Predicting the Probability of Successful Trials of single-outcome Quantum Circuits using Deep Learning

Misbah Ali, Leon H. Kloker, Karthik Nataraj
Mentor: Daniel Hothem
Institute for Computational and Mathematical Engineering
CME 291, Stanford University
fathali@stanford.edu, leonkl@stanford.edu, kartnat@stanford.edu

1 Problem statement

Quantum computers are theoretically able to solve a new class of problems, which previously could not be dealt with efficiently as there are no known polynomial-time algorithms that can run on classical computers. In spite of the major advances in quantum computing hardware in recent years, however, all computations on quantum computers are still error prone. Hence, the full theoretical potential of quantum computers has yet to be unlocked. Thus, the development of quantum circuits that minimize the errors in the computation result is crucial in order to make quantum computers practically useful.

The fidelity of two quantum states measures how close the two states are to each other and can be used to quantify the distance between the desired target state and the noisy result of a quantum computation. Hence, predicting the fidelity of the final state of a quantum circuit, i.e. the fidelity of the circuit, is an important step towards determining which implementation of a quantum algorithm will yield the least amount of errors.

For single-outcome circuits, the quantification of errors can be further simplified as the probability of successful trials (PST) can be used to approximate the fidelity. Hence, in this work, we are only focusing on single-outcome circuits, also called definite outcome circuits. For this class of circuits, we employ different neural networks in order to predict the PST for a given quantum circuit.

2 Previous work

Sandia National Laboratory has already trained self-built convolutional neural networks (CNN) in order to predict the PST of running definite-outcome circuits on a given architecture. Moreover, there are some papers on the subject: Namely a paper by Vadali et al. [2022] trying to predict the circuit fidelity also using a CNN and a paper by Wang et al. [2022], who employed a graph transformer architecture to predict the PST of definite-outcome circuits. Both approaches yield comparable results and serve as a reference for our work.

CME291: Xplore, Winter 2023, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 1st approach: ResNet-50

In this research project we built on that work in a couple ways. First, by seeing how well ResNet-50, a large off-the-shelf pre-trained convolutional neural network, performs on this regression task. Regarding the first approach, there were a couple challenges to overcome including dataset and task alignment. ResNet50 is designed to operate on inputs of a fixed size, namely (224, 224, 3) shaped arrays. Sandia's circuit datasets are not shaped in this way, the most notable difference being that the last dimension is 10 rather than 3, in order to encode the various gate (7 different ones) and state (3 different ones) information. We experimented with a couple different approaches to alter Sandia's inputs and/or alter the network (by prepending trainable layers) in order to ensure compatibility:

1. Reshape the circuit array, padding with zero rows if necessary, into the shape $(n_1, n_2, 3)$. "Stretch" this $(n_1, n_2, 3)$-shaped array into a $(224, 224, 3)$ one.
2. Reshape circuit array into $(224, n_2, 10)$ by padding first dimension with $224 - n_1$ rows (all provided circuit datasets have $n_1 < 224, n_2 > 224$), prepend trainable layers to get to $(224, n_2, 3)$ shaped array, and average pooling to get the $(224, 224, 3)$ input.

The "stretching" mentioned in point 1 was done by keras built-in resize method, with operates on images. The following is the pre-resnet architecture for the second model, for the original circuit data input:

1. Input: (224, 1825, 10)
2. (15, 3, 4) convolutional layer, output (224, 1825, 15)
3. (3, 3, 4) convolutional layer, output (224, 1825, 3)
4. padding layer, output (224, 2016, 3)
5. (1, 9) pooling layer, output (224, 224, 3)

The architecture for the other dataset, with input circuits of shape $(224, 273, 10)$, is very similar. Moreover, as ResNet50 is designed for classification rather than regression, we needed to substitute the last 1000 node layer (representing output probabilities for each of 1000 different classes) with a trainable single node layer (representing predicted PST).

3.1 Results

Although we obtained good preliminary results with the reshaped input, it's important to assess the effect of leaving original, contextual (gate, state) information intact. We ran a total of four experiments, under the two different architectures. Each architecture was employed with the original input and also the non-Markovian input. Model performance was evaluated using KL Divergence:

$$\begin{aligned} D_{KL}(y_{\text{true}}||y_{\text{pred}}) &= H_{y_{\text{pred}}}(y_{\text{true}}) - H_{y_{\text{true}}}(y_{\text{true}}) \\ &= -\frac{1}{N}\sum_{i=1}^{N} y_{i,\text{true}} \log(y_{i,\text{pred}}) + (1 - y_{i,\text{true}}) \log(1 - y_{i,\text{pred}}) \\ &- \left[-\frac{1}{N}\sum_{i=1}^{N} y_{i,\text{true}} \log(y_{i,\text{true}}) + (1 - y_{i,\text{true}}) \log(1 - y_{i,\text{true}}) \right] \\ &= -\frac{1}{N}\sum_{i=1}^{N} y_{i,\text{true}} \log\left(\frac{y_{i,\text{pred}}}{y_{i,\text{true}}}\right) + (1 - y_{i,\text{true}}) \log\left(\frac{1 - y_{i,\text{pred}}}{1 - y_{i,\text{true}}}\right), \end{aligned}$$

where $y_{\text{true}}, y_{\text{pred}}$ represent the true/predicted label distributions, resp. and $H_p(q)$ is the binary cross entropy. And L1 error:

$$\text{L1 error} := \frac{1}{N}\sum_{i=1}^{N} |y_{i,\text{pred}} - y_{i,\text{true}}|.$$

with a "good" L1 error being $< .05$. PST prediction results are below:

Experiment	Training/Val/Test	BCE training/val	Test KL/MAE
RegularReshaped	11620/3320/1000	0.3603/0.3622	0.001978/0.0156
NonMarkovianReshaped	4000/1000/5000	0.4358/0.4383	0.007561/0.0360
Regular	1000/100/1000	0.3610/0.3413	0.006367/0.0294
NonMarkovian	4000/1000/5000	0.4317/0.4334	0.002575/0.0210

The experiment titled “regular" could only be run with the 1000 circuits due to memory constraints imposed by the data and model. However, the same 1000 test circuits were used between this and “RegularReshaped", and even training on the same 1000 circuits we saw a smaller TestKL and MAE. Some plots of experimental results and associated commentary are in the Appendix.

3.2 Next steps

Although the modeling results confirmed intuition about the relationship between the regular and Markovian data, the original question about whether or not ResNet50 can improve prediction accuracy remains largely unanswered. We could run experiments with and without ResNet50 to assess it’s value-add–it’s possible that since ResNet50 is trained on images from daily life, it cannot help much with identifying features from circuits encoded as images.

4 2nd approach: Graph Neural Networks

As graph neural networks can deal with input graphs of arbitrary size, this approach comes up naturally when considering quantum circuits with varying number of qubits and layers. In order to feed the data into an implementation of a graph neural network, however, the data first needs to be transformed into a graph representation.

4.1 Qubit-centric encoding

The first graph encoding we employ for quantum circuits is qubit-centric, i.e. nodes represent qubits. The general idea is depicted in figure 1. We can see that the graph depends on the structure of the underlying quantum computer the circuit is run on. Moreover, every node in the graph has an associated 10-dimensional vector, describing the gate acting on the qubit at the specific layer as previously mentioned in section 3. This graph is saved in form of a n x 10 feature matrix X, and a sparse n x n adjacency matrix, where n is the overall amount of nodes in the graph. Additionally, a 6-dimensional global feature vector containing the number of qubits involved in the circuit, the amount of layers of the circuit, the amount of Z, X, SX and CNOT gates is also fed into the network. This feature vector is normalized with the mean and standard deviation of the training set.

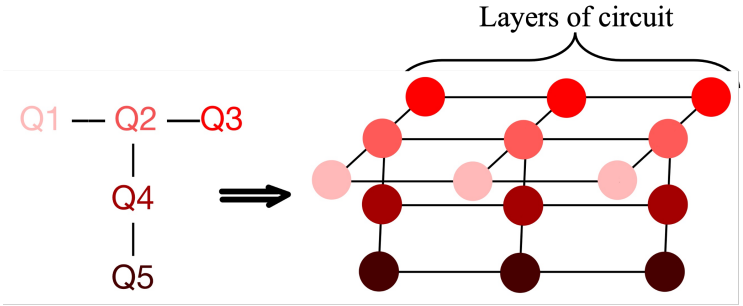


Figure 1: The quantum computer architecture on the left is repeated for every layer of the circuit, where each qubit is also connected to itself in the previous and next layer. The coloring is done for visualization purposes.

Table 1

Model	#Parameters	BCE training/val	Test MAE	TEST KL
Baseline	19665	0.3967/0.4039	0.0705	0.0485
1 layer GAT	20085	0.4628/0.4517	0.1382	0.1022
3 layer GAT	20565	0.4262/0.4182	0.0906	0.0552
5 layer GAT	21045	0.4196/0.4123	0.0845	0.0570

4.2 Architecture

The GNN models that were employed generally use a feedforward neural network (FFN) to process the global feature vector and run graph attention layers on the input graph. Each graph-attention (GAT) layers updates the state vector of each node based on:

$$
\begin{aligned}
X^{(l)} &= \alpha^{(l)} X^{(l-1)} W^{(l)} + b^{(l)} \quad , \\
\alpha_{i,j}^{(l)} &= \frac{\exp(\text{LeakyReLU}(a_1^{(l)T}(X^{(l-1)}W^{(l)})_i + a_2^{(l)T}(X^{(l-1)}W^{(l)})_j))}{\sum_{k \in \mathcal{N}(i) \cup i} \exp(\text{LeakyReLU}(a_1^{(l)T}(X^{(l-1)}W^{(l)})_i + a_2^{(l)T}(X^{(l-1)}W^{(l)})_k))} \quad ,
\end{aligned} \tag{1}
$$

where a_1, a_2, W and b are trainable weights for each GAT layer l, $X^{(0)} = X$ and $\mathcal{N}(i)$ denotes the set of neighbours of node i. In between every GAT layer, we employ a dense layer with ReLU activation as skip connection, add it to the output of the graph-attention layer and run layer normalization, similar to Wang et al. [2022]. After l GAT layers, global average pooling is employed on the graph and we end up with a 10 dimensional vector representing the entire graph. This vector is concatenated with the output of the FFN processing the global features. Finally, a FFN regressor network is run on the combined features to predict the PST of the entire circuit.

Similar to Wang et al. Wang et al. [2022], we employ a 3 layer FFN to process the global features with dimensions 12, 12, 6 and ReLU activation. The regressor consists of 3 layers with dimensions 128, 128 and 1 with ReLU activations and a sigmoid activation for the output layer.

4.3 Results

3 models with 1, 3 and 5 graph-attention layers as well as a baseline model that only takes the global features without graph input were employed. All models were trained for 100 epochs with Adam using a learning rate of 10^{-3}, which led to a convergence in all evaluation metrics. The learning rate was optimized by running experiments with a range of different values. The model with the lowest validation binary cross entropy between the predicted PST and ground truth was saved.

Figures 4 and 5 show the distribution of predictions on the train and test data for all 4 models that were employed. Moreover, table 1 contains the binary cross entropy, mean absolute error and Kullback-Leibler divergence for all 4 models.

It is immediately visible that the models did not perform well in general. As the train and validation binary cross entropy is very similar, there does not seem to be a variance problem but rather a bias problem with the models. This could be caused by insufficient complexity to fully capture the relationship between the given graph encoding and the actual PST of the circuit or by issues during the model training. As Wang et al. [2022] have achieved good results with an equally small model and our models stopped learning after only one or two epochs, i.e. the cross entropy did not decrease anymore, we suspect that some issues during training are responsible for the poor performance. The fact that the baseline model that only takes the 6-dimensional global feature vector as an input exhibits similar or even lower KL divergence than the other models also corroborates this hypothesis.

4.4 Conclusions

The GAT networks did not perform as well as expected. This is most likely caused by the non-trivial training process for GNNs. We believe, however, that using GNNs is the right approach as they allow for arbitrarily-sized input graphs and do not require any form of padding. As regular graph convolution operations just pass messages between neighbouring nodes and the graphs under consideration have

a length equal to the amount of layers of the quantum circuit, it might be worthwhile to incorporating virtual nodes in the graph that are connected to everything or to look at recurrent GNNs. These networks would for example apply an attention mechanism successively to a graph representation of every layer of a quantum circuit. As this is in accordance with the physical evolution of the quantum system, such a network structure might be a better fit to predict the PST. Moreover, we highly recommend to henceforth use PyTorch instead of Tensorflow, as the libraries for GNNs there seem to be much more flexible and matured.

4.5 Gate-centric encoding

The second graph encoding we employ for quantum circuits is gate-centric. Here, qubits, gates and measurements are represented as nodes. and the connections between them in the quantum circuit are represented as directed edges between the corresponding nodes.
The node features used are one hot encoding that identifies the type of the node (qubit, measurement, Z-gate, X-gate, SX-gate, CNOT-gate). For every node, depending on what the node represents, exactly one of these features will be one and all the others will be zeros. In addition to these, there are 5 more features that represent which qubits this gate is acting upon. These 5 features are multi-hot encoding (since some gates act on more than one node). This graph 2 illustrates a quantum circuit with its corresponding graph.

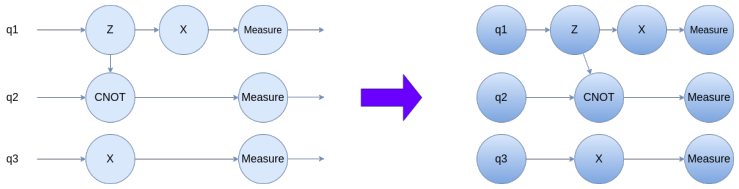


Figure 2: The quantum circuit on the left is converted to the graph on the right.

4.6 Experiments

For this graph encoding, I applied GNNs to compute embeddings for the nodes and then aggregate them to find an embedding for the entire graph. This embedding is fed to a Feed-Forward fully connected Neural Netowrk to predict the PST. I tested three hypothesis; each one with different GNN types (GCN, Transformer, GAT, GraphSAGE), extensive hyper-parameter search (learning-rates, hidden dimension, number of GNN layers, batch size). The hypothesis that were tested are:

1. Aggregating the embeddings of only the measurement nodes instead of aggregating all of the nodes, the KL-divergence of the model will segnificantly be improved.

2. Modifying the graphs of the circuits by fully connecting the qubit nodes together - instead of leaving them dis-connected except by gates that operate on two qubits - the KL-divergence of the model will segnificantly be improved.

3. Modifying the graphs of the circuits by connecting all the nodes in a channel to the measurement node at the end of that channel, the KL-divergence of the model will segnificantly be improved.

I ran experiments to test these hypothesis against the null hypothesis of using the standard GNNs with the encoding shown in 4.5. The expiremnts including training a bunch of models with the different combinations of the hyper-parameters and GNN layers types and then take the top 10% of all the models. I took these models, because these represent the space that we are intersted in. I created a table that summarizes the results of these models. For each model, the table has an indicator variable for each of the expriments, in addition to variables that specify the GNN type and the hyper-parameters. The tables also have the training loss and the validation loss. I then perform t-tests for all the variables against the validation loss to check for segnificant association between validation loss (kl-divergence) and the variables in the table.

4.7 Results

The top model achieved .0024 KL divergence. The model uses 3 GCN layer, with embedding size of 64. It aggregates only from measurement nodes and connects every node in every channel to its measurement.
The top factors (minimum p-values) in predicting the validation loss are reported in the table 2. The top factor indicates that using Graph Transformer layer with a relatively larger number of GNN layers (up to 10) is significant in improving the validation loss. A note here is that, this doesn't mean that more layers are better as long as we are using Graph Transformer. This is because of the well known oversmoothing problem in GNNs with large number of layers. The second most important factor is not to use Transformer layer and fully connecting the qubit nodes which corresponds to hypothesis (ii) above, this is because the coefficient is positive while the loss is to be minimized. The 3rd and 4th factors include aggregating only from the measurement nodes which correspond to hypothesis (i). However, in both factors this hypothesis is not used on its own. The 3rd factor includes this one with hypothesis (iii) connecting all the nodes in channel to the measurement node in that channel. The 4th factor includes using a larger number of GNN layers. This is expected since, if we aggregate only from the measurement nodes with a small number of GNN layers and wihtout connections in the channel, some of the information from the channel will not have a chance to be used for the final prediction which means the model will be ignoring important information.

Factor	Coef.	p-value
I(number of GNN layers * Transformer)	-0.17189513	0.01899611
I(Transformer * connect qubits)	0.05506688	0.04085572
I(aggregate measurement * connect channels)	-0.04398634	0.04716320
I(aggregate measurement * number of GNN layers)	0.13951615	0.06187569
I(number of GNN layers * connect qubits)	-0.08933936	0.17583209
Intercept	0.01028977	0.58066121

Table 2: The coeficients of the experiment setting vaiables in predicting the validation loss. Negative coefficient means that the factor improves the loss. Smaller p-values means the factor is statistically significant in affecting the validation loss.

4.8 Conclusions and Future work

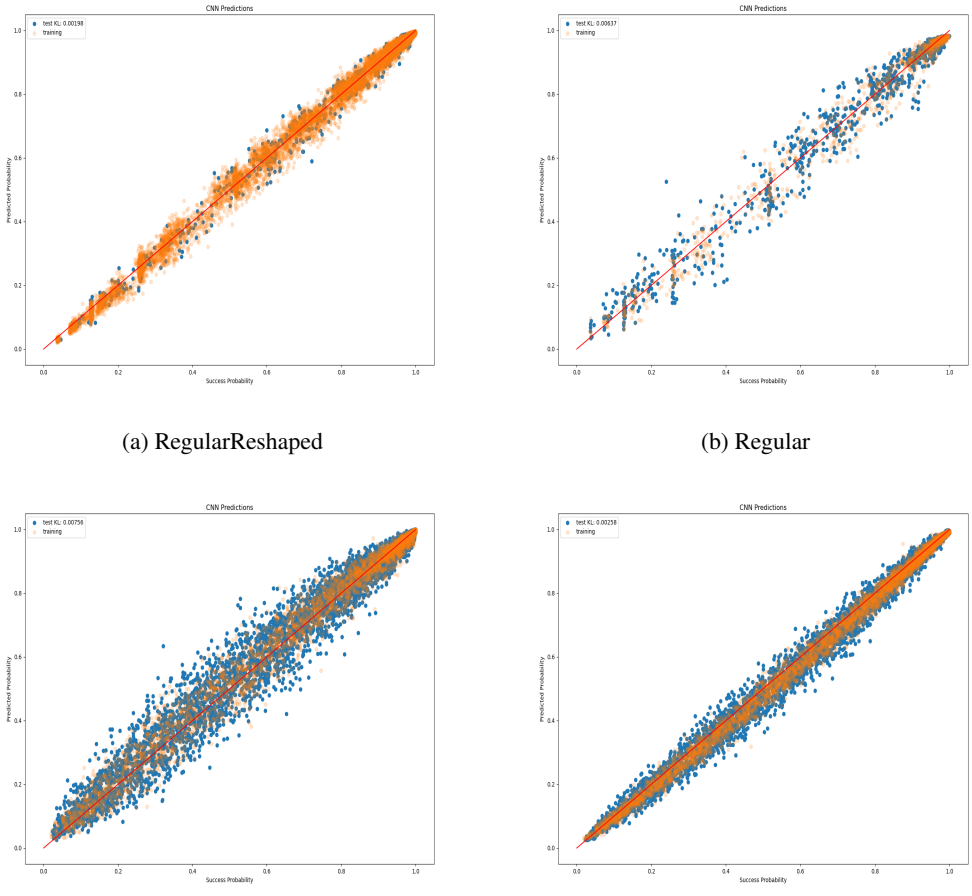
Based on these results, here are some important conclusions when using GNNs to predict PST using the current encoding:
- aggregating only the embeddings of the measurement nodes and using the results of that to predict the PST significantly improves the validation loss but it needs to be associated with either large number of GNN layers (at least the diameter of the Graph) or with connecting at least some of the earlier nodes in the channel to the measurement nodes in that channel. The reason is to make sure that all the important information is utilized when making the predictions. The issues with using more GNN layers is that the diameter of the Graph will vary from a graph to another. Which means we will need to use a large enough number of layers. But the problem with this is that this can lead to oversmoothing especially if there are many two qubit gates.
- The type of the GNN layer is in general not significant in improving the loss. Except for using Transformer with large number of layers. Again, large here doesn't mean arbitrarily large because of the oversmoothing problem.
- These experiments proved that modifying the connectivity structure of the Graph - to get a computational graph that is optimized for the ML task - is actually segnificant (more segnificant than most of the hyper-parameters). Which means using Graph Neural Networks has a good potential for this problem.

Possible future directions for this work are:
- Utitlize the connectivity structure of the qubits in the quantum circuits, which basically means combine the two embeddings shown in this paper. This is because fully connecting the qubits in the gate-centric encoding is an initial step towards that. Even though it was not significant in improving the loss; when combined with larger number of GNN layers, it was the fifth factor in the results.
- Use Recurrent GNNs which better depict the time dependence quantum circuits have by nature. In quantum circuits, the information flows from the qubits towards the measurement. This behavior is captured in the GNNs by the fact that the edges are directed. Even though, it is not perfectly captured,

since the GNN starts passing messages from all the nodes from the begening.
Include Graph-level features in the gate-centric encoding approach since the current approach doesn't use graph level features as opposed to the approach in node-centric encoding.

Author contributions
Misbah Ali: Sections on the Gate-centric GNNs
Leon H. Kloker: Sections on the Qubit-centric GNNs, previous work, problem statement
Karthik Nataraj: Sections on ResNet-50, previous work

5 Appendix



(a) RegularReshaped

(b) Regular

(c) NonMarkovianReshaped

(d) NonMarkovian

Figure 3: Predicted vs. True for Above 4 Experiments. In the non-Markovian data the outputs of gates in later layers are influenced by those of all previous layers, and so there is greater correlation amongst different rows of data. So when the data is jumbled by reshaping this correlation is harder to unpack by the model, resulting in worse predictions in (c) than (a). (d) passes in the data as is, where the correlation between row x and row y of data has a fixed meaning in terms of a given qubit, the layer of the circuit it is at, and the gates/states in that layer. Therefore it's easier for the model to rediscover this correlation, resulting in better performance than in (c). This data structure doesn't matter as much in the markovian case, and so we see more comparable predictions between (a) and (b).

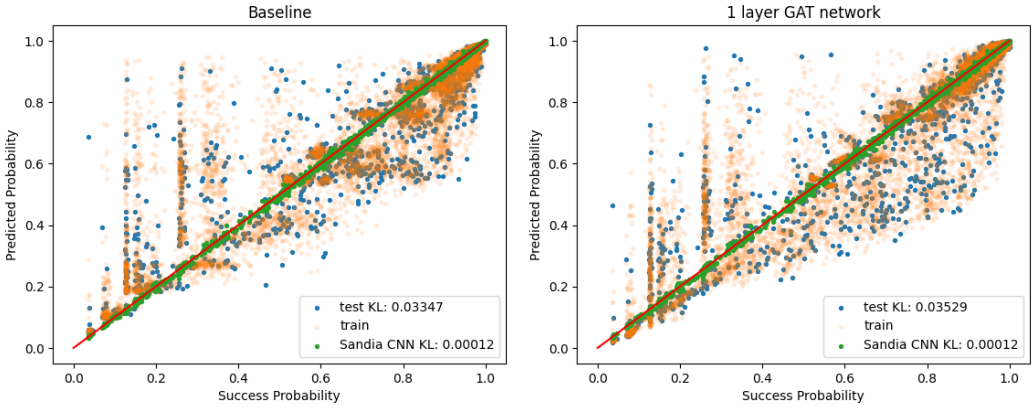


Figure 4: The figure shows the distribution of train and test predictions for the baseline model on the left side and the 1 layer GAT model on the right side. As we plot predicted probability over ground truth, the red line corresponds to perfect predictions.

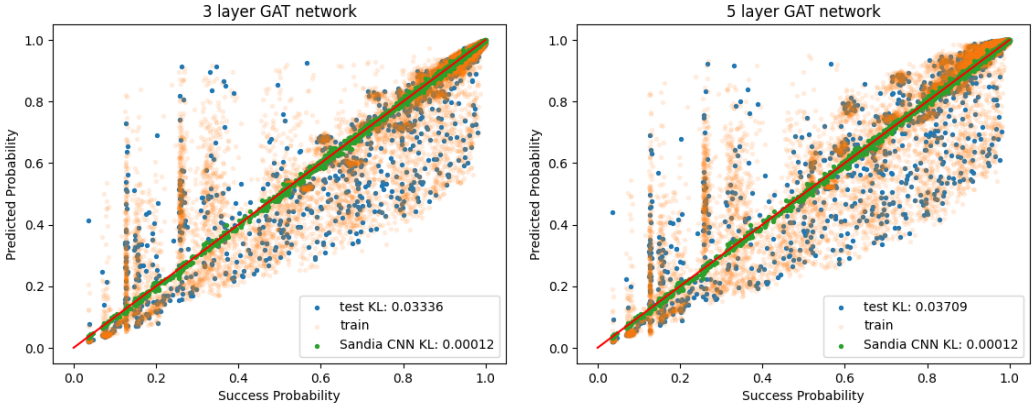


Figure 5: The figure shows the distribution of train and test predictions for the 3 and 5 layer GAT model on the left and right side, respectively.

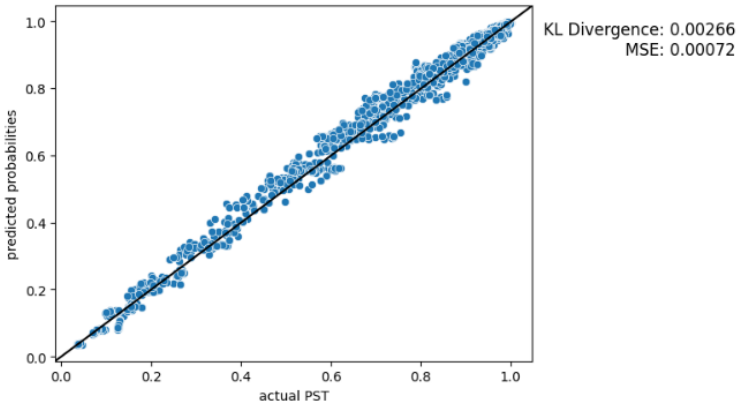


Figure 6: The figure shows the predictions of the GCN model on the gate-centric encoding. The model has 3 GCN layers and 64 hidden dimension.

References

Avi Vadali, Rutuja Kshirsagar, Prasanth Shyamsundar, and Gabriel N Perdue. Quantum circuit fidelity estimation using machine learning. *arXiv preprint arXiv:2212.00677*, 2022.

Hanrui Wang, Pengyu Liu, Jinglei Cheng, Zhiding Liang, Jiaqi Gu, Zirui Li, Yongshan Ding, Weiwen Jiang, Yiyu Shi, Xuehai Qian, et al. Quest: Graph transformer for quantum circuit reliability estimation. *arXiv preprint arXiv:2210.16724*, 2022.