High-Speed Adder Design Space Exploration via Graph Neural Processes

Hao Geng, *Student Member, IEEE*, Yuzhe Ma, *Member, IEEE*, Qi Xu, *Member, IEEE*, Jin Miao, Subhendu Roy, *Member, IEEE*, and Bei Yu, *Member, IEEE*

***Abstract*—Adders are the primary components in the datapath logic of a microprocessor, and thus, adder design has been always a critical issue in the very large-scale integration (VLSI) industry. However, it is infeasible for designers to obtain optimal adder architecture by exhaustively running EDA flow due to the extremely large design space. Previous arts have proposed the machine learning-based framework to explore the design space. Nevertheless, they fall into suboptimality due to a two-stage flow of the learning process and less efficient nor effective feature representations of prefix adder structures. In this article, we first integrate a variational graph autoencoder and a neural process (NP) into an end-to-end, multibranch framework, which is termed the *graph neural process*. The former performs automatic feature learning of prefix adder structures, whilst the latter one is designed as an alternative to the Gaussian process. Then, we propose a sequential optimization framework with the graph NP as the surrogate model to explore the Pareto-optimal prefix adder structures with tradeoff among Quality-of-Result (QoR) metrics, such as power, area, and delay. The experimental results show that compared with state-of-the-art methodologies, our framework can achieve a much better Pareto frontier in multiple QoR metric spaces with fewer design-flow evaluations.**

***Index Terms*—Design space exploration, graph autoencoder, graph neural process, high speed adder, neural process, sequential model-based optimization.**

I. Introduction

VERY large-scale integration (VLSI) design methodologies have developed for about 50 years, from manually-crafted design to computer-aided design (CAD) with increasingly higher levels of design specifications. Unfortunately, with the aggressive and amazing scaling down of semiconductor technology nodes, design complexity increases dramatically. As a result, efficient design space exploration (DSE) [1]–[12] has emerged as a promising solution due to the exponentially increasing size of design space of microprocessors and the consequent time-consuming synthesis runs.

Quintessential DSE optimizes a single objective when other objectives are considered as constraints. But, for the VLSI design, exploration in multiple Quality-of-Result (QoR) metrics space, such as performance, power, and area, is required, which naturally involves a tradeoff. In this case, multiobjective DSE seeks optimal solutions [3], [6], which tradeoff multiple objectives rather than finding one single optimal point. In other words, without any further information, none of these Pareto-optimal points can be regarded as being better than the others in all the objectives simultaneously. Nevertheless, obtaining a set of Pareto-optimal points with an acceptable cost is a big challenge. One main reason is that the objective functions are often unknown, and can only be determined through a pointwise evaluation, which is expensive and necessitates large computational and time resources. Existing EDA design flow, including synthesis tool and physical design tool, can only return one implementation per call and even worse, the implementation is not guaranteed to be in a Pareto set. Therefore, it is not difficult to imagine that searching for the Pareto-optimal set is time consuming since massive evaluations are required. In summary, the goal of multiobjective DSE is to find a set of Pareto-optimal points in as few evaluations of the multiple objective functions as possible so that the total expense is minimized.

In the VLSI domain, DSE is widely exploited but not limited to handling analog circuit synthesis [5], [9], FPGA CAD flow [4], FPGA HLS directives design optimization [10], DNN hardware deployment [11], processor architecture design [12], and adder design [3], [6]. To search for Pareto-optimal solutions as efficiently and effectively as possible, most of them are based on machine learning techniques. Although the above DSE technologies have achieved great success in different scenarios, most of them utilize the Gaussian process (GP), which has a high computational complexity as the regressor for performance estimation.

In this article, we focus on DSE techniques for the adder design. The adder design is one of the fundamental problems in VLSI, where designing carry-propagation units plays the most critical role. Although the unit can be implemented by enormous parallel prefix structures, real synthesis and physical design running are still needed. Regular adder structures proposed in [13] and [14] and some recently developed works [15], [16], which try to generate a single prefix adder network under a set of structural constraints, cannot cover a large design space yet. By incorporating two pruning

Manuscript received 18 February 2021; revised 18 June 2021; accepted 30 August 2021. Date of publication 21 September 2021; date of current version 19 July 2022. This work was supported in part by the HiSilicon Technologies Company, ACCESS–AI Chip Center for Emerging Smart Systems, Hong Kong SAR, and in part by The Research Grants Council of Hong Kong SAR under Grant CUHK14209420. This article was recommended by Associate Editor L. C. Wang. *(Corresponding author: Bei Yu.)*

Hao Geng, Yuzhe Ma, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Qi Xu is with the School of Microelectronics, University of Science and Technology of China, Hefei 230052, China.

Jin Miao is with Google, Mountain View, CA 94043 USA.

Subhendu Roy is with the Design and Sign-Off Group-Machine Learning Group, Cadence Design Systems, San Jose, CA 95134 USA.

Digital Object Identifier 10.1109/TCAD.2021.3114262

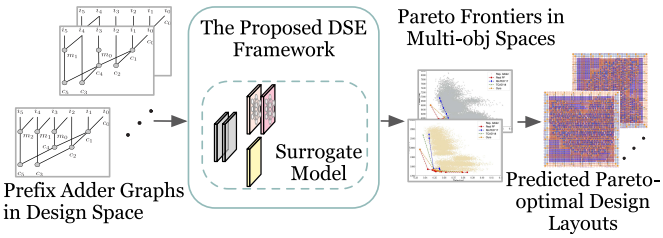

Fig. 1. High-speed adder DSE overview. The objective spaces include area versus delay space, power versus delay space, and area versus power versus delay space.

techniques in the prior, Ma *et al.* [6] proposed a state-of-the-art algorithm to generate the prefix graph structures, and exploited an active learning-based optimization model to explore Pareto-optimal adder designs.

Nevertheless, there exist two main issues in [6]. One is that [6] exploits the two-stage framework where the feature extractor for prefix adders and subsequent machine learning models is separated. Namely, the feature extraction process lacks guided information from the learning models. The other is feature representations are still manually crafted based on domain knowledge, which would be likely to lose useful latent information. More specifically, the manually crafted attributes, such as sum-path-fan-out and maximum-fan-out, are calculated to characterize the prefix adder structures. However, these attributes only partially represent the prefix adder structure. Therefore, even with a lot of hand-engineered efforts, information loss is still inevitable. We argue that learning representations of prefix adder networks plays a critical role in our framework because they enable many downstream learning tasks, and a unified feature learning and model training paradigm could boost the DSE process.

In this article, we propose an end-to-end deep learning model, graph neural process (GNP), which outputs predictions and uncertainties based on the feature representations automatically learned from adder structures. With GNP as the surrogate model, in this article, we harness a sequential optimization algorithm [17], [18] to perform DSE in the physical solution space. The visualization of adder DSE is displayed in Fig. 1. Our main contributions are summarized as follows.

1) A variational graph autoencoder (VGAE) is built to extract features from prefix adder structures automatically.
2) A neural process (NP) is exploited as an alternative to the GP to reduce computational complexity.
3) A multibranch, end-to-end surrogate model (i.e., GNP), which incorporates a VGAE and a NP, is proposed.
4) A GNP-based sequential optimization algorithm to explore Pareto-optimal solutions is investigated.
5) The proposed optimization framework with the developed surrogate model uses less labeled data and achieves better Pareto frontiers.

The remainder of this article is organized as follows. Section II introduces some prior knowledge about prefix adder synthesis, multiobjective optimization, and then gives the problem formulation. Section III proposes our graph neural process, while Section IV discusses the sequential optimization framework for design space exploration. Section V presents the experimental results, followed by the conclusion in Section VI.

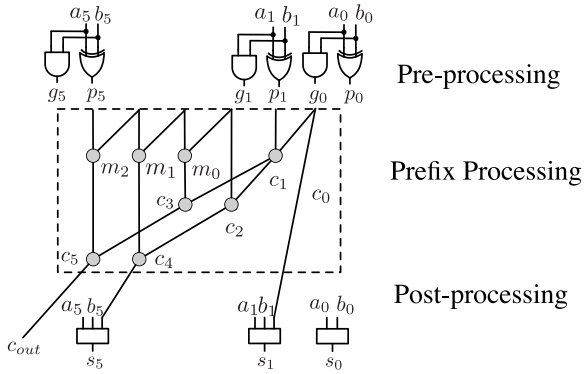

Fig. 2. 6-bit prefix computation process.

II. Preliminaries

In this section, the backgrounds of the prefix adder synthesis and multiobjective optimization are offered, and then we give the problem formulation.

A. Prefix Adder Network

An l bit binary adder eats two l bit inputs $\boldsymbol{A} = \{a_{l-1}..a_1, a_0\}$ and $\boldsymbol{B} = \{b_{l-1}..b_1, b_0\}$, and outputs the sum $\boldsymbol{S} = \{s_{l-1}..s_1, s_0\}$ with a carry out $C_{out} = c_{l-1}$, where $s_i = a_i \oplus b_i \oplus c_{i-1}$ and $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$. According to the recent study in [19], given the bitwise generation function and propagate function, l-bit binary addition can be represented as a prefix computation process. We visualize the computation process in Fig. 2. It can be seen that the process consists of three main steps. The first one is preprocessing where inputs for prefix processing (i.e., $\boldsymbol{g}$ and $\boldsymbol{p}$) are generated bitwisely. The second is prefix processing, which is the main carry-propagation step, and the last is postprocessing or carry-out generation.

The prefix processing or carry propagation network can be mapped to a prefix graph problem by taking the outputs of the preprocessing part as inputs and generates c_{out}. Each gray node in Fig. 2 is called a prefix node, which represents a certain logic operation. The logic operation of each prefix node in any prefix adder graph is the same except for the different inputs. In the prefix processing part showed in Fig. 2, we first extend g and p to multiple bits and define $G_{[i:j]}$, $P_{[i:j]}$ $(i \geq j)$ as

$$P_{[i:j]} = \begin{cases} p_i, & \text{if } i = j \\ P_{[i:k]} \cdot P_{[k-1:j]}, & \text{otherwise} \end{cases} \tag{1}$$

$$G_{[i:j]} = \begin{cases} g_i, & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, & \text{otherwise.} \end{cases} \tag{2}$$

The associative logic operation $\circ$ (i.e., any gray node in Fig. 2) is defined for (G, P) as

$$\begin{aligned} (G, P)_{[i:j]} &= (G, P)_{[i:k]} \circ (G, P)_{[k-1:j]} \\ &= \left(G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, P_{[i:k]} \cdot P_{[k-1:j]}\right). \end{aligned} \tag{3}$$

Especially, we use c_i to indicate the logic operation whose output involves in the s_i computation.

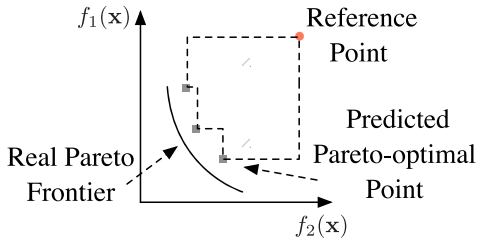

Fig. 3. Example of hypervolume in a biobjective space.

B. Multiobjective Optimization

Assume a multiobjective optimization problem has a set of feasible solutions $\mathcal{X}$. There are n objective functions, $f_1(\cdot), f_2(\cdot), \ldots, f_n(\cdot)$, which map an input $\boldsymbol{x}$ to corresponding results $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})$ and form an n-d result vector $\boldsymbol{f}(\boldsymbol{x})$. A result vector $\boldsymbol{f}(\boldsymbol{u})$ is said to dominate another result vector $\boldsymbol{f}(\boldsymbol{v})$ if $\boldsymbol{f}(\boldsymbol{u})$ is at least as good as $\boldsymbol{f}(\boldsymbol{v})$ in all the objectives, namely, $f_i(\boldsymbol{u}) \geq f_i(\boldsymbol{v})\ \forall i \in [1, n]$ if the objectives are to be maximized, and $f_i(\boldsymbol{u}) \leq f_i(\boldsymbol{v})\ \forall i \in [1, n]$ if the objectives are to be minimized. Hence, we say that a solution $\boldsymbol{x}$ is Pareto-optimal if it is not dominated by other solutions in the feasible solution set $\mathcal{X}$. Some prior arts [17], [18] in the machine learning field are proposed to handle the practical multiobjective problems where the evaluation of the multiobjective functions is expensive.

In our context for high-speed adder design, a feasible solution $\boldsymbol{x}$ is the feature representation of an adder design implementation, which satisfies the predetermined constraints, while a Pareto-optimal design is where none of the QoR metrics (or objectives), such as area, power, and delay, can be minimized without worsening at least one of the others. The Pareto set is the set consisting of all the Pareto-optimal solutions, and the Pareto frontier contains objective space values (i.e., the values of QoR metrics) associated with the Pareto set.

C. Problem Formulation

Definition 1 (Hypervolume [20]): The metric hypervolume is a Pareto-compliant evaluation, which refers to the volume fenced by the Pareto frontier and a reference point in the objective space. It measures how well distributed the points are on the Pareto frontier approximation.

In Fig. 3, the area filled with dash lines is an example of the hypervolume of a predicted Pareto-optimal set in a biobjective space. The hypervolume error for a predicted Pareto-optimal set $\hat{\mathcal{P}}$ is defined

$$\eta = \frac{V(\mathcal{P}) - V\left(\hat{\mathcal{P}}\right)}{V(\mathcal{P})} \tag{4}$$

where $\mathcal{P}$ is the golden Pareto-optimal set, and $V(\mathcal{P})$ is the ground truth of hypervolume. If a solution set $\mathcal{P}'$ is better than another set $\mathcal{P}''$, $V(\mathcal{P}')$ is greater than $V(\mathcal{P}'')$. It can be observed that a prediction $\hat{P}$, which contains the whole design space, has an error of 0.

Definition 2 (Average Distance From Reference Set (ADRS) [21]): Given a reference Pareto-optimal set $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots | \boldsymbol{a} = ({m_1}^a, {m_2}^a, \ldots, {m_n}^a)\}$ and an approximated Pareto-optimal set $\mathcal{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots | \boldsymbol{p} = ({m_1}^p, {m_2}^p, \ldots, {m_n}^p)\}$ in n-objective DSE problem

$$\mathrm{ADRS}(\mathcal{A}, \mathcal{P}) = \frac{1}{(|\mathcal{A}|)} \sum_{\boldsymbol{a} \in \mathcal{A}} \min_{\boldsymbol{p} \in \mathcal{P}} \delta(\boldsymbol{a}, \boldsymbol{p}) \tag{5}$$

where

$$\delta(\boldsymbol{a}, \boldsymbol{p}) = \max\left\{\left|\frac{{m_1}^p - {m_1}^a}{{m_1}^a}\right|, \ldots, \left|\frac{{m_n}^p - {m_n}^a}{{m_n}^a}\right|\right\}.$$

ADRS is used to quantify how close a set of nondominated points is from the Pareto frontier in the objective space. The smaller ADRS value is, the closer the approximate set P is to the reference set A.

With the aforementioned knowledge, our problem can be formulated.

Problem 1 (Adder Design Space Exploration): Given a collection of prefix adder networks, the aim of adder DSE is to search for the Pareto-optimal adder designs with tradeoff among multiple objectives, such as power, area, and delay over a wide design space.

III. Adder Feature Extraction and Regression

In traditional machine learning techniques, most of the features are manually crafted and based on domain knowledge [22], [23]. Followed up on the idea, previous machine learning-based adder DSE works, such as [3] and [6], exploit hand-engineered features, and the feature extractor and the consequent learning model are separated into two independent components. Isolating the two parts makes the whole framework be susceptible to converge to suboptimal performance.

Fortunately, deep Learning algorithms attempt to use a general-purpose learning procedure to automatically learn high-level features from data, which eliminates the need for domain expertise and formidable feature extraction [22], [23]. Many of deep learning models (e.g., AlexNet [24], GoogleNet [25], and ResNet [26]) perform feature extraction and classification in an end-to-end fashion or in a unified network structure. Recently, the end-to-end learning structure has also achieved notable success in oceans of EDA applications [27]–[32]. Based on the observations and arguments, we design an end-to-end, deep learning-based model, GNP, which incorporates the customized automatic feature extractor for prefix adder networks and the regressor.

In this section, the proposed GNP, which adopts a well-designed multibranch flow, is introduced. The proposed multibranch flow, shown in Fig. 4, has a backbone (i.e., the encoder part of a graph autoencoder) and simultaneously works on two branches: one is the decoder part of graph autoencoder and the other is the NP working as an alternative to the traditional Gaussian process. For the backbone and stream I, a graph autoencoder (GAE) that aims at summarizing the graph structure and attaining the latent representation of an input prefix adder is adopted. For stream II, an NP built upon the encoder–decoder structure outputs the regression values with corresponding uncertainties.

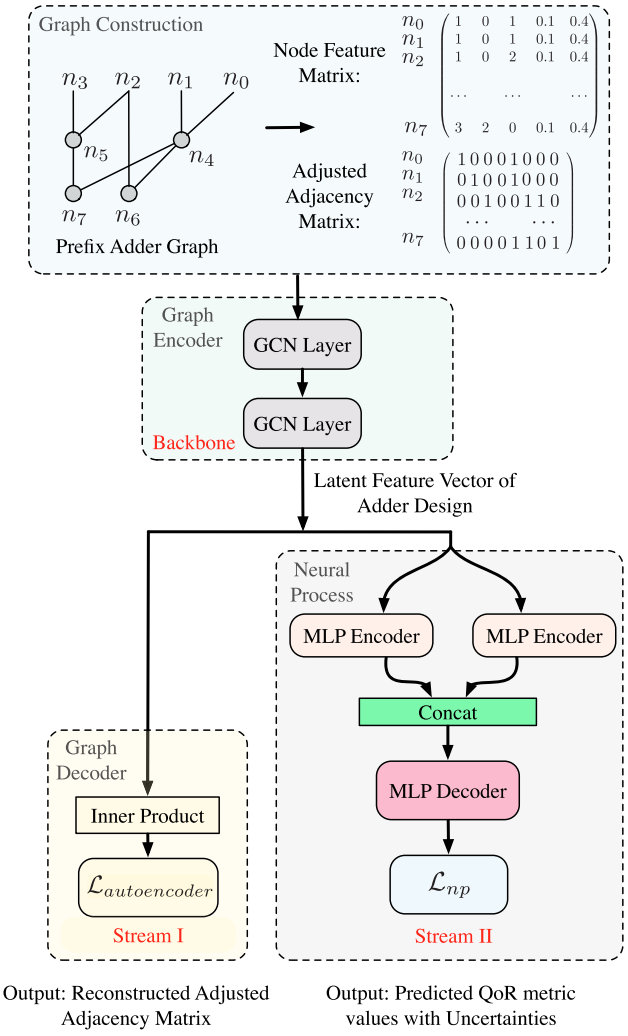

Fig. 4. Diagram of the proposed graph neural process.

A. Graph Construction of a Prefix Adder Network

Because the prefix adder structure is graph-like, the adjacency matrix and node feature matrix can describe it without any information loss. In the following descriptions, an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n vertices and m edges refers to a prefix adder network. $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is the adjusted adjacency matrix of $\mathcal{G}$, whilst $\boldsymbol{X} \in \mathbb{R}^{n \times s}$ indicates the attribute matrix of nodes. Note that it is customary to set diagonal elements in an adjacency matrix be zeros; however, it is added by an identity matrix $\boldsymbol{I}$ in the operation of a graph convolutional network (GCN). Accordingly, we term $\boldsymbol{A}$ as the adjusted adjacency matrix.

As aforementioned in Section II, the prefix adder network can be mapped to a prefix graph structure. For a better understanding, we list an example of the mapping in Fig. 4. According to our design, we regard inputs, outputs, and prefix nodes in a prefix graph as vanilla vertices in graph $\mathcal{G}$, while the logic relationship between two nodes is equivalent to an edge. Until now, the basic structure of $\mathcal{G}$ is constructed, and then the adjusted adjacency matrix $\boldsymbol{A}$ is built. Each row in $\boldsymbol{X}$ represents corresponding node attributes, which is composed of the logic level, in-degree, out-degree, and EDA tool settings. For each vertex, the in-degree means the number of vertices taking part in one logic operation, while the out-degree counts the number of vertices in higher logic levels connecting to the current vertex. Apart from the prefix graph structural features, tool settings from logical synthesis stage and physical design stage as other features are also considered as part of node features. We synthesize the adder structures by an industry-standard EDA synthesis tool [33], where we can configure the synthesis parameters for the adder. Different values of the synthesis parameters can result in different QoR metric (e.g., power/timing/area) values. More specifically, target delay and utilization are crucial parameters in synthesis flows, which define timing and area constraints. The tool adopts different strategies internally to satisfy that target delay, which we can hardly consider during prefix graph synthesis. On the other hand, changing utilization values can lead to significantly distinct layouts. So we consider these two synthesis parameters as attributes of a node. In fact, besides the target delay and utilization, we also attempt other tool settings. The optimization level setting in logical synthesis potentially impacts on the performance of adders, which can be configured by `compile` and `compile_ultra` commands with different options. However, after synthesizing, it is observed that the solutions generated with `compile_ultra` can significantly dominate the solutions generated by `compile`. Hence, this setting is fixed to `compile_ultra` level as we are aiming at superior designs.

For a better understanding, we have drawn the 4-bit prefix adder graph with its adjusted adjacency matrix and node feature matrix as an explicit example of mapping in the top part of Fig. 4. It can be seen that the 4-bit prefix adder graph has eight nodes, and the shape of the adjusted adjacency matrix is 8×8. "1" in the adjusted adjacency matrix refers to the existence of a logical relationship between two nodes, and "0" vice versa. The first row in $\boldsymbol{X}$ in Fig. 4 means that the n_0 node is in the logic level 1, and it is an input node connecting one node in the next logic level, while the EDA tool settings (target delay and utilization) are 0.1 and 0.4, respectively.

B. Backbone: The Encoder of Graph Autoencoder

It is widely known that in the graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph. Hence, the structure of a prefix adder network can be recognized as a tree-like graph. Recently, the GAEs [34] and VGAEs [34] are proposed as powerful node embedding methods to describe graphs. In view of this, we adopt a GAE structure to comprehensively capture the latent information of prefix adder. The encoder–decoder architecture is made up of two components. For the encoder part, it encodes adder graphs into the latent feature representations in a low dimensional vector space, while the decoder component tries to reconstruct the original graph structure based on the information passing through the encoding network.

Our design for Stream I is based on a two-layer GCN [35] encoder. The intuitive idea of the design is mapping each node

$i \in \mathcal{V}$ to a latent vector $\boldsymbol{z}_i \in \mathbb{R}^d$ ($d \ll n$). More precisely, the $n \times d$ matrix $\boldsymbol{Z}$ with all $\boldsymbol{z}_i$ vectors as rows is generated by the two-layer GCN processing $\boldsymbol{A}$.

We build a probabilistic model inferring latent variable $\boldsymbol{z}_i \in \mathbb{R}^d$ ($d \ll n$) that is the latent representation in an embedding space for each node i. The encoder parameterized by $\boldsymbol{\phi}$ is defined in (6), which is also called the inference model of the VGAE

$$q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A}) = \prod_{i=1}^{n} q_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{X},\boldsymbol{A}) = \prod_{i=1}^{n} \mathcal{N}\Big(\boldsymbol{z}_i|\boldsymbol{\mu}_i, \operatorname{diag}\Big(\boldsymbol{\sigma}_i{}^2\Big)\Big). \tag{6}$$

The parameters of approximated Gaussian distribution, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, are learned by a two-layer GCN. It takes the adjusted adjacency matrix $\boldsymbol{A}$ and the feature matrix $\boldsymbol{X}$ as inputs and generates the distribution parameters for the latent variable $\boldsymbol{Z}$. The first layer of the GCN outputs a lower dimensional feature matrix $\bar{\boldsymbol{X}}$ shown in

$$\bar{\boldsymbol{X}} = \operatorname{ReLU}\Big(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_0\Big) \tag{7}$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}(\boldsymbol{A})\boldsymbol{D}^{-1/2}$ with the degree matrix $\boldsymbol{D}$ is the symmetrically normalized adjacency matrix, and $\operatorname{ReLU} = \max(0, \cdot)$. The second layer of the GCN generates $\boldsymbol{\mu}$ and $\log\boldsymbol{\sigma}$, respectively

$$\boldsymbol{\mu} = \operatorname{GCN}_{\boldsymbol{\mu}}\Big(\tilde{\boldsymbol{A}}, \bar{\boldsymbol{X}}\Big) = \tilde{\boldsymbol{A}}\bar{\boldsymbol{X}}\boldsymbol{W}_1 \tag{8}$$

$$\log\boldsymbol{\sigma} = \operatorname{GCN}_{\boldsymbol{\sigma}}\Big(\tilde{\boldsymbol{A}}, \bar{\boldsymbol{X}}\Big) = \tilde{\boldsymbol{A}}\bar{\boldsymbol{X}}\boldsymbol{W}_1'. \tag{9}$$

Note that $\operatorname{GCN}_{\boldsymbol{\sigma}}(\tilde{\boldsymbol{A}}, \bar{\boldsymbol{X}})$ and $\operatorname{GCN}_{\boldsymbol{\mu}}(\tilde{\boldsymbol{A}}, \bar{\boldsymbol{X}})$ only share the first layer parameter $\boldsymbol{W}_0$. Two-layer GCN generates the parameters for a distribution where the latent variable $\boldsymbol{Z}$ can be sampled. Since sampling is unexpected to be involved in the backpropagation while $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ need to be kept in the computational graph to update the GCN layers, the reparameterization trick is exploited. So $\boldsymbol{Z}$ is not directly sampled from a normal distribution that is approximated by the encoder, instead, an auxiliary variable $\boldsymbol{\epsilon}_i$ is sampled from the standard normal distribution. By making use of the reparameterization trick, each row in the latent variable $\boldsymbol{Z}$ can be obtained according to (10). By taking the mean of rows of $\boldsymbol{Z}$, the latent representation for an adder is acquired

$$\boldsymbol{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_i \odot \boldsymbol{\sigma}_i \tag{10}$$

where $\boldsymbol{z}_i$, $\boldsymbol{\mu}_i$, and $\boldsymbol{\epsilon}_i$ are the rows of matrix $\boldsymbol{z}$, $\boldsymbol{\mu}$, and $\boldsymbol{\epsilon}$, respectively, and $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ with $\odot$ as the Hadamard product. Here is the proof in 1-D case for the reparameterization trick applied in (10), which can be readily extended to multidimensional case

$$\begin{aligned}
&\int \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right) dz \\
&= \int \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2\right] d\left(\frac{z-\mu}{\sigma}\right) \\
&= \int \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(\epsilon)^2\right] d(\epsilon).
\end{aligned} \tag{11}$$

Therefore, $z = \mu + \epsilon\sigma$.

C. Stream I: The Decoder of Graph Autoencoder

To reconstruct the graph, a simple inner product decoder parameterized by $\boldsymbol{\theta}$ is stacked to the aforementioned GCN layers. This module leverages the inner product between latent variable pairs $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$, shown in (12), to output the reconstructed adjusted adjacency matrix $\hat{\boldsymbol{A}}$. The reason for performing the inner product is that it could calculate the cosine similarity between rows in the latent variable $\boldsymbol{Z}$ with being invariant to the magnitude of the vectors. By applying the inner product on the latent variables $\boldsymbol{Z}$ and $\boldsymbol{Z}^\top$, the similarities among nodes in latent vector space can be learned to predict $\boldsymbol{A}$

$$p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p_{\boldsymbol{\theta}}\big(A_{ij}|\boldsymbol{z}_i, \boldsymbol{z}_j\big) \tag{12}$$

in which $p_{\boldsymbol{\theta}}(A_{ij} = 1|\boldsymbol{z}_i, \boldsymbol{z}_j) = \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$, $\sigma(\cdot)$ is the sigmoid activation function with $\sigma(x) = 1/(1 + e^{-x})$. Obviously, the larger the inner product $\boldsymbol{z}_i^\top \boldsymbol{z}_j$ is, the more likely nodes i and j will be connected.

As minimizing a reconstruction loss in GAE, the weights of neural networks in our model of Stream I are updated by maximizing a tractable variational evidence lower bound (ELBO) of the model's marginal likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{A})$ through gradients. This kind of trick is employed in [36] and [37] as well. Hence, the loss function $\mathcal{L}_{\text{autoencoder}}$ can be rewritten as minimizing the opposite of the ELBO, which is shown as follows:

$$\min_{\boldsymbol{\phi},\boldsymbol{Z},\boldsymbol{\theta}} \operatorname{KL}\big[q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})\|p(\boldsymbol{Z})\big] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z})\big]. \tag{13}$$

The complete derivation of (13) is based on Jensen's inequality. Bear in mind that for a convex function f with a random variable $\boldsymbol{x}$ as input, $\mathbb{E}[f(\boldsymbol{x})] \geq f(\mathbb{E}[\boldsymbol{x}])$ holds. On the contrary, if f is concave, $\mathbb{E}[f(\boldsymbol{x})] \leq f(\mathbb{E}[\boldsymbol{x}])$ holds

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{A}) &= \log \int p(\boldsymbol{Z}) p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}) d\boldsymbol{Z} \\
&= \log \int q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}) \frac{p(\boldsymbol{Z})}{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})} p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}) d\boldsymbol{Z} \\
&= \log\left(\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[\frac{p(\boldsymbol{Z})}{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})} p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z})\right]\right) \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[\log\left(\frac{p(\boldsymbol{Z})}{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})} p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z})\right)\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[\log \frac{p(\boldsymbol{Z})}{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] \\
&\quad + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z})\big] \\
&= -\operatorname{KL}\big[q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})\|p(\boldsymbol{Z})\big] \\
&\quad + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z})\big].
\end{aligned} \tag{14}$$

In (13), the first part, $\operatorname{KL}(q(\cdot)\|p(\cdot))$, is the Kullback–Leibler divergence between $q(\cdot)$ and $p(\cdot)$. It measures the "distance" between $q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})$ and $p(\boldsymbol{Z})$, where $p(\boldsymbol{Z}) = \prod_i p(\boldsymbol{z}_i) = \prod_i \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{0},\boldsymbol{I})$. The computational rule for KL divergence term is shown as follows. Since the loss can be calculated by each dimension of the latent variable $\boldsymbol{Z}$, we exemplified the calculation in 1-D dimension. Note that from (17) and (18), the integral of the probability density over

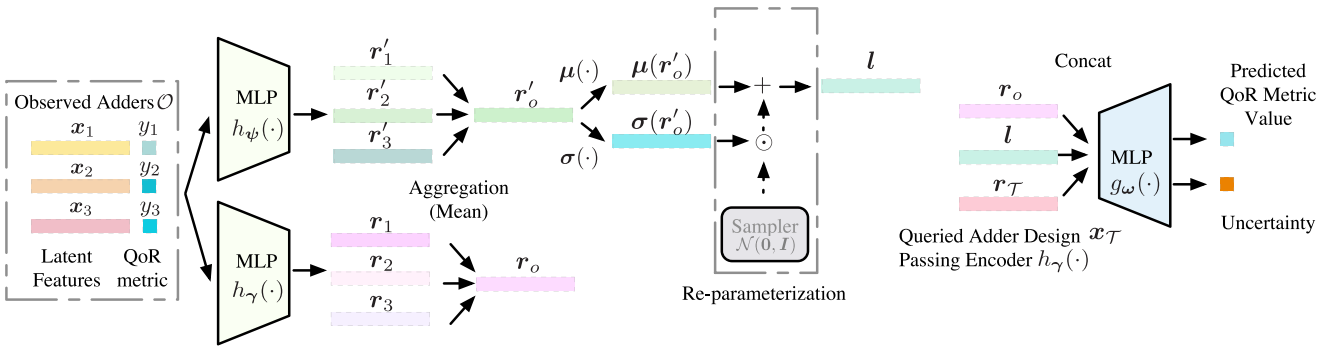

Fig. 5. Detailed regime of the neural process.

the whole space equalling to one, the mathematical definitions of the second moment and the variance are used. Since our decoder is a Bernoulli-based model, the second term can be readily transformed into the binary cross-entropy loss function. Generally, by gradually minimizing the reconstruction loss [i.e., (13)], we obtain increasing better latent representations of adders

$$\begin{aligned}
&\mathrm{KL}\Big(\mathcal{N}(\mu,\sigma^2)||\mathcal{N}(0,1)\Big) \\
&= \int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \left(\log \frac{e^{-(x-\mu)^2/2\sigma^2}/\sqrt{2\pi\sigma^2}}{e^{-x^2/2}/\sqrt{2\pi}}\right) dx \qquad (15)\\
&= \int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \\
&\quad \times \log\left\{\frac{1}{\sqrt{\sigma^2}} \exp\left\{\frac{1}{2}\left[x^2-(x-\mu)^2/\sigma^2\right]\right\}\right\} dx \qquad (16)\\
&= \frac{1}{2}\int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \\
&\quad \times \left[-\log\sigma^2 + x^2 - (x-\mu)^2/\sigma^2\right] dx \qquad (17)\\
&= -\frac{1}{2}\left[\log\sigma^2 - \left(\mu^2+\sigma^2\right)+1\right]. \qquad (18)
\end{aligned}$$

D. Stream II: Neural Processes

Function approximation plays a core role in numerous machine learning problems. One representative approach is the GP [38]. GPs do not require an expensive training stage and can perform inference about the ground-truth function conditioned on some observations, which makes them very flexible at testing. Nevertheless, traditional GPs are computationally costly due to cubical scaling concerning the number of data points. Thus, they are computationally expensive and their applicabilities are limited. On the other hand, a neural network (NN) can be regarded as a parameterized function. Recent works reveal that combining desirable properties of GPs and NNs, a collection of latent variables, which are modeled as neural networks, can learn an approximation of a stochastic process [39]–[41]. Like GPs, this kind of formulation termed as NP can provide the predictions as well as uncertainties on the QoR metric value of an adder design. Inheriting from NNs, NPs are more computationally efficient than GPs. Rudner *et al.* [42] mathematically demonstrated that under certain conditions, NPs are mathematically equivalent to GPs with deep kernels, while Garnelo *et al.* [40] experimentally demonstrated that when the number of context points is small, the NP outperforms the GP in terms of mean squared error metric in the image completion task. The conclusion is essential to our task since the adder DSE process works in a circumstance where the tool evaluation is expensive. Therefore, we adopt the NP in lieu of the Gaussian process.

1) Structure: In Stream II, we harness the NP as a good replacement to the traditional Gaussian process, which takes the latent representations of adders as input $\boldsymbol{x}$ and the associated QoR metric values as input $\boldsymbol{y}$ during training. NP is implemented in an encoder–decoder framework, which is given in Fig. 4. More specifically, the architecture includes two multilayer perceptrons (MLPs)-based encoders and an MLP-based decoder. One encoder $h_{\boldsymbol{\gamma}}(\cdot)$ maps the adder representations from the original space into the representation space to produce a representation $\boldsymbol{r}_i = h_{\boldsymbol{\gamma}}(\boldsymbol{x}_i, \boldsymbol{y}_i)\ \forall(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{O}$ for each of the pairs, where $\mathcal{O}$ represents a dataset consisting of observations in pairs $\{\boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}\}$ and $h_{\boldsymbol{\gamma}}(\cdot)$ stands for the encoder parametrized by $\boldsymbol{\gamma}$. Then, by taking the mean of these representations, $\boldsymbol{r}_o$ is generated. Another encoder $h_{\boldsymbol{\psi}}(\cdot)$ is built for parameterizing the distribution of a latent variable $\boldsymbol{l}$. Conditioned on observations, NPs define conditional distributions of the latent variable and thus, model a dataset $\mathcal{T}$ that is made up of a collection of target pairs defined as $\{\boldsymbol{x}_\mathcal{T}, \boldsymbol{y}_\mathcal{T}\}$. By virtue of the idea, the conditional decoder $g_{\boldsymbol{\omega}}(\cdot)$ treats the sampled global latent variable $\boldsymbol{l}$ and the input representation as well as target data $\boldsymbol{x}_\mathcal{T}$ (new queried data) as input and outputs the corresponding predictions with their uncertainties. The details of the NP are visualized in Fig. 5. To describe an NP, a Gaussian likelihood is given in

$$p\left(\boldsymbol{y}_\mathcal{T}|\boldsymbol{l}, \boldsymbol{x}_\mathcal{T}, \boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}\right) = p(\boldsymbol{l}) \prod_{j=1}^{|\mathcal{T}|} \mathcal{N}\Big(\boldsymbol{y}|g_{\boldsymbol{\omega}}(\boldsymbol{l}, \boldsymbol{r}_\mathcal{O}), \tau^{-1}\boldsymbol{I}\Big) \quad (19)$$

where the prior $p(\boldsymbol{l})$ is assumed as a multivariate standard normal distribution function. Following the same variational idea in VGAEs [34], to perform approximate inference in the NP, a variational posterior is defined in

$$q\left(\boldsymbol{l}|\boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}\right) = \mathcal{N}\Big(\boldsymbol{l}|\boldsymbol{\mu}\left(h_{\boldsymbol{\psi}}\left(\boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}\right)\right), \boldsymbol{\sigma}\left(h_{\boldsymbol{\psi}}\left(\boldsymbol{x}_\mathcal{O}, \boldsymbol{y}_\mathcal{O}\right)\right)\Big) \quad (20)$$

where $\boldsymbol{\mu}(\cdot)$ (i.e., mean or location) and $\boldsymbol{\sigma}(\cdot)$ (i.e., the diagonal of the covariance matrix) take aggregated and encoded input–output pairs as inputs and parameterize a normal distribution from which $\boldsymbol{l}$ is sampled [39]. The dense layer is often applied to mimic $\boldsymbol{\mu}(\cdot)$ and $\boldsymbol{\sigma}(\cdot)$. Intuitively, the latent variable $\boldsymbol{l}$ is designed to capture all information about the data-generating process needed to make predictions on the target inputs. Besides, introducing such a latent variable gives the most expressive model. Similarly, the reparameterization trick is performed for latent variable $\boldsymbol{l}$ due to exploiting the variational idea.

2) Loss Function: By using the variational distribution in (20), an ELBO on the log marginal likelihood is given as follows. Considering the observations and targets, (21) reflects the desired model behavior of an NP

$$\log p\big(\boldsymbol{y}_{\mathcal{T}}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}}\big) \geq \mathbb{E}_{q(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}})}\big[\log p\big(\boldsymbol{y}_{\mathcal{T}}|\boldsymbol{l},\boldsymbol{x}_{\mathcal{T}}\big)\big] - \mathrm{KL}\big(q\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}}\big)\|p\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}}\big)\big). \tag{21}$$

In (21), the intractable conditional prior $p(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}})$ replaces the prior $p(\boldsymbol{l}) = \mathcal{N}(\boldsymbol{l};\boldsymbol{0},\boldsymbol{I})$. By approximating the intractable conditional prior, (21) can be reformulated as

$$\log p\big(\boldsymbol{y}_{\mathcal{T}}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}}\big) \geq \mathbb{E}_{q(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}})}\big[\log p\big(\boldsymbol{y}_{\mathcal{T}}|\boldsymbol{l},\boldsymbol{x}_{\mathcal{T}}\big)\big] - \mathrm{KL}\big(q\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}}\big)\|q\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}}\big)\big). \tag{22}$$

Meanwhile, we acquire the loss function $\mathcal{L}_{np}$ [the opposite of the RHS of (22)]

$$\min_{\boldsymbol{\psi},\boldsymbol{\gamma},\boldsymbol{\omega}} \ \mathrm{KL}\big(q\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}}\big)\|q\big(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{O}},\boldsymbol{y}_{\mathcal{O}}\big)\big) - \mathbb{E}_{q(\boldsymbol{l}|\boldsymbol{x}_{\mathcal{T}},\boldsymbol{y}_{\mathcal{T}})}\big[\log p\big(\boldsymbol{y}_{\mathcal{T}}|\boldsymbol{l},\boldsymbol{x}_{\mathcal{T}}\big)\big]. \tag{23}$$

According to (23), the NP learns to reconstruct targets, regularized by a KL term that encourages the summary of the observations to be not too far from the summary of the targets.

E. Graph Neural Process

The loss function for the whole framework is the simple addition of losses of two branches [i.e., $\mathcal{L}_{\text{autoencoder}}$ in (13) and $\mathcal{L}_{np}$ in (23)]. During training, via backpropagation, the guide information (i.e., gradients) from the graph decoder and the NP updates each part, respectively, and then calibrates the encoder in GAE collectively. The two branches are jointly performed and mutually benefited. When conducting inference, the prefix adder first goes through the graph encoder to convert into the latent feature embedding, and then the embedding will be fed into the NP to acquire the prediction and uncertainty of the associated QoR metric value. Our GNP not only finds good graph feature embeddings of prefix adders but also builds a pretty good regressor for adder performance value. By the merit of the end-to-end nature, GNP is fast and flexible for both training and inference phases.

IV. Proposed DSE Framework

In the adder DSE problem, exhaustively determining golden QoR metric (e.g., area/power/delay) values of each adder design by running EDA flow is time intensive. The sequential optimization-based algorithm approximates the tool evaluations with a surrogate, which is cheaper to evaluate. Therefore, it is the cure to combat our problem. On the other hand, considering that the sequential-optimization model [17], [18] iteratively and incrementally samples the data to calibrate the surrogate model, the number of data points to train a machine learning-based surrogate model can be much less than the one in conventional DSE flow. By virtue of the incremental sampling stage, a method for simultaneous prediction and uncertainty estimation is in demand. Prior art [6] utilizes a GP as the surrogate model to offer predictions with uncertainties, which has high computational complexity.

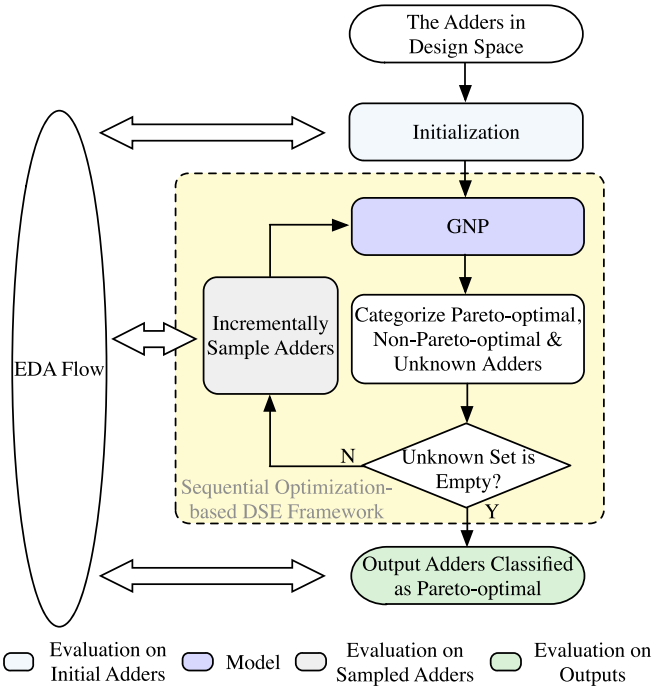

Fig. 6. Workflow of sequential optimization-based DSE framework.

As aforementioned in Section III, we incorporate the feature extractor into the sequential optimization framework so that the update of the learning model in the framework also benefits the feature extraction process. To reduce the high computation cost brought by a GP model, we exploit an NP, which is based on the NN structure. It can provide predictions as well as uncertainty estimations within a lower computational complexity [39], [40]. More importantly, the feature extractor is combined with the NP to form an end-to-end, multibranch model. Based on the above, we propose a sequential optimization-based DSE framework with the GNP as the surrogate model, which is shown in Fig. 6.

During the initialization, the proposed sequential optimization-based DSE framework interacts with EDA tools to obtain the golden QoR metric (area/power/delay) values of a small number of prefix adders, which are randomly sampled from the entire adder design space $\mathbb{E}$. Afterward, the DSE framework starts working iteratively. Our GNP is first calibrated with the initial data. The trained GNP outputs the QoR metric values of the adder designs in design space $\mathbb{E}$

with prediction uncertainties. The proposed DSE framework paradigm tries to classify the input adder designs based on the GNP's outputs into three classes: 1) Pareto-optimal; 2) nonPareto-optimal; and 3) unknown. During iterations, it incrementally selects the most representative adder designs as candidates for EDA flow (including synthesis, placement, and routing tools) evaluation toward a goal of minimizing the size of the unknown set. By harnessing the representative data along with their ground-truth QoR metric values, the GNP is updated. As more and more adder designs being selected, the GNP gets more and more accurate. The whole DSE process is terminated when the number of maximum iterations is reached or the unknown set is empty.

For the proposed DSE framework, the classification rules and selecting rules are based on the predictions and uncertainties offered by the GNP. More specifically, in each iteration, the GNP is invoked to infer the predictions as well as the uncertainties on QoR metrics over all unsampled adder $\boldsymbol{x}$ in the adder design space $\mathbb{E}$. A vector $\boldsymbol{m}(\boldsymbol{x})$, which includes concatenated QoR metric predictions (e.g., area, power, and delay) of $\boldsymbol{x}$ and a vector of corresponding standard deviations $\boldsymbol{\sigma}(\boldsymbol{x})$, is acquired. Consequently, a hyperrectangle $\mathcal{H}(\boldsymbol{x})$ is built to represent the prediction uncertainty in QoR metric space for a prefix adder design $\boldsymbol{x}$, which is defined as follows:

$$\mathcal{H}(\boldsymbol{x}) \coloneqq \left\{ \boldsymbol{y} \mid \boldsymbol{m}(\boldsymbol{x}) - \beta^{\frac{1}{2}}\boldsymbol{\sigma}(\boldsymbol{x}) \preceq \boldsymbol{y} \preceq \boldsymbol{m}(\boldsymbol{x}) + \beta^{\frac{1}{2}}\boldsymbol{\sigma}(\boldsymbol{x}) \right\} \quad (24)$$

where one element y_i in $\boldsymbol{y}$ (i.e., a point in the QoR metric space) refers to the coordinate of associated axes with $i \in \{1, 2, 3\}$ indicating different QoR metrics (area/power/delay), and β is a scaling parameter that controls the contribution of each element of $\boldsymbol{\sigma}$ to the hyperrectangle. The uncertainty information is exploited to guide the consequent sampling and to make a probabilistic assumption on the Pareto-optimality of every adder design $\boldsymbol{x}$. With the continuously updating GNP with new evaluated designs, the confidence of prediction increases. In light of ever-shrinking uncertainty, the uncertainty region of an adder design $\boldsymbol{x}$ in the tth iteration is written as

$$\mathcal{R}_t(\boldsymbol{x}) \coloneqq \mathcal{R}_{t-1}(\boldsymbol{x}) \cap \mathcal{H}(\boldsymbol{x}). \quad (25)$$

The initial $\mathcal{R}_{-1}$ is the entire objective space $\mathbb{R}^n$ with n as the number of QoR metrics. The iterative intersection guarantees the uncertainty regions are nonincreasing. In the uncertainty region $\mathcal{R}_t(\boldsymbol{x})$, the QoR metric performance upper bound comes from the optimistic prediction $\min(\mathcal{R}_t(\boldsymbol{x}))$, while the lower bound is associated with the pessimistic prediction $\max(\mathcal{R}_t(\boldsymbol{x}))$ of one certain adder design $\boldsymbol{x}$.

On account of the fact that the sequential optimization process in the DSE framework is monotonic, the numbers of designs in Pareto-optimal set $\mathcal{P}_t$ and nonPareto-optimal set $\mathcal{N}_t$ are nondecreasing regarding the iteration number t. In other words, at iteration t, the previous predicted design points in $\mathcal{P}_{t-1}$ and $\mathcal{N}_{t-1}$ remain their classification. Only designs in unknown set $\mathcal{U}_t$ need to be classified. With relaxing by a tolerance parameter $\boldsymbol{\epsilon}$ on both sides, the classification of a prefix adder design $\boldsymbol{x}$ abides by the following rules based on inequalities. If the pessimistic QoR metric prediction of $\boldsymbol{x}$, $\max(\mathcal{R}_t(\boldsymbol{x}))$, is not dominated by optimistic results of any other design $\boldsymbol{x}'$ in $\mathbb{E}$ [i.e., $\min(\mathcal{R}_t(\boldsymbol{x}'))$], then $\boldsymbol{x}$ is classified as Pareto-optimal

$$\max(\mathcal{R}_t(\boldsymbol{x})) - \boldsymbol{\epsilon} \preceq \min\left(\mathcal{R}_t\left(\boldsymbol{x}'\right)\right) + \boldsymbol{\epsilon}. \quad (26)$$

If the optimistic outcomes of $\boldsymbol{x}$, $\min(\mathcal{R}_t(\boldsymbol{x}))$, are dominated by the pessimistic QoR metric prediction of any other design $\boldsymbol{x}'$ in $\mathbb{E}$ [i.e., $\max(\mathcal{R}_t(\boldsymbol{x}'))$], then $\boldsymbol{x}$ is classified as nonPareto-optimal

$$\max\left(\mathcal{R}_t\left(\boldsymbol{x}'\right)\right) - \boldsymbol{\epsilon} \preceq \min(\mathcal{R}_t(\boldsymbol{x})) + \boldsymbol{\epsilon}. \quad (27)$$

If the above rules do not exist regarding $\boldsymbol{x}$, then $\boldsymbol{x}$ is still in the unknown set.

When the classification finishes, a prefix adder design $\boldsymbol{x}_t^s$ with the longest diagonal of its uncertainty region $\mathcal{R}_t(\boldsymbol{x})$ is sampled from Pareto-optimal and unknown categories for tool evaluation. The sampling rule can be written in

$$\boldsymbol{x}_t^s \coloneqq \arg\max_{\boldsymbol{y},\boldsymbol{y}' \in \mathcal{R}_t(\boldsymbol{x})} \left\| \boldsymbol{y} - \boldsymbol{y}' \right\|_2. \quad (28)$$

The GNP model calibration and prediction, the classification of prefix adder designs, and adder design incremental sampling perform alternatively in iterations until the stopping criteria (exceeds the maximum iterations or the unknown set is empty) meet. Eventually, the predicted Pareto-optimal adder designs are evaluated by running EDA flow. For a better understanding, we visualize the mathematical principles of the DSE framework in Fig. 7.

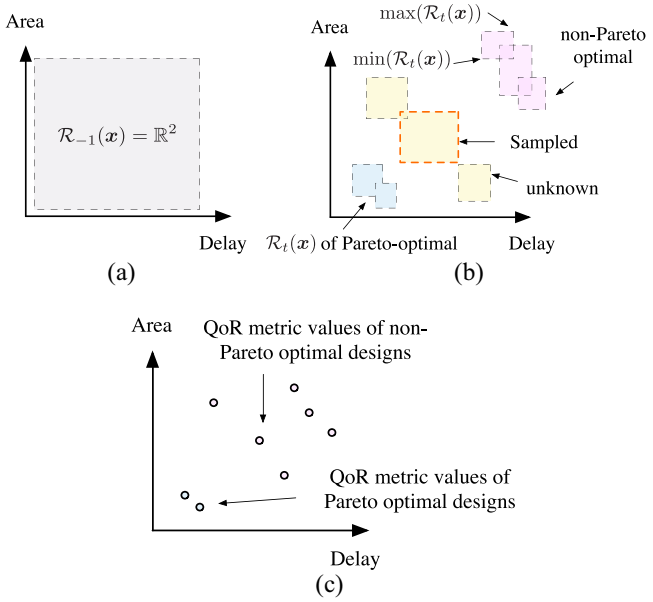

Fig. 7. Example of mathematical principles of the sequential optimization-based DSE framework. (a) Before starting. (b) Classification and sampling in one iteration. (c) After the termination of the stop criteria.

In a nutshell, the hyperparameters used in the DSE framework embrace the scaling parameter β controlling the hypervolume of a hyperrectangle when calculating the uncertainty region, the tolerance parameter ϵ in classification rules, and the max iterations $T_{\max}$ in the stop condition. For β, if we follow the value setting rule such as $\beta_t = 2\log(n|\mathbb{E}|\pi^2 t^2/(6\delta))$, a maximum hypervolume error can be achieved with a high probability $1 - \delta$ where $\delta \in (0, 1)$. This conclusion is proved in [18]. With regarding ϵ, we adopt the cross-validation

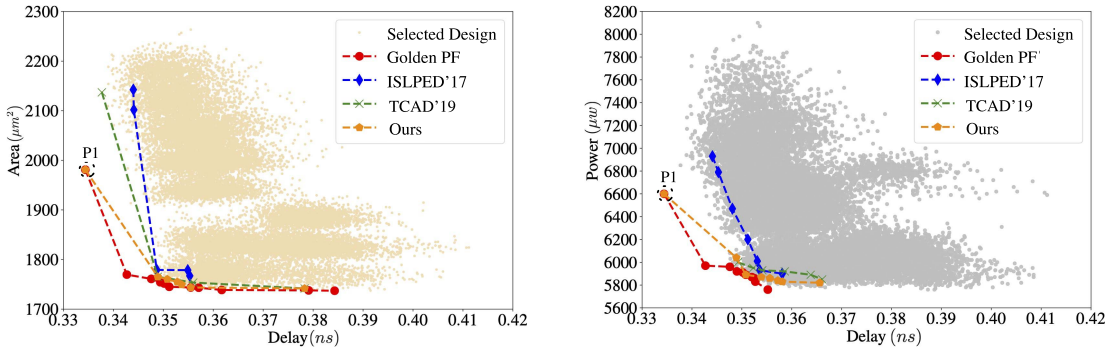

Fig. 8. (a) Pareto frontier: Area versus delay. (b) Pareto frontier: Power versus delay.

TABLE I
QUANTITATIVE COMPARISON OF THE PARETO FRONTIERS

Multi-objective	ISLPED'17 [3]			TCAD'19 [6]			Ours		
	HV	ADRS	Flows	HV	ADRS	Flows	HV	ADRS	Flows
`Area-Delay`	0.150	0.029	598	0.144	0.028	434	0.126	0.026	413
`Power-Delay`	0.155	0.017	615	0.168	0.021	482	0.128	0.010	422
`Area-Power-Delay`	0.110	0.019	1995	0.098	0.012	1636	0.073	0.008	1470
Average	0.138	0.022	1069	0.136	0.020	851	**0.109**	**0.015**	**768**
Ratio	1.266	1.467	1.392	1.248	1.333	1.108	**1.000**	**1.000**	**1.000**

method on our initial dataset to determine the value. When it comes to $T_{\max}$, theoretically, it is the smallest number satisfying

$$\frac{\sqrt{T\log\left(1-\sigma^{-2}\right)}}{4K\sqrt{\log\left(n|\mathbb{E}|\pi^2T^2/(6\delta)\right)\log^{d+1}T}} \geq \frac{na^{n-1}}{\eta(n-1)!} \tag{29}$$

where η refers to the hypervolume error, a indicates the maximum embedding distance between two adder designs after one iteration, d is the dimensionality of the latent feature vector of a prefix adder design, and n denotes the dimension of QoR metric space with σ as the standard deviation of the Gaussian distribution characterizing one QoR metric. In the practical experiment, we have found that the sequential optimization-based DSE process finishes classification for the entire design space $\mathbb{E}$ after ten iterations. We relax $T_{\max}$ to 20 so that the proposed whole framework can finish the optimization process within an acceptable sampling budget.

V. EXPERIMENTAL RESULTS

A. Experimental Settings

The implementation of our framework is in `Python` with the `Pytorch` library [43], and we test it on a platform with an Xeon Silver 4114 CPU processor and an nVIDIA TITAN Xp Graphic card. To verify the effectiveness and the efficiency of our framework, we compare our framework with the best-of-breed methods [3], [6] by exploring the 64-bit adder design space. First, we visually compare the quality of the Pareto frontiers found by the proposed framework and [3] and [6] in Fig. 8, and then quantitatively evaluate associated performance of these methods by the hypervolume error, the ADRS and the number of design flow runs in Table I. The former two

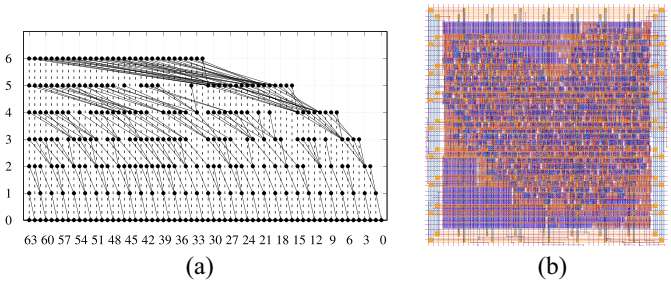

Fig. 9. Golden Pareto-optimal solution ("P1") that is not found by previous works. (a) Architecture overview: Bit width = 64, size = 234, Max. Level = 6, and Max. fanout = 6. The associated QoR metric values are 1981.13 μm^2 for the area, 6600 μw for the power, and 0.334 ns for the delay. (b) Corresponding layout snapshot.

metrics measure the quality of Pareto frontiers and the rest estimates time expense. Next, we exemplify one predicted Pareto-optimal prefix adder design found by the proposed algorithm in Fig. 9 to show the corresponding architecture and the layout. The corresponding time analyses of the exploration are provided by Figs. 10 and 11. Ultimately, the comparison between explored adders against some classical adders and a state-of-the-art adder synthesis algorithm [16] is depicted in Table II.

It is impossible to exhaustively enumerate all adder designs in the infinite design space for tool evaluation. A set of 22 000 adders, which is reasonable and comparatively large (almost 80 days are needed to run the flow), is generated and selected to represent the entire design solution space. This dataset is sampled in a quasirandom manner, which is based on a two-level (max-fan-out constraint and size) binning scheme. The approach evenly samples the prefix adders covering different

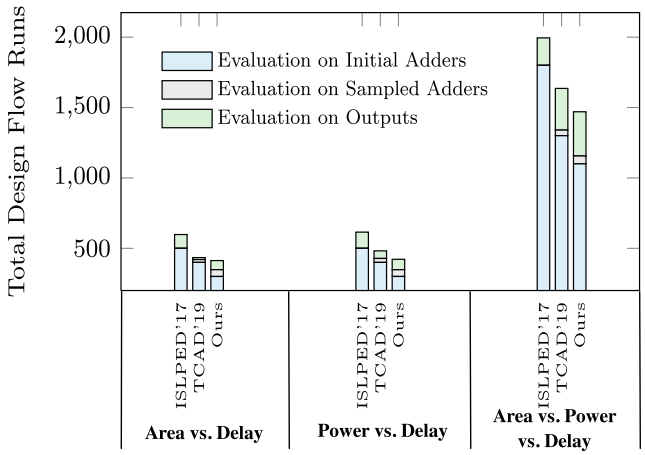

Fig. 10. Contributions of subprocesses of DSE works to total flow runs.

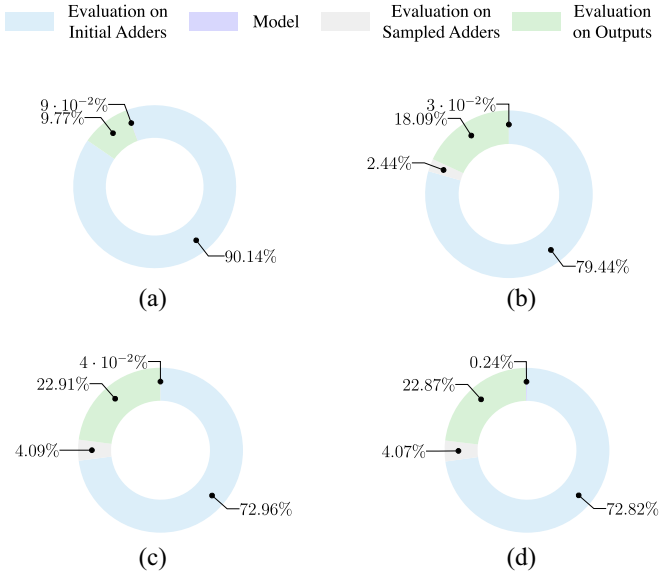

Fig. 11. Runtime breakdown of previous arts and our framework. (a) ISLPED'17. (b) TCAD'19. (c) Ours (with GPU). (d) Ours (only with CPU).

architectural bins. The first level of binning is based on the max-fan-out constraint. As many architectures may exist under the same max-fan-out constraint, another level relies on the size of the prefix adder. Note that we follow the prefix adder generation method proposed in [6], but the obtained design space is much larger than that in [6]. All the designs need to go through the whole design flow including front-end stages, such as logical synthesis and back-end steps, such as placement and routing. The tools in the design flow include design compiler [33] (version F-2011.09-SP3) for logical synthesis and IC Compiler [44] (version J-2014.09-SP5-3) for the placement and routing. "`tt1p05v125c`" corner and nonlinear delay model (NLDM) in 32-nm SAED cell-library for LVT class [45] (available by University Program) are used for technology mapping. The primary input activity of 0.1 is used along with 1-GHz operating frequency for power estimation. Concerning the tool settings, 0.1, 0.2, 0.3, and 0.4 ns are chosen as target delays. Utilization values are configured

TABLE II
COMPARISON WITH OTHER APPROACHES FOR 64-BIT ADDER

Method	Delay (ps)	Area (μm^2)	Energy (fJ/op)
DesignWare	346.5	2531.3	8160
Kogge-Stone	347.9	2563.7	8780
ISLPED'17-P1 [3]	344.1	2101.9	6930
TCAD'19-P1 [6]	339.0	2180.8	6930
Ours-P1	**334.4**	**1981.12**	**6600**
Sklansky	356.1	1792.5	6100
ISLPED'17-P2 [3]	353.5	1753.3	5980
TCAD'19-P2 [6]	353.0	1753.0	5900
Ours-P2	**353.0**	**1750.1**	**5830**
ASPDAC'15 [16]	348.7	1971.4	6980
ISLPED'17-P3 [3]	348.2	1969.8	6450
TCAD'19-P3 [6]	343.0	1912.6	6390
Ours-P3	**342.7**	**1769.6**	**5970**

with 0.4, 0.5, 0.6, 0.7, and 0.8. Per the design flow, it costs about 5.5 min. Although the involved generation overheads seem high, we utilize the dataset to demonstrate the superiority of the proposed methodology against SOTA DSE works and manual design processes for some classical adders. The proposed methodology can effectively aid and accelerate the adder design process under the current technology node. On the other hand, the dataset may lay the foundation of our future work, such as transfer learning among different bit-width adders or distinct technology nodes.

Our framework, as well as the previous works [3], [6], explores Pareto frontiers in area versus delay space, power versus delay space, and area versus power versus delay space. Fig. 8(a) and (b) visualizes the corresponding Pareto frontiers discovered by the prior arts [3], [6] and our framework in two kinds of 2-D objective spaces, respectively. In Fig. 8, each dot in the delay versus area or delay versus power space indicates the selected representative adder design after going through the design flow. Except the red dots indicate the associated positions of Pareto-optimal designs in QoR metric spaces with a straight line connecting these dots standing for the frontier, the other dash lines of different colors and with distinct markers refer to the Pareto frontiers found by the listed DSE algorithms. It is interesting to note that there is one point ["P1" outlined at the extreme left in Fig. 8(a) and (b)], which has 5ps better delay than those of other points, causing this point to be at some distance than the data cloud. Since we have connected the Pareto points by straight lines, the shift of this single point resulted in the "noticeable" shift from the cloud of data points. According to the observations of Fig. 8(a) and (b), both Pareto frontiers searched by our approach (orange lines) are much closer to the corresponding golden frontiers (red lines) than the frontiers explored by other methods. In other words, Fig. 8 qualitatively demonstrates our method finds the Pareto frontier of better quality.

B. Comparisons Against DSE-Based SOTA Works

The supplied Table I summarizes the qualities of Pareto frontiers including the Pareto frontiers of 2-D QoR metrics spaces presented in Fig. 8(a) and (b) and a 3-D case in quantification. Note that we utilize the golden Pareto set as the

reference set. Since the learning model establishment and calibration require far less time than design flows, we omit the model training and testing time. Our time metric mainly counts the cardinality of the dataset to train (or initialize) the machine learning-based surrogate model, and the final Pareto-optimal predictions for tool evaluation. The column "multiobjective" lists three QoR metric spaces: area versus delay, power versus delay, area versus power versus delay, whilst columns "HV," "ADRS," and "Flows" are the evaluation metrics in terms of hypervolume error, the ADRS, and the number of design flow runs. Columns "ISLPED'17," "TCAD'19," and "Ours" correspond to the results searched by [3] and [6] and our proposed sequential optimization-based DSE method with GNP as the surrogate. According to the results recorded in Table I, our algorithm averagely surpasses [3] by 21.0% less hypervolume error and 31.8% smaller ADRS value, and reduces 19.9% less hypervolume error and 25.0% ADRS value comparing to [6]. Moreover, the time expense is still less than those of [3] and [6]. In brief, our method performs better than the previous works [3], [6] with less hypervolume error, shorter ADRS, and fewer design flow calls. Here, we just illustrate one golden Pareto-optimal design ("P1"), which is not searched by [3] and [6] but found by ours. The architecture and layout are visualized in Fig. 9.

C. Time Analyses

The "Flows" metric in Table I indicates the number of design flow runs, coarsely consists of two parts.

1) The number of evaluated adder designs for training.
2) The number of predicted Pareto-optimal adder designs to go through the EDA flow for evaluation.

Different from the pure SVR regressor-based work [3], which does not involve an incremental sampling process, the training set in the active learning-based DSE flow [6] includes an initial dataset and incrementally sampled dataset. To extract the Pareto-frontier, EDA tools are invoked to acquire the real QoR metric values of the predicted Pareto-optimal adder designs of the three methods. We draw a stacked bar plot, Fig. 10, to show the contribution of each part to total flow runs in three QoR metric spaces. In Fig. 10, "Evaluation on Initial Adders" refers to the number of adder designs for evaluation during initialization, and "Evaluation on Sampled Adders" indicates the number of incrementally sampled adder designs during iterations in [6] and our work (both are four samples per iteration), while "Evaluation on Outputs" stands for the number of predicted Pareto-optimal designs by each methodology. As Fig. 10 suggests, the SVR regressor-based framework in [3] is trained with 500 adder designs when exploring Pareto frontier in area versus delay or power versus delay QoR metric space, and separately searches 98 and 115 Pareto-optimal adder designs for EDA flow evaluation in two metric spaces. The active learning-based work [6] initializes the GP with 400 adder designs in the area versus delay and power versus delay cases. After actively samples 20 (five iterations) and 28 (seven iterations) adder designs, it ultimately seeks 14 and 54 adder designs for evaluation in two metric spaces, respectively. The given bar charts in Fig. 10 represent only 300 adder samples are utilized for initialization in our proposed flow in 2-D QoR metric spaces, and both 48 (12 iterations) adder designs are incrementally selected to fine-tune the GNP model. Our flow predicts 65 and 74 Pareto-optimal samples to be evaluated by tools in area versus delay and power versus delay cases, respectively. For a more complicated task (i.e., optimization for area versus power versus delay QoR metrics), both the numbers of adders for training and EDA flow evaluation required by three methods surge upward. Roy *et al.* [3] calibrated the SVR regressor via 1800 samples and finds 195 samples, and Ma *et al.* [6] trained the surrogate model, GP regressor, by 1300 initial adder designs and 40 (10 iterations) additional selected samples and invokes EDA flow 296 times to evaluate the predictions, while our framework only harnesses 1100 adders to initialize the GNP model and 56 (14 iterations) adders for follow-up tuning, and 314 adders are regarded as Pareto-optimal solutions. It is conspicuous that compared with the active learning-based DSE framework [6] and ours, [3] needs more adders for initialization due to a lack of sampling process to update the model. On the other hand, despite the similar sampling process for updating surrogate models, the proposed framework reduces approximately 4.8%, 12.4%, and 10.1% time cost in area versus delay, power versus delay, and area versus power versus delay cases compared to [6]. Besides, the proposed framework requires the least adder designs for initialization, which alleviates the pressure for starting the high-quality design searching.

For a better demonstration, illustrations in Fig. 11 depict the time (in minute) spent on statistical models in area versus power versus delay case. A glance at the pie charts reveals that the time (denoted as "Model" in Fig. 11), including model training time and prediction time, is far less than the time spent on running EDA flows to generate golden QoR metric values of adder designs during initialization and outputs evaluation. As aforementioned, each EDA synthesis run takes about 5.5 min. The searching process totally costs [3] 10982.4 min, among which 3.8 and 6.1 min are used for building the machine learning model and performing predictions. Reference [6] spends about 9000.2 min to search for Pareto-optimal adder designs, and it takes only 1.7 and 0.5 min for training and inference. In our work with GPU acceleration, 7538.8 min are completely needed for design space exploration, where 1.26 and 2.53 min are consumed for training and testing. If we train and test the network only on CPU, the training and testing time climb to 8.5 and 9.4 min, respectively. Yet they still occupy extremely little proportion of whole runtime expenditure. It is worth mentioning that our framework directly extracts the features from adder designs without too much manual labor. On the contrary, both [3] and [6] feed 32-dimension, hand-crafted features to their machine learning model. The proposed GNP processes adjusted adjacency matrixes of the size up to 448×448 (64 bit and seven logic levels). Due to the different feature extraction processes and structures of features, our work behaves not as well as [6] when only considering model time. But the time cost on the model is still acceptable regarding the whole DSE process. Nevertheless, via the GPU acceleration, it outperforms [3] on model time.

D. Comparisons Against Classical Adders

Table II records the comparison between adder designs searched by aforementioned DSE methods against designware adders (best delay), an adder design generated by a highly sophisticated adder synthesis algorithm [16] as well as some classical adders, such as Kogge-Stone and Sklansky. These legacy adders are manually designed, which involve much human assiduous and masses of trials. When it turns to [16], a polynomial-time algorithm has been developed to generate prefix graph structures. The main idea behind [16] is to generate a single prefix graph network for a set of structural constraints, such as the logic level, fan-out, etc. Theoretically, the legacy adders and the traditional adder synthesis algorithm in [16] are not capable of handling the exploring task in a huge adder design space. To fairly compare with classical adders and conventional synthesis method [16], we select a slice of Pareto-optimal designs predicted by previous adder DSE methods[3], [6] and ours. For example, "ISLPED'17-P1" means one predicted Pareto-optimal design by [3], and "ISLPED'17-P2" stands for another solution in the explored Pareto-optimal set. It can be observed the designs searched by DSE methods behave better in all three targets (delay, area, and energy) than designware adders, Kogge-Stone adders, and the solution offered by [16]. Our solutions ("P1," "P2," and "P3") dominate the corresponding solutions explored by [3] and [6]. Notice that "P1" just happens to be the same solution point in the delay versus area and delay versus power Pareto-optimal curves, while other points ("P2" and "P3") are not outlined in Fig. 8 since they are just found in three QoR metric space case (area versus power versus delay). Table II implies that DSE methods are more effective than conventional adder solution providers, and more importantly, compared with the existing adder DSE methodologies, the proposed methodology can discover better adder designs.

In a nutshell, the proposed DSE method surrogated by GNP behaves better than previous arts. Additionally, our DSE framework has the potential to be generalized to different bit-width adder designs in theory. We utilize the 32-bit adder design as an exemplar for the following descriptions. There is no limitation on the size of data input of our DSE model. Besides, in terms of the graph structure and impact of tool settings, 32-bit adder designs are analogous to 64-bit adder designs to some extent. By harnessing certain transfer learning techniques [46]–[48] to achieve the domain adaptation of different bit-width adder designs, we can fine-tune a pretrained model (on 64-bit adder dataset) on a small amount of 32-bit adder designs.

VI. Conclusion

In this article, for the first time, we have proposed a new end-to-end learning model, graph neural process, where a GAE for prefix adder structures and a NP are simultaneously performed. The GNP provides a new solution to automatically extracting features from prefix adder structures. Besides, we have proposed a sequential optimization model-based DSE methodology with the GNP as its surrogate model to guide DSE for power-efficient, high-speed prefix adders. Our methodology is almost automatic from feature extraction to high-quality adder design space exploration. The experimental results have demonstrated the superiority of the proposed framework over the prior arts. With the VLSI designs becoming increasingly complicated, we expected to generalize our idea to settle more VLSI DSE problems (e.g., multiplier DSE problem, adder DSE issues among different bit width, and technology nodes.).

References

[1] Q. Guo, T. Chen, Y. Chen, Z.-H. Zhou, W. Hu, and Z. Xu, "Effective and efficient microprocessor design space exploration using unlabeled design configurations," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2011, pp. 1671–1677.

[2] D. Li, S. Yao, Y.-H. Liu, S. Wang, and X.-H. Sun, "Efficient design space exploration via statistical sampling and adaboost learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2016, pp. 1–6.

[3] S. Roy, Y. Ma, J. Miao, and B. Yu, "A learning bridge from architectural synthesis to physical design for exploring power efficient high-performance adders," in *Proc. IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2017, pp. 1–6.

[4] C. Lo and P. Chow, "Multi-fidelity optimization for high-level synthesis directives," in *Proc. Int. Conf. Field Programmable Logic Appl. (FPL)*, 2018, pp. 272–279.

[5] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *Proc. Int. Conf. Machine Learning (ICML)*, 2018, pp. 3312–3320.

[6] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A pareto driven machine learning approach," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 12, p. 2298–2311, Dec. 2019.

[7] D. Park and Y. Kim, "Fast pareto front exploration for design of reconfigurable energy storage," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 3, pp. 526–537, Mar. 2019.

[8] S. Zhang *et al.*, "An efficient multi-fidelity Bayesian optimization approach for analog circuit synthesis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[9] S. Zhang, F. Yang, D. Zhou, and X. Zeng, "An efficient asynchronous batch Bayesian optimization approach for analog circuit synthesis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[10] Q. Sun *et al.*, "Correlated multi-objective multi-fidelity optimization for HLS directives design," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, 2021, pp. 46–51.

[11] Q. Sun, C. Bai, H. Geng, and B. Yu, "Deep neural network hardware deployment optimization via advanced active learning," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, 2021, pp. 1510–1515.

[12] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: RISC-V boom microarchitecture design space exploration framework," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[13] J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electron. Comput.*, vol. EC-9, no. 2, pp. 226–231, Jun. 1960.

[14] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, Aug. 1973.

[15] T. Matsunaga and Y. Matsunaga, "Area minimization algorithm for parallel prefix adders under bitwise delay constraints," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI)*, 2007, pp. 435–440.

[16] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, "Polynomial time algorithm for area and power efficient adder synthesis in high-performance designs," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2015, pp. 249–254.

[17] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2011, pp. 2546–2554.

[18] M. Zuluaga, G. Sergent, A. Krause, and M. P$\acute{}$uschel, "Active learning for multi-objective optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013, pp. 462–470.

[19] B. R. Zeydel, T. T. J. H. Kluter, and V. G. Oklobdzija, "Efficient mapping of addition recurrence algorithms in CMOS," *Proc. IEEE Symp. Comput. Arithmetic (ARITH)*, 2005, pp. 107–113.

[20] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration," in *Proc. Int. Conf. Evol. Multi-Criterion Optim.*, 2007, pp. 862–876.
[21] C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, and L. Salomon, "Performance indicators in multiobjective optimization," *Eur. J. Oper. Res.*, vol. 292, no. 2, pp. 397–422, 2021.
[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 25, 2012, pp. 1097–1105.
[25] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.
[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.
[27] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 146.
[28] H. Geng *et al.*, "Hotspot detection via attention-based deep layout metric learning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2020, pp. 1–8.
[29] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Analog IC aging-induced degradation estimation via heterogeneous graph convolutional networks," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2021, pp. 898–903.
[30] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Deep H-GCN: Fast analog IC aging-induced degradation estimation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Aug. 24, 2021, doi: 10.1109/TCAD.2021.3107250.
[31] H. Geng, F. Yang, X. Zeng, and B. Yu, "When wafer failure pattern classification meets few-shot learning and self-supervised learning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–8.
[32] H. Geng *et al.*, "Hotspot detection via attention-based deep layout metric learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Sep. 14, 2021, doi: 10.1109/TCAD.2021.3112637.
[33] *Synopsys Design Compiler*. Accessed: Apr. 23, 2016. [Online]. Available: http://www.synopsys.com
[34] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. NeurIPS Workshop Bayesian Deep Learn.*, 2016, pp. 1–3.
[35] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 1024–1034.
[36] T. Chen, B. Lin, H. Geng, and B. Yu, "Sensor drift calibration via spatial correlation model in smart building," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
[37] T. Chen, B. Lin, H. Geng, S. Hu, and B. Yu, "Leveraging spatial correlation for sensor drift calibration in smart building," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1273–1286, Jul. 2021.
[38] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. Cambridge, MA, USA: MIT Press, 2006.
[39] M. Garnelo *et al.*, "Neural processes," in *Proc. ICML Workshop Theor. Found. Appl. Deep Generative Models*, 2018, pp. 1–11.
[40] M. Garnelo *et al.*, "Conditional neural processes," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 1690–1699.
[41] H. Kim *et al.*, "Attentive neural processes," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–18.
[42] T. G. J. Rudner, V. Fortuin, Y. W. Teh, and Y. Gal, "On the connection between neural processes and Gaussian processes with deep kernels," *Proc. NeurIPS Workshop Bayesian Deep Learn.*, 2018, pp. 1–6.
[43] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 8026–8037.
[44] *Synopsys IC Compiler*. Accessed: Apr. 23, 2016. [Online]. Available: http://www.synopsys.com
[45] *Synopsys SAED Library*. Accessed: Apr. 23, 2016. [Online]. Available: http://www.synopsys.com/Community/UniversityProgram/Pages/32–28nm-generic-library.aspx
[46] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[47] M. Kandemir, "Asymmetric transfer learning with deep Gaussian processes," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 730–738.
[48] M. Volpp *et al.*, "Meta-learning acquisition functions for transfer learning in Bayesian optimization," *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–22.

Hao Geng (Student Member, IEEE) received the M.E. degree from the Department of Electronic Engineering and Information Sciences, University of Science and Technology of China, Hefei, China, in 2015, and the M.Sc. degree in machine learning from the Department of Computing, the Imperial College London, London, U.K., in 2016. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include design space exploration, machine learning, deep learning, and optimization methods with applications in VLSI CAD.

Dr. Geng received the Best Paper Award nomination from ASPDAC in 2019.

Yuzhe Ma (Member, IEEE) received the B.E. degree from the Department of Microelectronics, Sun Yat-sen University, Guangzhou, China, in 2016, and the Ph.D. degree from the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, in 2020.

He has interned with Cadence Design Systems, San Jose, CA, USA, NVIDIA Research, Austin, TX, USA, and Tencent Youtu X-Lab, Shenzhen, China. His research interests include VLSI design for manufacturing, physical design, and machine learning on chips.

Dr. Ma received the Best Paper Award from ASPDAC in 2021, the Best Student Paper Award from ICTAI in 2019, the Best Paper Award Nomination from ASPDAC in 2019, and the Best Poster Research Award from Student Research Forum of ASPDAC in 2020.

Qi Xu (Member, IEEE) received the Ph.D. degree in electronic science and technology from University of Science and Technology of China (USTC), Hefei, China, in 2018.

He is currently an Associate Professor with the School of Microelectronics, USTC. His research interests include physical design automation and design for reliability for 3-D integrated circuits.

Jin Miao received the bachelor's degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2010, and the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin, Austin, TX, USA, in 2014.

After graduation, he has worked as a Research and Development Software Engineer with Cadence and Synopsys, San Jose, CA, USA. He is currently a Software Engineer with Google, Mountain View, CA, USA. He has broad interests in computer science and engineering, as well as emerging technologies.

Dr. Miao has been serving as a reviewer or TPC member for a number of journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, DAC, ASPDAC, and NEWCAS.

Subhendu Roy (Member, IEEE) received the B.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, in 2006, the M.Tech. degree in electronic systems from the Indian Institute of Technology Bombay, Mumbai, India, in 2009, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2015.

He is currently a Senior Principal Software Engineer with Cadence Design Systems, San Jose, CA, USA. Earlier he worked full-time in Intel, San Jose, and Atrenta, Noida, India (currently acquired by Synopsys), and as a summer intern with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, and Mentor Graphics, Fremont, CA, USA. He has worked on various areas, such as clock tree synthesis, adder synthesis, power optimization, gate-sizing, reliability, and machine learning-guided design space exploration.

Dr. Roy was a recipient of the Best Paper Award at ISPD'14. He has served as a reviewer in many journals/conferences, including IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I AND II, IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, TRANSACTION ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS, GLSVLSI, ISLPED, and ISCAS.

Bei Yu (Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received the seven Best Paper Awards from ASPDAC in 2021, ICTAI in 2019, Integration, the *VLSI Journal* in 2018, ISPD in 2017, SPIE Advanced Lithography Conference in 2016, ICCAD in 2013, ASPDAC in 2012, and six ICCAD/ISPD contest awards. He has served as the TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TCCPS Newsletter.