










ML-PTA: A Two-Stage ML-Enhanced Framework for Accelerating Nonlinear DC Circuit Simulation With Pseudo-Transient Analysis

Zhou Jin , *Member, IEEE*, Wenhao Li , *Graduate Student Member, IEEE*, Haojie Pei , Xiaru Zha ,
Yichao Dong , *Graduate Student Member, IEEE*, Xiang Jin , Xiao Wu , Dan Niu , *Member, IEEE*,
and Wei W. Xing 

Abstract—Direct current (DC) analysis lies at the heart of integrated circuit design in seeking DC operating points. Although pseudo-transient analysis (PTA) methods have been widely used in DC analysis in both industry and academia, their initial parameters and stepping strategy require expert knowledge and labor tuning to deliver efficient performance, which hinders their further applications. In this paper, we leverage the latest advancements in machine learning to deploy PTA with more efficient setups for different problems. More specifically, active learning, which automatically draws knowledge from other circuits, is used to provide suitable initial parameters for PTA solver, and then calibrate on-the-fly to further accelerate the simulation process using TD3-based reinforcement learning (RL). To expedite model convergence, we introduce dual agents and a public sampling buffer in our RL method to enhance sample utilization. To further improve the learning efficiency of the RL agent, we incorporate imitation learning to improve reward function and introduce supervised learning to provide a better dual-agent rotation strategy. We make the proposed algorithm a general out-of-the-box SPICE-like solver and assess it on a variety of circuits, demonstrating up to $3.10\times$ reduction in NR iterations for the initial stage and $285.71\times$ for the RL stage.

Index Terms—Circuit simulation, DC analysis, pseudo transient analysis, reinforcement learning, Gaussian process, active learning.

Received 18 May 2024; revised 29 March 2025; accepted 30 June 2025. Date of publication 10 July 2025; date of current version 10 September 2025. This work was supported by NSFC under Grant 62204265, Grant 62234010, Grant 92473107, and Grant 62374031; and by NSF of Jiangsu Province under Grant BK20240173. (*Corresponding authors: Dan Niu; Wei W. Xing.*)

Zhou Jin is with Zhejiang University, Hangzhou 311200, China (e-mail: z.jin@zju.edu.cn).

Wenhao Li and Xiaru Zha are with the College of Information Science and Engineering, China University of Petroleum (Beijing), Beijing 102249, China (e-mail: 2022216030@student.cup.edu.cn; 2019011712@student.cup.edu.cn).

Haojie Pei and Xiao Wu are with Huada Emphyrean Software Company Ltd., Beijing 100102, China (e-mail: peihj@emphyrean.com.cn; wuxiao@emphyrean.com.cn).

Yichao Dong and Dan Niu are with the School of Automation, Southeast University, Nanjing 210096, China (e-mail: 230238503@seu.edu.cn; 101011786@seu.edu.cn).

Xiang Jin is with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing 100191, China (e-mail: ZY2141113@buaa.edu.cn).

Wei W. Xing is with the University of Sheffield, Sheffield S10 2TN, UK (e-mail: wxing@buaa.edu.cn).

Digital Object Identifier 10.1109/TC.2025.3587470

I. INTRODUCTION

DIRECT current (DC) analysis, which finds DC operating points, is a crucial step in the integrated circuits design and verification. It is carried out before any other analysis in SPICE-like transistor-level circuit simulators [1], [2], providing an initial solution for transient analysis and determining small signal model parameters for nonlinear devices in AC analysis. Throughout the procedure, the primary challenge involves solving a set of nonlinear algebraic equations derived from the modified nodal analysis. In DC analysis, numerous numerical iterative algorithms have been researched to solve nonlinear algebraic equations, such as the Newton-Raphson (NR) method and continuation methods like Gmin stepping, source stepping, homotopy methods and others. These methods face challenges in achieving widespread adoption due to a series of reasons, including excessive dependence on a given initial point, poor convergence when solving highly nonlinear systems, or convergence relying too much on device models.

As an alternative, PTA [3] and its variants, such as pure PTA [4], Damped PTA (DPTA) [5], Ramping PTA (RPTA) [6], and Compound Element PTA (CEPTA) [7], have been proven to be the most practical solvers in the industry because they are easy to implement and strong ability to deal with discontinuity issues [8]. By introducing pseudo-components into the circuit, PTA methods transform the problem into solving the steady-state problem of ordinary differential equations (ODE) [9]. The difficulty of solving the resulting ODE system depends strongly on the pseudo-elements employed. Unfortunately, there is no universally applicable guideline for selecting the appropriate pseudo-elements for a given circuit, PTA solvers continue to suffer from low efficiency. Once the PTA solver generates an ODE system, it is subsequently solved iteratively using numerical integration techniques to approach the steady state. The stepping strategy in PTA determines the number of nonlinear equations being solved at discrete time points, which involves time-consuming NR iterations and resource-intensive computations. [10] presents a novel approach that uses supervised learning to select the time step by transforming the time step selection problem into a regression prediction problem. However, supervised learning methods cannot self-optimize in a

dynamically changing environment and fail to adapt well to the PTA process. Therefore, selecting the appropriate time step remains a challenging problem. Despite rapid advancements in PTA methods for DC analysis, these two challenges persist as the primary obstacles to their widespread application.

In this paper, we employ cutting-edge machine learning techniques to address both challenges simultaneously with a two stage-framework. In the first stage, we employ an offline active learning approach combined with online prediction to furnish the solver with refined initial parameters, thereby creating an optimal environment for our reinforcement learning (RL) [11], [12] setup. In the second stage, we present a RL scheme utilizing twin delayed deep deterministic (TD3) policy gradient, actor-critic agent [13], to expedite the PTA iterative process. We train the TD3 agent through interactions with the simulation process. Furthermore, we introduce Generative Adversarial Imitation Learning [14] (GAIL) to utilize historical stepping data for optimizing the agent's update strategy, avoiding the shortcomings associated with manually designed reward functions. Additionally, to minimize rejected PTA steps, we incorporate supervised learning to infer which agents should be activated at each subsequent time-step. This enables us to dynamically adjust the forward and backward time-step size, facilitating accelerated convergence to the final steady state.

The novelties of this work are as follows.

(1) To the best of our knowledge, our work represents the first application of active learning, reinforcement learning, imitation learning and supervised learning techniques together to enhance the PTA solver, resulting in improved time-stepping efficiency during numerical iterations and achieving a significant improvement in simulation performance compared to the state-of-the-art (SOTA) PTA solvers.

(2) We provide an effective initial parameters prediction model for any unseen circuit based on a limited number of training circuits. This is done efficiently through a novel self-training active learning procedure by extending the classic Bayesian optimization.

(3) Along with the TD3-based RL framework and a reward function optimized with imitation learning, our method provides a better time-step size to improve simulation performance. Imitation learning utilizes past converged samples to correct the real-time weight update direction of neural networks in RL, thereby avoiding the bias introduced by manually setting reward functions.

(4) Supervised learning is introduced to adaptively select between the forward TD3 RL agent with increasing time-step size and the rollback agent with reduced time-step size at each simulation time-point. This strategy can reduce the occurrence of time point rollbacks, thereby further enhancing simulation efficiency.

(5) The proposed two-stage framework has been implemented in an out-of-the-box SPICE-like simulator and is verified by extensive circuit simulations. Significant acceleration is achieved, with an average reduction of $2.19\times$ in NR iterations for the initial stage and $36.09\times$ for the second stage demonstrated on practical CMOS and BJT circuits.

II. BACKGROUND

A. Gaussian Process

The gaussian process (GP) [15] is a common choice as a surrogate model for building the input-output mapping for complex computer codes (e.g. the surrogate model of Bayesian optimization (BO)) due to its model flexibility and uncertainty quantification. For the sake of clarity, let us consider a case where the circuit is fixed and its index ξ is thus omitted. Assume that we have observation $y_i = \eta(\mathbf{x}_i) + \varepsilon$ and design points \mathbf{x}_i , $i = 1, \dots, N$, where y is the (determined) iteration numbers needed for convergence. In a GP model we place a prior distribution over $\eta(\mathbf{x})$ indexed by \mathbf{x} :

$$\eta(\mathbf{x})|\boldsymbol{\theta} \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})). \quad (1)$$

The covariance function can take many forms, the most common is the automatic relevance determinant (ARD) kernel. For any fixed \mathbf{x} , $\eta(\mathbf{x})$ is a random variable. A collection of values $\eta(\mathbf{x}_i)$, $i = 1, \dots, N$, on the other hand, is a partial realization of the GP. Realizations of the GP are deterministic functions of \mathbf{x} . The main property of GPs is that the joint distribution of $\eta(\mathbf{x}_i)$, $i = 1, \dots, N$, is multivariate Gaussian. Assuming the model inadequacy $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is also a Gaussian, with the prior and available data $\mathbf{y} = (y_1, \dots, y_N)^T$, we can derive the model likelihood

$$(\mathbf{y}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} - \ln|\mathbf{K} + \sigma^2\mathbf{I}| - \log(2\pi))/2, \quad (2)$$

where the covariance matrix $\mathbf{K} = [K_{ij}]$, in which $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, \dots, N$. The hyperparameters $\boldsymbol{\theta}$ are normally obtained from point estimates [16] by maximum likelihood estimate (MLE) of w.r.t. $\boldsymbol{\theta}$. The joint distribution of \mathbf{y} and $\eta(\mathbf{x})$ also form a joint Gaussian distribution. Conditioning on \mathbf{y} provides the conditional Gaussian distribution at \mathbf{x} [16] with mean and variance.

$$\begin{aligned} \hat{\eta}(\mathbf{x})|\mathbf{y}, \boldsymbol{\theta} &\sim \mathcal{N}(\mu(\mathbf{x}|\boldsymbol{\theta}), v(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})), \\ \mu(\mathbf{x}) &= \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \sigma^2\mathbf{I})^{-1} \mathbf{y} \\ v(\mathbf{x}) &= \sigma^2 + k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T(\mathbf{x}) (\mathbf{K} + \sigma^2\mathbf{I})^{-1} \mathbf{k}(\mathbf{x}). \end{aligned} \quad (3)$$

Based on the posterior in (3), we can optimize \mathbf{x} by sequentially quarrying points such that each point shows an improvement $I(\mathbf{x}) = \max(\hat{\eta}(\mathbf{x}) - y^\dagger, 0)$, where y^\dagger is the current optimal and $\hat{\eta}(\mathbf{x})$ is the predictive posterior in (3). Integrating out the posterior achieve the expected improvement (EI):

$$EI(\mathbf{x}) = \mathbb{E}_{\hat{\eta}(\mathbf{x})}[\max(\hat{\eta}(\mathbf{x}) - y^\dagger, 0)], \quad (4)$$

which has a closed form solution $(\mu(\mathbf{x}) - y^\dagger)\psi(u(\mathbf{x})) + v(\mathbf{x})\phi(u(\mathbf{x}))$, in which $\psi(\cdot)$ and $\phi(\cdot)$ are the probabilistic density function (PDF) and cumulative density function (CDF) of a standard normal distribution, respectively. The candidate for next iteration is selected by $\arg\max_{\mathbf{x} \in \mathcal{X}} EI(\mathbf{x})$ with on-convex optimizations, e.g., L-BFGS-B. This is the basic procedure of a Bayesian optimization.

B. PTA and Time-Step Control Method

PTA stands out as the most potent and promising continuation approach for addressing the non-convergence issues encountered in DC analysis due to discontinuity and strong nonlinearity. It inserts specific pseudo elements, e.g. capacitors and inductors, into the circuit, so that the original hard-to-solve nonlinear algebraic equations:

$$\mathbf{F}(\mathbf{x}) = 0, \quad (5)$$

where $\mathbf{F}(\cdot) : \mathbb{R}^m \rightarrow \mathbb{R}^m$, $\mathbf{x} = (\mathbf{v}, \mathbf{i})^T \in \mathbb{R}^m$, $m = N + M$, the variable vector $\mathbf{v} \in \mathbb{R}^N$ denotes node voltage, and vector $\mathbf{i} \in \mathbb{R}^M$ represents internal branch current, are transferred to an initial value problem for ODE:

$$\mathbf{P}(\mathbf{x}(t), d\mathbf{x}(t)/dt, t) = \mathbf{F}(\mathbf{x}) + \mathbf{D} * \dot{\mathbf{x}}(t) = 0, \quad (6)$$

where $\dot{\mathbf{x}}(t) = (\dot{\mathbf{v}}(t), \dot{\mathbf{i}}(t))$, and \mathbf{D} represents for the incidence matrix of inserted pseudo elements. With these modifications, implicit numerical integration algorithms, e.g. (7), are used to discretize in the time domain and finally obtain the steady state through iterative difference approximation of the differential term:

$$\dot{\mathbf{x}}(t)|_{t=t_{n+1}} = (\mathbf{x}_{n+1} - \mathbf{x}_n)/h_{n+1}. \quad (7)$$

Conventional PTA methods employ a straightforward iteration counting method [17] for determining the time-step size. This approach compares the number of NR iterations at each time point to determine the next time-step size. Another adaptive time-step control method, based on Switched Evolution/Relaxation (SER), was proposed in [18]. SER employs a heuristic approach leveraging domain experiences and has demonstrated significant potential in achieving speedup through adaptive time-step control. However, striking a balance between large step size for high efficiency and small step size for high solution convergence remains a challenge.

C. Reinforcement Learning

Reinforcement learning is an approach to learn where an agent interacts with the environment to maximize rewards or accomplish predefined goals. Many RL models are based on Markov Decision Processes (MDPs) [19], which serves as a fundamental framework for decision-making in uncertain environments. A MDP can be expressed as a tuple containing 5 main components, s (state), a (action), p (policy), r (reward) and γ (gamma). The Q-value function is commonly used to evaluate the expected sum of rewards obtain by taking a particular action a chosen by a Markov decision π in a given state s :

$$Q^\pi(s, a) = \mathbb{E}[r' + \gamma Q^\pi(s', a') | S_t = s, A_t = a], \quad (8)$$

where s', a', r' are the state, action and reward at the next moment. The ultimate goal of RL is to find the largest value function in the interaction between agent and environment as shown in (9).

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a), \quad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A} \quad (9)$$

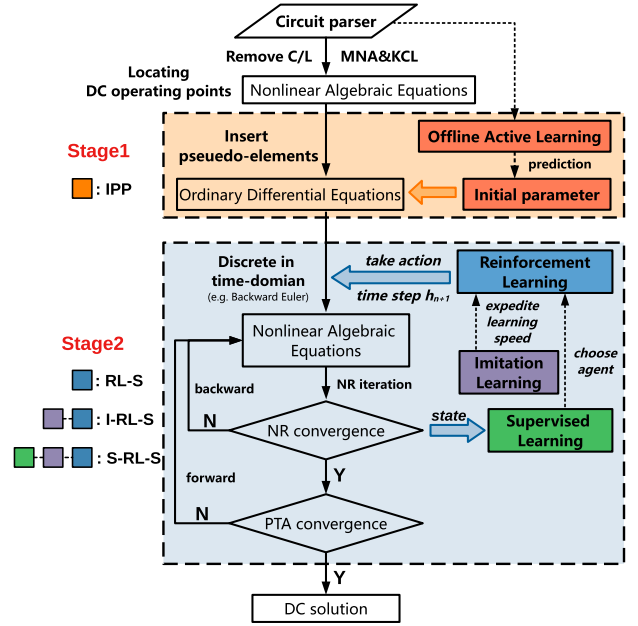


Fig. 1. Overall framework of proposed machine learning enhanced PTA.

III. OVERALL FRAMEWORK

Simulation efficiency of PTA algorithms highly depends on two aspects: (i) a good initial formula; and (ii) an efficient time-integration scheme while solving the formula. In the first stage, inserted pseudo-elements determine the ODE equations formulation and the matrix property. We need to fine-tune the initial parameters to avoid discontinuity caused by some strong nonlinearity designs. In the second stage, the core process of PTA is iteratively stepping towards the steady state through numerical integration. We need to design excellent stepping strategies to reduce the number of nonlinear equations that need to be solved at discrete time points.

In this paper, we propose a two-stage acceleration framework, depicted in Fig. 1, consists of two stages. This framework uses machine learning to speed up DC analysis. At the first stage, an offline trained model is evolved for initial parameter prediction (IPP), which provides appropriate insertion pseudo-elements prediction for better ODE equation formulation. At the second stage, we use Reinforcement Learning Stepping (RL-S) during the simulation process to learn and train the RL-S agent from simulated states. We will elaborate on the details of this part in Section V. Then in Section VI, we further leverage imitation learning to optimize RL-S, abbreviated as I-RL-S. This method uses imitation learning to eliminate errors introduced by manually designed reward functions and enabling the agent to adjust neural network weights more quickly and accurately. Finally in Section VII, we propose supervised learning optimized I-RL-S, abbreviated as S-RL-S. It employs supervised learning to activate the appropriate agent to provide the optimal stepping strategy at each discrete time point to help track the steady state fast enough.

IV. INITIAL PARAMETERS PREDICTION

A straight-forward way to predict the PTA solver parameters is to collect enough data and build a supervised learning model with deep learning to directly predict the best PTA solver parameter based on given circuit type and features. We introduce an off-line active learning to train the solver parameters predictor model. More specifically, the general BO is implemented to find the best solver parameters for the reference/training circuits. To emphasize the need for active learning, we adopt an adaptive acquisition to dynamically explore the parameter space and also to build a good reference database for online prediction of unseen circuits.

A. Problem Formulation

Consider a PTA solver g with initial parameters \mathbf{x} (indicating the value of inserted pseudo-capacitor, pseudo-inductor and time-constant Tau) that operates on a netlist file denoted as ξ and generates the steady state $\mathbf{u} = g(\mathbf{x}, \xi)$. We are interested in reducing the number of iterations, denoted as $\eta(\mathbf{x}, \xi) + \varepsilon$, for $g(\mathbf{x}, \xi)$. Here ε captures the model inadequacy and randomness that are not fully captured by \mathbf{x} and ξ . We aim to seek a function.

$$\mathbf{x}^*(\xi) = \arg \min_{\mathbf{x} \in \mathcal{X}} \eta(\mathbf{x}, \xi), \quad (10)$$

where $\mathbf{x}^*(\xi)$ is the optimal PTA solver parameters for any given netlist ξ , and \mathcal{X} is the feasible domain for \mathbf{x} . Note that this is not an optimization problem because we are not allowed to run $\eta(\mathbf{x}, \xi)$ for an unseen circuit. Instead, our goal is to find the mapping $\mathbf{x}^*(\xi)$, which is a straightforward supervised learning problem given that we have sufficient data.

We can simply run a classic MC method to locate the best solver parameters to provide the training dataset. However, there are critical issues with this approach: (1) This approach is in general computationally expensive as we need to search the solver parameters space X for all available circuits ξ ; (2) Only the best parameters are included, whereas the majority of the training data are wasted, leading to an inferior model; (3) Most critically, the best solver parameters are potentially multi-model, i.e., there is more than one best solver parameter that produces the best performance. To resolve these challenges simultaneously, we resolve $\mathbf{x}^*(\xi)$ approximately based on

$$\mathbf{x}^*(\xi) = \arg \min_{\mathbf{x} \in \mathcal{X}} \tilde{\eta}(\mathbf{x}, \xi | \mathcal{D}), \quad (11)$$

where $\tilde{\eta}$ is a surrogate model approximating η based on data \mathcal{D} . Our solution is a two-stage approach including (1) the online stage of solving Equation (11) and the offline stage which builds the training dataset \mathcal{D} for the online stage.

B. Dual Parameter Feature Learning

Our model needs to handle two types of parameters: circuit characters and solver parameters, which have different correlations, scales, and ranges. To resolve this issue, we assume

a separable kernel function and follow the work of [20] to introduce different transformations for them.

$$k([\mathbf{x}, \xi], [\mathbf{x}', \xi']) = k_x(\Psi(\mathbf{x}), \Psi(\mathbf{x}')) \cdot k_\xi(\Phi(\xi), \Phi(\xi')), \quad (12)$$

where $\Psi(\mathbf{x})$ and $\Phi(\xi)$ are functional transformation for \mathbf{x} and ξ , respectively. In this work, $\Phi(\xi)$ utilizes a deep feed forward fully connected network as an automatic feature extraction for ξ as in [20], whereas $\Psi(\mathbf{x})$ denotes a reparameterization for \mathbf{x} such that the range and scale is handled properly. Specifically, we force $\log(z_d) = (7 \cdot \text{sigmoid}(w_d))$, where $\text{sigmoid}(w_d) = 1/(1 + \exp(w_d))$ is the sigmoid function. Here, z_d is an intermediate variable introduced by the log-sigmoid transformation for optimizing the CEPTA solver, where $x_d = (7 \cdot \text{sigmoid}(z_d))^{10}$. With this transformation, \mathbf{x} is naturally constrained in the range of $[10^{-7}, 10^7]$. Also, the new parameters w_d focuses on the scale of \mathbf{x} rather than its particular value. This is particularly handy when we later optimize the number of iterations w.r.t. \mathbf{w} instead of \mathbf{x} .

For the original circuit characters, we follow [21] and use seven key factors (the number of nodes, MNA equations, independent current sources, resistors, voltage sources, bipolar junction transistor, and MOS field-effect transistor) to characterize a netlist and a flag $\{0, 1\}$ denoting whether this circuit is a BJT or MOS type circuit. We know that BJT and MOS type circuits have completely different *a-priori*. Thus, the transformation should reflect this knowledge. To this end, we further modify the kernel structure as

$$k([\mathbf{x}, \xi], [\mathbf{x}', \xi']) = (k_x^1(\Psi(\mathbf{x}), \Psi(\mathbf{x}')) \cdot k_\xi^1(\Phi(\xi), \Phi(\xi')))^{\tau} \times (k_x^2(\Psi(\mathbf{x}), \Psi(\mathbf{x}')) \cdot k_\xi^2(\Phi(\xi), \Phi(\xi')))^{1-\tau}, \quad (13)$$

where $\tau = \{0, 1\}$ indicates the BJT and MOS type circuit.

C. Offline Training and Online Predictions

With the circuit characters and solver parameters being handled properly, we now discuss the offline stage of building the model and training data $\tilde{\eta}(\mathbf{x}, \xi | \mathcal{D})$ and the online stage as shown in Fig. 2, in which a new circuit is given, and a set of predictive best solver parameters is suggested.

Let ξ^\dagger denotes a validating data circuits unseen to our system and \mathbf{x}_{ξ^\dagger} the optimal solver parameters for it. Without loss of generality, assuming that \mathcal{D} is constructed in a sequential manner, i.e., adding a point at a time. At t -th iteration, the regret for ξ^\dagger is $R_t^{\xi^\dagger} = \eta(\hat{\mathbf{x}}, \xi^\dagger) - \eta(\mathbf{x}_{\xi^\dagger}, \xi^\dagger)$, where $\hat{\mathbf{x}}$ is the proposition, \mathbf{x}_{ξ^\dagger} is the (unknown) best solver parameters. The proposed solver parameters are normally obtained based on Bayesian optimization. For instance, based on the EI approach, we can optimize,

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \mathbb{E}_{\tilde{\eta}(\mathbf{x})} \left[\max \left(y^\dagger - \tilde{\eta}(\mathbf{x}, \xi^\dagger), 0 \right) \right], \quad (14)$$

where y^\dagger indicates the minimum number of iterations so far or simply the number of iterations for the default PTA solver parameters. For our problem, we need to modify the Bayesian

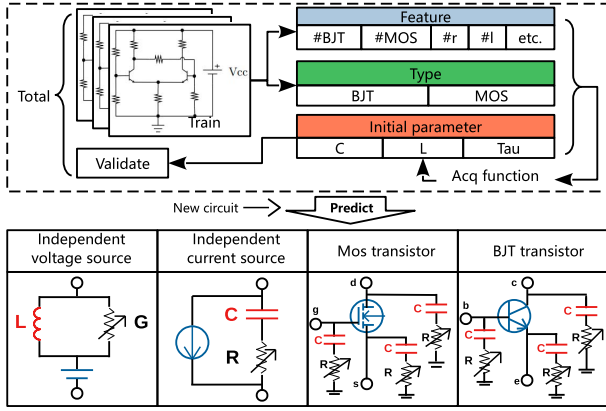


Fig. 2. Intelligent initialization for better equation formulation.

optimization formulation because we neither know ξ^\dagger in advance nor are allowed to run simulation to approach the best solver parameter with a few iterations. Notice that any candidate among our training circuits $\xi = [\xi_1, \dots, \xi_N]^T$ can be used as ξ^\dagger , if any data associated with it is excluded. For our training, we thus define a total regret as $R_T = \sum_{t=1}^T \sum_{n=1}^N r_t^{\xi_n}$. For each iteration when circuit ξ_n is selected, its associated data is excluded from the GP surrogate. More specifically, the optimization of (14) is augmented as

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x} \in \mathcal{X}} \mathbb{E}_{\tilde{\eta}(\mathbf{x})} \left[\max \left(y^\dagger - \tilde{\eta}(\mathbf{x}, \xi^\dagger | \mathcal{D}_{\xi^\dagger}^r), 0 \right) \right], \quad (15)$$

where $\min \tilde{\eta}(\mathbf{x}, \xi^\dagger | \mathcal{D}_{\xi^\dagger}^r)$ is the optimal solver iteration based on the solver parameters for the current $\mathcal{D}_{\xi^\dagger}^r$, which indicates that any data corresponding to ξ^\dagger circuit is excluded. We can now achieve the minimum total regret by iteratively solving Equation (15) with enumerations of all candidate circuits.

At the online stage, with the model after offline training and given a new circuit ξ^* , we can simply optimize $\tilde{\eta}(\mathbf{x}, \xi^* | \mathcal{D})$ w.r.t. \mathbf{x} , i.e., solving Equation (11) to propose the best predictive solver parameters.

V. REINFORCEMENT LEARNING ENHANCED STEPPING

For the ODE solving part in PTA, the time-integral stepping strategy significantly affects simulation efficiency. Therefore, in this section, we propose a method called RL-S to accelerate the PTA stepping process using reinforcement learning. RL-S treats the iterative tracking process of PTA solutions as a Markov Decision Process (MDP) problem and interacts with the circuit simulation environment through two agents: one for forward and one for backward steps. RL-S also addresses sample imbalance issues by equipping public sample buffers. The key elements of RL-S will be comprehensively discussed in this section. In addition, the training algorithm in RL-S belongs to the online learning category. Based on this, we combine offline pre-training with online learning to enhance adaptability to any new circuit during the actual simulation process.

A. TD3-Based Reinforcement Learning Stepping

State, action, and reward can be considered as the three key components in RL, where the essence lies in taking actions in different states to maximize rewards. By treating the iterative solution trace processes of PTA as MDP, that is to predict an appropriate time-step size according to simulation state at each PTA iteration, we can map it to a classic RL problem:

1) **State**: We define the simulation state at each time point t_n in the PTA process as the RL state S_n , which includes *Iters*: the number of NR iterations at current time point, *Res*: residual of the ODE equation, Γ : relative change in solution compared to the previous step, *NRflag*: absolute change in solution compared to the previous step, and *PTAflag*: limit of the *Res*. These simulation states evaluate whether the simulation is gradually converging, representing a measure of “convergence distance”.

2) **Action**: We normalize the action a obtained from the agent using the tanh function to the interval $(-1, 1)$. Then, we multiply it by the relevant coefficient and constrain its range using the exponential function. Finally, the transformed action a is multiplied by the time-step h_{n-1} of the previous time-step to obtain the next time-step h_n . The equation is $h_n = e^{(a+p) \cdot k} \cdot h_{n-1}$, where p and k are parameters to ensure reasonable ratio for time-step.

3) **Reward**: We prioritize PTA convergence as the primary objective and consider other states such as NR convergence as secondary objectives. We assign different coefficients to each simulation state and design the following reward function:

$$R = c_1 \Gamma + c_2 \text{Iters} + c_3 \text{Res} + c_4 \text{NRflag} + c_5 \text{PTAflag}. \quad (16)$$

Through these steps, we map the prediction of time-steps in the PTA simulation process to a classic RL problem.

TD3 [22] is a SOTA reinforcement learning algorithm that belongs to the deterministic strategy RL family and is based on the actor-critic agent. Compared to traditional reinforcement learning methods like DQN and Q-learning, which are typically used for discrete action spaces, TD3 can directly handle continuous action spaces. This means that the TD3 algorithm is better suited for calculating the time step, which belongs to a continuous action space, in the PTA solving process. Additionally, compared to other actor-critic methods like DDPG [23], TD3 mitigates the impact of high variance estimates and overfitting through techniques such as dual target networks and delayed policy updates, significantly improving stability. Therefore, we choose to build the RL-S algorithm based on it.

Dual-agent approach and public sample buffer have also been proposed and integrated into the RL-S algorithm. During simulation, most of the training data are large time-step samples collected during the convergence phase. However, in certain cases, smaller steps are needed to ensure convergence. This results in a single agent being unable to effectively handle both scenarios simultaneously during training. To address this, we introduce two agents, each responsible for controlling time-steps in different scenarios. The “forward” agent is responsible

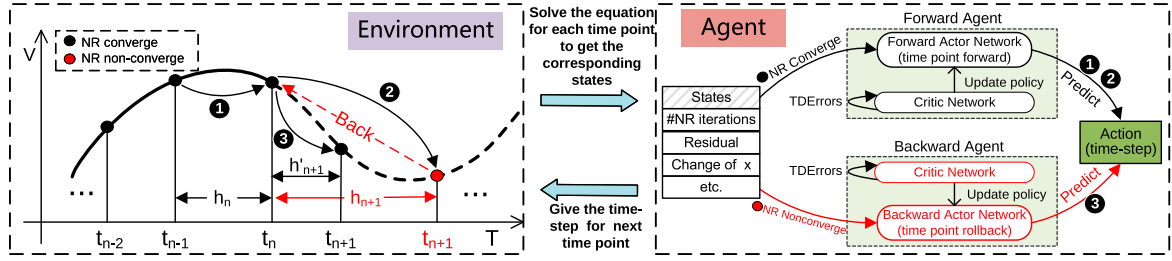


Fig. 3. Reinforcement learning workflow based on TD3 in simulation iterations. ① ② predicted through the forward agent, ③ predicted through the backward agent.

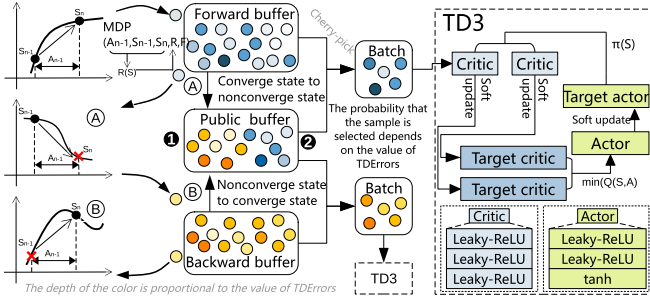


Fig. 4. Updating strategy of reinforcement learning for predicting step size. ① expansion of the public sample buffer. ② cherry-pick of the samples are used for training.

to produce an increased time-step (that is $h_{n+1} > h_n$). If the time-step size leads to non-convergence, the simulation needs to rollback to the previous converged time-point and call the “backward” agent to predict a decreased time-step. This concept is illustrated in Fig. 3.

However, the dual-agent approach leads to the original sample buffer being divided into two. This results in the backward agent’s sample pool being less abundant, affecting its convergence speed. Fortunately, in actual simulations, we find that some experiences of the forward agent are often beneficial to the backward agent as well. Therefore, as shown in Fig. 4, we introduce a public sample buffer to enhance sample utilization and accelerate the convergence speed of the agents. The algorithm pseudo code for the RL-S based on the TD3 algorithm is comprehensively explained in Algorithm 1.

VI. IMITATION LEARNING OPTIMIZED RL-S

Although RL-S proposed in Section V demonstrates significant efficiency improvements compared to traditional time-step control algorithms, the reward function in (16) is a simple linear one and often cannot fit the distribution of actual reward values well, leading to certain biases in the calculated reward values. This may not only cause the agent to learn incorrect strategies but also affect the speed of convergence, thereby becoming a key limitation to further improve simulation efficiency.

In RL-S, the actor network of the TD3 agent updates its network parameters through gradient ascent, with the gradient defined as shown in Equation (19):

$$\nabla_{\theta_{\pi}} J(\theta_{\pi}) = \mathbb{E}_{s \sim \rho^{\beta}, a \sim \pi_{\theta_{\pi}}} [\nabla_{\theta_{\pi}} \log(\pi_{\theta_{\pi}}(a|s)) \nabla_a Q(s, a)], \quad (19)$$

Algorithm 1 TD3-based Reinforcement Learning Stepping

- 1: Initialize critic networks Q_{θ_1} , Q_{θ_2} , and actor network π_{ϕ}
- 2: Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
- 3: Initialize private sample buffer \mathcal{B}_{α} , \mathcal{B}_{β} and public buffer \mathcal{B}
- Input:** Netlist τ
- Output:** The optimal time stepping strategy π^*
- 4: Get initial state $s \leftarrow s_0$
- 5: **while** PTA iteration does not converge **do**
- 6: Select agent from **dual-agents** according to NR flag
- 7: Determine action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \in \mathcal{N}(0, \sigma)$
- 8: Observe reward r and new state s' by selected agent
- 9: Store transition tuple (s, a, r, s') using **collaborative learning**
- 10: Sample mini-batch size of N transitions (s, a, r, s') from buffer \mathcal{B} and \mathcal{B}_{α} or \mathcal{B} and \mathcal{B}_{β}
- 11: $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
- 12: $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
- 13: Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
- 14: **if** $\text{step mod } d$ **then**
- 15: Update ϕ by the deterministic policy gradient
- 16: $\nabla_{\phi} J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_{\phi}(s)} \nabla_{\phi} \pi_{\phi}(s)$
- 17: Update target networks
- 18: $\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$; $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$
- 19: **end if**
- 20: **end while**

where $J(\theta_{\pi})$ represents the objective function of the actor network, ρ^{β} represents the state distribution, $\pi_{\theta_{\pi}}(a|s)$ denotes the action output of the actor network, and $Q(s, a)$ represents the estimated action-value function of the critic network. And the parameter update gradient for the critic network is given by Equation (20) (where the loss function is exemplified by Mean Squared Error, MSE):

$$\nabla_{\theta_Q} J(\theta_Q) = \frac{1}{N} \sum_{i=1}^N \nabla_{\theta_Q} \left[\frac{1}{2} (Q(s_i, a_i, \theta_Q) - y_i)^2 \right], \quad (20)$$

where $J(\theta_Q)$ represents the objective function of the critic network, and y_i represents the TD target:

$$y_i = r_i + \gamma \min(Q'_1(s'_i, \pi'(s'_i)), Q'_2(s'_i, \pi'(s'_i))). \quad (21)$$

Since r_i in y_i is not accurate enough, it implies that inaccurate reward values will affect the update of critic network parameters, further influencing the update of actor network parameters. We combine the actor network of TD3 as a generator network and pair it with a discriminator network to form a GAIL agent. The generator is primarily responsible for generating original

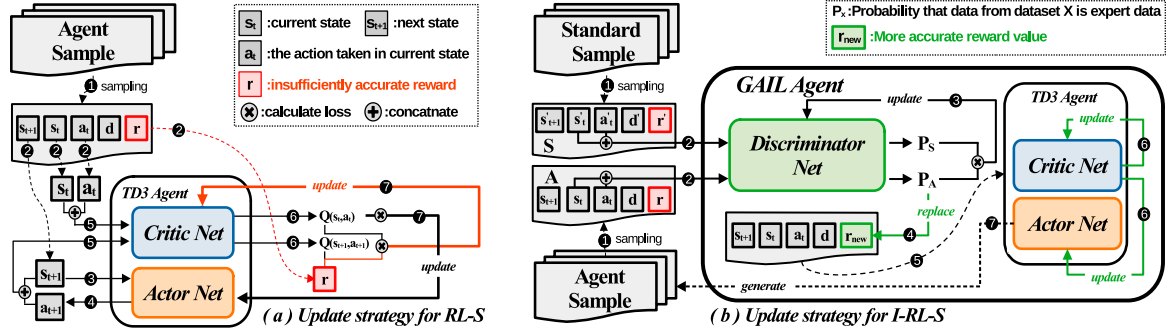


Fig. 5. Update strategies for RL-S and I-RL-S.

stepping samples, while the discriminator evaluates the likelihood that the input samples are from the standard distribution. We input the stepping samples A generated by the generator and the standard simulation samples S (collected from previous RL-S simulations) into the discriminator simultaneously. The discriminator provides the probabilities P_A and P_S that A and S are standard samples, respectively. We calculate the binary cross-entropy between these probabilities as the discriminator's loss function and update its weights using gradient descent, as shown in Equation (22):

$$L_D = -\mathbb{E}_{S \sim p(S)}[\log(D(S))] - \mathbb{E}_{A \sim p(A)}[\log(1 - D(A))], \quad (22)$$

where $D(\cdot)$ represents the discriminator's probability of judging the input sample as a standard sample, and $p(\cdot)$ is the sample distribution. Finally, we take the negative logarithm of P_A , which represents the similarity between the generated sample and the standard sample. We replace the inaccurate reward with it, effectively avoiding the bias introduced by manually designing the reward function. The process is illustrated in Fig. 5.

We refer to the RL-S improved by GAIL as the I-RL-S algorithm. This algorithm effectively utilizes the characteristics of generative adversarial networks to enable self-improvement of the TD3 agent through adversarial learning with past sample data, greatly reducing the involvement of human expert experience. The basic algorithm idea of GAIL is shown in Algorithm 2.

VII. SUPERVISED LEARNING OPTIMIZED I-RL-S

Although the dual agents proposed in Section V effectively handles two different simulation scenarios, the backward agent is only active when the given time-step size leads to a NR non-convergence and it needs to roll-back to the previous time-point. The backward agent will give a smaller time-step size to ensure NR convergence. This results in additional rollback overhead, significantly impacting simulation efficiency.

To address this issue, we further propose the S-RL-S algorithm, which incorporates supervised learning on top of I-RL-S to minimize the occurrence of rollbacks. S-RL-S labels the information collected from past PTA steps and uses it as training data. Then, it employs the random forest algorithm to train a model. The trained model determines whether the "forward"

Algorithm 2 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0

- 1: **for** $i = 0, 1, 2, \dots$ **do**
- 2: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
- 3: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E} [\nabla_w \log(1 - D_w(s, a))] \quad (17)$$

- 4: Take a policy step from θ_i to θ_{i+1} , using the TRPO rule with cost function $\log(D_{w_{i+1}}(s, a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q(s, a)] - \lambda \nabla_{\theta} H(\pi_{\theta}), \quad (18)$$

where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i} [\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$

5: **end for**

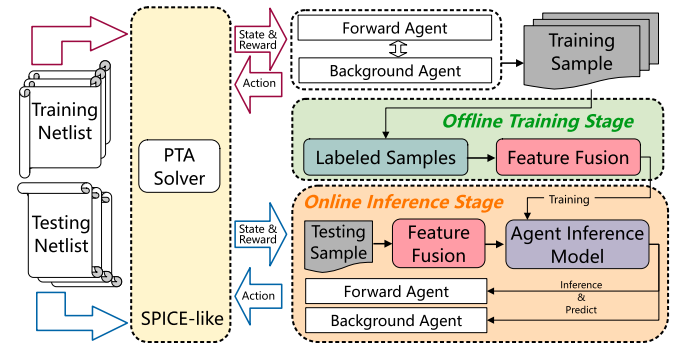


Fig. 6. Flow of S-RL-S.

or "backward" agent should be used in the current simulation state to reduce the incidence of rollbacks and further enhance simulation efficiency. The framework of the S-RL-S algorithm, as shown in Fig. 6, includes steps such as data labeling, feature fusion, agent selection, training, and model evaluation. Fig. 7 illustrates the details of data labeling and feature fusion.

For data labeling, we default to using the forward agent and set the label to $Label = 0$ when it converges. If the forward agent fails to converge, we set the label to $Label = 1$ and

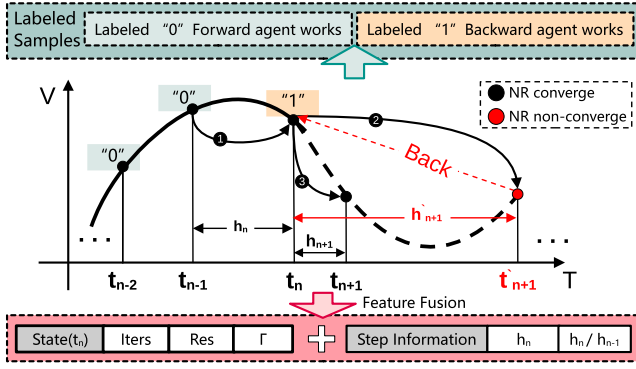


Fig. 7. Demo of labeled samples and feature fusion for dual-agents with supervised learning.

switch to the backward agent. Besides data labeling, selecting appropriate features is crucial. Beyond the initial features we choose—NR iterations (*Iters*), residual (*Res*), and relative change in the solution (Γ)—we also add two new features based on the characteristics of the time-step calculation process: the current time-step (h_n) and the ratio between the current and previous time-step (h_n/h_{n-1}). The detailed process is illustrated in Fig. 7.

The S-RL-S algorithm employs the widely used supervised learning method, Random Forest (RF), to perform prediction tasks for the agent selection. After combining labeled samples and features, the stepping information generated by the I-RL-S algorithm is selected as training set samples \mathcal{D} . The data collected are then processed and used to train the constructed RF model, with the training process completed during the offline pre-training phase, which does not incur additional computational overhead during simulation. In practical applications, the pre-trained RF model is used to predict which agent should be used at each time point of the PTA simulation, and then the corresponding RL agent is called to produce a time-step. This process is illustrated in Algorithm 3. Additionally, the extra inference overhead introduced by the prediction will be explained in Section VIII.

VIII. NUMERICAL EXAMPLE

A. Experimental Setup

The proposed two-stage acceleration framework has been implemented in the SPICE-like simulator WSPICE and tested on a server with a 2.6GHz Intel Core i5-11400H CPU and 32GB main memory. In our experiments, we collect benchmark circuits [24] and some practical circuits commonly used in nonlinear DC analysis from previous works [25], [26], [27]. We randomly select 70% circuits for the training set and 30% for the test set. By including various types of nonlinear circuits (e.g., BJT, MOS) into both training and test sets, we ensure the model's generalization ability in practical applications.

B. Initial Parameters Prediction

We demonstrate the reduction in NR iterations over default CEPTA setting based on our IPP in Table I. In this evaluation,

Algorithm 3 Inference Agent Based on Random Forest

- 1: **Offline training stage**
- 2: Input training data \mathcal{D}
- 3: **for** $i = 1$ to B **do**
- 4: 1) Draw a bootstrap sample Z^* of size N from \mathcal{D}
- 5: 2) Grow a random forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
- 6: a) Select m variables at random from the p variables.
- 7: b) Pick the best variable/split-point among the m .
- 8: c) Split the node into daughter nodes.
- 9: **end for**
- 10: Get ensemble of trees $\{T_b\}_1^B$ as trained predictor \mathcal{P}
- 11: **Online inference stage**
- 12: Load forward agent and backward agent
- 13: Get initial state $s \leftarrow s_0$ and time-step information h_{info}
- 14: **while** PTA is not convergence **do**
- 15: Fuse state s and time-step information h_{info} as feature x
- 16: **if** $\mathcal{P}(x) = 0$ and $preNRflag = 0$ **then**
- 17: Forward agent works
- 18: **else**
- 19: Backward agent works
- 20: **end if**
- 21: Update state s and time-step information h_{info}
- 22: **end while**

TABLE I
SIMULATION EFFICIENCY OF INITIAL PARAMETERS PREDICTION (#OF NR)

Circuits	Type	#Nodes	#Elem	CEPTA	IPP	Reduction
Adding	MOS	15	18	272	95	2.86x
MOSBandgap	MOS	34	16	153	93	1.65x
6stageLimAmp	BJT	80	35	72	34	2.12x
TRCKTorig	BJT	15	20	53	34	1.56x
UA709	BJT	42	39	407	223	1.83x
UA733	BJT	22	31	121	39	3.10x
D22	BJT	6	8	N/A	111	-

we utilize a training set consisting of 43 canonical benchmark circuits, while testing is conducted on 7 practical circuits (including two MOS circuits and five bipolar junction transistor circuits) using WSPICE. The results demonstrate a consistent $2 \times \sim 3 \times$ reduction in the number of NR iterations. Importantly, the IPP technique enables the convergence of previously non-converging cases, which is highly desirable in our applications. This is achieved by providing a feasible working space for the subsequent reinforcement learning (RL) procedure.

C. Simulation Efficiency Comparison

To evaluate the effectiveness of our proposed S-RL-S algorithm, we first compare our S-RL-S with two conventional stepping strategies (SOTA adaptive stepping method [18] and widely used simple stepping method [17]) for CEPTA, pure

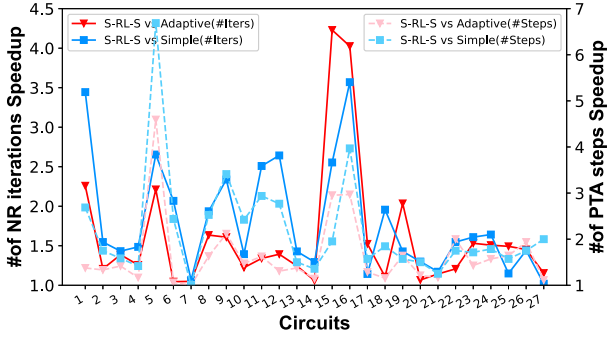


Fig. 8. Speed-up of S-RL-S over two conventional methods for CEPTA.

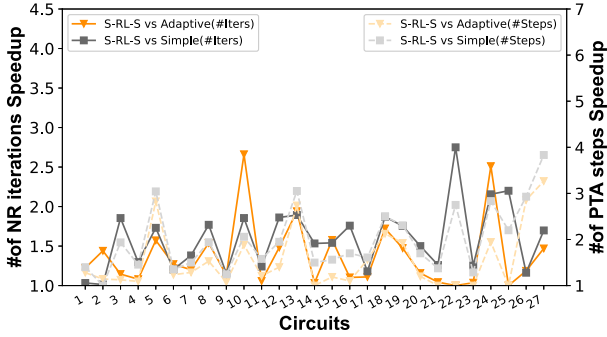


Fig. 9. Speed-up of S-RL-S over two conventional methods for pure PTA.

PTA, and RPTA. For DPTA, we not only compare our S-RL-S with the adaptive stepping method, but also further compare it with two SOTA methods: the RL-e method proposed in [25] and the OSSP method proposed in [26].

For CEPTA, we illustrate the performance comparison of our S-RL-S method against adaptive stepping [18] and simple stepping [17] in Fig. 8. The left y-axis represents the reduction in the number of NR iterations, corresponding to solid lines. The right y-axis represents the reduction in the number of PTA iterations, corresponding to dashed lines. The figure demonstrates that, compared to the simple stepping strategy, S-RL-S achieves an average reduction of $1.81\times$ (maximum $3.57\times$) in the number of NR iterations and $2.17\times$ (maximum $6.69\times$) in the number of PTA iterations. In comparison to the adaptive stepping strategy, S-RL-S attains an average reduction of $1.61\times$ (maximum $4.23\times$) in the number of NR iterations and $1.66\times$ (maximum $4.59\times$) in the number of PTA iterations. Similar results can also be observed with other PTA methods, such as pure PTA as shown in Fig. 9, and RPTA as depicted in Fig. 10.

For DPTA, we present the performance comparison of our S-RL-S method with adaptive stepping method [18], a supervised learning-enhanced stepping method [10], and two other SOTA methods RL-e [25] and OSSP [26] in Table II:

1) S-RL-S achieves an average reduction of $36.09\times$ (maximum $285.71\times$) compared to the adaptive stepping method in terms of the number of NR iterations. Compared to supervised learning-enhanced stepping method [10], S-RL-S achieves an average reduction of $2.57\times$ (maximum $8.28\times$), demonstrating

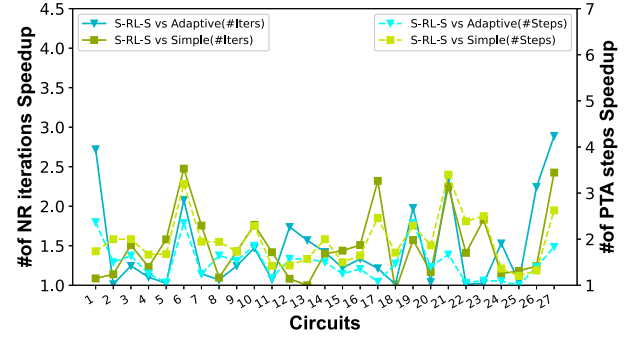


Fig. 10. Speed-up of S-RL-S over two conventional methods for RPTA.

that RL exhibits superior effectiveness over simplistic supervised approaches in dynamic decision-making tasks requiring adaptive multi-step optimization. When compared to RL-e, S-RL-S achieves an average reduction of $1.42\times$ (maximum $2.92\times$). Similarly, compared to OSSP, S-RL-S exhibits an average reduction of $1.32\times$ (maximum $1.87\times$).

2) S-RL-S achieves an average reduction of $76.92\times$ (maximum $589.86\times$) compared to the adaptive stepping method in terms of the number of PTA steps. Compared to [10], S-RL-S achieves an average reduction of $2.53\times$ (maximum $5.68\times$). When compared to OSSP, S-RL-S achieves an average reduction of $1.56\times$ (maximum $2.29\times$). Since RL-e does not present experimental results on the number of PTA iterations in the paper, we are temporarily unable to compare it with our method.

The three figures and one table showcase that the S-RL-S algorithm is not only applicable across various PTA contexts but also demonstrates superior simulation efficiency performance compared with SOTA PTA stepping methods. They underscore the immense potential and practical value of the S-RL-S algorithm.

D. Ablation Experiments

In this subsection, we will delve into the simulation efficiency performance improvements brought about by pre-training, reinforcement learning in RL-S, imitation learning in I-RL-S, and supervised learning in S-RL-S, respectively.

Firstly, we discuss the performance improvement brought by pre-training the agents before applying the RL-S algorithm to practical simulations. We select 6 canonical benchmark circuits as the training set for offline pre-training of the RL-S algorithm. Completing one round of pre-training for the agent on the entire training set using the RL-S algorithm is referred to as one epoch. Then, we use 7 unseen circuits as the test set. Fig. 11 illustrates the performance of the agents on these 7 circuits after experiencing different numbers of epochs. After approximately 20 epochs, the number of NR iterations stabilizes, indicating convergence of the agents. It can be observed that pre-training enables the agents to achieve PTA convergence faster when facing unseen circuits. Hence, in the practical application of RL-S, we typically utilize a model that has undergone pre-training for 20 epochs.

TABLE II
SIMULATION EFFICIENCY COMPARISON AMONG FOUR STEPPING METHODS FOR DPTA

Circuits	Adaptive [18]		LSTM [10]		RL-e [25]		OSSP [26]		S-RL-S		Reduction (S-RL-S vs. Adaptive)		Reduction (S-RL-S vs. LSTM)		Reduction (S-RL-S vs. RL-e)		Reduction (S-RL-S vs. OSSP)	
	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps	#Iters	#Steps
ab_integ	4406	2167	402	85	177	-	158	35	155	40	28.43x	54.18x	2.59x	2.13x	1.14x	-	1.02x	0.88x
fadd32	1859	917	284	73	141	-	129	39	75	17	24.79x	53.94x	3.79x	4.29x	1.88x	-	1.72x	2.29x
todd3	9341	4645	1259	167	444	-	199	49	152	30	61.45x	154.83x	8.28x	5.57x	2.92x	-	1.31x	1.63x
mosamp	239	97	263	65	129	-	140	30	108	23	2.21x	4.22x	2.44x	2.83x	1.19x	-	1.30x	1.30x
THM5	5324	2660	127	44	124	-	116	36	116	35	45.90x	76.00x	1.09x	1.26x	1.07x	-	1.00x	1.03x
UA733	152	37	133	34	69	-	71	21	66	14	2.30x	2.64x	2.02x	2.43x	1.05x	-	1.08x	1.50x
mux8	118	46	154	58	84	-	101	35	54	18	2.19x	2.56x	2.85x	3.22x	1.56x	-	1.87x	1.94x
rca	119	28	64	13	60	-	62	19	49	9	2.43x	3.11x	1.31x	1.44x	1.22x	-	1.27x	2.11x
gm1	47	21	69	34	40	-	41	20	29	13	1.62x	1.62x	2.38x	2.62x	1.38x	-	1.41x	1.54x
e1480	5514	2741	369	50	179	-	207	42	155	39	35.57x	70.28x	2.38x	1.28x	1.15x	-	1.34x	1.08x
mosrect	826	407	84	18	103	-	83	27	48	13	17.21x	31.31x	1.75x	1.38x	2.15x	-	1.73x	2.08x
RCA3040	119	28	64	13	60	-	62	19	49	9	2.43x	3.11x	1.31x	1.44x	1.22x	-	1.27x	2.11x
UA709	339	50	711	142	297	-	137	24	137	25	2.47x	2.00x	5.19x	5.68x	2.17x	-	1.00x	0.96x
cram	87	36	110	45	55	-	62	23	51	17	1.71x	2.12x	2.16x	2.65x	1.08x	-	1.22x	1.35x
schmitfast	5691	2820	176	50	129	-	-	-	105	22	54.20x	128.18x	1.68x	2.27x	1.23x	-	-	-
slowlatch	9353	4649	264	60	186	-	-	-	145	25	64.50x	185.96x	1.82x	2.40x	1.28x	-	-	-
MOSMEM	26000	12977	95	18	101	-	-	-	91	22	285.71x	589.86x	1.04x	0.82x	1.11x	-	-	-
ab_ac	3947	1959	265	66	123	-	-	-	106	30	37.24x	65.30x	2.50x	2.20x	1.16x	-	-	-
ab_opamp	2536	1210	430	87	208	-	-	-	191	40	13.28x	30.25x	2.25x	2.18x	1.09x	-	-	-
Average	-	-	-	-	-	-	-	-	-	-	36.09x	76.92x	2.57x	2.53x	1.42x	-	1.32x	1.56x

#Iters:number of NR iterations.

#Steps:accepted and rejected steps of PTA.

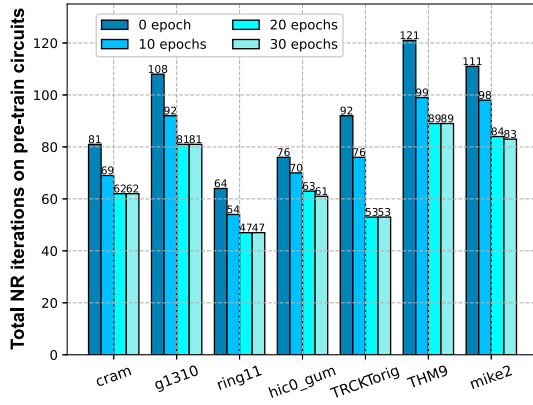


Fig. 11. Performance of pre-trained agent on total NR iterations.

Next, we will analyze the performance improvements brought about by introducing imitation learning and supervised learning. Additionally, we also compare the performance of RL-S based on DQN and RL-S based on TD3. Table III illustrates the performance of RL-S, I-RL-S, and S-RL-S on 12 test circuits. It is worth noting that the performance in the table is with pre-training process. From the table, it can be clearly seen that:

1) Compared to DQN, TD3 achieves a maximum reduction of $2.17\times$ and an average reduction of $1.16\times$ in terms of the number of NR iterations. In terms of PTA steps, it exhibits a maximum reduction of 64.41% and an average reduction of 13.62%. The results demonstrate that TD3 achieves superior time-step over DQN in PTA's continuous action space, leading to enhanced simulation efficiency.

2) I-RL-S achieves a maximum reduction of $5.24\times$ and an average reduction of $1.76\times$ in terms of the number of NR iterations compared to RL-S. It also exhibits a maximum reduction

of 74.38% and an average reduction of 34.38% in total PTA steps.

3) S-RL-S demonstrates further performance improvement compared to I-RL-S. It achieves a maximum reduction of $1.49\times$ and an average reduction of $1.10\times$ in terms of the number of NR iterations. Moreover, it achieves a maximum reduction of 100.00% and an average reduction of 25.84% in rollback PTA steps.

Fig. 12 presents the detailed simulation process of the three stepping methods on the circuit REGULATOR. The three bar charts above display the rate of change between adjacent step sizes in logarithmic form, making the forward (green bars) and backward (red and yellow bars) steps symmetric about the zero axis. Additionally, we also mark the average rate of change of forward steps with gray dashed lines in the three upper charts. The three lower charts show the voltage variation curve of node V(1) in the REGULATOR circuit, where red "X" indicates non-convergent PTA steps, and green "O" indicates convergent PTA steps. Combining these six charts, we can observe that introducing imitation learning makes the step size change rate of I-RL-S (average $2^{1.50}\times$) more aggressive than that of RL-S (average $2^{0.87}\times$), avoiding excessive unnecessary calculations caused by overly small time-steps and achieving PTA convergence with fewer PTA steps. Moreover, from the voltage curve, we can see that the larger step size adopted by I-RL-S also makes it easier to skip local non-convergent points.

However, I-RL-S does not effectively address the problem of time point rollbacks caused by non-convergence, which remains one of the limiting factors affecting performance. To address this issue, we introduce supervised learning into I-RL-S, thereby forming S-RL-S. The introduction of supervised learning enables S-RL-S to maintain the aggressive step size change rate of I-RL-S while proactively predicting the adoption of backward agent in regions where convergence is challenging. By stepping through these regions with a series

TABLE III
SIMULATION EFFICIENCY COMPARISON AMONG PROPOSED RL-S(DQN), RL-S, I-RL-S, AND S-RL-S FOR DPTA

Circuits	RL-S(DQN)			RL-S			I-RL-S			S-RL-S			Reduction (TD3 vs. DQN)		Reduction (I-RL-S vs. RL-S)		Reduction (S-RL-S vs. I-RL-S)	
	#Iters	#Asteps	#Rsteps	#Iters	#Asteps	#Rsteps	#Iters	#Asteps	#Rsteps	#Iters	#Asteps	#Rsteps	#Iters	#Steps	#Iters	#Steps	#Iters	#RSteps
fadd32	187	60	0	152	42	8	75	17	0	75	17	0	1.23x	16.67%	2.03x	66.00%	1.00x	0%
todd3	1818	280	60	838	92	29	160	28	3	152	28	2	2.17x	64.41%	5.24x	74.38%	1.05x	33.33%
mosamp	148	30	1	144	27	0	108	23	0	108	23	0	1.03x	12.90%	1.33x	14.81%	1.00x	0%
UA733	93	22	0	95	19	2	66	13	1	66	13	1	0.98x	4.55%	1.44x	33.33%	1.00x	0%
rca	81	20	0	69	18	0	49	9	0	49	9	0	1.17x	10.00%	1.41x	50.00%	1.00x	0%
mosrect	88	28	0	85	25	1	48	13	0	48	13	0	1.04x	7.14%	1.77x	50.00%	1.00x	0%
REGULATOR	485	95	9	474	86	15	371	64	13	295	79	0	1.02x	2.88%	1.28x	23.76%	1.26x	100.00%
UA741	501	69	8	485	38	23	449	29	23	362	44	10	1.03x	20.78%	1.08x	14.75%	1.24x	56.52%
UA741PFBVINNEG	323	71	5	311	64	7	268	54	6	249	41	5	1.04x	6.58%	1.16x	15.49%	1.08x	16.67%
Multiplier	289	37	9	275	28	14	198	31	7	187	37	5	1.05x	8.70%	1.39x	9.52%	1.06x	28.57%
add32	155	33	1	134	30	1	110	24	1	110	24	1	1.16x	8.82%	1.22x	19.35%	1.00x	0.00%
Suntraction	190	29	5	201	25	9	113	16	4	76	19	1	0.95x	0.00%	1.78x	41.18%	1.49x	75.00%
Average	-	-	-	-	-	-	-	-	-	-	-	-	1.16x	13.62%	1.76x	34.38%	1.10x	25.84%

#Iters:number of NR iterations.

#Asteps:accepted steps of PTA.

#Rsteps:rejected steps of PTA.

#Steps:accepted and rejected steps of PTA.

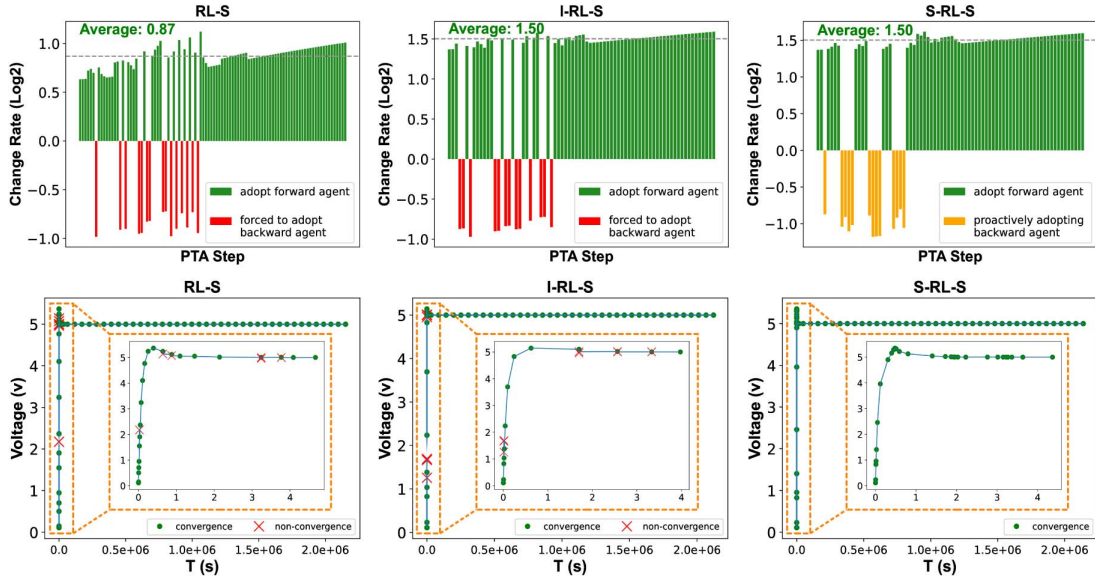


Fig. 12. Detailed simulation process of circuit REGULATOR with three stepping methods.

of smaller time-steps, S-RL-S fundamentally prevents the occurrence of time point rollbacks. From the upper bar charts, it can be observed that S-RL-S actively invokes backward agents near the non-convergent time points where I-RL-S failed to converge. From the curve graphs, it can be seen that S-RL-S steps with smaller time-steps near these non-convergent time points, resulting in denser time points and thereby avoiding the occurrence of rollbacks caused by excessively large time-steps.

E. Model Performance Analysis

Furthermore, within this subsection, we present the ultimate training outcomes of the RF model in S-RL-S. We train an RF model using 6900 samples collected during PTA simulation as the training set, and use an additional 9664 samples as the test set. The test results are depicted in the heatmap shown in Fig. 13(a). This heatmap illustrates the comparison between the model's predictions and the actual results, where "0" represents the forward model and "1" represents the backward model. From this heatmap, we can calculate that our RF model achieves an accuracy of 98.73% and a precision of 99.10% on this test

set. Additionally, Fig. 13(b) displays the ROC curve of the test circuit UA741, with the False Positive Rate (FPR) on the horizontal axis and the True Positive Rate (TPR) on the vertical axis. The area below the ROC curve is called AUC [28] ($AUC \in [0,1]$), where a value closer to 1 indicates better classification accuracy. This plot demonstrates that our RF model achieves an AUC of 0.93 on the UA741 circuit. The data displayed in both figures indicate that our model possesses strong predictive capabilities.

F. Overhead Analysis

Since the S-RL-S algorithm calls both a RL model and a RF model at each numerical integration step, it introduces additional time overhead T_{RL} and T_{RF} . Consequently, the total simulation time becomes: $T_{total} = T_{solver} + T_{RL} + T_{RF}$. In contrast, conventional adaptive time-stepping methods perform only simple numerical checks to determine the step size. Their overhead is negligible compared to the $O(N^3)$ cost of the matrix solves and is thus omitted in Fig. 14. Meanwhile, other SOTA RL-based solutions typically have model complexities

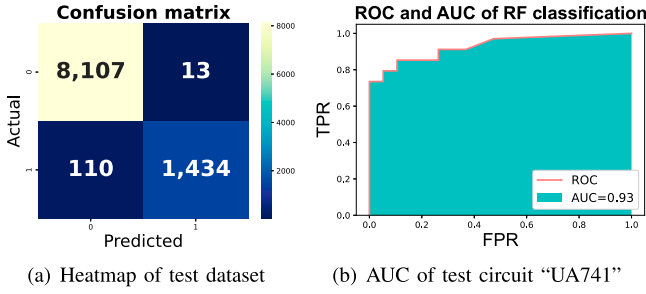


Fig. 13. Heatmap of test dataset and ROC curve of test circuit.

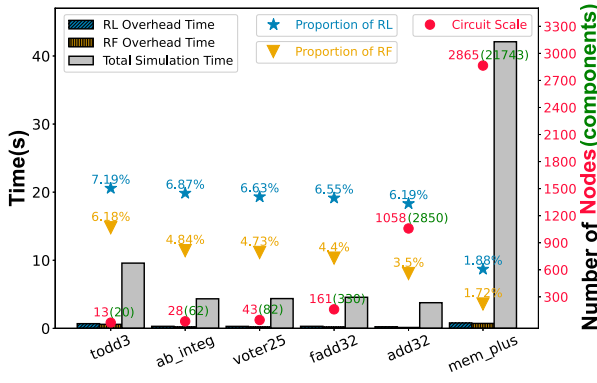


Fig. 14. Overhead analysis for S-RL-S with total NR iteration time.

comparable to our RL network, resulting in a similar overhead to our T_{RL} , yet they do not incur T_{RF} because they lack the RF component. Fig. 14 illustrates how the combined RL and RF inferences compare with the solver's runtime, showing that although these inferences add to the total simulation time, the solver itself remains the dominant cost. In Fig. 14, the proportion of T_{RL} and T_{RF} in the total simulation time progressively diminishes to 3.5% as the circuit scale and component count increase. The increase in circuit components and nodes reflects a single larger interconnected circuit, leading to an $O(N^3)$ increase in matrix solution time while the inference overhead unchanged. This implies that time T_{solver} escalates dramatically with the growing size and complexity of the system matrices, while T_{RL} and T_{RF} remain constant. Consequently, the S-RL-S framework demonstrates exceptional efficiency for ultra-large-scale circuit simulation, as its overhead becomes negligible in large-scale scenarios.

IX. CONCLUSION

In this paper, we propose a fast DC analysis method using cutting-edge techniques such as active learning, reinforcement learning, imitation learning and supervised learning. Our approach aims to accelerate the powerful and promising continuation approach, PTA, in a two-stage acceleration framework.

In the first stage, we employ an offline-trained model for IPP to provide accurate predictions of insertion pseudo-elements, which in turn improves the formulation of the ODE equations. Subsequently, we obtain effective stepping strategies to track

steady states by utilizing reinforcement learning stepping (RL-S) during the simulation process. Furthermore, we introduce imitation learning on top of RL-S (I-RL-S) to avoid biases introduced by manually setting reward functions. And we incorporate supervised learning on top of I-RL-S (S-RL-S) to intelligently select the best-performing agent for each PTA stage, thereby avoiding unnecessary computations caused by rollback. Combining these techniques together, the simulation performance is effectively enhanced. Experiment results demonstrate that our framework achieves a maximum reduction of 3.10x in NR iterations at the first stage and 285.71x compared to the SOTA adaptive stepping approach.

REFERENCES

- [1] J. Deng, K. Batselier, Y. Zhang, and N. Wong, "An efficient two-level DC operating points finder for transistor circuits," in *Proc. 51st ACM/EDAC/IEEE Des. Automat. Conf. (DAC)*, 2014, pp. 1–6.
- [2] T. Nechma and M. Zwolinski, "Parallel sparse matrix solution for circuit simulation on FPGAs," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1090–1103, Apr. 2015.
- [3] C. T. Kelley and D. E. Keyes, "Convergence analysis of pseudo-transient continuation," *SIAM J. Numer. Anal.*, vol. 35, no. 2, pp. 508–523, 1998.
- [4] R. Wilton, "Supplementary algorithms for DC convergence," in *Proc. IEE Colloq. SPICE: Surviving Problems Circuit Eval.*, 1993, pp. 311–319.
- [5] X. Wu, Z. Jin, and Y. Inoue, "Numerical integration algorithms with artificial damping for the PTA method applied to DC analysis of nonlinear circuits," in *Proc. Int. Conf. Commun. Circuits Syst. (ICCCAS)*, 2013, pp. 421–424.
- [6] Z. Jin, X. Wu, Y. Inoue, and N. Dan, "A ramping method combined with the damped PTA algorithm to find the DC operating points for nonlinear circuits," in *Proc. Int. Symp. Integr. Circuits (ISIC)*, 2014, pp. 576–579.
- [7] Z. Jin, X. Wu, and Y. Inoue, "An effective implementation and embedding algorithm of PTA method for finding DC operating points," in *Proc. Int. Conf. Commun. Circuits Syst. (ICCCAS)*, 2013, pp. 417–420.
- [8] Z. Jin, M. Liu, and X. Wu, "An adaptive dynamic-element pta method for solving nonlinear dc operating point of transistor circuits," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2018, pp. 37–40.
- [9] K. S. Kundert and P. Gray, *The Designer's Guide to Spice and Spectre*. Norwell, MA, USA: Kluwer Publishers, 1995.
- [10] X. Zha, H. Pei, D. Niu, X. Wu, and Z. Jin, "Deep learning enhanced time-step control in pseudo transient analysis for efficient nonlinear dc simulation," in *Int. Symp. Electron. Des. Automat. (ISED)*, 2023, pp. 23–28.
- [11] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 228–239, Feb. 2021.
- [12] J. Zhang, Z. Ning, M. Waqas, H. Alasmay, S. Tu, and S. Chen, "Hybrid edge-cloud collaborator resource scheduling approach based on deep reinforcement learning and multiobjective optimization," *IEEE Trans. Comput.*, vol. 73, no. 1, pp. 192–205, Jan. 2024.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.
- [14] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4572–4580.
- [15] R. M. Dudley, "Sample functions of the gaussian process," *Ann. Probability*, vol. 1, no. 1, pp. 66–103, 1973.
- [16] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2005.
- [17] F. N. Najm, *Circuit Simulation*. Hoboken, NJ, USA: Wiley, 2010.
- [18] X. Wu, Z. Jin, D. Niu, and Y. Inoue, "An adaptive time-step control method in damped pseudo-transient analysis for solving nonlinear DC circuit equations," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 100, no. 2, pp. 2274–2282, 2015.
- [19] R. Bellman, "A Markovian decision process," *Indiana Univ. Math. J.*, vol. 6, no. 5, pp. 679–684, 1957.

- [20] W. W. Xing, X. Jin, T. Feng, D. Niu, W. Zhao, and Z. Jin, "BOA-PTA: A Bayesian optimization accelerated PTA solver for spice simulation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 2, pp. 1–26, Dec. 2022.
- [21] S. Zhang, W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Bayesian optimization approach for analog circuit synthesis using neural network," in *Proc. Des. Automat. & Test Eur. Conf. & Exhib. (DATE)*, 2019, pp. 1463–1468.
- [22] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
- [23] F. Rezazadeh, H. Chergui, L. Alonso, and C. Verikoukis, "Continuous multi-objective zero-touch network slicing via twin delayed DDPG and openAI gym," in *Proc. IEEE Global Commun. Conf.*, 2020, pp. 1–6.
- [24] J. Barby and R. Guindi, "Circuitsim93: A circuit simulator benchmarking methodology case study," in *Proc. 6th Annu. IEEE Int. ASIC Conf. Exhibit.*, 1993, pp. 531–535.
- [25] Y. Dong, D. Niu, Z. Jin, C. Zhang, Q. Li, and C. Sun, "Adaptive stepping PTA for DC analysis based on reinforcement learning," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 70, no. 1, pp. 266–270, Jan. 2023.
- [26] D. Niu, Y. Dong, Z. Jin, C. Zhang, Q. Li, and C. Sun, "OSSP-PTA: An online stochastic stepping policy for PTA on reinforcement learning," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 42, no. 11, pp. 4310–4323, Nov. 2023.
- [27] Z. Jin et al., "Accelerating nonlinear dc circuit simulation with reinforcement learning," in *Proc. 59th ACM/IEEE Des. Automat. Conf. (DAC)*, 2022, pp. 619–624.
- [28] N. A. Obuchowski, "Roc analysis," *Amer. J. Roentgenol.*, vol. 184, no. 2, pp. 364–372, 2005.

Zhou Jin (Member, IEEE) received the Ph.D. degree from the Graduate School of Information, Production and Systems, Waseda University, Japan, in 2015. She has been a ZJU 100 Young Professor in Zhejiang University. Her research interests include verification technologies for nonlinear circuits and systems, AI and GPU acceleration for EDA, parallel linear algebra, etc.

Wenhao Li (Graduate Student Member, IEEE) received the B.Eng. degree in computer science and technology from China Three Gorges University, in 2022. He is currently working toward the master's degree with China University of Petroleum-Beijing. His research interests include verification technologies for nonlinear circuits and systems, LSI simulation technologies, and AI for EDA.

Haojie Pei received the master's degree from the College of Information Science and Engineering, China University of Petroleum-Beijing, in 2023. He has been with R&D, Huada Emperyrean Software Company Ltd. His research

interests include verification technologies for nonlinear circuits and systems, LSI simulation technologies, and AI for EDA.

Xiaru Zha received the B.Eng. degree in computer science and technology in 2023 from China University of Petroleum-Beijing, where she is currently working toward the master's degree. Her research interests include integrating AI with EDA, enhancing EDA processes, and semiconductor innovation through AI-assisted design optimization.

Yichao Dong (Graduate Student Member, IEEE) received the B.Eng. degree in electrical engineering and automation from Jiangsu University, Zhenjiang, China, in 2020. He is currently working toward the Ph.D. degree with the Graduate School of Automation, Southeast University, Nanjing, China. His research interests include verification technologies for nonlinear circuits and systems, LSI simulation technologies, and AI for EDA.

Xiang Jin received the master's degree from Beihang University, where his research focused on accelerating EDA processes using Gaussian processes and Bayesian optimization. He is currently a Recommendation Algorithm Engineer with the Model and Application Department, Kuaishou Technology Company Ltd. His work involves developing and optimizing recommendation systems to enhance user experience and platform efficiency.

Xiao Wu received the Ph.D. degree from the Graduate School of Information, Production and Systems, Waseda University, Japan, in 2017. He has been with R&D, Huada Emperyrean Software Company Ltd. His research interests include verification technologies for nonlinear circuits and systems, LSI simulation technologies, and AI for EDA.

Dan Niu (Member, IEEE) received the Ph.D. degree from the Graduate School of Information, Production and Systems, Waseda University, Japan, in 2013. He has been an Associate Professor with the School of Automation, Southeast University. His research interests include verification technologies for nonlinear circuits and systems, LSI simulation technologies, and AI for EDA and AI for weather spatiotemporal sequence prediction.

Wei Xing received the Ph.D. degree in engineering from the University of Warwick, U.K., in 2017. He became a Lecturer with the University of Sheffield, in 2022. His research interests include intelligent computation in engineering and he has developed a set of AI-based design tools, including a yield optimization system, a reinforcement-learning-accelerated SPICE solver, multifidelity fusion and optimization, AI-driven analog, and mixture signal circuit design.