procedure, we can achieve better learning performance in terms of NashConv. Moreover, the comparison between the two cases: "α-Rank $\rightarrow$ PRD" and "PRD $\rightarrow$ α-Rank", again verifies the previously observed fact that none of the meta-solvers can consistently beat all the others in terms of NashConv during the PSRO procedure[5].

The above observations motivate us to think about a natural question: How to determine the meta-solvers during the PSRO procedure such that we can obtain better learning performance? Note that the examples in Figure 1 are NFGs. For extensive-form games (EFGs), in addition to the meta-solver, the hyperparameters also include the initialization of a BR policy $\pi^{i,\text{BR}}$ and the number of updates K for training the BR policy. Most of the existing works determine the values of these hyperparameters by hand-crafted tuning, which typically requires extensive domain knowledge.

Thus, a critical problem to be addressed is: *how to automatically determine the optimal hyperparameter values during the PSRO running?* Specifically, at each epoch, we need to i) choose one or multiple meta-solver(s), ii) determine how to initialize the new BR policies, e.g., random initialization, copy from one of the previous BRs, or mix, and iii) determine the number of updates K of the new BR policy of each player. In this work, we make the first attempt to develop a novel framework that can self-adaptively determine the optimal hyperparameter values during the PSRO running.

5 Self-adaptive PSRO

In this section, we establish the Self-adaptive PSRO (SPSRO) through two steps: (1) We parameterize the PSRO algorithm (PPSRO) by introducing several hyperparameters. (2) We cast the hyperparameter value selection of PPSRO as a hyperparameter optimization (HPO) problem where our objective is to learn an HPO policy that will self-adaptively determine the optimal hyperparameter values of PPSRO.

5.1 Parametric PSRO

First, inspired by the observations in the previous section, in this work, instead of considering a single meta-solver as most of the existing works, we use the meta-solver set $\boldsymbol{\mathcal{M}}^{\boldsymbol{\alpha}}$ with m different meta-solvers and associate it with a vector $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_m)$ specifying the weight of each meta-solver. Intuitively, by combining multiple meta-solvers, we could obtain better performance. As $\boldsymbol{\alpha}$ is only dependent on

[4]Note that throughout this work, e always represents the index of PSRO epoch, neither the power of e nor the Euler's number.

[5]These observations are different from [Wang *et al.*, 2022] which regards the strategy exploration and its evaluation as two orthogonal components. For example, one can use PRD to guide the BR policies computation but use α-Rank to compute the meta-distribution for decision-making after the new BR policies of players are added to their respective policy spaces. However, it is worth noting that our observations do not cause inconsistency with [Wang *et al.*, 2022] as we just evaluate PSRO as an online algorithm.

the meta-game payoff tensor M regardless of the underlying games (normal-form or extensive-form games), we refer to the weights in it as *game-free* hyperparameters.

Second, we introduce a parametric BR oracle $\mathcal{O}^i(\boldsymbol{\sigma}; \beta, K)$ where the hyperparameters include[6]: (1) the initialization parameter $\beta \in [0, 1]$ which determines the initialization of the new BR policy of player i by mixing the BR policy obtained in the last epoch with a randomly initialized policy $\pi^{i,\text{random}}$, and (2) the number K which determines the number of updates needed for training the new BR policy. Formally, at each epoch e, we initialize player i's BR policy as $\pi^{i,\text{BR},e} = \beta\pi^{i,\text{BR},e-1} + (1-\beta)\pi^{i,\text{random}}$, and then update this BR policy $\pi^{i,\text{BR},e}$ for K steps. After that, the trained BR policy $\pi^{i,\text{BR},e}$ is added to player i's policy space Π^i. As β and K are highly dependent on the underlying games (e.g., poker, soccer), we refer to them as *game-based* hyperparameters.

By specifying $\boldsymbol{\alpha}$, β and K, we can obtain GDA and various PSRO variants (Table 1). For example, GDA can be instantiated as follows. Suppose that $\mathcal{M}_b$ is the meta-solver "Last-One", then we set $\alpha_b = 1$ and $\alpha_{d\neq b} = 0$, which implies that the meta-strategy of player i is $\sigma_b^i = (\sigma_b^{i,1} = 0, \cdots, \sigma_b^{i,e-1} = 1)$ at epoch e. Then, player i initializes the BR policy $\pi^{i,\text{BR},e}$ with $\beta = 1$ and trains $\pi^{i,\text{BR},e}$ with $K = 1$ update.

Algorithm	$\mathcal{M}^{\boldsymbol{\alpha}}$	$\mathcal{O}^i(\boldsymbol{\sigma}; \beta, K)$
GDA	Last-One	$\beta = 1$ & $K = 1$
InRL	Last-One	$\beta = 1$ & $K = \bar{K}$
PSRO_P	Penultimate	$\beta \in [0,1]$ & $K = \bar{K}$
PSRO_U	Uniform	$\beta \in [0,1]$ & $K = \bar{K}$
PSRO_N	Nash	$\beta \in [0,1]$ & $K = \bar{K}$
PSRO_rN	Rectified Nash	$\beta \in [0,1]$ & $K = \bar{K}$
$\text{PSRO}_{\alpha\text{-Rank}}$	α-Rank	$\beta \in [0,1]$ & $K = \bar{K}$
PSRO_CCE	Coarse Correlated	$\beta \in [0,1]$ & $K = \bar{K}$

Table 1: Specifications of PSRO variants. $\bar{K}$ is the number of updates needed to obtain a converged BR policy. The references for these methods: GDA [Fiez and Ratliff, 2021], InRL [Matignon *et al.*, 2012], PSRO_P [Ho *et al.*, 1998], PSRO_U [Heinrich and Silver, 2016], PSRO_N [Lanctot *et al.*, 2017], PSRO_rN [Balduzzi *et al.*, 2019], $\text{PSRO}_{\alpha\text{-Rank}}$ [Muller *et al.*, 2020], PSRO_CCE [Marris *et al.*, 2021].

5.2 HPO Policy Optimization

With the PPSRO introduced in the previous section, a natural problem is how to determine the values of $\boldsymbol{\alpha}$, β, and K in PPSRO. To address this problem, we propose a novel algorithmic framework, Self-adaptive PSRO (SPSRO), which is shown in Figure 2 and Algorithm 1. In the following, we first present the overall procedure of SPSRO, then define the performance metric of a given selection of hyperparameter values, and finally formalize the hyperparameter optimization (HPO) problem where our objective is to learn an HPO policy that will self-adaptively select the optimal hyperparameter values of PPSRO during game solving.

Let $\tau \in \Gamma$ denote an HPO policy where Γ is the policy space. Let $u^e = (\boldsymbol{\alpha}^e, \beta^e, K^e) \in \mathcal{U}$ denote the hyperparameter values selected according to the HPO policy τ at each epoch $e \geq 1$ of SPSRO, where $\mathcal{U}$ is the admissible set of the hyperparameter values. That is, we have $u^e \sim \tau$. We run one epoch of SPSRO as follows. (1) Compute the payoff tensor M through game simulation (Line 3). (2) Compute the final joint meta-strategy $\boldsymbol{\sigma}^e$ (Line 4). (3) Expand each player's policy space (Line 5). Specifically, for each player i, we initialize the BR policy as $\pi^{i,\text{BR},e} = \beta^e\pi^{i,\text{BR},e-1} + (1-\beta^e)\pi^{i,\text{random}}$, and then train the BR policy $\pi^{i,\text{BR},e}$ for K^e updates and add it to player i's policy space $\Pi^i = \Pi^i \cup \{\pi^{i,\text{BR},e}\}$. (4) Compute the performance metric $y^e(u^e)$ of the current selection (Line 6). (5) Select new hyperparameter values u^{e+1} according to τ (Line 7).

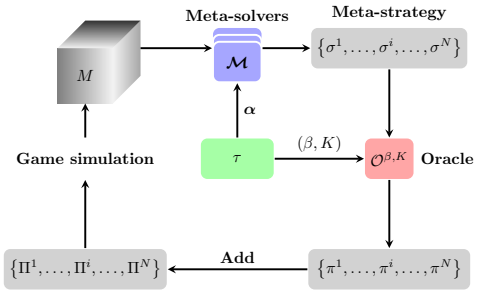


Figure 2: Illustration of Self-adaptive Policy-Space Response Oracles (SPSRO). PSRO and PPSRO are two special cases of SPSRO. Illustration inspired by [Muller *et al.*, 2020].

Algorithm 1 SPSRO

1: Initialize Π^i with random policies, $\forall i \in \mathcal{N}$, $e \leftarrow 1$, select initial hyperparameter values $u^1 = (\boldsymbol{\alpha}^1, \beta^1, K^1)$, $\tau \in \Gamma$
2: **for** epoch $e \in \{1, 2, 3, \ldots\}$ **do**
3: Update payoff tensor M via game simulation
4: Compute $\boldsymbol{\sigma}^e$ using $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\alpha}^e$: $\boldsymbol{\sigma}^e = \sum_{b=1}^m \alpha_b^e \boldsymbol{\sigma}_b^e$
5: Expand policy spaces $\mathcal{O}^i$: $\Pi^i \leftarrow \Pi^i \cup \mathcal{O}^i(\boldsymbol{\sigma}^e; \beta^e, K^e)$
6: Compute the performance metric $y^e(u^e)$
7: Select new hyperparameter values $u^{e+1} \sim \tau$
8: **end for**

Given a selection of the hyperparameter values u^e, we define its performance metric as $y^e(u^e) = \frac{\mathcal{R}(\boldsymbol{\sigma}^e)}{\mathcal{R}(\boldsymbol{\sigma}^1)} + \frac{h^e}{h^1}$. Roughly speaking, it consists of two parts: the NashConv of all players and the BR training effort, which implies that using larger K^e could obtain lower NashConv $\mathcal{R}(\boldsymbol{\sigma}^e)$ but at the cost of longer BR training time h^e, and using smaller K^e could shorten the BR training time while at the cost of higher NashConv $\mathcal{R}(\boldsymbol{\sigma}^e)$.

Given the performance metric, our objective is to learn an HPO policy τ by solving the following HPO problem: $\forall e \geq 1$,

$$\arg\min_{\tau \in \Gamma} y^e(u^e \sim \tau). \tag{1}$$

[6]There could be other hyperparameters such as batch size in the BR oracle. Nevertheless, as the first attempt to explore the possibility of self-adaptively determining the optimal hyperparameter values, we focus on the ones that enable us to unify various PSRO variants. Moreover, it is worth noting using the same type of BR oracle (DQN in this work) with the same configuration for all the other hyperparameters is important to ensure a fair comparison between our approach and baselines. See Appendix A for more discussion.

6 A Novel Offline HPO Algorithm

The most straightforward method to optimize the HPO policy is to employ the classic HPO methods such as Bayesian optimization [Snoek *et al.*, 2012]. However, most HPO methods typically predict hyperparameter values based on online generated data (history of past epochs in the context of our work), and consequently, the performance could be poor. To overcome the limitations of online HPO, we propose a novel offline HPO method to optimize the HPO policy, which possesses the potential to transfer to different games without fine-tuning.

6.1 HPO as Sequence Modeling

As presented in Algorithm 1, selecting the hyperparameter values in SPSRO can be naturally regarded as a sequence modeling problem where we model the probability of the next token x^e conditioned on all prior tokens: $P_\theta(x^e|x^{<e})$, similar to the decoder-only sequence models [Zhao *et al.*, 2023; Touvron *et al.*, 2023]. Specifically, we consider the sequence of hyperparameter values up to e-th epoch:

$$\mathcal{H}^e = (\cdots, \alpha_1^e, \cdots, \alpha_m^e, \beta^e, K^e, y^e). \tag{2}$$

Figure 3 presents the overview of the architecture.

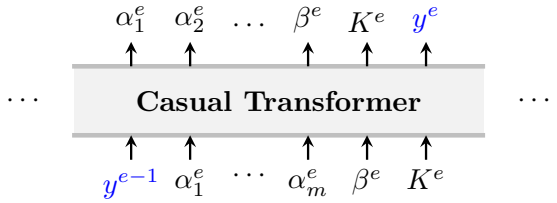


Figure 3: HPO based on Transformer. At each epoch, the Transformer model predicts parameter values in an autoregressive manner using a causal self-attention mask, i.e., each predicted parameter value will be fed into the model to generate the next parameter value.

6.2 Tokenization

We convert each element in Eq. (2) into a single token. The idea is to normalize and discretize each element such that it falls into one of Q bins each with size 1. Q is referred to as the quantization level. Specifically, we have:

$$\bar{x}^e = \text{int}[x^e_{\text{norm}} \cdot Q], \tag{3}$$

where $x^e_{\text{norm}} = (x^e - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}})$. x_{min} and x_{max} are determined by the space $\mathcal{U}$ and the range of y^e is determined by observed values in the offline dataset (introduced in the next subsection) or the underlying games (e.g., normal-form and extensive-form). After tokenization, we have:

$$\bar{\mathcal{H}}^e = (\cdots, \bar{\alpha}_1^e, \cdots, \bar{\alpha}_m^e, \bar{\beta}^e, \bar{K}^e, \bar{y}^e). \tag{4}$$

6.3 Training Dataset

To train the Transformer model θ, one of the critical steps is to generate the offline dataset, which is non-trivial due to the computational complexity of running SPSRO. Specifically, to generate a dataset $\mathcal{D}$ consisting of $|\mathcal{D}|$ sequences of Eq. (2) or Eq. (4), we need to run the SPSRO for $|\mathcal{D}|$ times. However, it is well-known that running PSRO can be computationally difficult in complex games. One of the main difficulties is that solving the meta-game using α-Rank could be NP-hard [Yang *et al.*, 2020] and as it requires enumerating all the joint strategies to construct the response graph [Omidshafiei *et al.*, 2019], it is time-consuming as the progress of the PSRO procedure. To more efficiently generate the training dataset, we use a simple pruning technique to constrain the size of the support set of the meta-strategy when using α-Rank to solve the meta-game in each epoch of the PSRO algorithm.

Let C denote the maximum size of the support set of the meta-strategy. At the first C epochs, we follow the standard PSRO$_{\alpha\text{-Rank}}$. After that, at each epoch $e > C$, we construct the meta-game of size $(C+1) \times (C+1)$ where a new BR policy is obtained by deep RL algorithms such as DQN [Mnih *et al.*, 2015]. Next, we compute the meta-distribution by solving the meta-game through α-Rank. Let π^i_{min} denote the policy with the minimum probability in the meta-distribution. Then, we get the final meta-strategy by setting $\sigma^{i,e}_{\alpha\text{-Rank}}(\pi^i_{\text{min}}) = 0$ and normalizing the resulting distribution.

During the dataset generation, we employ the widely used tool, Optuna [Akiba *et al.*, 2019], to determine the value of u^e at each epoch e. Using the terminology of offline RL [Levine *et al.*, 2020; Chen *et al.*, 2021a], Optuna is a *behavior policy* to generate the offline training dataset (see Appendix B for an example code of Optuna showing how to generate the dataset). Furthermore, we distinguish between normal-form games (NFGs) and extensive-form games (EFGs) when generating the dataset. The primary reason is that the parameters of interest are different. In EFGs, in addition to the weights of different meta-solvers $\boldsymbol{\alpha}$, Eq. (2) also consists of the parameters related to the BR oracle, β and K. Therefore, the transformer model trained on the NFG dataset cannot be directly applied to EFGs. In addition, generating the NFG dataset is relatively easier as the computational difficulty mainly resulted from the meta-game solving using α-Rank, which can be effectively addressed by the previously proposed pruning technique. For the EFG dataset, obtaining the BR policies requires extra computational overhead as the BR policies are typically approximated via deep RL algorithms such as DQN [Mnih *et al.*, 2015].

6.4 Loss Function and Inference

Given the dataset $\mathcal{D}$, we train the Transformer model θ by maximizing the log-likelihood for each sequence $\bar{\mathcal{H}}^{\bar{e}} \sim \mathcal{D}$:

$$\mathcal{L}(\theta; \bar{\mathcal{H}}^{\bar{e}}) = \sum\nolimits_{n=1}^{\bar{e}(m+3)} \log P_\theta(\bar{\mathcal{H}}^n | \bar{\mathcal{H}}^{1:n-1}), \tag{5}$$

where $\bar{e}$ is the maximum number of epochs, $\bar{\mathcal{H}}^n$ is the n-th token in Eq. (4), and $\bar{\mathcal{H}}^{1:n-1}$ is all tokens up to the $(n-1)$-th token in Eq. (4). After training, we can apply the Transformer θ to a given game to predict the value of u^e. Specifically, we reverse the tokenization to obtain the token distribution:

$$p_\theta(x|\cdot) = \frac{Q \cdot P_\theta(\bar{x}|\cdot)}{(x_{\text{max}} - x_{\text{min}})}. \tag{6}$$

Then, we can sample u^e from the model's prior distribution and thus, define the HPO policy as follows:

$$\begin{aligned} \tau(u^e|\mathcal{H}^{e-1}) = &\prod\nolimits_{b=1}^{m} p_\theta(\alpha_b^e|\mathcal{H}^{e-1}, \alpha_1^e, \cdots, \alpha_{b-1}^e) \\ &\times p_\theta(\beta^e|\mathcal{H}^{e-1}, \{\alpha_b^e\}_{1\le b\le m}) \\ &\times p_\theta(K^e|\mathcal{H}^{e-1}, \{\alpha_b^e\}_{1\le b\le m}, \beta^e). \end{aligned} \tag{7}$$

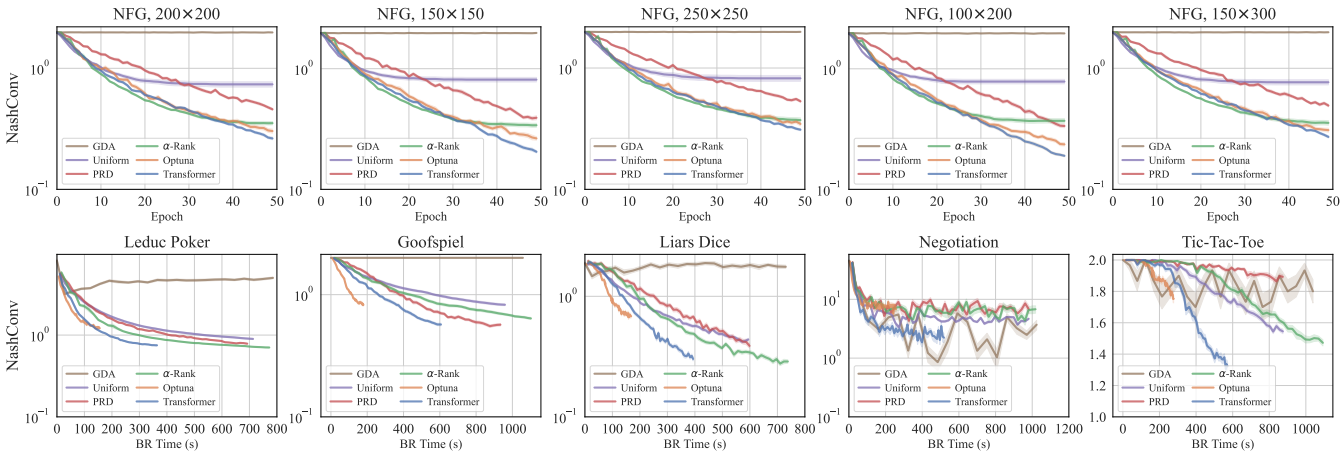


Figure 4: Evaluation performance. The top and bottom rows correspond to NFGs and EFGs, respectively.

That is, at epoch e, τ predicts each parameter value in u^e conditioned on: i) the sequence of past epochs $\mathcal{H}^{e-1}$, and ii) the values of the preceding predicted parameters.

7 Experiments

In this section, we evaluate the effectiveness of SPSRO.

7.1 Experimental Setup

All experiments are performed on a machine with a 24-core 3.2GHz Intel i9-12900K CPU and an NVIDIA RTX 3060 GPU. All the results are averaged over 30 independent runs. More experimental details can be found in Appendix C.

Games. We consider the following games. (1) Normal-form games (NFGs) of size $|\mathcal{A}_1| \times |\mathcal{A}_2|$. The payoff matrices are randomly sampled from the range $[-1, 1]$. The set of size is $\{150 \times 150, 200 \times 200, 250 \times 250, 100 \times 200, 150 \times 300\}$. (2) Extensive-form games (EFGs): Leduc, Goofspiel, Liar's Dice, Negotiation, and Tic-Tac-Toe, which are implemented in OpenSpiel [Lanctot *et al.*, 2019]. Please refer to Section 6.3 for the reasons to distinguish between NFGs and EFGs.

Methods. (1) GDA [Fiez and Ratliff, 2021]. At each epoch, a player only best responds to the opponent's newest BR action (NFGs) or policy (EFGs). (2) Uniform [Heinrich and Silver, 2016]. The meta-distribution is the uniform distribution. (3) PRD [Lanctot *et al.*, 2017; Muller *et al.*, 2020], an approximation of Nash equilibrium. We choose PRD instead of an exact Nash solver as it has been widely adopted in PSRO-related research. (4) α-Rank [Muller *et al.*, 2020]. (5) Optuna [Akiba *et al.*, 2019]. The parameter values are determined by Optuna. (6) Transformer. The parameter values are determined by the Transformer model. Among these methods, (2) to (4) are classic PSRO methods that only involve a single meta-solver, while (5) and (6) are SPSRO methods that involve multiple meta-solvers and use different HPO approaches.

Training and Testing. We generate the training datasets for NFGs and EFGs separately. For NFGs, we generate the dataset on the game of size $|\mathcal{A}_1| \times |\mathcal{A}_2| = 200 \times 200$. For EFGs, we generate the dataset on the Leduc Poker. During testing, in addition to the games used to generate the dataset, we directly apply the trained Transformer model to the other games to verify the zero-shot generalization ability of the model.

7.2 Results

The results are summarized in Figure 4. From the results, we can draw several conclusions as follows.

By combining multiple meta-solvers, we could obtain better performance than using a single meta-solver. In all the NFGs, the final NashConvs of Optuna and Transformer are lower than that of the classic PSRO baselines considering a single meta-solver (Uniform, PRD, or α-Rank). For EFGs, in Negotiation and Tic-Tac-Toe, the final NashConvs of Transformer are lower than the classic PSRO baselines. The results clearly verify the necessity of synergistically integrating multiple meta-solvers during game solving.

Transformer-based HPO could achieve better performance. Transformer-based HPO can learn a better prior distribution of hyperparameter values from offline data, providing a better scheme for weighting multiple meta-solvers, and therefore, achieving better performance than Optuna which is an online method and only relies on past epochs to obtain the prior distribution of hyperparameter values. In addition, in EFGs, we found that the NashConv of Optuna decreases quickly, but converges to a high value (also shown in Figure 5). In contrast, the Transformer can converge to a lower NashConv, though it needs a longer time in terms of BR training.

Transformer has the potential to provide a universal and plug-and-play hyperparameter value selector. As shown in Figure 4, the trained Transformer model can be applied to the games that are different from the training dataset: for NFGs, it can be applied to games with different sizes of action (strategy) spaces, and for EFGs, it can be applied to different games even with different reward scale (e.g., the maximum reward in Goofspiel is 1 while in Negotiation it is 10). This corresponds to the desiderata of a universal and plug-and-play hyperparameter value selector as mentioned in Section 6.

Given a set of meta-solvers, none of them can consistently beat (dominate) all the others during game solving (the observation in Section 4). This can be derived by comparing the performance of the three single-solver-based PSRO base-

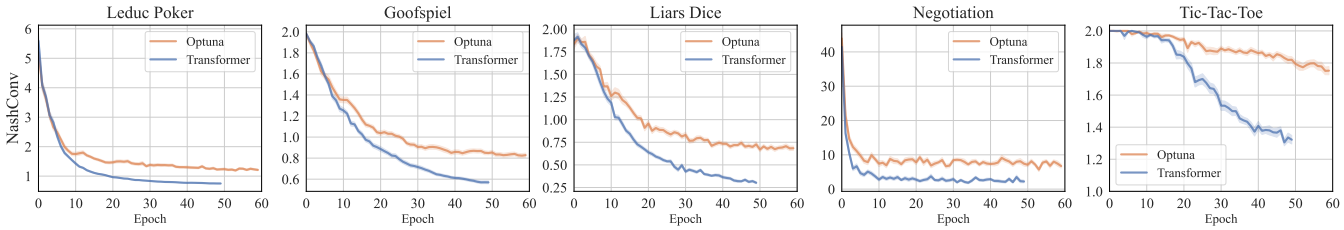


Figure 5: NashConvs of Optuna and Transformer in different extensive-form games.

lines: Uniform, PRD, and α-Rank. For example, consider the NFG of size 200×200. In the early stage of PSRO, the NashConv of Uniform is lower than PRD and α-Rank. In the middle stage, α-Rank quickly surpasses Uniform and PRD, but Uniform still performs better than PRD. However, in the final stage, Uniform is beaten by PRD and α-Rank. Moreover, we note that in the final stage, PRD could also perform better than α-Rank, as shown in the NFG of size 100×200. Similar results are observed in EFGs, further verifying the conclusion.

For EFGs, in Figure 4, at first glance, one may come to the conclusion that Optuna is a better option than Transformer. However, we note that the x-axis in Figure 4 is the BR running time (which is appropriate as in our experiments only one BR policy is added to each player's policy space). To avoid this misleading conclusion, we plot the NashConv versus epoch in Figure 5. The results clearly show that instead of terminating too early, Optuna cannot further decrease the NashConv even if it is given more epochs (10 epochs more than Transformer). The primary reason we hypothesize is that, as the SPSRO progresses, it is a struggle for Optuna to balance the two parts (NashConv and BR training time) in the performance metric y^e. Specifically, at the latter stage of SPSRO, the second term in y^e would dominate the first term, even though they have been normalized by using the values obtained at the first epoch, making Optuna suggest a smaller number of updates for the BR policies (see Figure 7 in Section 7.3), which on the contrary cannot further decrease the NashConv because the quality of the BR policies would be low without providing enough training amount. The results demonstrate that Transformer is more effective in handling such a dilemma.

7.3 More Discussion

In this section and Appendix C, we provide more discussion to further deepen our understanding of our approach.

Figure 6 shows the weights of different meta-solvers (Uniform, PRD, and α-Rank) determined by Optuna and Transformer during SPSRO running. We can see that the weights determined by Optuna vary dramatically throughout SPSRO running, while Transformer's predictions are more stable (around $1/3$ for each solver). We hypothesize that such a relatively stable weighting scheme for multiple meta-solvers is necessary to obtain better performance. In addition, an interesting observation is that the weights of Uniform and PRD change almost in a mirror form. More results can be found in Appendix C.

In Figure 7, we found that Optuna and Transformer follow very different patterns to select the computing amount used for training the BR policy at each epoch. For Optuna, the number of episodes suddenly decreases to a very low value after

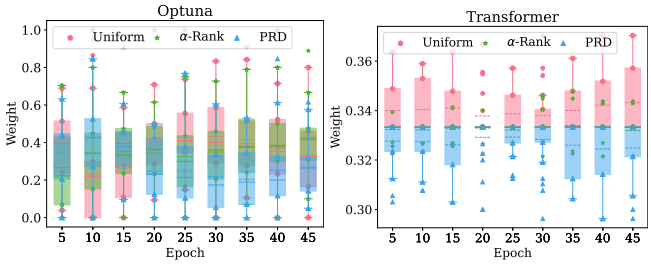


Figure 6: Weights of three meta-solvers determined by Optuna and Transformer in the NFG of size 200×200.

about 10 epochs. Intuitively, when K is much smaller than $\bar{K}$ (the maximum number of episodes to obtain a converged BR policy), the policy obtained through the BR oracle may be far away from the true BR policy, resulting in poor performance. This is also reflected in Figure 5 where Optuna cannot obtain a lower NashConv even if it is given more epochs. In contrast, by offline learning, Transformer could better trade-off between the NashConv and BR training time and hence, performs better than Optuna. More results can be found in Appendix C.

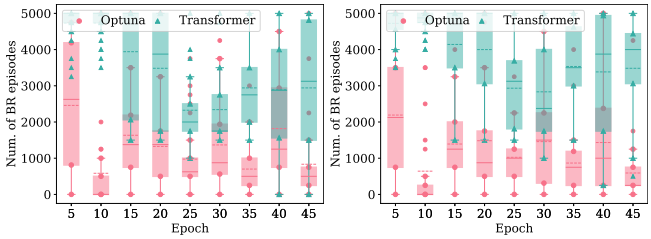


Figure 7: Numbers of BR training episodes determined by Optuna and Transformer in EFGs. (Left) Leduc. (Right) Goofspiel.

8 Conclusions

In this work, we first attempt to explore the possibility of self-adaptively determining the optimal hyperparameter values in the PSRO framework and provide three contributions: (1) the parametric PSRO (PPSRO) which unifies GDA and various PSRO variants; (2) the self-adaptive PSRO (SPSRO) where we aim to learn a self-adaptive HPO policy; (3) a novel offline HPO approach to optimize the HPO policy based on the Transformer architecture. The well-trained Transformer-based HPO policy has the potential of transferring to different games without fine-tuning. Experiments on different games demonstrate the superiority of our approach over different baselines.

References

[Akiba *et al.*, 2019] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, pages 2623–2631, 2019.

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Balduzzi *et al.*, 2019] David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *ICML*, pages 434–443, 2019.

[Bardenet *et al.*, 2013] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *ICML*, pages 199–207, 2013.

[Beltagy *et al.*, 2020] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[Brown, 1951] George W Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374, 1951.

[Charton, 2021] François Charton. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898*, 2021.

[Chen *et al.*, 2017] 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 *ICML*, pages 748–756, 2017.

[Chen *et al.*, 2021a] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, pages 15084–15097, 2021.

[Chen *et al.*, 2021b] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

[Chen *et al.*, 2022] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, MarcAurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In *NeurIPS*, pages 32053–32068, 2022.

[Duan *et al.*, 2016] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[Feng *et al.*, 2021] Xidong Feng, Oliver Slumbers, Ziyu Wan, Bo Liu, Stephen McAleer, Ying Wen, Jun Wang, and Yaodong Yang. Neural auto-curricula in two-player zero-sum games. In *NeurIPS*, pages 3504–3517, 2021.

[Feurer *et al.*, 2015] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, 2015.

[Fiez and Ratliff, 2021] Tanner Fiez and Lillian J Ratliff. Local convergence analysis of gradient descent ascent with finite timescale separation. In *ICLR*, 2021.

[Heinrich and Silver, 2016] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[Ho *et al.*, 1998] Teck-Hua Ho, Colin Camerer, and Keith Weigelt. Iterated dominance and iterated best response in experimental "p-beauty contests". *The American Economic Review*, 88(4):947–969, 1998.

[Jain *et al.*, 2013] Manish Jain, Vincent Conitzer, and Milind Tambe. Security scheduling for real-world networks. In *AAMAS*, pages 215–222, 2013.

[Janner *et al.*, 2021] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, pages 1273–1286, 2021.

[Jordan *et al.*, 2010] Patrick R Jordan, L Julian Schvartzman, and Michael P Wellman. Strategy exploration in empirical games. In *AAMAS*, pages 1131–1138, 2010.

[Krause and Ong, 2011] Andreas Krause and Cheng Ong. Contextual Gaussian process bandit optimization. In *NeurIPS*, pages 2447–2455, 2011.

[Lample and Charton, 2019] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *ICLR*, 2019.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pages 4193–4206, 2017.

[Lanctot *et al.*, 2019] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo

Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

[Lee *et al.*, 2022] Kuang-Huei Lee, Ofir Nachum, Sherry Yang, Lisa Lee, C. Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-game decision transformers. In *NeurIPS*, pages 27921–27936, 2022.

[Levine *et al.*, 2020] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[Li *et al.*, 2022] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

[Liu *et al.*, 2021a] Xiangyu Liu, Hangtian Jia, Ying Wen, Yujing Hu, Yingfeng Chen, Changjie Fan, Zhipeng Hu, and Yaodong Yang. Towards unifying behavioral and response diversity for open-ended learning in zero-sum games. In *NeurIPS*, pages 941–952, 2021.

[Liu *et al.*, 2021b] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pages 10012–10022, 2021.

[Marris *et al.*, 2021] Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In *ICML*, pages 7480–7491, 2021.

[Matignon *et al.*, 2012] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[McAleer *et al.*, 2020] Stephen McAleer, John B Lanier, Roy Fox, and Pierre Baldi. Pipeline PSRO: A scalable approach for finding approximate Nash equilibria in large games. In *NeurIPS*, pages 20238–20248, 2020.

[McMahan *et al.*, 2003] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML*, pages 536–543, 2003.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Muller *et al.*, 2020] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. A generalized training approach for multiagent learning. In *ICLR*, 2020.

[Muller *et al.*, 2022a] Paul Muller, Mark Rowland, Romuald Elie, Georgios Piliouras, Julien Perolat, Mathieu Lauriere, Raphael Marinier, Olivier Pietquin, and Karl Tuyls. Learning equilibria in mean-field games: Introducing mean-field PSRO. In *AAMAS*, pages 926–934, 2022.

[Müller *et al.*, 2022b] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. In *ICLR*, 2022.

[Omidshafiei *et al.*, 2019] Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific Reports*, 9(1):9937, 2019.

[Perez-Nieves *et al.*, 2021] Nicolas Perez-Nieves, Yaodong Yang, Oliver Slumbers, David H Mguni, Ying Wen, and Jun Wang. Modelling behavioural diversity for learning in open-ended games. In *ICML*, pages 8514–8524, 2021.

[Robinson, 1951] Julia Robinson. An iterative method of solving a game. *Annals of Mathematics*, pages 296–301, 1951.

[Rothfuss *et al.*, 2021] Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. PACOH: Bayes-optimal meta-learning with PAC-guarantees. In *ICML*, pages 9116–9126, 2021.

[Schvartzman and Wellman, 2009a] L Julian Schvartzman and Michael P Wellman. Exploring large strategy spaces in empirical game modeling. *Agent Mediated Electronic Commerce (AMEC 2009)*, page 139, 2009.

[Schvartzman and Wellman, 2009b] L Julian Schvartzman and Michael P Wellman. Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. In *AAMAS*, pages 249–256, 2009.

[Smith *et al.*, 2021] Max Smith, Thomas Anthony, and Michael Wellman. Iterative empirical game solving via single policy best response. In *ICLR*, 2021.

[Snoek *et al.*, 2012] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *NeurIPS*, pages 2951–2959, 2012.

[Swersky *et al.*, 2013] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *NeurIPS*, pages 2004–2012, 2013.

[Touvron *et al.*, 2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[Vaswani *et al.*, 2017] 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 *NeurIPS*, pages 5998–6008, 2017.

[Volpp *et al.*, 2019] Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. *arXiv preprint arXiv:1904.02642*, 2019.

[Wang *et al.*, 2016] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[Wang *et al.*, 2019] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *AAAI*, pages 1401–1408, 2019.

[Wang *et al.*, 2022] Yongzhao Wang, Qiurui Ma, and Michael P Wellman. Evaluating strategy exploration in empirical game-theoretic analysis. In *AAMAS*, pages 1346–1354, 2022.

[Wellman, 2006] Michael P Wellman. Methods for empirical game-theoretic analysis. In *AAAI*, volume 980, pages 1552–1556, 2006.

[Wen *et al.*, 2022] Muning Wen, Jakub Grudzien Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. Multi-agent reinforcement learning is a sequence modeling problem. *arXiv preprint arXiv:2205.14953*, 2022.

[Wistuba and Grabocka, 2020] Martin Wistuba and Josif Grabocka. Few-shot Bayesian optimization with deep kernel surrogates. In *ICLR*, 2020.

[Xu *et al.*, 2022] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *ICML*, pages 24631–24645, 2022.

[Yang *et al.*, 2020] Yaodong Yang, Rasul Tutunov, Phu Sakulwongtana, and Haitham Bou Ammar. α^{α}-Rank: Practically scaling α-Rank through stochastic optimisation. In *AAMAS*, pages 1575–1583, 2020.

[Yuan *et al.*, 2021] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token ViT: Training vision transformers from scratch on ImageNet. In *ICCV*, pages 558–567, 2021.

[Zahavy *et al.*, 2020] Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. A self-tuning actor-critic algorithm. In *NeurIPS*, volume 33, pages 20913–20924, 2020.

[Zhao *et al.*, 2021] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *ICCV*, pages 16259–16268, 2021.

[Zhao *et al.*, 2023] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[Zheng *et al.*, 2022] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *ICML*, pages 27042–27059, 2022.

A Frequently Asked Questions

Q1. Why self-adaptive HPO policy is necessary in PSRO?

PSRO [Lanctot *et al.*, 2017] since pioneered has been the main algorithm for solving various games. However, determining the hyperparameter values in PSRO could be difficult in complex games. On one hand, at each epoch, it needs to solve the meta-game to obtain the meta-strategies of all players, which could be computationally difficult, e.g., solving α-Rank may be NP-hard [Yang *et al.*, 2020]. On the other hand, at each epoch, it needs to compute the BRs of all players, which could be time-consuming because, in complex games such as poker and soccer, the BRs are often approximated by deep RL algorithms such as DQN [Mnih *et al.*, 2015] and obtaining good approximated BRs will require a large number of updates. Moreover, existing works have suggested various meta-solvers such as Uniform [Heinrich and Silver, 2016], Nash [Lanctot *et al.*, 2017], α-Rank [Muller *et al.*, 2020], correlated equilibrium [Marris *et al.*, 2021], and neural meta-solver [Feng *et al.*, 2021]. However, there is no explicit rule of which one is the best in terms of learning performance.

Therefore, there is a challenging dilemma: (1) we must determine these hyperparameter values before running PSRO, and (2) evaluating a particular setting is hard because the performance is unknown until the end of the PSRO running due to its nature of iterative policy-space expansion. Thus, instantiating and running PSRO typically involve extensive domain knowledge, rendering it unfriendly to researchers, especially those who are not familiar with game theory.

To further boost PSRO-based research and make it more applicable to real-world problems, a natural idea is to establish a hyperparameter value selector that can be employed to self-adaptively determine the optimal hyperparameter values for different games (even if the games are very different from each other such as normal-form games versus extensive-form games) with or without fine-tuning, which is exactly the motivation of this work. Ideally, such a self-adaptive hyperparameter value selector can help reduce the effort required for researchers to conduct the costly hyperparameter tuning when applying PSRO to different games, and hence, we believe it will benefit the game community.

Q2. More explanations on mixing multiple meta-solvers.

As PSRO is a general algorithmic framework, any meta-solver can be used to compute the meta-distribution, though different meta-solvers are designed to optimize toward different solution concepts. However, it is worth noting that the objective that PSRO achieves is primarily determined by the evaluation metric. For example, in [Muller *et al.*, 2020], even if α-Rank is chosen as the meta-solver, one can still evaluate the convergence of PSRO to Nash equilibrium by computing the NashConv, the metric to measure the distance between the current policy to Nash equilibrium. In this sense, we note that the observations in Section 4 hold in terms of NashConv, which supports our motivation of combining multiple meta-solvers during game solving using PSRO. Moreover, as mentioned in Section 4, our observations do not cause inconsistency with [Wang *et al.*, 2022] as we just evaluate PSRO as an online algorithm. Finally, we note that combining multiple meta-solvers or changing from one meta-solver to another has been explored and demonstrated better performance in PSRO [Feng *et al.*, 2021] and double oracle (DO) [Wang *et al.*, 2019], which provides support to our motivation.

Q3. Discussion on the hyperparameters for the BR oracle.

In this work, we consider two hyperparameters in the BR oracle: the initialization parameter β and the number of BR policy updates K. Here, we provide more explanations on the parameter K. (1) PSRO is a deep learning extension of double oracle (DO), where RL is adopted to compute the best response. However, RL is not guaranteed to compute the exact best response [Lanctot *et al.*, 2017], a more suitable notion is "better response", i.e., a new response better than the existing responses, which is also considered in DO [Jain *et al.*, 2013]. In this sense, K is a hyperparameter determining the amount of computing used to learn the better response. (2) Although different methods can be adopted to learn the BR policy, e.g., DQN and PPO, it is essential to use the same method (DQN in this work) to ensure a fair comparison when comparing different PSRO methods. In other words, the comparison between PSRO_N-DQN and PSRO_N-PPO would not induce any valid argument *on the parameter* K. Furthermore, even for the same BR learning method, keeping the same configuration for all the other hyperparameters (e.g., batch size) is also indispensable. (3) As this is the first attempt to explore the possibility of self-adaptively determining the optimal hyperparameter values in the PSRO framework, we only focus on some of the hyperparameters. As for self-adaptively determining more hyperparameters such as learning rate and batch size, more techniques may be required [Chen *et al.*, 2022; Zahavy *et al.*, 2020] and we leave for future works.

Q4. More explanations on the NashConv values.

In some EFGs, the NashConv values seem relatively high, e.g., in Tic-Tac-Toe. Nevertheless, we note that this does not violate the goal achieved by our framework: achieve a better balance between minimizing the NashConv of all players and the BR training effort. In this sense, Transformer performs better than Optuna (as well as all other baselines). Moreover, though not fully converged, from Figure 5 and Figure 6 in the main text we can see that, as the learning process progresses, the NashConv of Transformer shows a continuously and relatively quickly decreasing trend, while the NashConv of Optuna decreases much more slowly.

B More Related Works

Transformers [Vaswani *et al.*, 2017] have demonstrated outstanding performance in various real-world tasks including natural language processing (NLP) [Vaswani *et al.*, 2017; Beltagy *et al.*, 2020], computer vision (CV) [Liu *et al.*, 2021b; Yuan *et al.*, 2021], and 3D point cloud [Zhao *et al.*, 2021]. Transformers have also shown the ability of symbolic manipulation [Lample and Charton, 2019; Chen *et al.*, 2021b; Li *et al.*, 2022], numerical manipulation [Charton, 2021; Müller *et al.*, 2022b], or both [Chen *et al.*, 2022]. Recently, Transformers have also been applied to offline RL and achieved state-of-the-art performance in various RL benchmarks [Chen *et al.*, 2021a; Janner *et al.*, 2021; Zheng *et al.*, 2022; Xu *et al.*, 2022; Lee *et al.*, 2022; Wen *et al.*, 2022]. In this work, it is natural to model the HPO policy learning as a sequence modeling

problem and thus, Transformers can be used to predict hyperparameter values. Intuitively, we aim to learn an HPO policy that self-adaptively determines *a sequence of choices of the hyperparameter values* to optimize the objective, which is similar to the RL problem where an agent tries to maximize his return through *a sequence of actions*. Therefore, we use the Decision Transformer [Chen *et al.*, 2021a] to predict the hyperparameter values in SPSRO.

Our work is also related to empirical game theoretic analysis (EGTA), an empirical methodology that bridges the gap between game theory and simulation for practical strategic reasoning [Wellman, 2006]. In EGTA, an empirical game is estimated via simulation over the combinations of a set of strategies and then can be analyzed with standard methods. In iterative approaches to EGTA, the game model is extended by iteratively generating and adding new strategies to the strategy spaces of players. One of the most challenging problems in EGTA is the strategy exploration problem (how to direct the iterative strategy generation process to construct effective game models with minimal iteration) and a set of works have been done toward investigating this problem [Schvartzman and Wellman, 2009b; Schvartzman and Wellman, 2009a; Jordan *et al.*, 2010; Wang *et al.*, 2022]. In [Schvartzman and Wellman, 2009b], tabular RL has been adopted to serve as the BR oracle to generate new strategies. Investigation of strategy exploration was advanced significantly by the introduction of PSRO [Lanctot *et al.*, 2017] which is a flexible framework for iterative EGTA, where at each epoch, new strategies are generated through deep reinforcement learning (DRL) algorithm such as DQN [Mnih *et al.*, 2015].

C More Experimental Details

In this section, we provide more experimental details.

Dataset. We separately generate the datasets for normal-form games and extensive-form games. For the normal-form game, we generate the dataset on the game of size $|\mathcal{A}_1| \times |\mathcal{A}_2| = 200 \times 200$, and the dataset contains 1000 sequences with the form of Eq. (4) with $\bar{e} = 50$. For the extensive-form game, we generate the dataset on the Leduc Poker, and the dataset consists of a total of 1359 sequences with the form of Eq. (4) with $\bar{e} = 50$. During dataset generation, we use Optuna [Akiba *et al.*, 2019] as the *behavior policy*. Using the terminology of offline RL [Levine *et al.*, 2020; Chen *et al.*, 2021a; Lee *et al.*, 2022], each sequence is a *trajectory* recording the process of SPSRO under the behavior policy. To more clearly show how to generate the dataset, we present an example code in Figure 9. At each epoch e, Optuna first suggests new values for the hyperparameters (Line 6–10), including the weights of multiple meta-solvers and the hyperparameters related to the BR oracle, then runs one epoch of PSRO and computes the metric y^e (Line 13–15). After $\bar{e}$ epochs, Optuna returns the sequence of Eq. (4) (Line 17–20). This process will be repeated for $|\mathcal{D}|$ times to generate the training dataset.

Model Architecture. We utilize Decision Transformer [Chen *et al.*, 2021a] as our underlying model architecture. Specifically, 4 linear layers are used to project the weights of the three meta-solvers (Uniform, PRD, and α-Rank) $\boldsymbol{\alpha}^e$, the initialization parameter value β^e, the number of BR episodes K^e, and the objective y^e to the token embeddings which will be fed into a Transformer block followed by layer normalization [Ba *et al.*, 2016]. The outputs are then mapped by a linear layer to the logits which will be used to compute the loss during training or infer the next token during inference. Moreover, an embedding for each epoch is learned and added to each token embedding. More details can be found in [Chen *et al.*, 2021a] and the corresponding codebase[7].

```
1  import optuna
2  import ···
3
4  def objective(trial):
5      # suggest new values of the parameters
6      ᾱ_1 = trial.suggest_int    # weight of meta-solver 1
7      ···
8      ᾱ_m = trial.suggest_int    # weight of meta-solver m
9      β̄ = trial.suggest_int     # initialization of BR
10     K̄ = trial.suggest_int     # number of updates of BR
11
12     # run one epoch of PSRO, compute the objective
13     α,β,K = reverse_tokenization(ᾱ,β̄,K̄)
14     y = PSRO(α,β,K)
15     return y
16
17 study = optuna.create_study(direction="minimization")
18 study.optimize(lambda trial: objective(trial), n_trials=ē)
19
20 H̄^ē = study.trials_dataframe()  # one sequence of Eq.(4)
```

Figure 9: An example code for generating one sequence of Eq. (4).

Hyperparameters. The hyperparameters are provided in Table 2 where those of the Decision Transformer are similar to [Chen *et al.*, 2021a]. We give some remarks on some of the hyperparameters as follows. (1) The *context length* used in the Transformer context corresponds exactly to the *maximum number of epochs* of SPSRO. (2) The *return-to-go* is simply set to 0 and the operation of decreasing return-to-go with reward is removed. In other words, the objective value at each intermediate epoch is only used to serve as a signal to guide the selection of new hyperparameter values for the next epoch. (3) In this work, we consider three meta-solvers: Uniform, PRD, and α-Rank ($m = 3$), the most commonly used ones in most of the existing works. As for considering more meta-solvers such as correlated equilibrium and neural meta-solver, we leave it for future works.

More Experimental Results. For NFGs $\{150 \times 150, 250 \times 250, 100 \times 200, 150 \times 300\}$, the weights of the three meta-solvers (Uniform, PRD, and α-Rank) determined by Optuna and Transformer are shown in Figure 10 to Figure 13, respectively. For Leduc Poker, Goofspiel, Liar's Dice, Negotiation, and Tic-Tac-Toe, the weights of the three meta-solvers determined by Optuna and Transformer are shown in Figure 14 to Figure 18, respectively. For Liar's Dice, Negotiation, and Tic-Tac-Toe, the numbers of BR training episodes determined by Optuna and Transformer are shown in Figure 19.

[7] https://github.com/kzl/decision-transformer

Hyperparameter	Value
Number of Transformer blocks	2
Number of attention heads	4
Embedding dimension	128
Batch size	64
Context length (i.e., $\bar{e}$)	50
Return-to-go	0
Nonlinearity	GeLU
Transformer training epochs	50
Dropout	0.1
Learning rate	$3 * 10^{-4}$
Adam betas	$(0.9, 0.95)$
Grad norm clip	1.0
Weight decay	0.1
Warmup tokens	5000
Final tokens	NFG: $2 * 50 * 1000 * 4$ EFG: $2 * 50 * 1359 * 6$
Quantization level Q	20
Number of meta-solvers m	3
Size of the support set C	10
Max number of BR episodes $\bar{K}$	5000

Table 2: Hyperparameters.

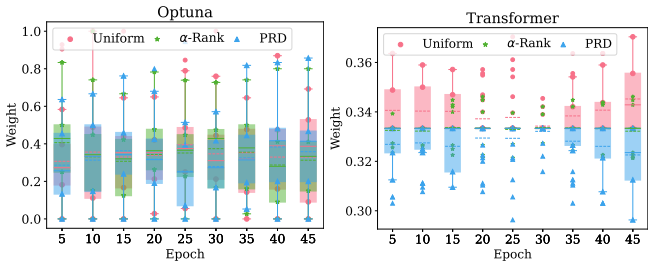


Figure 10: Weights of meta-solvers in **NFG 150×150**.

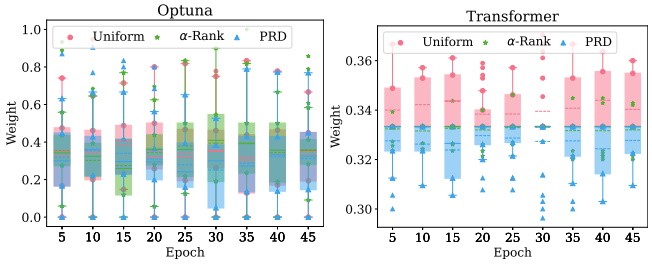


Figure 11: Weights of meta-solvers in **NFG 250×250**.

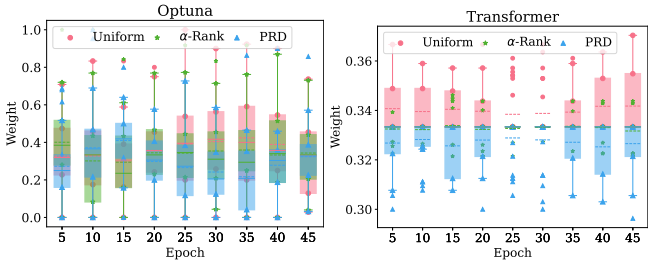


Figure 12: Weights of meta-solvers in **NFG 100×200**.

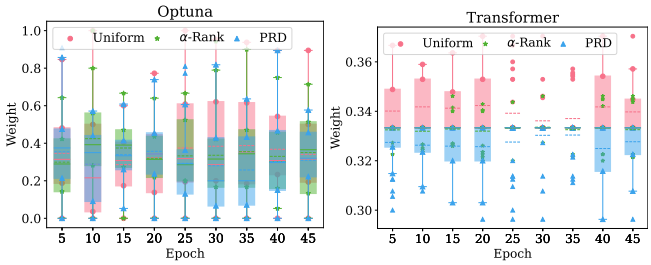


Figure 13: Weights of meta-solvers in **NFG 150×300**.

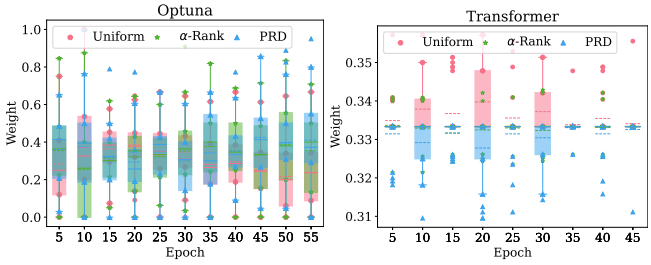


Figure 14: Weights of meta-solvers in **Leduc Poker**.

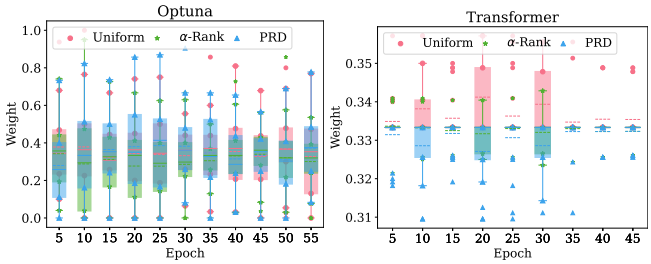


Figure 15: Weights of meta-solvers in **Goofspiel**.

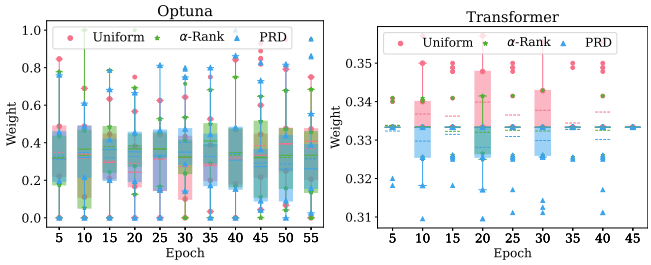


Figure 16: Weights of meta-solvers in **Liar's Dice**.

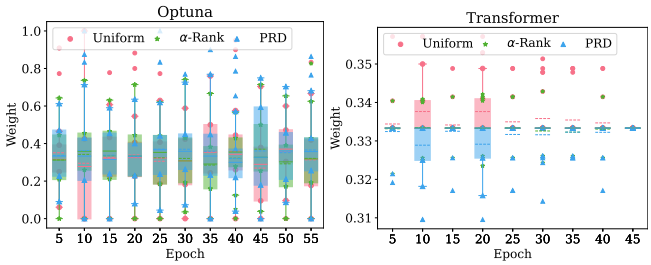


Figure 17: Weights of meta-solvers in **Negotiation**.

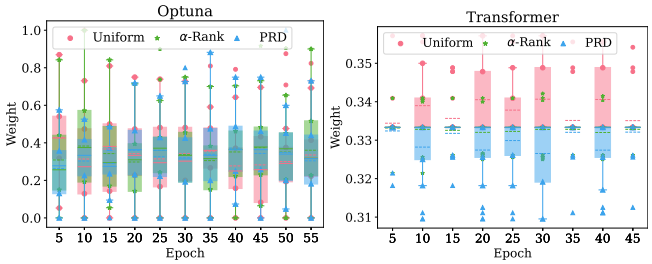
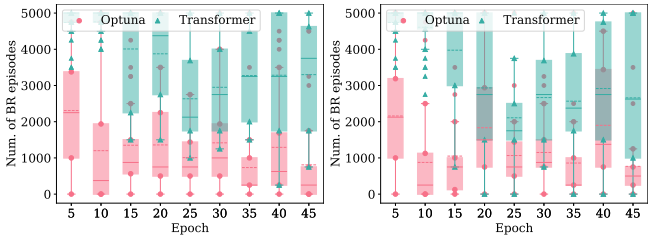
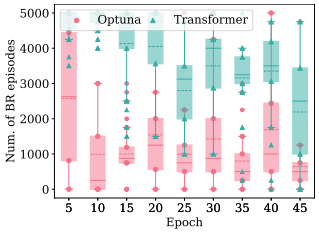


Figure 18: Weights of meta-solvers in **Tic-Tac-Toe**.



(a) Liar's Dice (b) Negotiation



(c) Tic-Tac-Toe

Figure 19: Numbers of BR training episodes determined by Optuna and Transformer in **Liar's Dice**, **Negotiation**, and **Tic-Tac-Toe**. The boxes are obtained by counting 30 independent runs.