

GENERATING QUANTUM CIRCUITS WITH FINETUNED PRETRAINED
LARGE LANGUAGE MODELS

by

Matthew Choi

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Department of Computer Science
University of Toronto

© Copyright 2024 by Matthew Choi

Generating Quantum Circuits with Finetuned Pretrained Large Language Models

Matthew Choi
Master of Science

Department of Computer Science
University of Toronto
2024

Abstract

Large Language Models (LLMs) have recently demonstrated remarkable capabilities in natural language processing tasks, coding tasks, and beyond. Quantum computing is an emerging technology with the potential to accelerate many computational tasks exponentially. I investigate the efficacy of large language models for solving quantum algorithms by finetuning an LLM to learn quantum state preparation, a critical subroutine in quantum computing. I create models for two-, three-, and four-qubit datasets representing quantum circuits in OPENQASM2.0 and a Python program with Qiskit code and report good fidelity between the target state and the state transformed from the generated quantum circuits. I also introduce expressibility as a necessary measure when creating quantum circuit datasets to ensure they sample the Hilbert space sufficiently.

Acknowledgements

It truly takes a community to raise a child. I would firstly like to thank my supervisor, Dr Alán Aspuru-Guzik, for his advice, supervision, and the opportunity to conduct research in his group.

Next, my fellow lab mates who contributed to discussions for this thesis and made my time as a graduate student enjoyable. Starting with Dr Kouhei Nakaji, Philipp Schleich, Dr Mohsen Bagherimehrab, Dr Abdulrahman Aldossary, Ella Rajaonson, Stanley Lo, Cher-Tian Ser, Austin Cheng, Marko Huang, Luis Mantilla, Andreas Burger, Dr Abhinav Anand, and Dr Rodrigo Vargas.

Lastly, I want to thank my family for their continuous support: my parents for their love and patience, and my brother for our conversations and tennis sessions.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Related Works	3
2	Methods	6
2.1	Preliminaries	6
2.1.1	Quantum Computing	6
2.1.2	Transformers and Large Language Models	9
2.1.3	Low-Rank Adaptation (LoRA)	14
2.1.4	Expressibility of Quantum Circuits	15
2.2	Dataset Generation	16
2.2.1	Circuit Generation	16
2.2.2	Cleanup and Extracting State vectors	19
2.2.3	Create Prompt	20
2.3	Optimization and Training Details	21
2.3.1	Language Models of Code	21
2.3.2	Tokenization	22
2.4	Inference	22
2.4.1	Large Language Model Inference	22
2.4.2	Sampling Methods	23
3	Results	25
3.1	Evaluation strategy	25
3.1.1	Fidelity	25
3.1.2	Evaluation Pipeline	26
3.2	Two qubit system	27

3.3	Scaling to larger qubit systems	31
3.4	Benchmarking to other works	33
4	Conclusion	35

Chapter 1

Introduction

Pretrained Large Language Models (LLMs) have shown incredible capabilities in natural language processing. Since their infancy [1], these models have spread into different fields of science. Today, LLMs have been applied to tackle problems in chemistry, such as generating metastable inorganic compounds [2] and biology, like generating protein sequences [3]. LLMs have also started to enter the realm of physics and quantum computing. These models have been explored for quantum architecture design [4], ground state search [5], and a Qiskit code assistant for aiding Qiskit Users [6]. A new Qiskit quantum computing dataset has been released to evaluate an LLM’s ability to solve quantum computing tasks as well [7]. Many classical machine learning algorithms have been proposed for various quantum problems. This work focuses on quantum state preparation, a crucial subroutine in many quantum algorithms.

Classical machine learning methods for quantum state preparation have been explored extensively. This work presents another flavor of machine learning for quantum state preparation. I propose Low-Rank Adaptation (LoRA) [8], a finetuning method on a pre-trained LLM where, given some state vector, the model learns a quantum circuit that approximately prepares that state vector. Formally, given some natural language prompt containing information on a desired state vector, can a finetuned LLM produce a string representation of a quantum circuit that transforms some initial state ($|0^n\rangle$) to the desired quantum state?

I create models for two, three, and four qubit systems and report good fidelities between target quantum states and states from the generated quantum circuits with fidelities of 0.9, 0.81, and 0.72, respectively. While I cannot achieve the 0.99 fidelity in related works with a similar design pipeline, I show that their datasets do not

sufficiently sample the Hilbert space, creating misleading results. While I acknowledge that many important quantum algorithms require more qubits, I conduct experiments to show that with more data, the model could achieve better fidelity and scale to more qubits. Quantum state preparation usually involves complex techniques that require at least a decent amount of domain knowledge to understand, let alone implement. My work proves that an LLM can learn quantum computing problems through natural language responses that are intuitive to humans, potentially reducing the barrier of entry into the field.

1.1 Motivation

Quantum State Preparation is a critical subroutine in quantum computing, which aims to load a classical $N = 2^n$ complex vector $v = (v_1, \dots, v_N)^T \in \mathbb{C}^N$ into a quantum computer by preparing the n -qubit state

$$|\psi\rangle = \sum_{k=0}^{N-1} v_k |k\rangle$$

where $\sum_{k=0}^{N-1} |v_k|^2 = 1$ and $|k\rangle \equiv \{|k_n k_{n-1} \dots k_1\rangle\}$ being the basis with bits k_j for $j = 1, 2, \dots, n$. Quantum state preparation finds a quantum circuit that transforms the state to a desired point and is used in many quantum algorithms. For example, state preparation is necessary for solving linear systems of equations in quantum computers. Harrow et al. [9] present an efficient way of solving $Ax = b$ on a quantum computer by first performing state preparation on the b vector. A follow-up algorithm for solving the quantum linear systems problem by Childs et al. [10] also requires state preparation of the b vector. State preparation algorithms are also widely used in quantum machine learning as well, the process of training machine learning models on a quantum computer to encode the classical data into the quantum computer [11–13]. It has significant applications in chemistry, drug design, and materials design, which often try to find the ground state energy. Full configuration interaction (FCI) is a classical method used to find the ground state energy with high accuracy, but the computational cost increases exponentially with the system size. Thus, quantum algorithms such as the Variational Quantum Eigensolver (VQE) [14] and the Quantum Phase Estimation (QPE) [15] have been developed to find good approximations of the ground state energy. However, both techniques only significantly reduce computational cost if the initial states of the algorithm have a good overlap with the ground

state of the system and state preparation is used to generate these such states.

Given the importance of state preparation, this work aims to contribute to the use of classical machine learning for quantum state preparation. I first investigate whether LLMs can be used as a submodule for arbitrary state preparation and present another flavor of machine learning models used for this task. Secondly, quantum state preparation techniques require extensive expertise in quantum algorithms and quantum computing; I want to investigate whether an LLM can learn quantum state preparation only through natural language. An LLM that is excellent at performing quantum state preparation could significantly reduce the barrier to entry into quantum applications and research. Researchers and quantum computing beginners can easily download these models and supply a natural language input to infer from the LLM a string representation of a circuit that prepares the desired target state.

1.2 Related Works

Brute force quantum state preparation is typically expensive because encoding any general n -qubit quantum state requires $O(2^n)$ circuit depth since all 2^n parameters have to be encoded using different quantum gates [16]. Furthermore, we are in the NISQ [17] era and multi-qubit Controlled-NOT (CNOT) gates are expensive to implement on a quantum computer, so researchers have another constraint to use as little of them as possible in general. Indeed, the sole contribution of some works is to reduce the CNOT count for state preparation, [16, 18]. Sun et al. [19] showed that they achieved an optimal result with $\Theta(2^n/n)$ circuit depth. More importantly, they showed they could achieve subexponential scaling using ancillary qubits and reduced the gate count to $\Theta(n)$. Unfortunately, these ancillary qubits scale by $O(2^n)$. Rosenthal et al. [20] proposed another solution that had $\tilde{O}(2^n)$ ancillary qubits. Araujo et al. [21] show that using a divide and conquer quantum algorithm and ancillary qubits, they can perform state preparation with $O(n)$ qubits with polylogarithmic depth, albeit for somewhat prohibitive experiments.

Optimization-based methods have also been explored for Quantum State Preparation. Variational methods [22] and Quantum Generative Adversarial Networks (GANs) [23] have been proposed by training circuits with low depth and qubit count to approximate the quantum state. More recently, tensor networks [24] and Walsh Series Loaders [25] have been used to approximate the target quantum state.

Given the rise of neural networks since AlexNet [26], machine learning has been

used in quantum physics as a black-box model to learn relationships between different scientific concepts [27]. When using classical machine learning for quantum state preparation, there are two glaring problems. As we increase the number of qubits, the problem space, known as the Hilbert space, scales exponentially. One needs exponentially more data to ensure the Hilbert space is well represented, leading to significantly longer training times. Secondly, the size of the state vector scales by 2^n , so the model has to learn a mapping with exponentially more degrees of freedom, making it challenging to learn something meaningful. Given these restrictions, researchers using machine learning to solve quantum state preparation have mostly remained within two boundaries. First, the number of qubits is small but aims to generalize to arbitrary state vectors. Later, I show that works proclaiming excellent fidelity (> 0.99) do not generalize well. Second, they target larger qubit systems (> 6 qubits) but only on states with unique properties such as symmetries and sparse state vectors. This reduces the dimensionality of the problem and, consequently, will not generalize as well as arbitrary state vectors.

Despite the issues, reinforcement learning has been a popular technique for learning arbitrary state vectors because it is natural to view state preparation as a culmination of building blocks to achieve the target state vector. Bukov et al. [28] use a modified version of the popular Q-learning RL algorithm to teach a computer to find the best driving protocol to prepare a quantum system from some initial state by controlling a time-dependent field. They extend their methodology to multi-coupled qubits as well. Some works [29, 30] approach quantum state preparation from the Bloch vector representation and use RL to control the spin of the qubit to bring the initial state to some target state. However, since it is not trivial to scale the Bloch sphere representation for larger qubit systems, their work only targets one to two qubits. Zhang et al. [31] perform a comparative study to investigate whether RL is even the best way to approach quantum state preparation by comparing it to non-machine learning methods such as Stochastic Gradient Descent [32] and Krotov algorithms [33]. They show that Krotov and SGD perform better for some settings, proving that RL is not the be-all-end-all algorithm for quantum control.

Moving on to larger qubit systems, Cincio et al. [34] used a simple regression machine learning method to learn W states, states that represent multipartite entanglement, for 4 and 5 qubits. Furruter et al.'s [35] diffusion model approach is also trained on quantum circuits that produce states with specific entanglement, as described by their Schmidt rank vector (SRV), a numerical vector of dimension equal to the number of qubits that contains the rank of the reduced-density matrix for each subsystem.

Porrotti et al. [36] use deep RL to perform state preparation in a cavity subject to quantum-non-demolition photon number detection, with a simple linear drive as control. They show that Fock states ($|1\rangle$ to $|7\rangle$) can be produced and stabilized with very high fidelity. In the realm of variational methods, finding a suitable initial state for VQE is critical because it significantly reduces the computational cost of optimizing for the ground state energy of a Hamiltonian and Shen [37] uses a diffusion model to generate quantum circuits similar to unitary coupled cluster (UCC) circuits.

Lastly, I analyze the works that took a similar approach to mine by sampling quantum circuits from a gate set and then training on the resulting dataset. Moro et al. [38] use reinforcement learning (RL) by treating actions as an elementary gate, using two types of gates: 1. Rotation gates from $\{R_x(\pm\frac{\pi}{128}), R_y(\pm\frac{\pi}{128}), R_z(\pm\frac{\pi}{128})\}$ and 2. Harrow–Recht–Chuang efficiently universal gates. They show promising results in solving at least 95% of their target unitaries, albeit only for one qubit system. Selig et al. [39] generate random circuits from the basis gate set $X, Y, Z, H, T, T^\dagger, S, S^\dagger$, and CX for two to five qubit circuits and show 0.99 fidelity with the inferred circuits and the target state. While not quantum state preparation, Furrutter et al. also used their diffusion model architecture [35] to solve the problem of unitary synthesis. They learn and infer on target unitaries based on the gate set $\{X, Z, CCX, SWAP, H, CX\}$ for three-qubit systems. I show later that these discrete gate sets on larger qubit systems generate circuits that are not well representative of the Hilbert space, and the main reason why they can achieve such high fidelity is that they evaluate the performance of their models in the already small space. Overall, every paper discussed provides an ensemble of methods to sample high-quality quantum circuits, which can serve as oracles to refine the dataset used to train the LLM. I leave that as a future work.

Chapter 2

Methods

2.1 Preliminaries

2.1.1 Quantum Computing

Quantum Computing deals with harnessing quantum-mechanical phenomena to perform computation. Like gate operations and wires in an electric classical circuit, quantum computation is built from a quantum circuit with wires and gates to pass and manipulate quantum information. One of the big differences between a classical and quantum bit is that it can be in a state other than $|0\rangle$ or $|1\rangle$. A quantum bit can be in a superposition of states $\alpha|0\rangle + \beta|1\rangle$, where $|0\rangle$ and $|1\rangle$ are computational basis states that form an orthonormal basis for the \mathbb{C}^2 vector space and α, β are L2 normalized. The restrictions on α and β arise because when extracting information from the quantum bit, quantum mechanics tells us we can only acquire restricted information within the projection of a computational basis state. So when we measure the qubit, we get the state $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Naturally, from the axioms of probability theory, $|\alpha|^2 + |\beta|^2 = 1$. Geometrically, we can interpret this as the condition that the qubit's state vector be normalized to length 1. One can represent a single qubit as a vector in the same form of

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

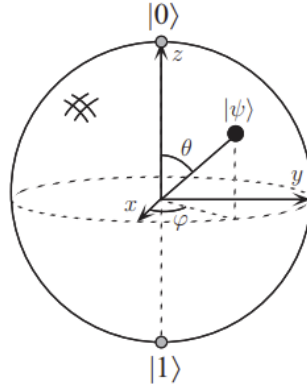


Figure 2.1: An illustration of the Bloch sphere. Taken from [40].

where α and β are complex numbers. Also, since, $|\alpha|^2 + |\beta|^2 = 1$, we can also represent 2.1 now as

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \right) \quad (2.2)$$

We can then formulate ψ and γ as points on a three-dimensional sphere, usually called the Bloch Sphere 2.1. The Bloch sphere is a useful representation because any single-qubit operations are well described on the Bloch Sphere. Unfortunately, it is impractical to use the Bloch sphere representation in higher dimensions because it becomes difficult to visualize.

Single Qubit Gates

$|\alpha|^2 + |\beta|^2 = 1$ also forms constraints on what can be a quantum gate because the norm of the quantum state has to be preserved after a gate applies a transformation to the state. It turns out that the only constraint on quantum gates is that they must be unitary, i.e., $UU^\dagger = U^\dagger U = I$, where U^\dagger is the conjugate transpose of U and I is the identity matrix. Firstly, I introduce the Pauli Matrices, one of the important operators in quantum computing:

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (2.3)$$

X is the bit-flip operator (also known as the NOT gate), Z is the phase operator, and Y is the bit-phase flip because $XZ = iY$. When we exponentiate the Pauli matrices σ

by $e^{-i\theta\sigma/2}$, we get the useful rotational matrices, which take θ as an input. The Pauli matrices are hermitian as well and are used as observables.

$$R_x \equiv \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad R_y \equiv \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad R_z \equiv \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \quad (2.4)$$

These matrices are called rotational because, for some angle θ and Pauli matrix σ_n , the operation $R_n(\theta)$ translates to rotating the quantum state by angle θ on the n -axis on the Bloch sphere. The remaining single-qubit quantum gates used in this work are the Hadamard (H) gate, The Phase (S) gate, and the $\pi/8$ (T) gate:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S \equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad T \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \quad (2.5)$$

The H gate is one of the most used gates in quantum computing because it sets $|0\rangle$ into a superposition of both states $|0\rangle$ and $|1\rangle$ with equal probability. The S gate is sometimes also known as the Z90 gate, where in the Bloch sphere representation, it rotates the state 90 degrees along the z-axis. The T gate is $S = T^2$.

Multi-qubit gates

The only multi-qubit gate we use is the CNOT gate, also called the CX gate. The CNOT gate is a two-qubit gate that performs a Pauli-X gate on the target qubit when the control qubit is in state $|1\rangle$ and leaves the target qubit unchanged when the control qubit is in state $|0\rangle$. It is represented by the 16x16 matrix:

$$CNOT \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.6)$$

The $\{CNOT, H, S\} + T$ gate set forms a universal gate set, meaning all unitary operations can be expressed as a finite sequence from that gate set and is especially useful when designing datasets for arbitrary state preparation.

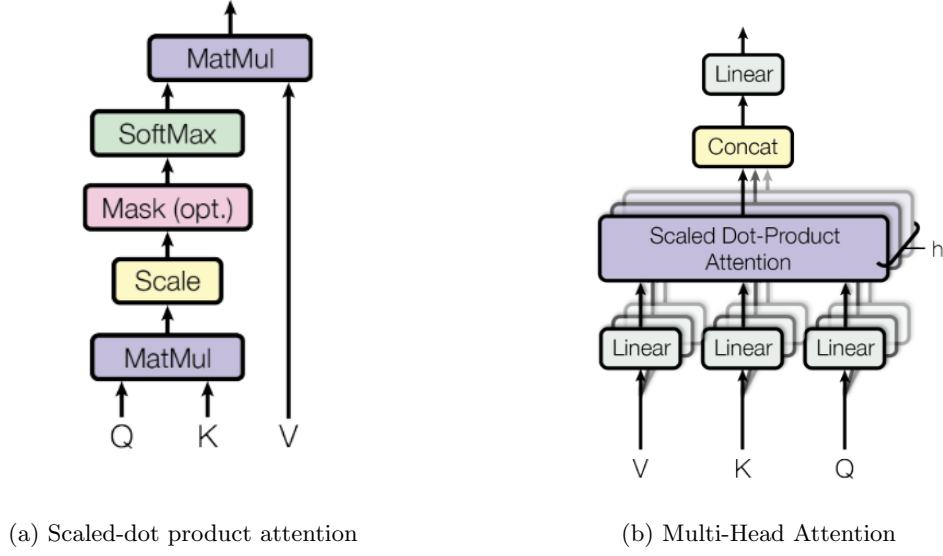


Figure 2.2: Images were taken from Vaswani et al. [1]

2.1.2 Transformers and Large Language Models

Attention

Most LLMs are built off the transformers architecture [1]. The core of this is the attention mechanism, which calculates the highest correlations between words within a sentence. Suppose we had an input sequence x and three weight matrices W_Q, W_K, W_V that project the inputs into query, key, and value projections, abbreviated by Q, K, V and have dimensions of d^k, d^k, d^v respectively. The attention function, which they coin as scaled dot product attention 2.2a, is then:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d^k}}\right) V \quad (2.7)$$

where QK^T is known as the attention matrix and $QK^T[i, j]$ contains the pairwise attention between x_i and x_j , the i -th index of the input sequence and j -th of the input sequence. Softmax is then applied to ensure each attention score is scaled between 0 and 1, forming a valid probability distribution. The authors divide the attention matrix by $\sqrt{d^k}$ because they suspect that for large values of d^k , the dot products might grow in magnitude, resulting in extremely small gradients when calculating the

softmax. They add the constant to help with training.

Multi-Head Attention

Multi-head attention [2.2b](#) parallelizes scaled dot product attention into smaller chunks by linearly projecting Q, K, V h times with different linear projection weights. The multi-head attention is given by

$$\begin{aligned} \text{MultiHeadAttention}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h) \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^W) \end{aligned} \quad (2.8)$$

where W_i^Q, W_i^K, W_i^W are projection matrices for a particular head i . Each attention head is then concatenated to form the multi-head attention. The number of heads h is a hyperparameter, and the dimensions for d^k, d^v scale with the number of heads to keep the computational cost of multi-head attention similar to scaled-dot product attention.

Masked Multi-Head attention

The only difference with masked multi-head attention is that when the attention matrix QK^T is calculated, the values of the indexes above the diagonal are set to a large negative number to ensure it does not contribute any weight to the attention function. Alternatively, one can view masking the attention scores of the current token as ensuring it does not depend on future tokens in the sequence.

Feed Forward Transformation

Besides attention, the transformer also uses a pointwise feed-forward transformation. It applies a fully connected neural network layer through every position of an input token vector separately and identically. It consists of 2 linear layers with an activation function in between and is given by:

$$FFT(x) = \Sigma(W_1x + b_1)W_2 + b_2 \quad (2.9)$$

where Σ is the chosen non-linear activation function, W_1, b_1, W_2, b_2 are the weights and biases of the first and second linear layer, respectively. While the original transformer paper uses the ReLu activation function, most LLMs today use GeLu [\[41\]](#) or SwiGlu [\[42\]](#), which supposedly yield the best performance when training these models.

Layer Normalization

Neural network models were not always easy to train, especially when the values of the data had different magnitudes of order. For example, if input sample x_i has a range of 0 to 1 and x_{i+1} has a range of 50000-100000, vanilla neural network models tend to produce unstable gradients during backpropagation, leading to longer training times or even failure to converge. Hence, preprocessing the data is usually beneficial to training these models. One of the methods of preprocessing is layer normalization [43], which normalizes each input along all layers/features independently. Ba et al. created layer normalization to tackle the issues of batch normalization [44], which depended heavily on the batch size used for training. Batch normalization normalized each input across each feature along the batch, so having a small batch size would skew the batch statistics heavily, making the model more difficult to train, which is especially true for larger models, which generally train on smaller batch sizes due to its size.

Instead, suppose $a^l = W_i^{lT} h^l$ where a^l is the vector representation of the summed inputs to the neuron in a layer l . W_i^l is the weight matrix for layer l at the i -th neuron and h^l be the bottom up inputs. The mean and standard deviation of the layer can be calculated as:

$$\begin{aligned}\mu^l &= \frac{1}{H} \sum_{i=1}^H a_i^l \\ \sigma^l &= \sqrt{\sum_{i=1}^H (a_i^l - \mu^l)^2}\end{aligned}\tag{2.10}$$

Where H denotes the number of hidden units in a layer, one can normalize the layer with those statistics.

Positional Encoding

The transformer does not use recurrence or convolution. However, it still needs to inform the model of the sequence order by injecting some information about the relative or absolute position of a token in the sequence. The transformer does this through positional encoding. A vector v' is created with the same dimension as the token embedding, allowing both v' and the token embedding to be summed. In the paper,

the positional encoding is then defined as:

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{model}}) \\ PE(pos, 2i+1) &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \quad (2.11)$$

where pos is the token's position in the sequence, and i is the dimension. d_{model} is usually the same as d^k and d^v . The authors hypothesized that using a sinusoid as the positional encoding allowed the model to extrapolate to longer sequences than the data found in the training dataset.

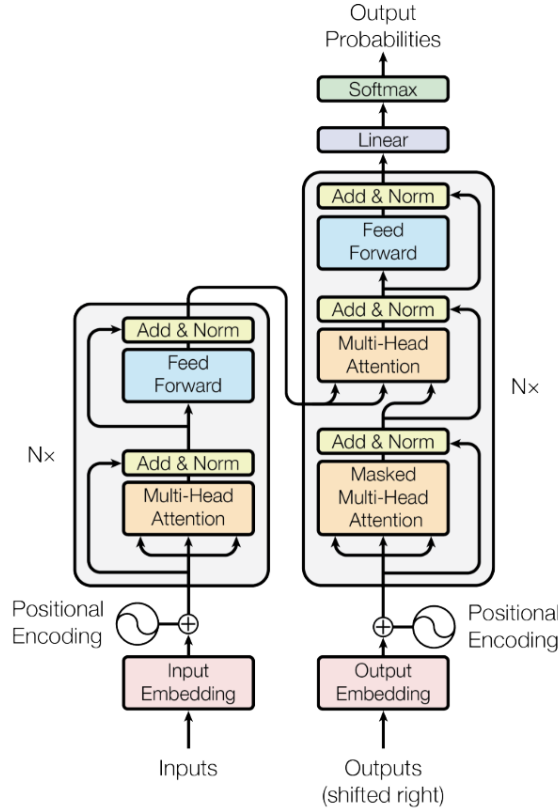


Figure 2.3: The architecture for the encoder-decoder transformer. Decoder-only architectures generally consist of the right portion of the architecture only. Image is taken from Vaswani et al. [1]

Encoder-Decoder Transformers

The transformer architecture is the culmination of all the techniques mentioned above. Given some tokenizer that maps an input x into an array containing the tokenized input t_x , it is summed with its positional encoding and passed through a single encoding layer. Note that there can be more than 1 of these layers, depending on the size of the model. Between each sub-layer, there is a residual connection with the original input vector, introduced in ResNets [45]. The residual connection aims to reduce exploding or vanishing gradients by providing a shortcut path for gradients to back-propagate over, which often happens with very deep neural networks.

The transformer’s decoder module follows a similar pipeline to the encoder module, but instead of multi-head attention, it uses masked multi-head attention. Lastly, it passes through the language modeling head, which is a linear layer followed by the softmax to form a valid probability distribution for each token in the vocabulary.

Decoder only Transformer Models

Many LLMs today, such as GPT and Llama, use a variant of the original transformer architecture known as the decoder-only transformer. While we can attribute this to the success of OpenAI’s GPT models, Liu et al.’s [46] initial paper introducing the decoder-only transformer explains that it reduces the amount parameters needed by half because it does not contain an encoder module. Furthermore, it can be trained with unlabeled data, reducing the cost incurred with gathering and labeling the data before training the model.

Training a decoder-only LLM

The heart of training an LLM is next-token prediction. Suppose we had some token sequence $U = \{u_1, \dots, u_n\}$, we want to maximize the conditional probability of the current token given its preceding tokens and the model. In particular,

$$\arg \max_{\theta} \mathcal{L}_U = \sum_{i=1}^n P(u_i | u_{i-1}, \dots, u_1, \theta) \quad (2.12)$$

The benefit of training decoder-only transformers is that since the attention matrix is masked, the logits can be calculated for every token in a sequence with one forward pass. Similarly to encoder-decoder transformers, the training is self-supervised, as we do not require labels to train. After passing through the feed-forward transformation

layers, the cross-entropy loss is computed between the ground truth input of the token sequence and the logits predicted from the language modeling head layer for each token. The loss is finally averaged over all the tokens in the sequence and then back propagated through all the layers.

2.1.3 Low-Rank Adaptation (LoRA)

One of the most popular techniques used today for finetuning LLMs, Low-Rank Adaptation (LoRA) [8], is one of the early techniques that gave rise to parameter-efficient finetuning (PEFT) [47], a subfield of research within the LLM space. The main goal of PEFT is to find ways to reduce the number of parameters needed to train big models, such as LLMs and diffusion models. Finetuning was first introduced in *InstructGPT* [48], which is an OpenAI model that learned to follow instructions when trained further on a new dataset on top of OpenAI’s pre-trained GPT3. Researchers quickly realized an issue: finetuning on new datasets would not scale. Firstly, these models were large; training them even for a few epochs will require extreme amounts of compute and memory. Secondly, training these models for n datasets would result in n X number of parameters of memory needed to store these models. For example, finetuning **Llama2-70b** on ten datasets would require almost 230GB of storage for these ten new models. PEFT aims to alleviate these issues.

There are a few methods that directly inspired LoRA [49, 50]. Based on these works, Hu et al. hypothesized that the update matrices, when training a transformer, lay in a low intrinsic dimension. Hence, we can project these update matrices into a smaller subspace, potentially reducing the number of parameters. Suppose we define the pre-trained weights of the model as W_0 , one can constraint the update matrices ΔW to $\Delta W = BA$ as a matrix multiplication approximation, where $\Delta W \in \mathbb{R}^{d \times k}$, $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$. Notice that if $r \ll \min(d, k)$, this significantly reduces the number of trainable parameters, which the authors show that even for small values of r , it can retain good accuracy from the original model while reducing the number of trainable parameters by up to 99%. A full forward iteration of LoRA starts by passing the inputs through both the pre-trained model and the LoRA weights and sums their respective outputs coordinate-wise. The weights of the original model remain frozen, and we only back-propagate through the LoRA weights. To summarize, for the original $h = W_0x$, the modified forward pass is now:

$$h = W_0x + \Delta Wx = W_0x + BAx \quad (2.13)$$

where A is initialized as a random gaussian matrix and $B = 0$, the new update matrix is scaled by $\frac{\alpha}{r}$, the two most important hyperparameters when finetuning with LoRA. r , as defined above, stands for rank, which controls the number of parameters of the finetuned model and the expressiveness of the model to learn the new dataset. α is a scaling factor that controls how much weight we put on the output from the LoRA matrices compared to the pre-trained model. For example, $\alpha > r$ would entail that we favor the outputs from LoRA matrices more and vice versa. Typically, α is greater than r as we want the finetuned model to pick up the nuances in the new dataset.

In the original paper, the authors only applied LoRA to the attention layers of the transformer. Since the attention layers only process how the tokens are related to one another, one can symbolically view applying LoRA to only attention layers as changing the way the LLM responds to a user, which is helpful for AI alignment tasks [51] where the corpus of the new dataset does not change much. However, for other tasks that require an injection of knowledge, such as this work, we apply LoRA to the linear layers of the LLM as well.

2.1.4 Expressibility of Quantum Circuits

Our models should be able to approximate any given state vector. Thus, we must ensure we generate a training dataset that can produce circuits well representative of the Hilbert space. Sim et al. [52] introduced a simple mathematical method to quantify the expressibility of a quantum circuit. Specifically, expressibility is the circuit's ability to generate pure states representative of the Hilbert space. In their work, they introduce expressibility for parameterized quantum circuits (PQC) and compare the distribution of fidelities measured from sampling different sets of rotation parameters of a PQC to the fidelities from the uniform distribution of states, known as the ensemble of Haar Random States. They perform this comparison by utilizing a popular technique in machine learning literature known as the Kullback–Leibler (KL) divergence [53].

They define expressibility as:

$$Expr = D_{KL}(P_{PQC}(F, \theta), P_{Haar}(F)) \quad (2.14)$$

where $P_{PQC}(F, \theta)$ is the distribution of fidelities between 2 states sampled from the PQC and $P_{Haar}(F)$ is the distribution of fidelities of two states sampled from the Haar random measure. Notice that there is an analytical form of the probability density function of Haar random states $p_{Haar}(F) = (N-1)(1-F)N-2$, where F corresponds

to the fidelity and N is the dimension of the Hilbert space. [54]. In her paper, Sim tests her method on expressible circuit architectures in the literature, and in general, these circuits have D_{KL} values of less than 0.1. Hence, this is the baseline we are trying to reach when generating our quantum circuit datasets.

While our dataset does not use PQC's, this technique can still be applied for my purposes. Suppose we have 2 pure states $|\psi_y\rangle = U_y|0\rangle$ and $|\psi_x\rangle = U_x|0\rangle$. Since they are pure states, we know the fidelity between those two states is $|\langle\psi_y|\psi_x\rangle|^2$ (a proof can be found at 3.1.1). Next, suppose we have some unitary U_z expressed as $U_z = U_y^\dagger U_x$. A simple proof is that any unitary U can be represented as $V(V^\dagger U)$, since $VV^\dagger = I$. Hence, we can view the unitaries (quantum circuits) generated by our dataset as the matrix multiplication of 2 random unitaries and calculate the fidelity $|\langle\psi_y|\psi_x\rangle|^2$. Notice that this is equivalent to $|\langle 0|U_z|0\rangle|^2$, which is the expectation value of U_z when measuring the $|0\rangle$ state.

Instead of using the rotational parameters of PQC's as samples of the distribution, we can view a randomly generated circuit in our dataset as U_z . This provides a measure to quantify the expressiveness of our datasets. Furthermore, we do not need to compare different circuits in our dataset, which is done in Sim's work, simplifying the method overall.

2.2 Dataset Generation

To investigate whether a transformer can learn to map a string representation of quantum state vectors to a string representation of quantum circuits, I create a question-answering dataset with an English natural language prompt containing the desired state vector and its corresponding string representation of a quantum circuit. A summarized illustration of the pipeline for dataset creation is at 2.4.

2.2.1 Circuit Generation

Intuitively, since the model should generalize well, one could simply sample Haar Random circuits. Unfortunately, LLMs have historically struggled with mathematics. Dziri et al. [55] showed that long decimals can be detrimental to the performance of LLMs. Hence, we have to play a balancing act between the expressibility of the generated circuits and the string representation of the quantum circuit itself. I limit the gates and rotational parameters selected for the quantum circuit so that the LLM can reasonably learn this string representation. I settled for circuits with a maximum

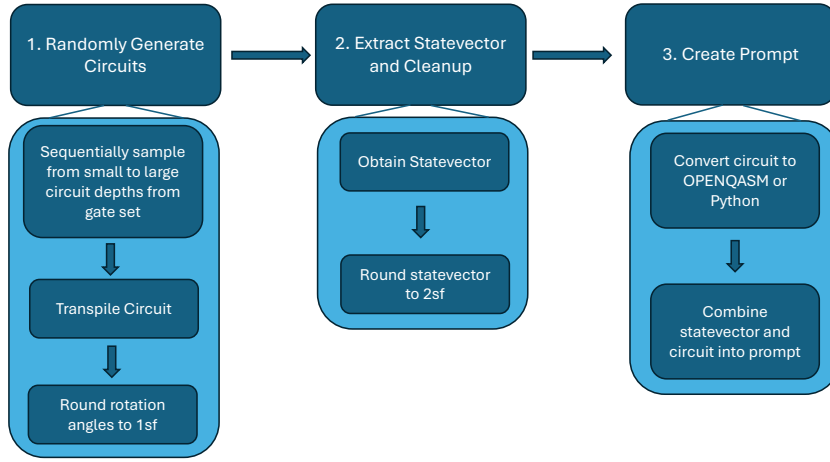


Figure 2.4: Dataset creation pipeline

of ten gates per qubit and an additional Hadamard Gate at the beginning of each qubit.

Initially, I started with the universal set of discrete gates $[X, Y, Z, S, H, CNOT]$ in hopes that the gate set would make it easy for the LLM to learn as the only numbers present in the circuit would be the respective qubit number for the gate. However, the gate set led to average expressibility values of around 0.8, a far cry from the 0.1 baseline. I replaced X, Y, Z with their rotational variants, with rotational angles in multiples of π as an angle set for these gates. I chose these angles as one can represent π as `pi` in a string instead of some decimal, reducing the complexity of the string representation of the circuit. Based on the expressibility comparisons in 2.5, while the distributions of the generated circuits have unwanted peaks and valleys compared to the Haar random measure, all three datasets approached the 0.1 baseline. I found it a good compromise for the LLM to learn. The final gate set chosen is $[Rx, Ry, Rz, H, CNOT, T, S]$ and the rotational angle set is $[\pi/5, \pi/4, \pi/3, \pi/2, \pi, 2\pi]$. All of these parameters are randomly sampled.

I build the dataset by sampling from a minimum of 1 gate per qubit to a maximum of 10 gates per qubit. We process circuits in ascending order, i.e., we generate circuits with one gate per qubit and store the state vector in a set before moving on to more gates per qubit. I perform this sequential process because if we reencountered the same

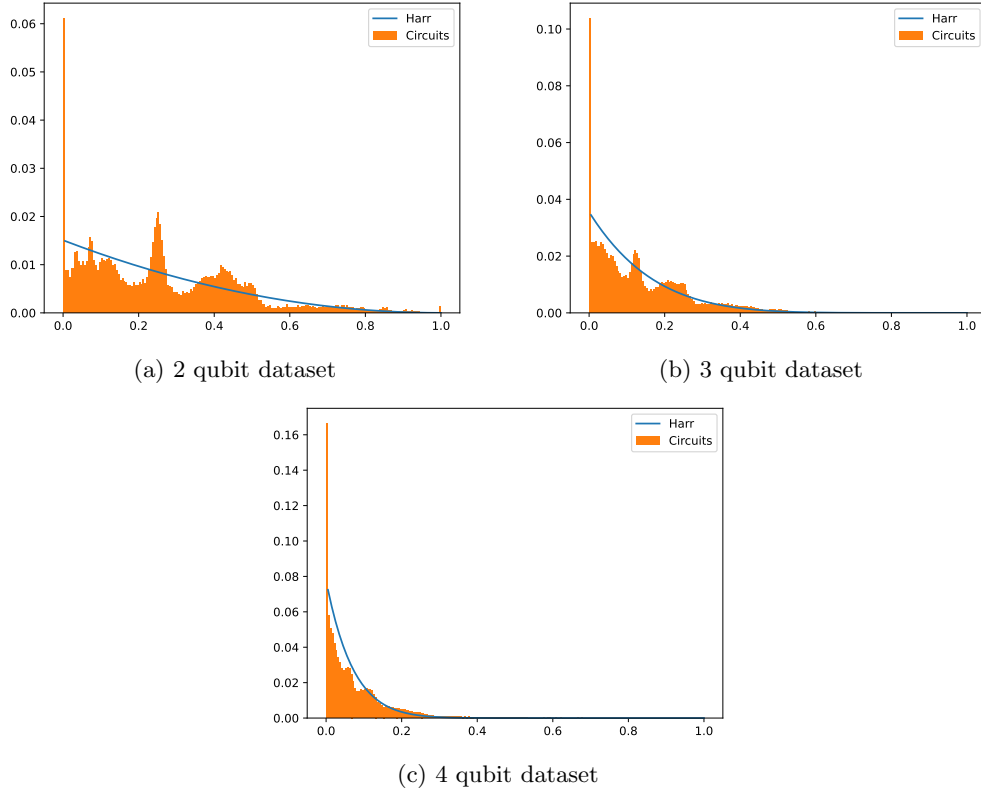


Figure 2.5: The expressibility graph for each n qubit dataset (orange bins) compared to the Haar random measure (blue line). The values of expressibility for the two, three, and four qubit datasets are 0.146, 0.111, and 0.116, respectively.

state vector, there would already be a more efficient quantum circuit in the dataset, so I could safely discard the current one. This design decision also implies that our dataset only contains unique state vectors.

However, purely generating random circuits without further processing has some pitfalls. For example, if we sample two Hadamard gates back to back for the same qubit, the two gates cancel out. Consider the case where two generated circuits in the dataset have similar state vectors, but one has the two Hadamard gates while the other does not; it will likely confuse the model. Hence, I chose to transpile the circuit as well. I used the Qiskit transpiler with optimization level 1 and the same gate set parameters as the original dataset. Upon inspection, the transpilation generally only removed back-to-back Hadamard gates and optimized a series of gates to $RzRyRz$

decompositions. Overall, the transpiling step removed redundancies and decreased the gate count of the generated circuits.

2.2.2 Cleanup and Extracting State vectors

Since LLMs are not good at mathematical problems, rounding is necessary for the pipeline. Decimals in the rotation angles were rounded to one significant figure, making the circuit's string presentation more concise and straightforward. I converted all instances of π represented in decimal into multiples of π . For example, 6.28318530718... would become $2 * \pi$. Thankfully, Qiskit had an implementation for this functionality called `pi_check`, which I tweaked for this project. After cleaning up the circuit, I extracted the state vector and rounded the complex values to two significant figures. I create two datasets for two, three, and four qubit circuits. The first is OPENQASM 2.0 code, and the other is a Python program in Qiskit code.

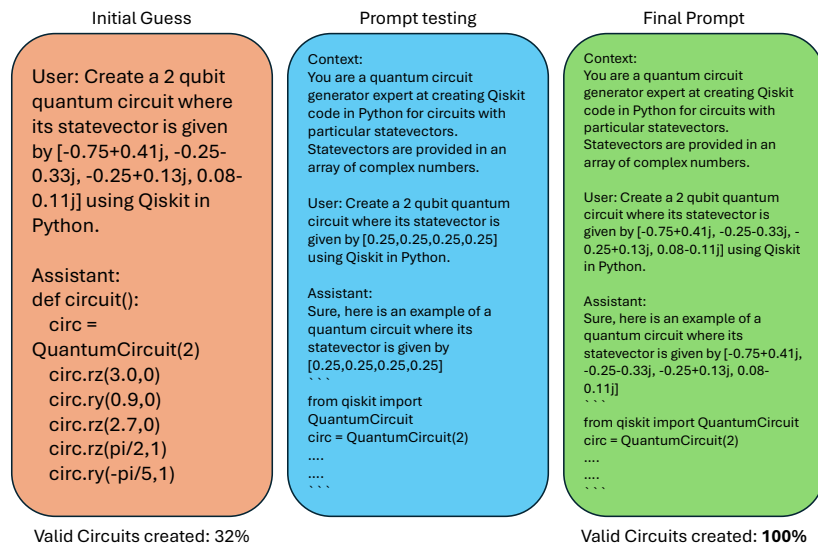
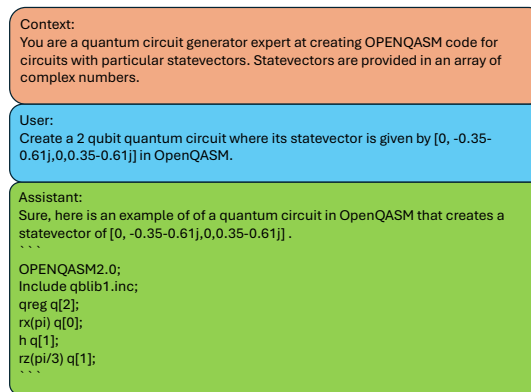


Figure 2.6: Prompt Testing. Instead of forcing the LLM to conform to a prompt format, prompt the base model with a small portion of the dataset to understand the output format and build your instruction fine-tuning dataset around the outputs. When trained on a forced dataset, the number of valid circuits created during inference is 32%, compared to 100% with a dataset built around the outputs from the base LLM.

2.2.3 Create Prompt

Now that we have all our quantum information, we can design a prompt to train the model. Interestingly, choosing how to represent the data as a natural language prompt posed a significant challenge. I started with a concise prompt format to reduce the dataset’s average token lengths (shown in orange in 2.6). However, as seen in 2.6, training a model on this initial prompt format resulted in only 32% valid circuits during inference. I started testing the base LLM to determine what it “likes” to respond by testing easy state vectors like $[0.25, 0.25, 0.25, 0, 25]$ for a two-qubit system. I call this method prompt testing (shown in blue in the same plot). After prompt testing, I crafted a new prompt format similar to the new template and trained another model. Surprisingly, the latest model generated 100% valid circuits on inference. This highlights the importance of the prompt format and can profoundly impact the performance of the fine-tuned models.

A data entry of the final training dataset can be found at 2.7.



```

Context:
You are a quantum circuit generator expert at creating OPENQASM code for
circuits with particular statevectors. Statevectors are provided in an array of
complex numbers.

User:
Create a 2 qubit quantum circuit where its statevector is given by [0, -0.35-
0.61j, 0, 0.35-0.61j] in OpenQASM.

Assistant:
Sure, here is an example of of a quantum circuit in OpenQASM that creates a
statevector of [0, -0.35-0.61j, 0, 0.35-0.61j] .
...
OPENQASM2.0;
Include qelib1.inc;
qreg q[2];
rx(pi) q[0];
h q[1];
rz(pi/3) q[1];
...
  
```

Figure 2.7: One of the training samples for the OPENQASM dataset. The Python dataset follows a similar structure, but where there are words regarding OPENQASM, I replace them with Python in Qiskit. The OPENQASM quantum circuit is replaced with a Python program containing Qiskit code that implements the circuit.

2.3 Optimization and Training Details

2.3.1 Language Models of Code

The original idea was to finetune with OPENQASM2.0 quantum circuits. OPENQASM2.0 is a well-established string representation in quantum computing, so naturally, I wanted to use code LLMs as the base model for finetuning. I was inspired by Madaan et al. [56], who showed that language models of code showed good reasoning capabilities when structuring a reasoning problem in code instead of text. I decided to create another dataset using Qiskit code in Python. The hypothesis is that since Python is prevalent today, the datasets for training these code LLMs will probably train on vast amounts of Python code. If they trained well in Python, they could focus on mapping the relationship between the quantum gates and the state vectors instead of trying to learn the syntax of Qiskit code.

One of my philosophies for this project is that the models have as few trainable parameters as possible, given limited compute. Hence, I chose LLMs with at most 10 billion parameters. As of Apr 2024, I chose the three most prevalent models that perform well on the code portion of the Masked Multi-Language Understanding (MMLU) benchmark [57]. These are **Llama3-8b**, **CodeLlama-7b** and **Deepseek-6.7b**. While **Llama3-8b** is not solely a language model for code generation, it has been trained with significantly more code as compared to its predecessor, **Llama2** [58] and performs better than most code LLMs. My datasets are in a question-answering format, so I chose the **Instruct** variants of all three models as they were already finetuned on instruction-based datasets.

With an average of seven billion parameters, LoRA finetuning with a rank of 16 would be equivalent to about 40 million parameters or 0.8% of the total parameters, which I thought was ample to learn the dataset and set that maximum rank. The models were trained for ten epochs with a learning rate of $2e - 4$ using the cosine annealing schedule with warm restarts [59] at the beginning of every epoch. The learning rate was also warmed up from 0 to $2e - 4$ in the first 250 steps. Loshchilov et al. showed that warming produced good results, especially for the attention mechanisms.

Given that the token length of each prompt increases with system size, the batch size used to train each model decreases but remains a power of two. The two-, three-, four-qubit training batch sizes were 64, 32, and 16, respectively. I split each dataset into 90% training and 10% validation. All models were trained on A100 GPUs with 80GB VRAM. The two— and three-qubit models required 1 GPU, while the four-qubit

model required 2 GPUs.

2.3.2 Tokenization

Finetuning allows for the addition of new tokens as well. Initially, it felt natural to tokenize the circuits by setting the string representation of a gate as one token. For example, the Hadamard gate on the first qubit of the circuit is `h q[0]`, and with the new tokenizer, that would form a single token. Intuitively, this is a great idea, as I can reduce the number of tokens for each data sample, directly leading to faster training times and better inference. Unfortunately, the model did not train well compared to letting the pre-trained tokenizer tokenize the prompt as it would see fit. Gruver et al. also reported this phenomenon in the paper, whose work finetuned `Llama2-7b` on crystalline structures. A hypothesis as to why this happens is that we would need to train the embedding layer of the pre-trained LLM, but it does not respond well to the reduced dimensions when using LoRA. Using the base tokenizer, the same Hadamard gate on the first qubit would look something like `[h, q, [, 0,]]`. I used the base tokenizer for all experiments moving forward.

2.4 Inference

2.4.1 Large Language Model Inference

To perform inference, decoder-only transformer models use autoregressive decoding, an iterative process of predicting the next token given what has already been predicted until either the maximum number of tokens has been reached or the end-of-sentence token has been predicted. For some input token embedding x_i in an input sequence, where i is the i -th index of the sequence, passing the input embeddings through the LLM will result in an output embedding y_i , the output embedding for that particular token. These embeddings then pass through the language modeling head that maps that input embedding to the vocabulary size and outputs raw unnormalized prediction scores for all tokens in the vocabulary, also known as logits. A final softmax layer converts the logits into a valid probability distribution from which we can sample possible tokens. An illustration 2.3 is provided in the decoder portion of the transformer architecture.

2.4.2 Sampling Methods

Now that we have a valid probability distribution of the next token, what is the best way to sample it? While this is still an active area of research, a couple of techniques have been proposed, which we can split into deterministic and stochastic methods.

Deterministic Methods

The first method and the most intuitive is greedy sampling. The token with the top probability at every inference step is chosen until the algorithm terminates. Beam search, another deterministic method, builds upon greedy sampling and searches multiple branches through a hyperparameter, beam width b . For example, if $b = 2$, beam search considers the best two tokens with the highest probabilities at each inference step and builds two possible sequences until it terminates. The sequence with the highest probability is returned as the response. While beam search is more computationally expensive, it usually performs better than greedy sampling. Both methods fall under deterministic methods because they do not involve any randomness in their outputs.

Stochastic Methods

These methods consist of top- k and top- p sampling, which are similar as both try to cut off the tail-end of the probability distribution so that tokens with low probability are not sampled. Top- k forms an ensemble of tokens by choosing the top k tokens with the highest probabilities, while top- p chooses the ensemble of tokens that make up p probability. For example, if we set $p = 0.95$, the ensemble of tokens to be considered would have a sum of at most 0.95. After the ensemble of tokens has been found for both techniques, the probabilities are re-weighted for the ensemble and then randomly sampled. Both methods are stochastic because the token chosen at each inference step is sampled from some distribution. Typically, top- p and top- k produce more coherent results and perform better than deterministic ones.

Temperature

Temperature, τ , controls the randomness of the probability distribution. In particular,

$$\text{Final } L = \frac{L}{\tau} \quad (2.15)$$

Where L are the unnormalized logits, if τ is lower than 1, the probability distribution will be more extreme, i.e., larger probabilities mass will have even more mass and smaller probabilities will have even less mass, leading to more consistent responses across similar prompts. With τ larger than 1, the probability distribution moves towards uniformity, potentially giving rise to more "creative" responses.

Chapter 3

Results

3.1 Evaluation strategy

3.1.1 Fidelity

Fidelity [60] is a distance measure that calculates the overlap between 2 states, ρ and σ , and is given by

$$F(\rho, \sigma) \equiv (tr \sqrt{\rho^{1/2} \sigma \rho^{1/2}})^2 \quad (3.1)$$

where tr is the trace of the matrix. Since our datasets are valid quantum circuits, all state vectors considered are pure states, and there is an even simpler way to calculate fidelity. Suppose we represent $\rho = |\psi_\rho\rangle\langle\psi_\rho|$ and $\sigma = |\psi_\sigma\rangle\langle\psi_\sigma|$. Firstly, notice that $\sqrt{\rho} = \rho$ since the consequence of a pure state is only having a single eigenvalue; thus, the trace will always be 1. Hence, we can rewrite the equation above as

$$\begin{aligned} F(\rho, \sigma) &= (tr \sqrt{\rho \sigma \rho})^2 \\ &= \left(tr \sqrt{|\psi_\rho\rangle\langle\psi_\rho| |\psi_\rho\rangle\langle\psi_\rho| |\psi_\sigma\rangle\langle\psi_\sigma| |\psi_\sigma\rangle\langle\psi_\sigma|} \right)^2 \\ &= \left(tr \sqrt{|\psi_\rho\rangle\langle\psi_\rho| (|\langle\psi_\rho|\psi_\sigma\rangle|^2)} \right)^2 \\ &= |\langle\psi_\rho|\psi_\sigma\rangle|^2 \left(tr \sqrt{|\psi_\rho\rangle\langle\psi_\rho|} \right)^2 \\ &= |\langle\psi_\rho|\psi_\sigma\rangle|^2 \end{aligned} \quad (3.2)$$

$|\langle\psi_\rho|\psi_\sigma\rangle|^2$ is just the overlap between pure states, sometimes known as the transition probability between two states. Consequently, the fidelity of two pure states is easy to compute and bounded from 0 to 1, where 0 signifies they have support on orthogonal subspaces and are thus perfectly distinguishable. Vice versa, when fidelity is 1, $\rho = \sigma$.

In this work, I introduce two evaluation metrics based on fidelity: maximum and average fidelity. During evaluation, I sampled several circuits for each state vector to create an ensemble of generated circuits for each target state vector. Maximum Fidelity means that the circuit that achieves the highest fidelity within the ensemble of circuits is chosen. Average fidelity calculates the average fidelity of the ensemble of quantum circuits and is used as a benchmark to evaluate how the model performs in general. I put more emphasis on maximum fidelity when evaluating the model because the goal is to provide the user with the best circuit possible for a particular state vector.

3.1.2 Evaluation Pipeline

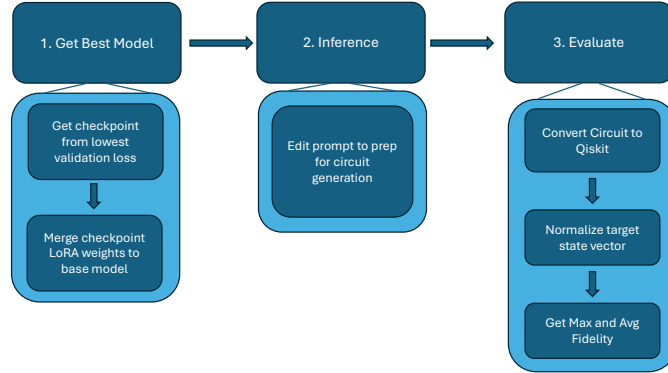


Figure 3.1: The inference pipeline when evaluating the fidelity of the target state against the state produced by the generated quantum circuit.

Across all the models trained, I found that with ten epochs, the model trained long enough to overfit on the data, which lets me pinpoint the LoRA checkpoint with the lowest validation loss. I conduct all experiments on that checkpoint by merging the LoRA weights to the base model. During inference, I remove the string representation

of the quantum circuit from the prompt to inform the model to generate a quantum circuit instead and perform inference on the evaluation set, which is data the model has not trained on. After the model generates a circuit, I convert it into a Qiskit quantum circuit and calculate the fidelity between the resulting state vector and the normalized version of the target state vector. We need to normalize the target state vector because we round the original unit state vector to 2 significant figures to reduce the complexity of the problem for the LLM to learn, which might break the unit vector constraint when using fidelity as a measure. An illustration is provided for the inference pipeline 3.1.

In my initial experiments, the results between top-p and top-k were similar, so I followed most conventional parameters used in research and opted for $top-p = 0.9$. I then sample the same prompt n number of times and calculate the maximum and average fidelity for the ensemble of quantum circuits generated for the target state vector. Using stochastic sampling is a natural approach for bottom-up problems such as circuit building, as it is challenging to map the entire state vector to the first gate when constructing the circuit. Hence, sampling circuits n times allows the model to explore different circuit-building paths to find one that has high fidelity to the target state vector. For consistency among experiments, I conducted ten runs for each experiment and reported their average statistics.

3.2 Two qubit system

A n -qubit system is represented by a 2^n state vector, resulting in exponential scaling with larger qubits. Furthermore, I create circuits with a maximum of 11 gates per qubit. These two factors increase the average token length of the dataset, as seen in 3.2, leading to increasingly long training and inference times. Thus, the motivation behind conducting the 2-qubit experiments is to find a set of inference parameters that achieve the best results before moving on to more computationally expensive three and four-qubit datasets. Essentially, we have a combinatorial problem at hand. However, given the limited compute and the high cost of training LLMs, my first approach was eliminating as many of these inference parameters as possible. The following subsections walk through my thought process of elimination chronologically within the two-qubit dataset. I assume that the best hyperparameters chosen for the two-qubit model will translate to the three, and four-qubit ones.

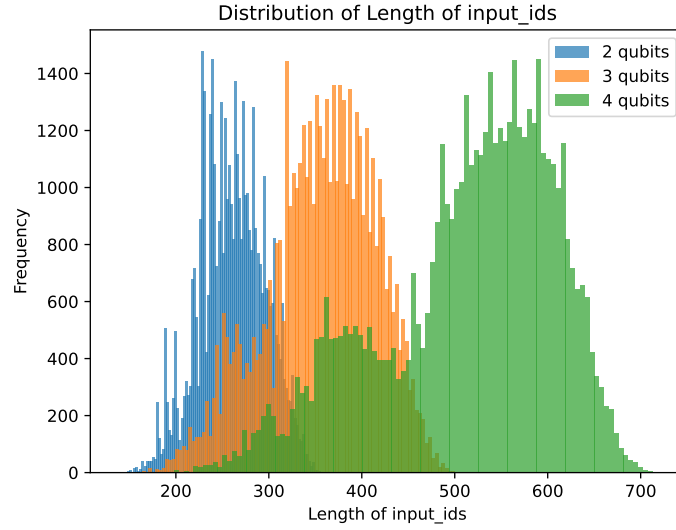


Figure 3.2: Distribution of lengths for `input_ids` for each dataset. The average lengths for the 2,3,4 qubit dataset are 261, 356, and 513, respectively.

Performance across models

The overarching theme of this work is to experiment with the efficacy of large language models on quantum problems; thus, eliminating the number of models to train is the logical first step with limited resources. All three instruct models in consideration were LoRA finetuned on the two-qubit OPENQASM 2.0 circuit dataset with a rank of 8 and an alpha of 32. They were then evaluated on a tiny configuration by sampling ten circuits for each state vector from a randomly chosen subset of 100 target state vectors

hyperparameters: rank=8, alpha=32, num_repeated_prompts=10, val_size=100		
Model	Maximum fidelity	Average fidelity
codellama-7b-Instruct	0.684±0.015	0.236±0.049
deepseek-6.7b-Instruct	0.719±0.008	0.333±0.015
Llama3-8b-Instruct	0.724±0.025	0.348±0.008

Table 3.1: Maximum and Average Fidelities on a low compute training and inference regime on the three models in consideration

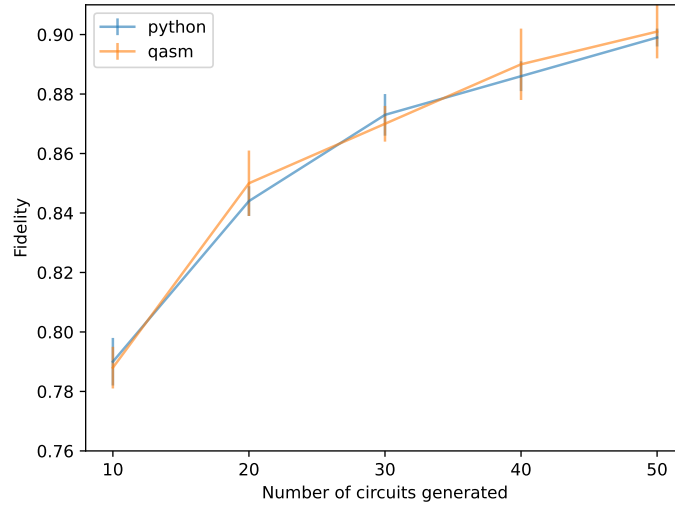


Figure 3.3: The maximum fidelity for a circuit generated compared to the target statevector against the number of circuits generated. Both the Python and QASM models perform similarly well. While the QASM model performs the best at 50 circuits generated at 0.901 fidelity, the Python model has less variance.

in the validation set. Summarizing the results from figure 3.3, **Llama3-8b-Instruct** performed the best with a maximum fidelity of 0.724 and an average fidelity of 0.348 (Table 3.3). Codellama was released first, followed by Deepseek, and lastly, Llama3 and its performance on the datasets following its release date. I hypothesize that each subsequent model trained on increasingly more Qiskit and OPENQASM data, leading to better performance. I only finetuned Llama3 moving forward.

Performance across number of circuits generated

I investigate the optimal number of circuits to sample per state vector that maximizes performance while staying within a reasonable amount of compute. While fifty circuits garnered the best results, the performance gain from forty to fifty circuits is relatively minimal, and decided to stop at 50 because it took around 40 secs for one target state vector (Figure 3.3). I leave sampling even more circuits as a future work, but I hypothesize that the maximum fidelity will eventually plateau, given the slowing increment of maximum fidelity. A table of results can be found at 3.2.

num. circuits generated	Python fidelity		QASM fidelity	
	max	avg.	max	avg.
10	0.79±0.008	0.409±0.009	0.788±0.005	0.412±0.007
20	0.844±0.005	0.406±0.004	0.85±0.008	0.417±0.011
30	0.873±0.007	0.404±0.01	0.87±0.003	0.409±0.006
40	0.886±0.005	0.408±0.012	0.89±0.007	0.412±0.012
50	0.899±0.003	0.401±0.008	0.901±0.005	0.415±0.009

Table 3.2: Maximum and Average fidelity for the number of circuits sampled for 500 randomly target state vectors in the validation set.

Performance across rank

Gruver et al. [2] inspired the decision to set $\alpha = 4 * \text{rank}$, whose finetuning pipeline is similar to mine. This experiment investigates the effect of rank on performance. I trained the Python and OPENQASM datasets on **LLama3-8b** with rank=8 and rank=16. I stopped at rank=16 because the models had around 40 million trainable parameters, which I considered ample to learn the intricacies of the dataset. The goal was also to minimize the number of trainable parameters while maximizing performance. Based on the results in table 3.3, rank=16 produces the best fidelity across both datasets. However, the maximum fidelities between r=8 and r=16 are relatively close for the Python model, suggesting we could save 20 million parameters while maintaining performance. While it is difficult to pinpoint precisely why the Python model performs better with fewer trainable parameters as compared to the OPENQASM model, I hypothesize that **Llama3** is trained on significantly more Python code than OPENQASM, thus requiring fewer parameters to learn the relationship between state vectors and quantum circuits.

Comparison against a random baseline

I quantify how our model performs better with the chosen parameters by comparing them to a random circuit generator for 500 validation samples. We take the rank=16 and alpha=64 models and generate 50 circuits per sample. The results (Table 3.4) show our model learns some representation between state vectors and quantum circuits by outperforming the random baseline by 0.14 fidelity. It also has a much lower standard

LoRA Config	Python fidelity		QASM fidelity	
	max	avg.	max	avg.
r=8, a=32	0.889±0.003	0.384±0.011	0.866±0.005	0.329±0.011
r=16, a=64	0.899±0.003	0.401±0.008	0.901±0.005	0.415±0.009

Table 3.3: Maximum and Average Fidelity across the LoRA configurations of rank and alpha. Rank directly affects the number of trainable parameters in the model, with a larger rank implying more parameters. At Rank=8, there are approximately 20 million parameters, and at rank=16, approximately 40 million. Fifty circuits were sampled for 500 target state vectors in the validation set.

setting	Python fidelity		QASM fidelity	
	max	avg.	max	avg.
random circuit	0.766±0.095	0.25±0.026	0.76±0.09	0.249±0.027
best model	0.899±0.003	0.401±0.008	0.901±0.005	0.415±0.009

Table 3.4: Comparison between a random circuit generator to form a baseline against our best-performing model and inference setting. The model achieves 0.14 more fidelity across both datasets.

deviation among all circuits generated, showing it is more consistent with creating good circuits.

3.3 Scaling to larger qubit systems

We move on to the three-qubit and four-qubit systems. Similarly, I train two models for both qubit systems on the Python and OPENQASM datasets and sample 50 circuits per target state vector. Since the Hilbert space is larger, creating datasets with the same size will undoubtedly result in poorer performance than the two-qubit models. Hence, I increase the dataset size to combat the exponential growth of the Hilbert space. I chose an increment step size of approximately 25k until the model achieved at least 0.7 fidelity. The three-qubit dataset had a size of 71760, and the four-qubit dataset had 100682.

Comparing against random baseline

We test the performance of the larger models by comparing them to their respective random baseline, as done in the 2-qubit system. The results in table 3.5 show that while there is a decrease in fidelity, the model can still generate circuits with decent fidelity and perform increasingly better compared to their random baseline as the number of qubits increases.

setting	Python fidelity		QASM fidelity	
	max	avg.	max	avg.
3-qubit random	0.48 ± 0.101	0.124 ± 0.016	0.47 ± 0.1	0.124 ± 0.016
3-qubit	0.813 ± 0.008	0.391 ± 0.016	0.796 ± 0.012	0.384 ± 0.019
4-qubit random	0.278 ± 0.081	0.063 ± 0.009	0.275 ± 0.078	0.063 ± 0.008
4-qubit	0.729 ± 0.015	0.349 ± 0.017	0.725 ± 0.01	0.333 ± 0.016

Table 3.5: Results on 3-qubit and 4-qubit models and their respective random baselines. I sampled fifty circuits per validation state vector.

Comparison against the size of the dataset

I conducted another experiment investigating the relationship between dataset size and fidelity. I created two smaller Python datasets with 25k and 50k sizes for the three and four-qubit models. I trained and evaluated the resulting models and plotted the average maximum fidelity for 500 validation state vectors (figure 3.4). I hypothesize that if the plot of dataset size against maximum fidelity is linear, then the dataset does not sufficiently sample the Hilbert space, and adding more circuits might achieve better performance. If the plot begins to plateau, there will be diminishing returns to increasing dataset size. The 3-qubit model does show signs of plateauing, while the four-qubit model shows a linear plot. However, increasing the dataset size comes with its problems. Training times will take significantly longer as it has more data to train on, as seen in table 3.6. It would be interesting to see if these larger models can achieve the same level of maximum fidelity as the two-qubit model. I leave this as a future work.

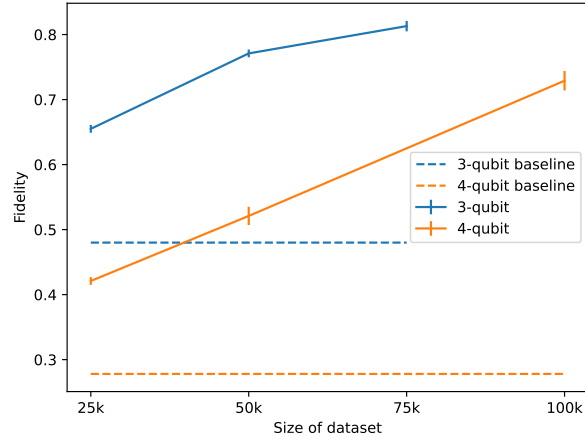


Figure 3.4: Maximum fidelity against the size of the Python training dataset. We train three different models with varying sizes for the three-qubit model (25k, 50k, 75k samples) and the four-qubit model (25k, 50k, 100k).

num. qubits	training time (days)
2-qubit	0.75
3-qubit	1.75
4-qubit	3.8

Table 3.6: Training times of the two, three, four qubit models. These numbers paint a bleaker picture than reality. Since the number of GPUs remains the same, the batch size for training models had to decrease because of the increase in token lengths described in 3.2. A smaller batch size increased each model’s total number of training steps, and since the model evaluation was kept constant at 500 steps, the larger datasets had significantly more evaluation steps, leading to drastically longer training times.

3.4 Benchmarking to other works

I finetuned an LLM on a similar problem statement to Furruter et al. [35]. In their work, they train a three-qubit quantum circuit dataset using an ensemble of gates with a diffusion model. They take a unitary matrix as input and specify the gates their model can choose from. They reported that their diffusion model achieved 92.6% of perfect matches (fidelity = 1) on a test set of arbitrary unitaries when sampling 1024 circuits per unitary. While I finetuned Llama3-8b on their datasets, I swapped out

Paper	num. qubits	Gate Set	Average Expressibility
Furrutter et al.	3	H,CX,Z, X,CCX,SWAP	5.235
Selig et al.	2,3,4,5	X, Y, Z, H, T, T^\dagger , S, S^\dagger , CX	1.319
This work	2,3,4	H, Rx, Ry, Rz, S, T, CX	0.124

Table 3.7: Comparing expressibility among works that randomly sample quantum circuits based on some gate set. My datasets perform better due to using rotational gates with randomly sampled angles and starting all qubits with Hadamard gates to put the state in a superposition.

the unitaries for state vectors. I note that training on state vectors is significantly less challenging than unitaries because the unitary provides information on all basis states, while the state vector only informs on one. In the future, I plan to investigate whether LLMs can perform unitary synthesis. Nonetheless, with the extra constraint that the model can only use certain gates, the finetuned LLMs achieved 92.9% perfect matches with only 20 samples from the Python model and 96.8% on the OPENQASM model with 500 randomly chosen validation target state vectors. The good results show the method’s versatility in adapting to different prompt formats.

I found the 0.99 fidelity in most works somewhat misleading. Hence, skeptical of Furrutter’s [35] and Selig’s [39] high fidelities, I generated random circuits with their chosen gate sets. The results in table 3.7 show that the circuits in my datasets are, on average, 42 times more expressive than Furrutter’s dataset and ten times more expressive than Selig’s dataset. These datasets are nowhere near the 0.1 baseline that Sim et al. showed, so I propose that expressibility be a necessary metric for works that perform arbitrary quantum state preparation.

Chapter 4

Conclusion

This work has demonstrated the effectiveness of learning quantum computing problems through natural language by fine-tuning LLMs on prompts about quantum state preparation for two-, three-, and four-qubit systems. I also propose that expressibility be a necessary metric for predicting arbitrary state vectors to ensure quantum computing datasets are well representative of the Hilbert space. I also show the versatility of LLMs by training on datasets used by other works. This supports the mode-agnostic pipeline I proposed to formulate a quantum computing problem into natural language, demonstrating that it can be applied to other quantum algorithms. Like many open-source LLMs, these models are easy to use with the low effort required to infer a quantum circuit from an input prompt, providing quick access for quantum computing novices and experts to quantum circuits that perform state preparation.

However, I acknowledge that four-qubit systems are relatively simple, and many interesting quantum problems typically require more qubits. With more time and compute, I plan to investigate larger qubit systems and more extensive inference parameters to show the true usefulness of LLMs for quantum computing problems.

Bibliography

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
- [2] N. Gruver, A. Sriram, A. Madotto, A. G. Wilson, C. L. Zitnick, and Z. W. Ulissi, “Fine-tuned language models generate stable inorganic materials as text,” in *AI for Accelerated Materials Design - NeurIPS 2023 Workshop*, 2023.
- [3] L. Lv, Z. Lin, H. Li, Y. Liu, J. Cui, C. Y.-C. Chen, L. Yuan, and Y. Tian, “ProLLaMA: A Protein Large Language Model for Multi-Task Protein Language Processing,” Feb. 2024. arXiv:2402.16445 [cs, q-bio].
- [4] Z. Liang, J. Cheng, R. Yang, H. Ren, Z. Song, D. Wu, X. Qian, T. Li, and Y. Shi, “Unleashing the Potential of LLMs for Quantum Computing: A Study in Quantum Architecture Design,” July 2023. arXiv:2307.08191 [quant-ph].
- [5] K. Nakaji, L. B. Kristensen, J. A. Campos-Gonzalez-Angulo, M. G. Vakili, H. Huang, M. Bagherimehrab, C. Gorgulla, F. Wong, A. McCaskey, J.-S. Kim, T. Nguyen, P. Rao, and A. Aspuru-Guzik, “The generative quantum eigensolver (gqe) and its application for ground state search,” 2024.
- [6] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, “Quantum computing with Qiskit,” 2024.
- [7] S. Vishwakarma, F. Harkins, S. Golecha, V. S. Bajpe, N. Dupuis, L. Buratti, D. Kremer, I. Faro, R. Puri, and J. Cruz-Benito, “Qiskit humaneval: An evaluation benchmark for quantum code generative models,” 2024.

- [8] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-rank adaptation of large language models,” in *International Conference on Learning Representations*, 2022.
- [9] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum Algorithm for Linear Systems of Equations,” *Physical Review Letters*, vol. 103, p. 150502, Oct. 2009.
- [10] A. M. Childs, R. Kothari, and R. D. Somma, “Quantum algorithm for systems of linear equations with exponentially improved dependence on precision,” *SIAM Journal on Computing*, vol. 46, no. 6, pp. 1920–1950, 2017.
- [11] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, “Quantum Circuit Learning,” *Physical Review A*, vol. 98, p. 032309, Sept. 2018. arXiv:1803.00745 [quant-ph].
- [12] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, pp. 195–202, Sep 2017.
- [13] N. Guo, Z. Yu, M. Choi, A. Agrawal, K. Nakaji, A. Aspuru-Guzik, and P. Rebentrost, “Quantum linear algebra is all you need for transformer architectures,” 2024.
- [14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature Communications*, vol. 5, p. 4213, Jul 2014.
- [15] A. Y. Kitaev, “Quantum measurements and the abelian stabilizer problem,” *Electron. Colloquium Comput. Complex.*, vol. TR96, 1995.
- [16] M. Plesch and Brukner, “Quantum-state preparation with universal gate decompositions,” *Physical Review A*, vol. 83, p. 032302, Mar. 2011. Publisher: American Physical Society.
- [17] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018.
- [18] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, “Heuristics for quantum compiling with a continuous gate set,” *ArXiv*, vol. abs/1912.02727, 2019.

- [19] X. Sun, G. Tian, S. Yang, P. Yuan, and S. Zhang, “Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 10, pp. 3301–3314, 2023.
- [20] G. Rosenthal and H. Yuen, “Interactive proofs for synthesizing quantum states and unitaries,” *CoRR*, vol. abs/2108.07192, 2021.
- [21] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, “A divide-and-conquer algorithm for quantum state preparation,” *Scientific Reports*, vol. 11, p. 6329, Mar. 2021. Publisher: Nature Publishing Group.
- [22] K. Nakaji, S. Uno, Y. Suzuki, R. Raymond, T. Onodera, T. Tanaka, H. Tezuka, N. Mitsuda, and N. Yamamoto, “Approximate amplitude encoding in shallow parameterized quantum circuits and its application to financial market indicators,” *Phys. Rev. Res.*, vol. 4, p. 023136, May 2022.
- [23] C. Zoufal, A. Lucchi, and S. Woerner, “Quantum generative adversarial networks for learning and loading random distributions,” *npj Quantum Information*, vol. 5, Nov. 2019.
- [24] A. A. Melnikov, A. A. Termanova, S. V. Dolgov, F. Neukart, and M. R. Perelshtein, “Quantum state preparation using tensor networks,” *Quantum Science and Technology*, vol. 8, p. 035027, June 2023. Publisher: IOP Publishing.
- [25] J. Zylberman and F. Debbasch, “Efficient quantum state preparation with Walsh series,” *Physical Review A*, vol. 109, p. 042401, Apr. 2024. Publisher: American Physical Society.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [27] M. Krenn, J. Landgraf, T. Foesel, and F. Marquardt, “Artificial intelligence and machine learning for quantum technologies,” *Phys. Rev. A*, vol. 107, p. 010101, Jan 2023.
- [28] M. Bukov, A. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, “Machine Learning Meets Quantum State Preparation. The Phase Diagram of Quantum Control,” May 2017.

- [29] T. Haug, W.-K. Mok, J.-B. You, W. Zhang, C. Eng Png, and L.-C. Kwek, “Classifying global state preparation via deep reinforcement learning,” *Machine Learning: Science and Technology*, vol. 2, p. 01LT02, Mar. 2021.
- [30] R.-H. He, H.-D. Liu, S.-B. Wang, J. Wu, S.-S. Nie, and Z.-M. Wang, “Universal quantum state preparation via revised greedy algorithm,” *Quantum Science and Technology*, vol. 6, p. 045021, Oct. 2021.
- [31] X.-M. Zhang, Z. Wei, R. Asad, X.-C. Yang, and X. Wang, “When does reinforcement learning stand out in quantum control? A comparative study on state preparation,” *npj Quantum Information*, vol. 5, pp. 1–7, Oct. 2019. Publisher: Nature Publishing Group.
- [32] J. Kiefer and J. Wolfowitz, “Stochastic estimation of the maximum of a regression function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
- [33] O. V. Morzhin and A. N. Pechen, “Krotov method for optimal control of closed quantum systems,” *Russian Mathematical Surveys*, vol. 74, p. 851, oct 2019.
- [34] L. Cincio, K. Rudinger, M. Sarovar, and P. J. Coles, “Machine Learning of Noise-Resilient Quantum Circuits,” *PRX Quantum*, vol. 2, p. 010324, Feb. 2021.
- [35] F. Furrutter, G. Muñoz-Gil, and H. J. Briegel, “Quantum circuit synthesis with diffusion models,” *Nature Machine Intelligence*, vol. 6, pp. 515–524, May 2024.
- [36] R. Porotti, A. Essig, B. Huard, and F. Marquardt, “Deep Reinforcement Learning for Quantum State Preparation with Weak Nonlinear Measurements,” *Quantum*, vol. 6, p. 747, June 2022. arXiv:2107.08816 [quant-ph].
- [37] Y. Shen, “Prepare Ansatz for VQE with Diffusion Model,” Oct. 2023. arXiv:2310.02511 [quant-ph].
- [38] L. Moro, M. G. A. Paris, M. Restelli, and E. Prati, “Quantum compiling by deep reinforcement learning,” *Communications Physics*, vol. 4, pp. 1–8, Aug. 2021. Publisher: Nature Publishing Group.
- [39] P. Selig, N. Murphy, D. Redmond, and S. Caton, “DeepQPrep: Neural Network Augmented Search for Quantum State Preparation,” *IEEE Access*, vol. 11, pp. 76388–76402, 2023. Conference Name: IEEE Access.
- [40] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

- [41] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” 2023.
- [42] N. Shazeer, “Glu variants improve transformer,” 2020.
- [43] J. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *ArXiv*, vol. abs/1607.06450, 2016.
- [44] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.
- [45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.
- [46] P. J. Liu*, M. Saleh*, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, “Generating wikipedia by summarizing long sequences,” in *International Conference on Learning Representations*, 2018.
- [47] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, “Peft: State-of-the-art parameter-efficient fine-tuning methods.” <https://github.com/huggingface/peft>, 2022.
- [48] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. J. Lowe, “Training language models to follow instructions with human feedback,” *ArXiv*, vol. abs/2203.02155, 2022.
- [49] N. Houlsby, A. Giurghi, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, “Parameter-efficient transfer learning for NLP,” in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799, PMLR, 09–15 Jun 2019.
- [50] X. L. Li and P. Liang, “Prefix-tuning: Optimizing continuous prompts for generation,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural*

- Language Processing (Volume 1: Long Papers)* (C. Zong, F. Xia, W. Li, and R. Navigli, eds.), (Online), pp. 4582–4597, Association for Computational Linguistics, Aug. 2021.
- [51] J. Ji, T. Qiu, B. Chen, B. Zhang, H. Lou, K. Wang, Y. Duan, Z. He, J. Zhou, Z. Zhang, F. Zeng, K. Y. Ng, J. Dai, X. Pan, A. O’Gara, Y. Lei, H. Xu, B. Tse, J. Fu, S. M. McAleer, Y. Yang, Y. Wang, S.-C. Zhu, Y. Guo, and W. Gao, “Ai alignment: A comprehensive survey,” *ArXiv*, vol. abs/2310.19852, 2023.
 - [52] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, “Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms,” *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.
 - [53] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006.
 - [54] K. Życzkowski and H.-J. Sommers, “Average fidelity between random quantum states,” *Phys. Rev. A*, vol. 71, p. 032313, Mar 2005.
 - [55] N. Dziri, X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, S. Welleck, P. West, C. Bhagavatula, R. L. Bras, J. D. Hwang, S. Sanyal, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi, “Faith and fate: Limits of transformers on compositionality,” in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
 - [56] A. Madaan, S. Zhou, U. Alon, Y. Yang, and G. Neubig, “Language models of code are few-shot commonsense learners,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), (Abu Dhabi, United Arab Emirates), pp. 1384–1403, Association for Computational Linguistics, Dec. 2022.
 - [57] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, “Measuring massive multitask language understanding,” in *International Conference on Learning Representations*, 2021.
 - [58] AI@Meta, “Llama 3 model card,” 2024.
 - [59] I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in *International Conference on Learning Representations*, 2017.
 - [60] R. Jozsa, “Fidelity for mixed quantum states,” *Journal of Modern Optics*, vol. 41, no. 12, pp. 2315–2323, 1994.