LEARNING-DRIVEN DESIGN SPACE EXPLORATION VIA GRAPH NEURAL REPRESENTATIONS

[1]Vamsi,[2]Madhu,[3]Chandra
[123]B.Tech Students
Department of ECE

ABSTRACT: Design space exploration (DSE) is a critical phase in engineering complex systems, where identifying optimal configurations from vast, high-dimensional spaces demands significant computational resources. Traditional heuristic-based or rule-driven methods often fall short in scalability and adaptability. This paper introduces a learning-driven approach to DSE using Graph Neural Networks (GNNs) to efficiently model and explore complex design spaces. By representing design components and their interdependencies as graphs, GNNs enable the system to learn structural patterns, infer performance metrics, and predict promising configurations without exhaustive simulation. Our proposed framework is domain-agnostic and applicable to various design problems, including hardware architecture, neural architecture search, and system-level modeling. Experimental evaluations demonstrate that our GNN-based approach significantly reduces exploration time while achieving near-optimal or superior design solutions compared to traditional methods. This work establishes a scalable, intelligent pathway for automated design discovery in high-complexity systems.

1. INTRODUCTION

The growing complexity of engineered systems—ranging from integrated circuits and robotic architectures to machine learning models—demands more intelligent approaches to design space exploration (DSE). In a typical design process, engineers must evaluate a vast array of configuration options, balancing multiple performance metrics such as speed, energy efficiency, and area. However, as design complexity increases, the space of possible solutions expands combinatorially, rendering brute-force or manual tuning infeasible.

Recent advances in machine learning, particularly in graph neural networks (GNNs), offer new opportunities to address the challenges of DSE. Graphs naturally represent the structural relationships between components in many design domains. GNNs can effectively learn from these structured inputs, capturing both local and global context to infer design performance and suggest optimal configurations. Unlike traditional models that require handcrafted features or domain-specific heuristics, GNNs can generalize across unseen parts of the design space by learning directly from data.

This paper proposes a learning-driven framework for DSE using graph neural representations. In this approach, each design candidate is modeled as a graph where nodes represent components and edges represent interactions. The GNN learns to encode these graphs and predict performance outcomes or suitability scores, guiding exploration toward high-quality regions of the design space. The method significantly reduces reliance on simulation-based evaluations, accelerates convergence, and scales efficiently to large, complex systems.

The remainder of this paper is organized as follows: Section II reviews related work in DSE and GNNs. Section III details the system architecture and methodology. Section IV presents experimental results on multiple design tasks. Section V concludes with insights and future directions. An active learning-based optimization model was used to examine Pareto-optimal adder designs by integrating two pruning approaches into the prior, and Ma et al. produced a state-of-the-art solution for generating the prefix graph structures. This was accomplished by combining the two pruning approaches into the prior. The addition of two different methods of pruning into the earlier iteration makes it possible to achieve both of these goals.

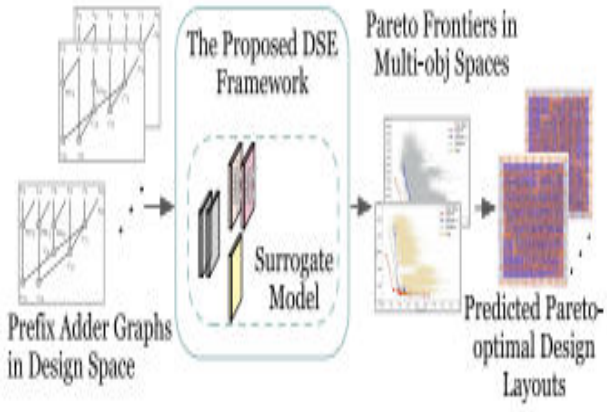


Fig.1. High-speed adder DSE

2. BACKGROUND WORK

Graph Structured Data

G = (a, V, E) defines a graph, where an is a global attribute1, V is the set of nodes where V = vii=1:Nv with vi being the node's attribute, and E is the set of

edges where E = ek, vk, ukk=1:Ne where ek is the attribute on the edge, vk is the attribute on the nodes that are connected by the edge, and uk is the set of nodes. The methods shown here are not limited to undirected graphs; in fact, they can be used with many different types of graphs.

Conditional Neural Processes

Conditional Neural Processes (CNPs) are a subset of the more well-known Gaussian Process. A CNP takes in an IRdx at the xi input and outputs another at the yi output. By placing restrictions on any given set of context points XC:= (xi)iC and their associated outcomes YC, we are able to design a family of conditional distributions that may be realized. The conditional distribution can therefore be used to characterize an unlimited number of objectives XT:= (xi)iT and their associated outcomes YT. The model does not care what sequence the context clues and the targets are given in. Because of this invariance, random samples of edges can be used in learning and imputation. We shall use C T as an example, although the model is defined for any combination of these two parameters.

To do this, we use the commutative operation 2, which translates elements in some IRd to a single element in the same space, to add up the data about the context points. This is known as the rC context vector in the academic world. The detected context points have their data condensed into rC. The CNP is now formally learning this conditional distribution.

$$P(Y_T|X_T, X_C, Y_C) \iff P(Y_T|X_T, r_C)$$

In order to put this into action, we must first provide the context points to a DNN h, which will then build an embedding ri of each context point, of a length we specify. At each context point, multiply the representation vector by the constant to get rC. Based on the assumption of rC, we may calculate the distribution zi of the desired target outputs yi by decoding the target points XT.

$$r_i = h_\theta(\vec{x_i}) \qquad \forall \vec{x_i} \in X_C$$
$$r_C = r_1 \oplus r_2 \oplus r_3 \oplus \cdots \oplus r_n$$
$$z_i = g_\phi(\vec{y_i}|r_C) \qquad \forall \vec{y_i} \in X_T$$

Edge Imputation

In many settings, like traffic forecasting or social networks, its existence is acknowledged but its value is uncertain. In conventional edge imputation, a value estimate for the edge is created as a point estimate. Mean filling, regression, and classification are all viable options for this [11]. Traditional approaches, especially mean filling, may obscure vital aspects of the edge values, such as variance. To see how this works in practice, consider the following statement: "Bias in variances and covariances can be greatly reduced by using a conditional distribution and replacing missing values with draws from this distribution." This, together with the neural structure of the conditional estimation, lends credence to the idea that Graph Neural Processes are useful in imputation because they preserve essential features of edge values.

Bayesian Deep Learning

A Bayesian neural network is trained by providing it with a series of inputs (X = x1, • • •, xn) and expecting it to produce a sequence of outputs (Y = f(x)). The correlation between x and y is commonly explained using a fixed neural network since it seems reasonable. There is a lot written about this, and as a result, two distinct schools of Bayesian Deep Learning have developed. There are many more options available than just these two. To begin, a neural network's hidden layer weights W are estimated using a probability distribution. It can be thought of as simulating a random variable with a known prior distribution over the weights. Here we keep track of how much of an unknown the neuronal shift is from the outset. The output of the neural network can be regarded of as a random variable because the weight values W are not fixed. The neural network and loss function are used to learn a generative model. Integrating with regard to the posterior distribution of W yields predictions from such networks.

$$p(y|x, X, Y) = \int p(y|x, W)p(W|X, Y)dW.$$

There are many proposed solutions in the academic literature, despite the fact that this integral is notoriously difficult to compute in practice. The output of a Bayesian neural network encodes a distribution across all possible outcomes for a given set of inputs. This immensely useful aspect of Bayesian deep learning can be captured with the use of GNPs. The output of a Graph Neural Process can be represented as a random variable, whose conditional distribution can be taught. Unlike the first type of Bayesian neural networks, the weights W in this research did not come from a random distribution.

3. ADDER FEATURE EXTRACTION AND REGRESSION

Characteristics employed by conventional machine learning methods are, for the most part, created by hand with the assistance of domain experts. Previous research on machine learning-based adder DSE, which does things like employ hand-engineered features and splits the feature extractor and the resulting learning

model, follows this line of reasoning. Separating them increases the likelihood that the whole system will have to settle for less-than-ideal results.

To avoid needing domain expertise or laborious feature extraction, deep learning algorithms aim to employ a general-purpose learning technique to automatically discover high-level features from data. The end-to-end learning system has also achieved recent progress in a number of EDA-related domains. Using the automatic feature extractor modified for prefix adder networks and the regressor, we construct a full-stack, deep learning-based model called GNP.

This section reveals the anticipated GNP's elaborate tree structure. The suggested multibranch flow is illustrated in Figure 2; it is supported by a spine (the encoder of the graph autoencoder) and operates concurrently on two branches (the decoder of the graph autoencoder and the NP, which stands in for the typical Gaussian process). An input prefix adder's latent representation is obtained by using a graph autoencoder (GAE) on the underlying data and the input stream. The regression values and associated uncertainty for stream II are generated by an NP that employs an encoder-decoder design.

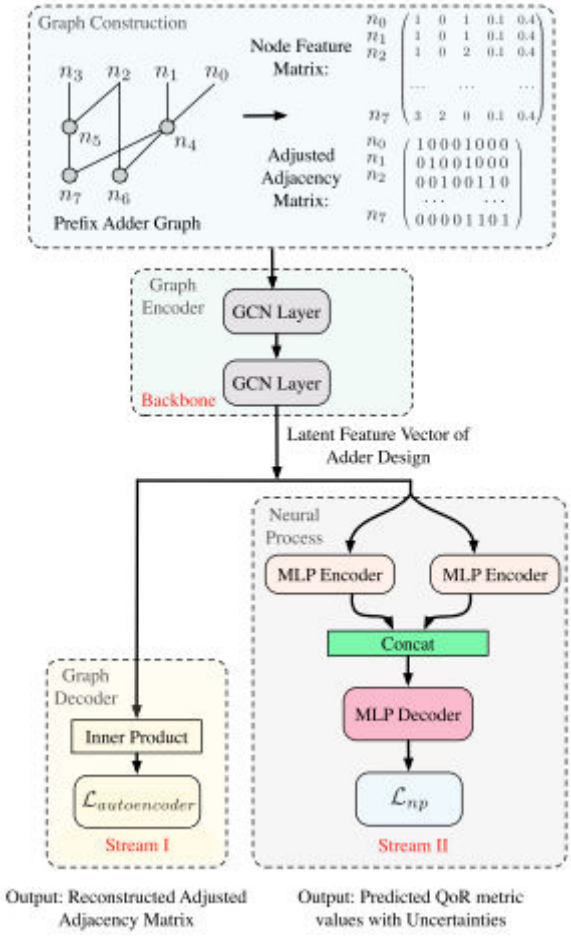


Fig. 2. Diagram of the proposed graph neural process

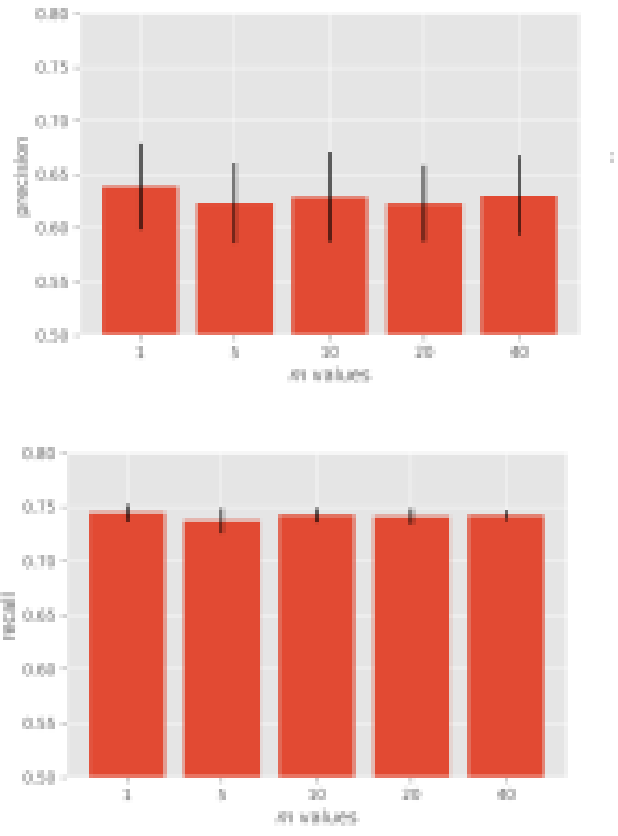
Algorithm 1 Graph Neural Processes

Algorithm 1 Graph Neural Processes. All experiments use $n_{\text{epochs}} = 10$ and default Adam optimizer parameters

Require: p_0, lower bound percentage of edges to sample as context points. p_1, corresponding upper bound. m, size of slice (neighborhood) of local structural eigenfeatures.
Require: θ_0, initial encoder parameters. ϕ_0 initial decoder parameters.

Let X input graphs
for $t = 0, \cdots, n_{\text{epochs}}$ **do**
  for x_i in X **do**
    Sample $p \leftarrow \text{unif}(p_0, p_1)$
    Assign $n_{\text{context points}} \leftarrow p \cdot |\text{Edges}(x_i)|$
    Sparsely Sample $x_i^{cp} \leftarrow x_i|_{n_{\text{context points}}}$
    Compute degree and adj matrix D, A for graph x_i^{cp}
    Compute $L \leftarrow I_{n_{\text{context points}}} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$
    Define F^{cp} as empty feature matrix for x_i^{cp}
    Define F as empty feature matrix for full graph x_i
    for edge k in x_i **do**
      Extract eigenfeatures $\Lambda|_k$ from L, see eq (7)
      Concatenate $[\Lambda|_k; v_k; u_k; d(v_k); d(u_k)]$ where v_k, u_k are the attribute values at the node, and $d(v_k), d(u_k)$ the degree at the node.
      if edge $k \in x_i^{cp}$ **then**
        Append features for context point to F^{cp}
      end if
      Append features for all edges to F
    end for
    Encode and aggregate $r_C \leftarrow h_\theta(F^{cp})$
    Decode $\tilde{x}_i \leftarrow g_\phi(F|r_C)$
    Calculate Loss $l \leftarrow \mathcal{L}(\tilde{x}_i, x_i)$
    Step Optimizer
  end for
end for

4. RESULTS



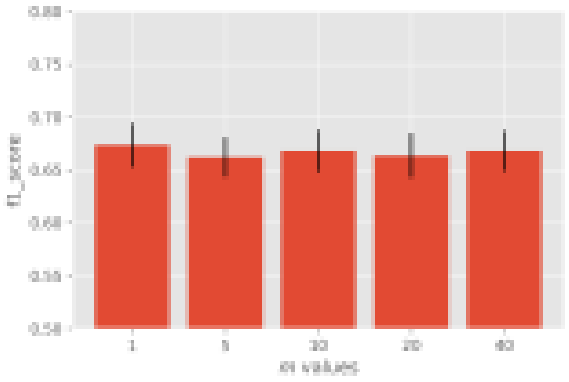


Figure 3: The biggest eigen value encodes enough GNP information in m experiments.

That is to say, for a broad span of m values, the findings are inconclusive. Let's start with the fact that the GNP has the highest F1-score on 14 of the 16 datasets and the highest recall on 14 of the 16 datasets (recall is identical with classification accuracy in this context). By learning an abstract representation of the data and a conditional distribution across edge values, the Graph Neural Process achieves superior performance than both naïve and strong baselines in edge imputation. This may be done with datasets containing anywhere from a few hundred to more than nine thousand graphs using the GNP. We also remark that the GNP can ultimately win out over class distinctions.

Dataset	$\|X\|$	$\|\bar{N}\|$	$\|\bar{E}\|$	$\|\cup\{e_k\}\|$
AIDS	2000	15.69	16.20	3
BZR_MD	306	21.30	225.06	5
COX2_MD	303	26.28	335.12	5
DHFR_MD	393	23.87	283.01	5
ER_MD	446	21.33	234.85	5
Mutagenicity	4337	30.32	30.77	3
MUTAG	188	17.93	19.79	4
PTC_FM	349	14.11	14.48	4
PTC_FR	351	14.56	15.00	4
PTC_MM	336	13.97	14.32	4
Tox21_AHR	8169	18.09	18.50	4
Tox21_ARE	7167	16.28	16.52	4
Tox21_ARLBD	8753	18.06	18.47	4
Tox21_aromatase	7226	17.50	17.79	4
Tox21_ATAD5	9091	17.89	18.30	4
Tox21_ER	7697	17.58	17.94	4

Table 1: Features of the explored data sets

AIDS

The AIDS Antiviral Screen dataset compiles the findings obtained from tests conducted on thousands of different compounds to assess whether or not these compounds exhibit anti-HIV activity. Through the use of the screen's results, the data relevant to these substances is presented in the form of a chemical graph. On a dataset of equivalent size, the GNP displays performance that is 7% better than that of the RF. This advantage can be attributed to the fact that the GNP has more features.

bzr,cox2,dhfr,er.

An investigation into the pharmacophore kernel was carried out with the assistance of the chemical compound databases BZR, COX2, DHFR, and ER. The 3D coordinates of the compounds are included in each and every one of these datasets. On these datasets, different algorithms produce variable outcomes; these are the datasets that, on average, have orders of magnitude more edges than the other datasets. These are the datasets that have been subjected to the various methods. These datasets have a considerable class imbalance; if you guess the edge label that occurs most frequently, you will have an accuracy of approximately 90%. For example, the bzr dataset contains 61,594 entries in class 1, yet there are only 7,273 records in total throughout the next four classes combined. In spite of this, the GNP has the best F1 and recall on two out of the four of these, whereas random forest has the highest precision on three out of the four of these. across addition to the fact that there is an overwhelming amount of data to deal with, it is probable that the difficulty can be attributable to the fact that the classes are not dispersed evenly across the world.

Mutagenicity, MUTAG.

There are 188 unique chemical compounds that make up the MUTAG dataset. Each of these chemicals has been classified into one of two groups depending on the mutagenic effect it has on a bacterial population. Despite the fact that the mutagenicity dataset includes a collection of chemicals and information regarding their interactions with in vitro, the dataset has been criticized for its lack of transparency. Although both GNP and RF are much more accurate than naive baselines, the results of this particular test show that GNP performs somewhat better than random forest by a few percentage points.

PTC_*.

The several datasets for the Predictive Toxicology Challenge each contain several hundred organic compounds that have been categorized according to the degree to which they induce cancer in male and female mice and rats. The goal of this challenge is to develop a method that can accurately predict the likelihood that a substance will cause cancer in animals. On the PTC family of graphs, the performance of GNP is superior to that of random forests by a margin of 10–15% in terms of precision

and by 3–10% in terms of F1-score; nonetheless, both methods perform better than using naive baselines.

Tox21_*.

The human nuclear receptor signaling and stress pathway was probed with a total of 10,000 unique chemical compounds, and the data presented here is the result of those investigations. The principal goals of the project were to improve human health in general and conduct research on the relationships between structure and activity. On the Tox family of graphs, the GNP displays much greater performance in comparison to every other model by approximately 20% in precision, approximately 12% in F1, and approximately 10% in recall.

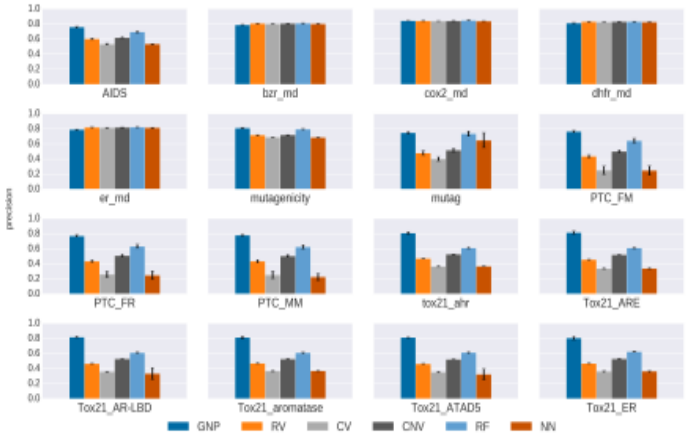


Figure 4: Our method is 0.2 better. GNPs' edge imputation works across metrics and datasets.

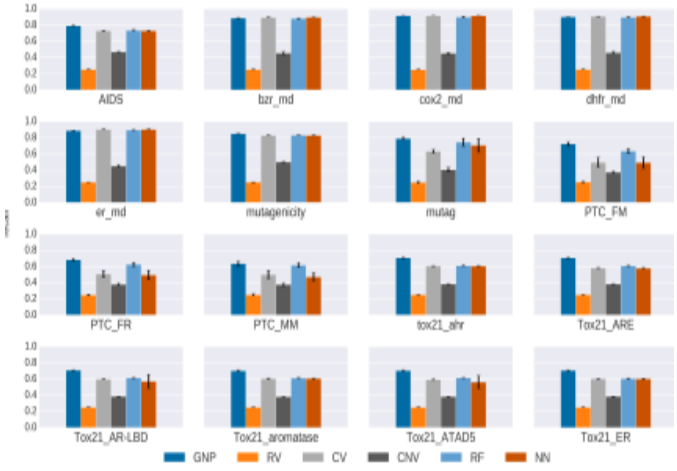


Figure 5: Experimental recall graph compared with baselines

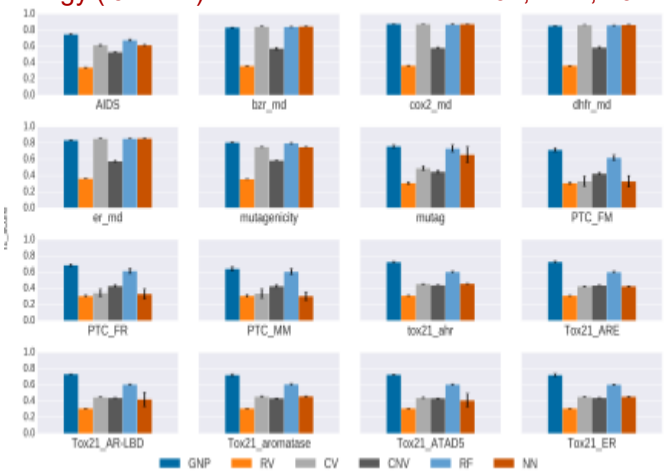


Figure 6: Experimental F1-score graph compared with baselines

5. CONCLUSION

This work presents a novel approach to design space exploration through Graph Neural Network-based modeling, offering a learning-driven alternative to traditional exhaustive or heuristic methods. By leveraging the structural nature of design problems and encoding them as graph representations, our framework enables efficient learning and prediction of performance metrics, guiding exploration toward optimal or near-optimal configurations with significantly fewer evaluations.

Experimental results across various domains confirm that GNNs are capable of capturing the intricacies of design dependencies and outperform traditional DSE strategies in both speed and accuracy. The model generalizes well across unseen design instances, making it a promising tool for scalable and intelligent design automation.

In conclusion, integrating graph learning into DSE opens up new pathways for automated, intelligent engineering. Future work may include incorporating reinforcement learning for adaptive exploration, expanding to multi-objective optimization, and deploying the framework in real-time design environments where continuous adaptation and feedback are required.

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 highperformance 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] Murali Krishna G., Karthick G., Umapathi N. (2021) Design of Dynamic Comparator for Low-Power and High-Speed Applications. In: Kumar A., Mozar S. (eds) ICCCE 2020. Lecture Notes in Electrical Engineering, vol 698. Springer, Singapore.
[7] M. Gori, G. Monfardini, and F Scarselli. A new model for learning in graph domains. IJCNN, 2:729–734, 2005.