2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129393

PreAxC: E
Approximate
G

Lakshmi Sath

[2]Indian
[3]UC Sant

Abstract—**While Approximate Comp
ing technique to trade off accuracy f
fundamental challenge is the lack of a
error models of AxC applications. In
PreAxC, a novel error modeling and
designs. Instead of using simple error
work, we use error distribution for A
with input awareness. We propose grap
based methods to predict the error d
grams, which are represented as data
propose two approaches: model-free a
the former directly predicts the error
and the latter models the distribution
Model (GMM) and predicts the GMM
results demonstrate that our approache
error statistics and can successfully pred
especially the model-free approach, eve
graphs (representing new AxC progran**

Index Terms—**Approximate Computi
ror Distribution Prediction, Graph Neu**

I. Introducti

Approximate Computing (AxC) i
that slightly trades off accuracy in
efficient circuit design for applicatio
error resilient [1]–[3]. AxC design ca
and circuit-level; in this work, we fo
where algorithms can be represente
(DFGs). In one DFG, some operatio
functional units (FUs), while others u
it one DFG *configuration.*

AxC Challenges. Despite the grea
a large amount of existing study, the
lenges. First, one of the greatest challe
although AxC applications are error to
still need to be carefully managed an
threshold. For instance, in image proc
to-noise ratio (PSNR) should be less t
error management requires **accurate error modeling and prediction**, meaning that the AxC design output error must be accurately predictable to avoid large quality degradation.

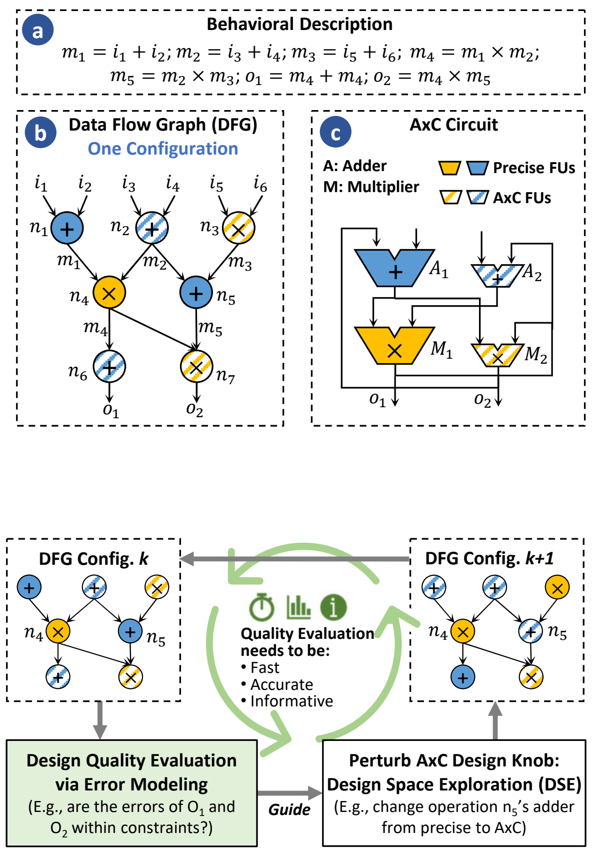
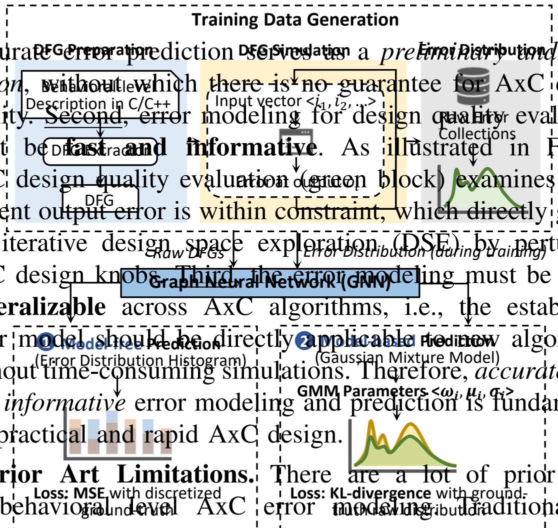

a
Behavioral Description
$m_1 = i_1 + i_2; m_2 = i_3 + i_4; m_3 = i_5 + i_6; m_4 = m_1 \times m_2;$
$m_5 = m_2 \times m_3; o_1 = m_4 + m_4; o_2 = m_4 \times m_5$
b
Data Flow Graph (DFG)
One Configuration
c
AxC Circuit
A: Adder
M: Multiplier
Precise FUs
AxC FUs
DFG Config. k
DFG Config. k+1
Quality Evaluation needs to be:
• Fast
• Accurate
• Informative
Design Quality Evaluation via Error Modeling
(E.g., are the errors of O_1 and O_2 within constraints?)
Guide
Perturb AxC Design Knob: Design Space Exploration (DSE)
(E.g., change operation n_5's adder from precise to AxC)

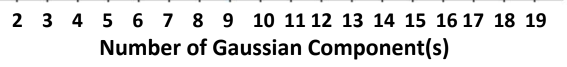
Fig. 1. **Motivation for fast, accurate, and informative error modeling for design quality evaluation.**

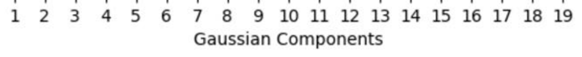

Training Data Generation
DFG Preparation
DFG Simulation
Error Distribution
Behavioral-level Description in C/C++
Input vector $<i_1, i_2, ...>$
Raw Error Collections
DFG Extraction
Error Output
DFG
Raw DFGs
Error Distribution (during training)
Graph Neural Network (GNN)
1 Model-free Prediction
(Error Distribution Histogram)
2 Model-based Prediction
(Gaussian Mixture Model)
GMM Parameters $<\omega_i, \mu_i, \sigma_i>$
Loss: MSE with discretized ground-truth
Loss: KL-divergence with ground-truth distribution

Accurate error prediction serves as a *preliminary and foundation*, without which there is no guarantee for AxC design quality. Second, error modeling for design quality evaluation must be **fast and informative**. As illustrated in Fig. 1, AxC design quality evaluation (green block) examines if the current output error is within constraint, which directly guides the iterative design space exploration (DSE) by perturbing AxC design knobs. Third, the error modeling must be **easily generalizable** across AxC algorithms, i.e., the established error model should be directly applicable to new algorithms without time-consuming simulations. Therefore, *accurate, fast, and informative* error modeling and prediction is fundamental for practical and rapid AxC design.

Prior Art Limitations. There are a lot of prior work on behavioral-level AxC error modeling. Traditional approaches apply statistical analysis [4]–[6] to roughly analyze the output error. Recently, ML-based algorithms have been proposed, such as Bayesian network [7], [8], Approxilyzer [9], Rumba [10], and AXNet [11]. Both static and ML-based approaches analyze output error statistics, including error mean, error variance, or error level (e.g., acceptable, unacceptable, precise). However, there are three major **limitations**. First, the modeling statistics are too simple, uninformative, and can be far from accurate (e.g., 10% to 20% estimation error [6]). One reason is that, predicting error mean and variance implies that the error follows Gaussian distribution, which is *not true* based on our observation. As shown in Fig. 2(a), we randomly visu-

Contact Authors: (LS, CH) {lsathidevi3, callie.hao}@gatech.edu





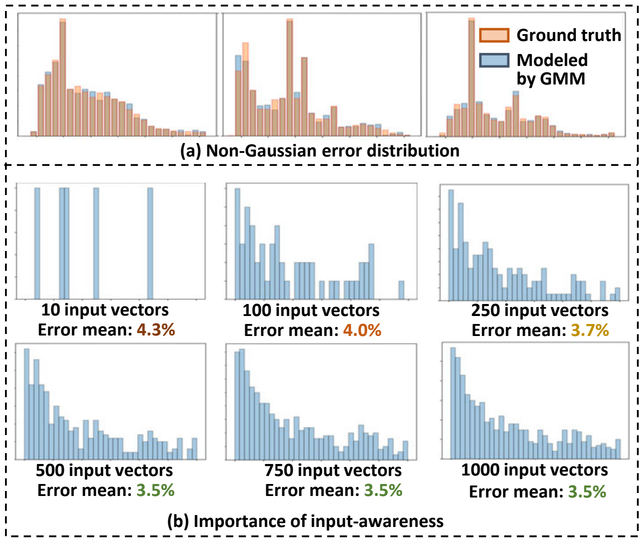


Fig. 2. **(a) Non-Gaussian error distribution can be nicely modeled by a 9-component Gaussian Mixture Model (GMM). (b) Importance of input-**

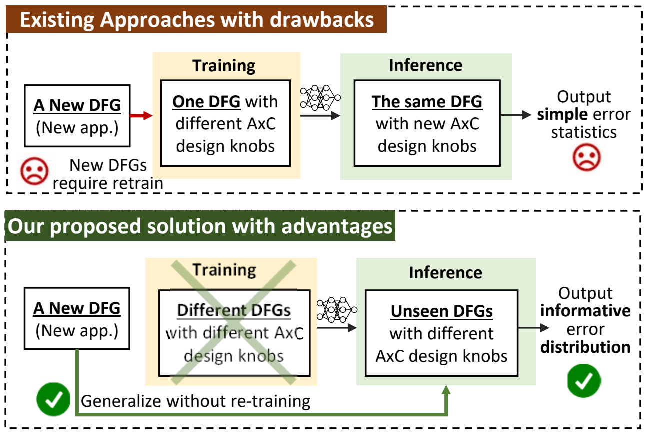
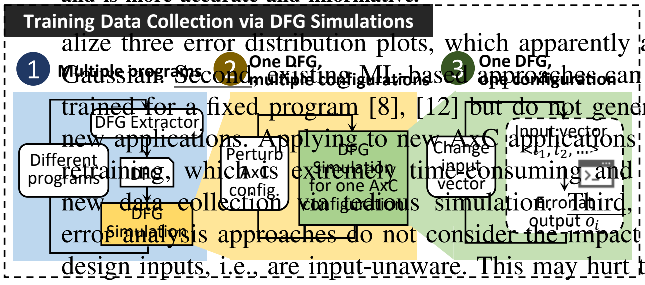
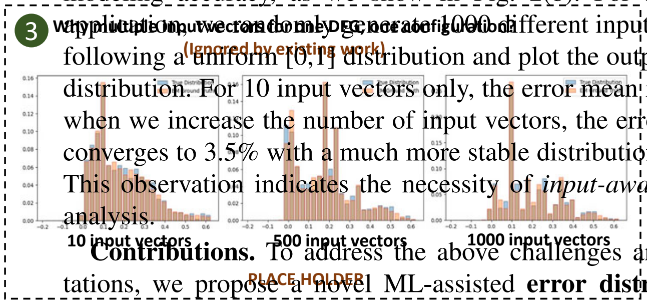

Existing Approaches with drawbacks
A New DFG
(New app.)
Training
One DFG with different AxC design knobs
Inference
The same DFG with new AxC design knobs
Output simple error statistics
New DFGs require retrain
Our proposed solution with advantages
A New DFG
(New app.)
Training
Different DFGs with different AxC design knobs
Inference
Unseen DFGs with different AxC design knobs
Output informative error distribution
Generalize without re-training

Fig. 3. **Existing approaches v.s. proposed PreAxC. PreAxC is generalizable without retrain, predicts error distribution with input-awareness, and is more accurate and informative.**


Training Data Collection via DFG Simulations
Multiple programs
One DFG, multiple configurations
One DFG, one configuration
DFG Extractor
Different programs
DFG
DFG Simulation
Perturb AxC config.
DFG Simulation for one AxC configuration
Change input vector
Input vector
Error of output

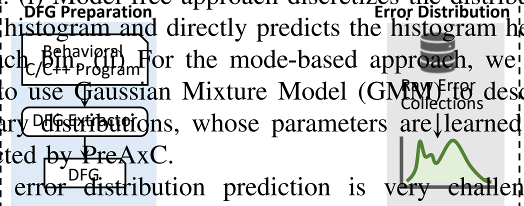


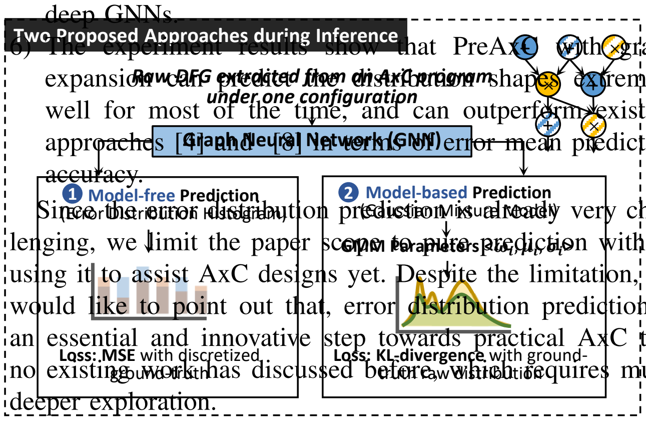
alize three error distribution plots, which apparently are non-Gaussian. Second, existing ML-based approaches can only be trained for a fixed program [8], [12] but do not generalize to new applications. Applying to new AxC applications requires retraining, which is extremely time-consuming and requires new data collection for tedious simulation. Third, existing error analysis approaches do not consider the impact of AxC design inputs, i.e., are input-unaware. This may hurt the error modeling accuracy, as we show in Fig. 2(b). For an AxC application, we randomly generate 1000 different input vectors following a uniform [0,1] distribution and plot the output error distribution. For 10 input vectors only, the error mean is 4.3%; when we increase the number of input vectors, the error mean converges to 3.5% with a much more stable distribution shape. This observation indicates the necessity of *input-aware* error analysis.

Contributions. To address the above challenges and limitations, we propose a novel ML-assisted **error distribution modeling and prediction for behavioral-level AxC**, named **PreAxC**, which is input-aware, informative, and generalizable, as shown in Fig. 3. It is equipped with a graph neural network (GNN) backbone and a distribution prediction head. Our contributions include:

1) PreAxC is **informative and accurate**: it is the first output **error distribution** modeling for AxC at behavioral-level. Comparing with the traditional error statistic analysis such as error mean and variance, error distribution is far more informative and precise (and yet much more challenging to predict).
2) PreAxC is **generalizable**: it can be directly applied to unseen AxC algorithms by running inference solely on the DFG, without any data re-collection and re-training (Fig. 3). It is empowered by a GNN backbone for graph learning, whose inductive capability enables generalization to unseen applications.
3) PreAxC is **input-aware**: we collect training data by running sufficient simulations using randomly sampled input vectors, until obtaining a stable error distribution. This ensures that the error prediction is robust against inputs that may be outliers.
4) On top of the GNN backbone, we propose two detection heads, to predict the distribution: **model-free** and **model-based**. (i) Model-free approach discretizes the distribution using histogram and directly predicts the histogram height for each bin. (ii) For the mode-based approach, we propose to use Gaussian Mixture Model (GMM) to describe arbitrary distributions, whose parameters are learned and predicted by PreAxC.
5) Since error distribution prediction is very challenging (more details in Sec. III-C), we propose a **graph expansion** technique by adding auxiliary edges to the DFG to improve the GNN message passing efficiency. It can greatly boost the prediction performance and reduce the necessity of deep GNNs.
6) The experiment results show that PreAxC with graph expansion can predict the distribution shapes extremely well for most of the time, and can outperform existing approaches [4] and [8] in terms of error mean prediction accuracy.

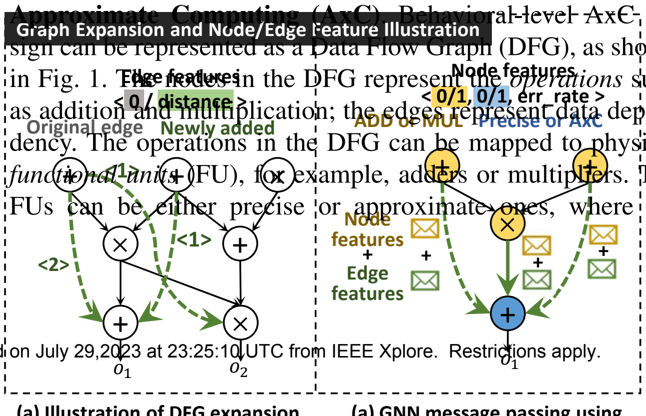
Since the error distribution prediction is already very challenging, we limit the paper scope to pure prediction without using it to assist AxC designs yet. Despite the limitation, we would like to point out that, error distribution prediction is an essential and innovative step towards practical AxC that no existing work has discussed before, which requires much deeper exploration.

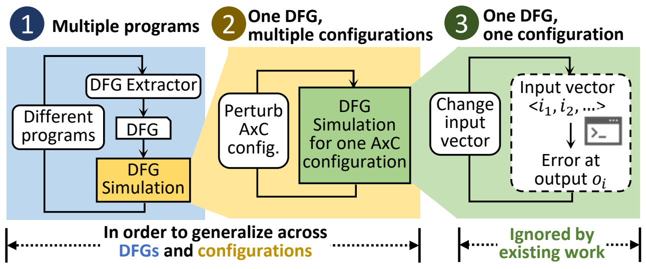



Two Proposed Approaches during Inference
Raw DFG extracted from an AxC program under one configuration
Graph Neural Network (GNN)
1 Model-free Prediction
(Error Distribution Histogram)
Loss: MSE with discretized ground-truth
2 Model-based Prediction
(Gaussian Mixture Model)
GMM Parameters
Loss: KL-divergence with ground-truth raw distribution

II. Preliminary

Approximate Computing (AxC). Behavioral-level AxC design can be represented as a Data Flow Graph (DFG), as shown in Fig. 1. The nodes in the DFG represent the *operations* such as addition and multiplication; the edges represent data dependency. The operations in the DFG can be mapped to physical *functional units* (FU), for example, adders or multipliers. The FUs can be either precise or approximate ones, where the





1 Multiple programs
2 One DFG, multiple configurations
3 One DFG, one configuration
DFG Extractor
Different programs
DFG
DFG Simulation
Perturb AxC config.
DFG Simulation for one AxC configuration
Change input vector
Input vector $<i_1, i_2, \ldots>$
Error at output o_i
In order to generalize across DFGs and configurations
Ignored by existing work

Fig. 4. **Input-aware training data collection.**

AxC FUs usually consume less power and area with faster execution speed. We refer the allocation of AxC FUs on the DFG as *one DFG configuration*. AxC DFG configurations will introduce errors by using AxC FUs, which will be propagated to the primary outputs. The output error is jointly determined by several factors: 1) the error introduced by AxC FUs; 2) error propagation along the DFG; 3) the primary inputs of the DFG, since different inputs will trigger different AxC FU error.

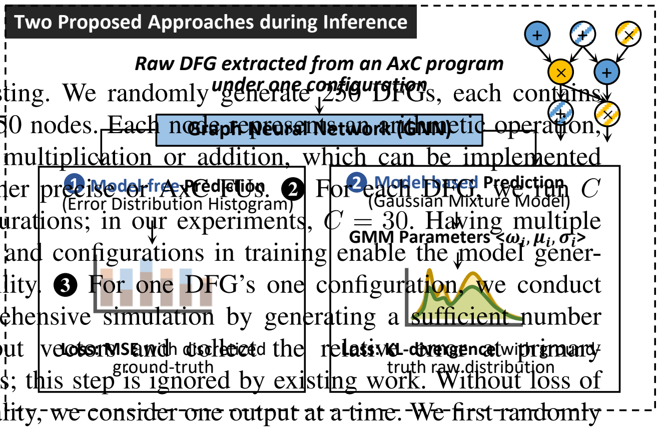
Graph Neural Network (GNN). GNNs operate by aggregating information along the edges of graphs. Each node v and/or each edge e is associated with a representation, i.e., embedding. A GNN layer updates each node representation by aggregating information of its neighbors, itself, and possibly connected edges; this process is called message passing. A GNN model can have multiple GNN layers; the more GNN layers, the larger the receptive field is, i.e., the information can be aggregated from farther nodes and edges. Example prevalent GNNs include graph convolutional network (GCN) [13], graph attention network (GAT) [14], and graph isomorphism network (GIN) [15]. Among these, GIN is provably as powerful as the Weisfeiler-Lehman graph isomorphism test, leveraging sum aggregators over a countable input feature space.

III. Proposed Approaches

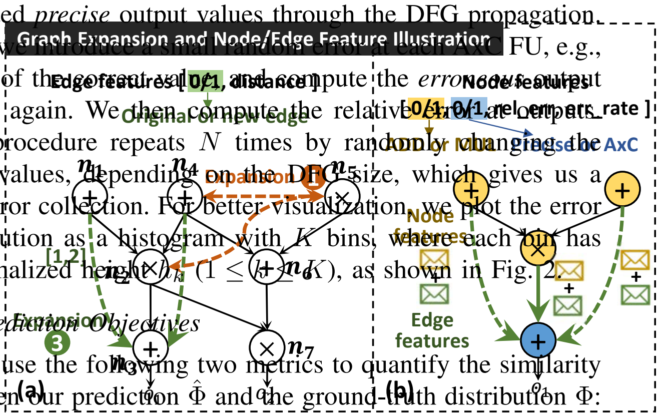
In this section, we first explain training data collection in Sec. III-A. In Sec. III-C, we introduce our error prediction workflow, **PreAxC**, using GNNs as the backbone; we further propose a very important **graph expansion** technique, which can significantly improve the prediction performance and reduce the required GNN complexity. We then discuss two prediction approaches on top of the GNN, **model-free** and **model-based**, in Sec. III-D and Sec. III-E, respectively. For the purpose of quantifying the similarity between the predictions and the ground truth distributions we use the metrics KL Divergence (KL), Bhattacharyya Distance (BD) and Relative Error Mean (REM).

A. Training Data Collection

Fig. 4 explains the three steps of training data collection. ❶ Extracting DFGs from multiple programs. Since real-world AxC applications are not sufficient enough for model training, we follow the work IronMan [16] and use synthetic DFGs for model training, and use both synthetic and real-world DFGs for testing. We randomly generate 250 DFGs, each contains 20 to 50 nodes. Each node represents an arithmetic operation, either multiplication or addition, which can be implemented by either precise or AxC FUs. ❷ For each DFG, we run C configurations; in our experiments, $C = 30$. Having multiple DFGs and configurations in training enable the model generalizability. ❸ For one DFG's one configuration, we conduct comprehensive simulation by generating a sufficient number of input vectors and collect the relative error at primary outputs; this step is ignored by existing work. Without loss of generality, we consider one output at a time. We first randomly assign each input value between [0,1]; then we compute the expected *precise* output values through the DFG propagation. Next, we introduce a small random error to each AxC FU, e.g., $\pm\omega\%$ of the correct value, and compute the *erroneous* output values again. We then compute the relative error at outputs. This procedure repeats N times by randomly changing the input values, depending on the DFG size, which gives us a raw error collection. For better visualization, we plot the error distribution as a histogram with K bins, where each bin has a normalized height h_k $(1 \le k \le K)$, as shown in Fig. 2.



Two Proposed Approaches during Inference
Raw DFG extracted from an AxC program under one configuration
Graph Neural Network (GNN)
1 Model-free Prediction (Error Distribution Histogram)
2 Model-based Prediction (Gaussian Mixture Model)
GMM Parameters $<\omega_i, \mu_i, \sigma_i>$
Loss: MSE with discretized ground-truth
Loss: KL-divergence with primary truth raw distribution



Graph Expansion and Node/Edge Feature Illustration
Edge features [0/1, distance]
Original or new edge
Node features
[0/1, 0/1, rel_err, err_rate]
Add or Mul
Precise or AxC
Expansion
[1,2]
Node features
Edge features
(a)
(b)

B. Prediction Objectives

We use the following two metrics to quantify the similarity between our prediction $\hat{\Phi}$ and the ground-truth distribution Φ:

- *KL divergence* [17] and *Bhattacharyya distance* (BD) [18], which are commonly used to describe how similar two distributions are. We compute both on the discretized histogram:

$$KL(\hat{\Phi}||\Phi) = \sum_{1 \le k \le K} \hat{h_k} \cdot log(\frac{\hat{h_k}}{h_k}) \tag{1}$$

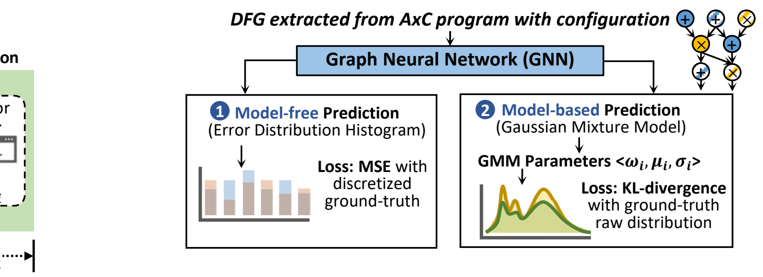
$$BD(\hat{\Phi}||\Phi) = \sum_{1 \le k \le K} \sqrt{h_k \cdot \hat{h_k}} \tag{2}$$

- *Relative Error Mean (REM)*. Existing works can only predict error mean and variance. To make a comparison with PreAxC, we compute the average of the relative error $\delta(\phi_j)$ within collection Φ for N input vectors, denoted by REM_{gt}, as the ground-truth error mean: $REM_{\text{gt}} = \frac{1}{N} \sum_{1 \le j \le N} \delta(\phi_j)$.

C. PreAxC Architecture using GNN

Given the natural graphs structures of DFGs, we propose to use GNN for representation learning. The *inductive* capability of GNNs also enables generalization to *unseen* graphs. On top of the GNN backbone, we propose two distribution detection heads (in Sec III-D and Sec. III-E). The overview of PreAxC is shown in Fig. 5. During inference, the input to PreAxC is the DFG extracted from an algorithm with its configuration; the output is the predicted distribution.

Challenges of Applying GNN to DFG. Although the DFGs are natural inputs to GNNs, applying GNNs directly on DFGs is *non-trivial*. In our preliminary experiments, learning on the raw DFGs do not produce satisfying results and the prediction error is extremely high. We attribute the failure to the


DFG extracted from AxC program with configuration
Graph Neural Network (GNN)
1 Model-free Prediction
(Error Distribution Histogram)
Loss: MSE with discretized ground-truth
2 Model-based Prediction
(Gaussian Mixture Model)
GMM Parameters $<\omega_i, \mu_i, \sigma_i>$
Loss: KL-divergence with ground-truth raw distribution

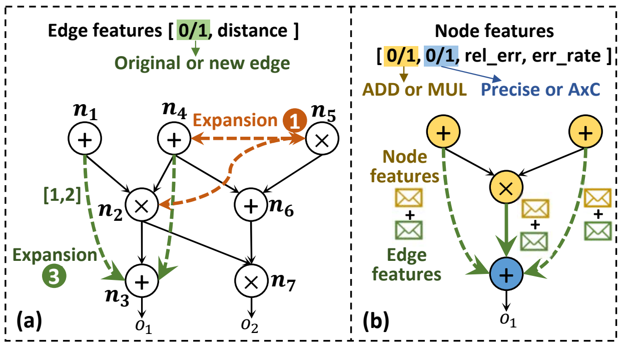


Fig. 6. **(a) Graph expansion methods (1) and (3). Ablation study is in Sec. IV. (b) GNN Message passing on the expanded DFG using both node and edge features.**

ineffectiveness of message passing between two nodes which are far away but one's (erroneous) output will eventually be propagated to another. We use Fig. 6(a) as an example, where the black edges are original edges in the DFG. In this example, node n_1's output will be propagated to node n_3 and output o_1. However, since the message passing mechanism in GNNs usually only aggregate information from direct neighborhood nodes per layer, in order to let node n_3's information be aggregated to node n_1, it requires at least two layers of GNNs. If the DFG is deep (which is usually the case), i.e., there are many nodes on the propagation path from one node to the output node, it will require an extremely deep GNN with many layers to capture the impact of the early node on the output. However, many empirical experiments have shown that deep GNNs do not perform well, sometimes even worse than shallow GNNs [19], [20]. In addition, deep GNNs require much more parameters, are hard to train, and do not scale in our problem since the depth of a DFG can be very large. For instance, a DFG with a longest path length 50 from input to output node will require at least a 50-layer GNN to propagate the error information, which is not applicable in practice.

Graph Expansion. Therefore, to address this challenge, we propose a graph expansion technique by adding auxiliary edges to connect nodes that are far away. We discuss three ways of expanding and we will provide an ablation study in Sec. IV-B. Expansion ❶: we connect every pair of nodes using bi-directional edges with the longest node distance as edge feature; it will result in a fully connected graph. Expansion ❷: the edges are the same as expansion ❶ but we remove all edge features. Expansion ❸: only one-directional edges are added to node pairs that are on the same path, with the longest distance as edge features. Fig. 6(a) shows an example of the DFG with newly added edges. The red edges are examples of expansion ❶, while the green edges are examples of expansion ❸. The purpose of graph expansion is to increase the efficiency of information propagation during GNN message passing and to remove the necessity of deep GNNs, since two faraway nodes may still collaboratively impact the output result. We recognize that such an approach will still face scalability issue, which must be addressed in future study.

Node/Edge Features and Message Passing. The node features include: arithmetic operation type (addition or multiplication), FU type (precise or approximate), relative error (in percentage), and error rate (in percentage). Because of graph expansion, the edge features include two values: (1) the first is a binary value indicating whether it is a newly added edge; (2) if a newly added edge, the second value is the shortest path in the original DFG between the two nodes. For example, for a new edge connecting two 2-hop nodes, the edge feature vector is [1, 2]. Initial node and edge features will be converted to hidden embeddings using a linear layer. Fig. 6(b) explains the message passing on the expanded DFG. For each node, it aggregates information by integrating its neighborhood nodes' messages as well as the incoming edge embeddings.

GNN Model. For the backbone GNN, we use GIN [15] as the graph learning model, which is capable of including both node and edge features and of discriminating large graphs. We choose GIN not only because it is provably as powerful as the Weisfeiler-Lehman graph isomorphism test, but also because it can flexibly incorporate edge features, which is especially important in our problem. In the experiments, we applied GCN and GAT as well but neither perform well enough to produce meaningful predictions.

Specifically, for the GIN used in PreAxC, the message passing aggregation function is $\phi(x, m) = x_l + \epsilon_l \cdot m_l$, where $\phi(\cdot)$ is the message transformation function, usually a fully-connect layer or multi-layer perceptron; x_l is the node embedding of l-th layer, m_l is the aggregated message, and ϵ_l is a learnable coefficient.

D. Model-free Distribution Prediction

After the GNN backbone for graph learning on the DFGs, we obtain a collection of node representations, which still need to be processed by a head for distribution prediction. We propose two approaches, model-free and model-based, as demonstrated in Fig. 5.

Model-free approach directly learns from the histogram of the error distribution without assuming any distribution models (e.g., Gaussian, Beta, Poisson). Given K histogram bins, where the j-th bin represents error ϵ_j with a height h_j $(1 \leq j \leq K)$, the goal is to directly predict the height of each bin $\hat{h_j}$. We use the Mean Squared Error (MSE) as the loss function during training:

$$\mathcal{L} = \frac{1}{K} \sum_{1 \leq j \leq K} (h_j - \hat{h_j})^2 \tag{3}$$

After the GIN backbone, we use a mea
and a multi-layer perceptron (MLP) with K
where the j-th neuron outputs the height of
Error Mean (REM) REM_{free} in model-fr
$REM_{\text{free}} = \sum_{1\le j\le K} \hat{h}_j \cdot \epsilon_j$.

E. Model-based Distribution Prediction

Gaussian Mixture Model (GMM). Th
based prediction is to assume a distributior
the goal is to learn the distribution model
the error distribution can be far from Gaussi
(as shown in Fig. 2(a)), we propose Gaussi
(GMM) to model the error distribution [21],
used approach to describe unknown distrib
the number of Gaussian components as M
function is expressed as:

$$p(\delta|\lambda) = \sum_{1\le i\le M} \omega_i \cdot \mathcal{N}(x|\mu_i,$$

where δ is a collection of the measured c
$\{\omega_i, \mu_i, \sigma_i\}$ $(1 \le i \le M)$ is a collection of C
$\mathcal{N}(x|\mu_i, \sigma_i^2)$ is a Gaussian function with mea
σ_i^2; ω_i are the mixture weights that satisfies
Relative Error Mean REM_{gmm} in model-ba
$REM_{\text{gmm}} = \sum_{1\le i\le M} \omega_i \cdot \mu_i$.

To verify the assumption that the error d
precisely modeled as GMM, we visualize t
lected test cases in Fig. 2(a). The orange bar
raw distribution; the blue bars are modeled b
by the Expectation–maximization (EM) alg
visualization confirms that using GMM to
distribution is valid.

GMM Parameter Learning. We let the GNN predict the GMM parameters $\hat{\lambda}$. In each training iteration, we sample N data points following the predicted distribution $\hat{\Phi}(\hat{\lambda})$, and use the KL divergence described in Eq. (1) as the loss function: $\mathcal{L} = -KL(\hat{\Phi}(\hat{\lambda})||\Phi)$, where $\hat{\Phi}(\hat{\lambda})$ is discretized as histogram with K bins the same as Φ.

The GNN model uses the same GIN followed by a mean global pooling and an MLP with $3M$ output neurons. The first M output neurons represent the weights of GMM components, which must sum up to 1, so for that we add a softmax before them. The second $2M$ neurons use an absolute function to guarantee that relative error mean $\hat{\mu}_i$ and error variance $\hat{\sigma}_i^2$ are positive.

Learning via Reparameterization. Since the sampling process from $\hat{\Phi}(\hat{\lambda})$ is non-differentiable, we adopt the reparameterization trick to sample from a distribution following GMM [23]. It is known that sampling from $x \sim \mathcal{N}(\mu, \sigma^2)$ is equivalent as sampling from $z \sim \mathcal{N}(0, 1)$ with the transformation of $x = z \cdot \sigma^2 + \mu$. Therefore for multiple Gaussian components, each time we sample from the i-th component with a probability of ω_i, where ω_i is the component weight. In this way, the loss function is differentiable with respect to the distribution parameter and can be updated via back propagation.

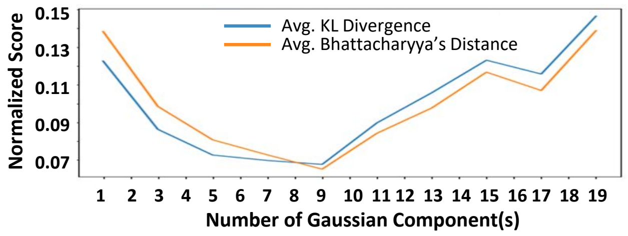


Fig. 7. **Normalized scores using different numbers of Gaussian components (M). In our experiments we let $M = 9$.**

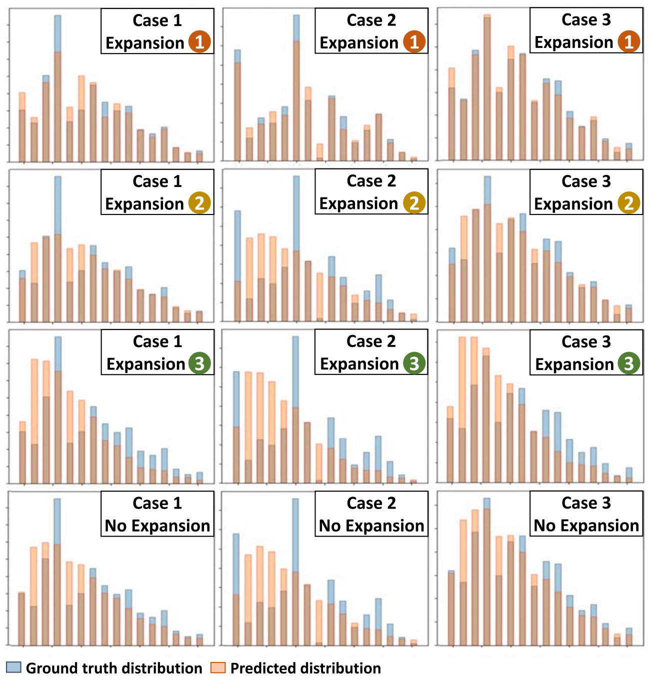


Fig. 8. **Ablation study on graph expansion methods. With graph expansion (1), the distribution prediction is very close to the ground-truth.**

GMM Component Count. Since the number of Gaussian components is critical since it directly determines the structure of the predictor, we conduct a preliminary experiment on the value of M. Fig. 7 shows the normalized similarity scores, including KL divergence and Bhattacharyya's Distance, between the predicted GMM-based distribution and the raw data distribution. When M increases at the beginning, the score decreases, indicating that the GMM becomes more expressive. The trend saturates at $M = 9$, indicating that larger M will introduce trivial components and will disturb the prediction. Therefore in our experiments, we let $M = 9$.

IV. Experiments

A. Experimental Setup and Baselines

Training Details. The training data collection is described in Sec. III-A. We let both AxC adders and multipliers have an error rate of 100% and relative error of 10%. Both models, the model-free and model-based, are trained for 1000 epochs with the Adadelta optimizer; the drop out is 0.4. The GIN backbone has two layers; the dimension of node and edge embeddings is 20. For the model-free predictor, we use a 20-20-20 MLP; for the model-based predictor, we use a 20-20-20-30 MLP.

TABLE I

RESULTS ON **TRANSDUCTIVE** (SEEN DFGS) AND **INDUCTIVE** (UNSEEN DFGS) SETTINGS. ALL REAL-CASES ARE **INDUCTIVE**. **BN**: BAYESIAN NETWORK; **REM**: RELATIVE ERROR MEAN; **KL:** KL-DIVERGENCE; **BD:** BHATTACHARYYA DISTANCE. ALL METRICS ARE THE SMALLER THE BETTER. *NOTE THAT BN DOES NOT GENERALIZE TO INDUCTIVE GRAPHS SO WE RETRAIN A NEW MODEL EACH CASE.

		DFG Trav [4]	**BN [8]**	**Ours** Model-free			**Ours** Model-based		
		REM	REM	REM	KL	BD	REM	KL	BD
Synthetic	Transductive	48.83%	19.51%	11.5%	0.1	0.02	18.7%	0.15	0.03
	Inductive	39.48%	15.23%*	8.6%	0.13	0.02	37.1%	0.16	0.03
Real-case (Inductive)	FFT	52.07%	27.61%*	3.7%	0.09	0.01	5.4%	0.22	0.03
	Sobel Filter	20.12%	16.48%*	6%	0.15	0.02	25.5%	0.3	0.06
	Linear	44.29%	23.79%*	12%	0.37	0.03	9.4%	0.17	0.03

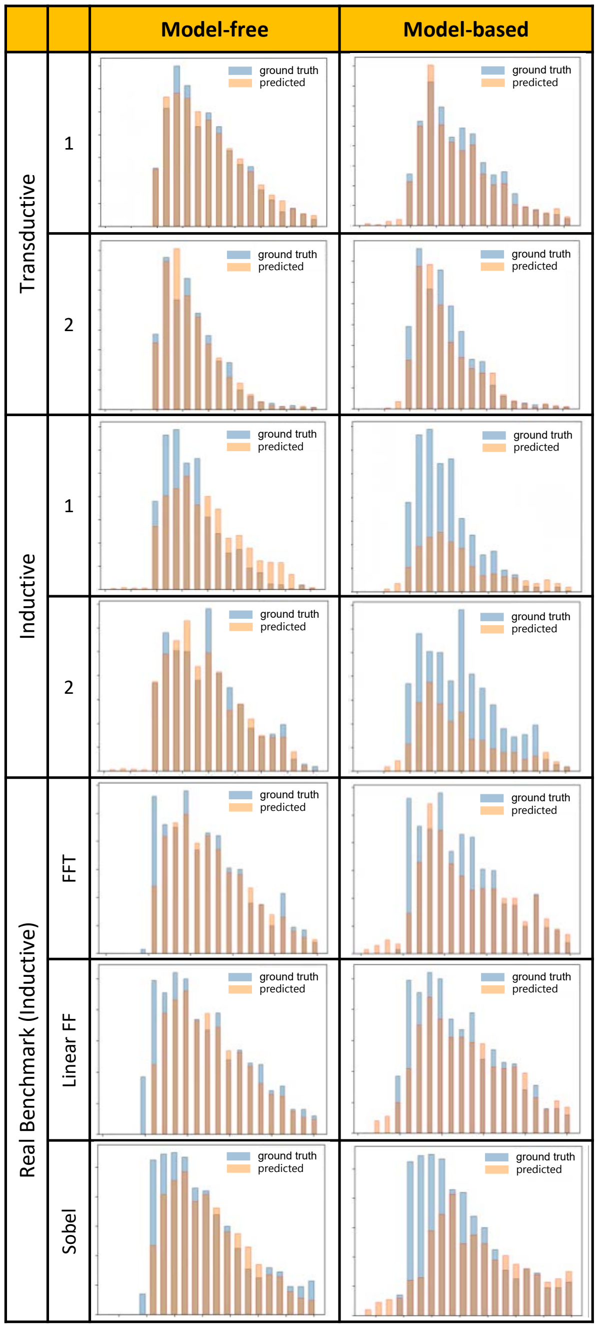


Fig. 9. **Visualization of error distributions predicted by model-free and model-based approaches on expanded DFGs.**

Transductive and Inductive Evaluations. We conduct evaluation on two settings: **transductive** and **inductive**. Transductive means that the model is trained on all the DFGs and is tested on *seen DFGs but with unseen configurations*; inductive means that the model is trained on a subset of the DFGs and is tested on *unseen DFGs and configurations*. Apparently, the inductiveness is much more difficult but important; it enables an ML model to be applied to new designs without any retraining or data collection, of which existing works fail to handle. For inductive setting, we use 160 DFGs for training and 90 new DFGs for testing. Finally, we use three representative real-cases: FFT, Sobel Filter, and a linear feedforward layer, which are held out for testing only.

Baselines. We compare with two baselines: static DFG traversal replicated from [4] and Bayesian Network (BN) replicated from [8]. Since neither of them can predict error distribution, we compute the error mean values from the distribution predicted by PreAxC. Note that BN [8] is trained on one DFG and infers on the same DFG with different configurations, which **does not generalize to unseen DFGs**; a new BN must be trained for every new DFG from scratch, which also requires data collection. Therefore, it is *not a fair comparison* since our PreAxC require no re-training at all.

B. *Evaluation*

Ablation Study of Graph Expansion. The necessity of graph expansion is demonstrated in Fig. 8. Each column is one randomly selected test case; each row shows one expansion method. The visualization clearly demonstrates that the full expansion ❶ works surprisingly well while others and no expansion perform worse. Comparing expansion ❶ and ❷, it highlights the importance of the edge features we introduced, the longest distance between a node pair. Comparing expansion ❶ and ❸, one plausible explanation is that, by fully connecting node pairs, the GNN can learn the collaborative effect of two operations, even though they are not on the same path. Using the example in Fig. 6(a), where node n_4 and node n_5 are both contributing to output o_2 but are not on the same path; adding an edge between n_4 and n_5 can be beneficial to learn their joint impact on o_2. This expansion can remove the necessity of using deep GNNs and enable much more effective message passing: in our experiments, just two GNN layers can perform good enough.

Comparisons with Baselines. We then compare our approach with baselines, static DFG traverse [4] and BN [8], as shown in Table I. Since the existing works do not predict distribution but only error mean values, we compare our computed error mean from GMM and the REM score (indicating the prediction error). Table I shows that, in average, the DFG transverse approach gives the largest prediction error, from 20% to 52%. BN works relatively well for transductive settings since it is trained per-DFG. Since it cannot handle inductive cases, we re-train on each new case and show their results (which is not a fair comparison with ours). Our approach works well for both synthetic and real-case DFGs, transductive and inductive. In synthetic transductive scenario, both the model-free and model-based approach demonstrate superior performance in REM comparing with existing works: 11.5% for model-free and 18.7% for model-based, comparing with 19.51% for BN. In inductive setting, model-free approach shows promisingly lower REM, 8.6%, comparing with 15.32% of BN, especially that BN is *retrained* on these cases. For the three real-world benchmarks, our approaches also demonstrate remarkable generalizability with a consistent low REM, especially much lower than DFG transverse. This result demonstrates the promising prediction accuracy of our proposed **PreAxC** workflow. Finally, we also evaluate the similarity of the predicted and ground-truth distributions using the metrics including KL and BD in Table I (Sec. III-B). The KL and BD scores are consistently low for both transductive and inductive cases as well as real-world cases, demonstrating the generalizability of PreAxC.

Visualizations. Fig. 9 visualizes the error distribution in the format of histogram. We plot four random synthetic cases for transductive and inductive settings, as well as the three inductive real-world benchmarks. Blue bars represent ground-truth distribution and orange bars represent predicted. We have the following observations. First, model-free approach generally excels model-based. One possible reason is that the learning task in model-free is easier. Second, in some cases, the model-based method describes the detailed distribution shape better than model-free. For example, for FFT, the model-based method is able to predict the second distribution spike towards the right, while model-free fails to do so. Similar observation can be found in transductive Case 2. This minute feature is only captured by the model-based approach. Third, since the model-based approach assumes GMM, there is always a misprediction around zero-value. Overall, the visualization confirms that PreAxC is effective in predicting the distribution shapes.

V. Conclusions and Future Work

In this work, we proposed PreAxC, the first work to predict output error distribution using GNNs for AxC applications. PreAxC is more accurate and informative with input-awareness and generalizability. We proposed two approaches: model-free using histogram, and model-based using GMM. We also proposed graph expansion to significantly improve learning efficiency. Experiments demonstrated that PreAxC works well for both transductive and inductive settings by accurately depicting the distribution. Future works include testing on real-world benchmarks and discussing how the primary output error distribution affect the final application quality, e.g., PSNR for image processing or accuracy for classification.

VI. Acknowledgments

This project is partially supported by the National Science Foundation under Grant No.2202329.

References

[1] Lukas Sekanina et al. Approximate circuits in low-power image and video processing: The approximate median filter. *Radioengineering*, 26(3), 2017.

[2] Matthieu Courbariaux et al. Binaryconnect: Training deep neural networks with binary weights during propagations. *NeurIPS*, 28:3123–3131, 2015.

[3] Zhaowei Cai et al. Deep learning with low precision by half-wave gaussian quantization. In *CVPR*, pages 5918–5926, 2017.

[4] Chaofan Li et al. Joint precision optimization and high level synthesis for approximate computing. In *DAC*, pages 1–6. IEEE/ACM, 2015.

[5] Omid Akbari et al. X-cgra: An energy-efficient approximate coarse-grained reconfigurable architecture. *TCAD*, 2019.

[6] Masoud Pashaeifar et al. A theoretical framework for quality estimation and optimization of dsp applications using low-power approximate adders. *TCAS*, 66(1):327–340, 2018.

[7] Xin Sui et al. Proactive control of approximate programs. *ACM SIGPLAN Notices*, 51(4):607–621, 2016.

[8] Marcello Traiola et al. Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. *Microelectronics Reliability*, 102:113309, 2019.

[9] Radha Venkatagiri et al. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. In *MICRO*, pages 1–14. IEEE, 2016.

[10] Daya S Khudia et al. Rumba: An online quality management system for approximate computing. In *ISCA*, pages 554–566, 2015.

[11] Zhenghao Peng et al. Axnet: Approximate computing using an end-to-end trainable neural network. In *ICCAD*, pages 1–8, 2018.

[12] Sana Mazahir et al. Probabilistic error modeling for approximate adders. *IEEE Transactions on Computers*, 66(3):515–530, 2016.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[15] Keyulu Xu et al. How powerful are graph neural networks? In *ICLR*, 2018.

[16] Nan Wu, Yuan Xie, and Cong Hao. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *GLSVLSI*, 2021.

[17] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[18] Thomas Kailath. The divergence and bhattacharyya distance measures in signal selection. *IEEE transactions on communication technology*, 15(1):52–60, 1967.

[19] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[20] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2019.

[21] Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741, 2009.

[22] Arthur P Dempster et al. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–22, 1977.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.