

Learning a quantum computer’s capability using convolutional neural networks

Daniel Hothem,¹ Kevin Young,¹ Tommie Catanach,² and Timothy Proctor¹

¹Quantum Performance Laboratory, Sandia National Laboratories, Livermore, CA 94550, USA

²Sandia National Laboratories, Livermore, CA 94550, USA

(Dated: April 24, 2023)

The computational power of contemporary quantum processors is limited by hardware errors that cause computations to fail. In principle, each quantum processor’s computational capabilities can be described with a *capability function* that quantifies how well a processor can run each possible quantum circuit (i.e., program), as a map from circuits to the processor’s success rates on those circuits. However, capability functions are typically unknown and challenging to model, as the particular errors afflicting a specific quantum processor are *a priori* unknown and difficult to completely characterize. In this work, we investigate using artificial neural networks to learn an approximation to a processor’s capability function. We explore how to define the capability function, and we explain how data for training neural networks can be efficiently obtained for a capability function defined using process fidelity. We then investigate using convolutional neural networks to model a quantum computer’s capability. Using simulations, we show that convolutional neural networks can accurately model a processor’s capability when that processor experiences gate-dependent, time-dependent, and context-dependent stochastic errors. We then discuss some challenges to creating useful neural network capability models for experimental processors, such as generalizing beyond training distributions and modelling the effects of coherent errors. Lastly, we apply our neural networks to model the capabilities of cloud-access quantum computing systems, obtaining moderate prediction accuracy (average absolute error around 2-5%).

I. INTRODUCTION

The information processing capabilities of modern quantum computers are limited by a wide variety of hardware and implementation errors [1–17]. These *a priori* unknown errors accumulate during a quantum computation, leading to results that are corrupted in ways that are difficult to predict in advance [3]. A quantum processor’s ability to implement a particular quantum algorithm depends on its ability to execute that algorithm’s quantum circuits with low error. So, a quantum processor’s computational power can be captured by a *capability function* that quantifies its execution error on quantum circuits.

A capability function is a map s from quantum circuits c to a measure of how successfully the processor can execute each circuit. Possible measures of “success” include the total variation distance (TVD) between c ’s ideal and actual output probability distributions, the trace distance between the ideal and actual quantum states output by c , or the process fidelity between the ideal and actual quantum processes implemented by c . For any circuit c and any reasonable definition of $s(c)$, $s(c)$ can be estimated simply by repeatedly running the circuit c —or a family of closely related circuits [7]—on the processor and then calculating the relevant performance metric from the data. However, in general, $s(c)$ cannot be estimated efficiently for an arbitrary circuit c , and it is not practical to test a processor’s performance on every possible circuit of interest. Accurate surrogate models for a processor’s capability function, i.e., a classical approximation of $s(c)$, would therefore aid the understanding of a processor’s capabilities.

Simple heuristics are often used to model capability functions, but they are typically inaccurate. A commonly-used heuristic approximation to a circuit’s process fidelity, $s_F(c)$, consists of modelling each of a processor’s native gates by a fidelity, and then estimating $s_F(c)$ as the product of the fidelities of the gates in c . This heuristic is accurate for proces-

sors experiencing completely unstructured errors—i.e., uniform depolarization—but it is often extremely inaccurate in practice due to the prevalence of structured errors [3]. For example, contemporary processors experience crosstalk errors [3–5, 11, 15, 16], which cause a native gate’s error process to depend on what other gates are applied in parallel with it. They also experience coherent errors [15, 18], which can add or cancel within a circuit. The complexity of the errors experienced by contemporary quantum computers necessitates more complex models for $s(c)$.

An alternative approach to modelling a processor’s capability is to do so indirectly by using a parameterized error model [7, 8]. A parameterized model for a device’s errors is chosen (e.g., process matrices within unknown entries [8]), and best-fit values for the model’s unknown parameters are estimated from data, using techniques such as gate set tomography (GST) [7], randomized benchmarking (RB) [2, 4–6], or Pauli noise estimation [16, 17]. The estimated model can then be used to simulate each circuit of interest c , to compute the model’s prediction for $s(c)$. Parameterized error models are a principled approach to predicting $s(c)$, e.g., they are often amenable to rigorous statistical methods and physical interpretation [15]. But they have some important limitations, including the ubiquitous problem of inaccurate predictions caused by error models that cannot describe all of the errors a processor experiences [19]. For example, conventional parameterized error models are based on process matrices, which means that they cannot model non-Markovian errors, such as $1/f$ noise [14] and drift [12, 13, 20]. Improvements to parameterized error models may enable accurate modelling of $s(c)$, but today there are no error models that have consistently enabled accurate predictions of $s(c)$.

In this paper, we investigate using classical artificial neural networks to model a quantum computer’s capability function (see schematic in Fig. 1). Neural networks are general-purpose function approximators [21] that have shown promise

for many tasks in quantum physics and computing [22, 23]—including calibration [24–27], circuit compilation [28], tomography [29–31], and solving many-body physics problems [32]. Neural network models for $s(c)$ are intriguing because they need not be constrained by the assumptions of a particular parameterized error model. Instead, neural networks can use a many-parameter ansatz for $s(c)$ that is highly expressive, potentially enabling them to be trained to model the effect of poorly understood or unexpected error modes. Furthermore, neural network models for capabilities have the potential to be scalable: a trained neural network model for $s(c)$ can be quickly queried even in the many-qubit setting. This is because neural network models for $s(c)$ need not rely on explicit classical simulations of quantum circuits, which contrasts with directly computing $s(c)$ from most parameterized models (e.g., process matrices).

There are many neural network architectures that could be applied to the problem of capability modelling, and in this work we use convolutional neural networks (CNNs) [33–35]. CNNs extract predictive features from their input using learned convolutional filters, and they have proven useful for image classification [33] and natural language processing [34]. CNNs are a promising architecture for capability modelling because convolutional filters can pick out particular patterns of gates in a circuit (encoded as an image, or tensor), and particular but *a priori* unknown patterns of gates can be correlated with $s(c)$ [4, 11]. Furthermore, it is possible to construct CNNs whose complexity—i.e., the number of parameters that must be learned—increases only slowly (or even not at all) with the number of qubits (n) in a processor, meaning that scalable capability modelling with CNNs is feasible.

The contributions and structure of this paper are as follows. In Section II we introduce the problem of learning an approximation to a capability function $s(c)$. In Section III we introduce the neural network architecture (i.e., CNNs) and the data encoding that we use in this work. Our representation of the quantum circuits includes a limited form of error sensitivity information that is designed to aid our CNNs in the task of accurately modelling $s(c)$ in the presence of Pauli stochastic errors (our approach is a simple kind of physics-informed machine learning [36]). In Sections IV and V we show that CNNs can be trained to accurately model $s(c)$ in the presence of both Markovian and non-Markovian Pauli stochastic errors, including in the many-qubit setting ($n = 49$). In Section VI we highlight some important challenges to creating useful and reliable models for $s(c)$ using neural networks. We demonstrate the difficulties presented by coherent errors, the challenge of generalizing beyond training distributions, and the challenge of extending beyond the limited error sensitivity information included within our data encoding. In Section VII we demonstrate the application of $s(c)$ learning using CNNs to data from cloud-access quantum computers, obtaining models with moderate prediction accuracy. Finally, we conclude in Section VIII.

This paper is the culmination of work previously presented (but unpublished) [37]. It is also not the only paper to propose modelling $s(c)$ using neural networks. Independent works by Wang *et al.* [38], Vadali *et al.* [39] and Amer *et al.* [40] have

investigated using neural networks for a variety of prediction problems closely related to modelling $s(c)$. Core differences are the capability metrics, network architectures, and encoding schemes used (see Appendix A for further details).

II. LEARNING A QUANTUM COMPUTER’S CAPABILITY

In this section we introduce the central problem considered in this work: modelling capability functions. The purposes of this section are to (1) introduce the general capability learning problem, and (2) specify the particular form of this problem that we address herein. A capability function s for a quantum processor maps a circuit c to how successfully that circuit is executed on that processor, $s(c)$. Therefore, defining $s(c)$ necessitates specifying: (i) what we mean by “quantum circuit,” i.e., the domain of s (see Section II A); and (ii) what we mean by “success” (see Section II B–II C). There are many well-motivated capability functions $s(c)$, each corresponding to a different way to quantify the difference between the ideal and actual implementation of c , and so we identify a particular choice—the process fidelity—that is particularly promising in this context (see Section II D). As we explain, process fidelity is both a useful metric for a processor’s performance on a circuit c and it is possible to efficiently gather training data, i.e., it is possible to efficiently estimate the process fidelity for any circuit c . We then introduce the problem that we address for the remainder of this paper: predicting the success probabilities of definite outcome circuits (see Section II E). We explain why predicting success probabilities is closely related to the problem of predicting process fidelities, and why we choose to consider the problem of predicting success probabilities.

A. Quantum circuits

Here we define what we mean by a quantum circuit, which enables defining capability functions $s(c)$. Quantum circuits are typically defined as a sequence of layers of quantum logic gates. In this work, a w -qubit logic layer (l) is an instruction to apply physical operations that ideally implements a particular unitary evolution $U(l) \in \text{SU}(2^w)$ on w qubits. This definition excludes logic operations that are intended to be non-unitary, such as mid-circuit measurements. A quantum-input quantum-output (QIQO) w -qubit circuit (c) over a w -qubit logic layer set $\mathbb{L}_w = \{l\}$ is a sequence of $d \geq 0$ layers

$$c = l_d l_{d-1} \cdots l_2 l_1, \quad (1)$$

where each $l_i \in \mathbb{L}_w$ [4]. The circuit c is an instruction to apply its constituent logic layers, l_1, l_2, \dots , in sequence, implementing the unitary evolution

$$U(c) = U(l_d) \cdots U(l_2)U(l_1). \quad (2)$$

Below, it will be convenient to use the superoperator representation of this unitary [$\mathcal{U}(c)$] given by:

$$\mathcal{U}(c)[\rho] = U(c)\rho U^\dagger(c), \quad (3)$$

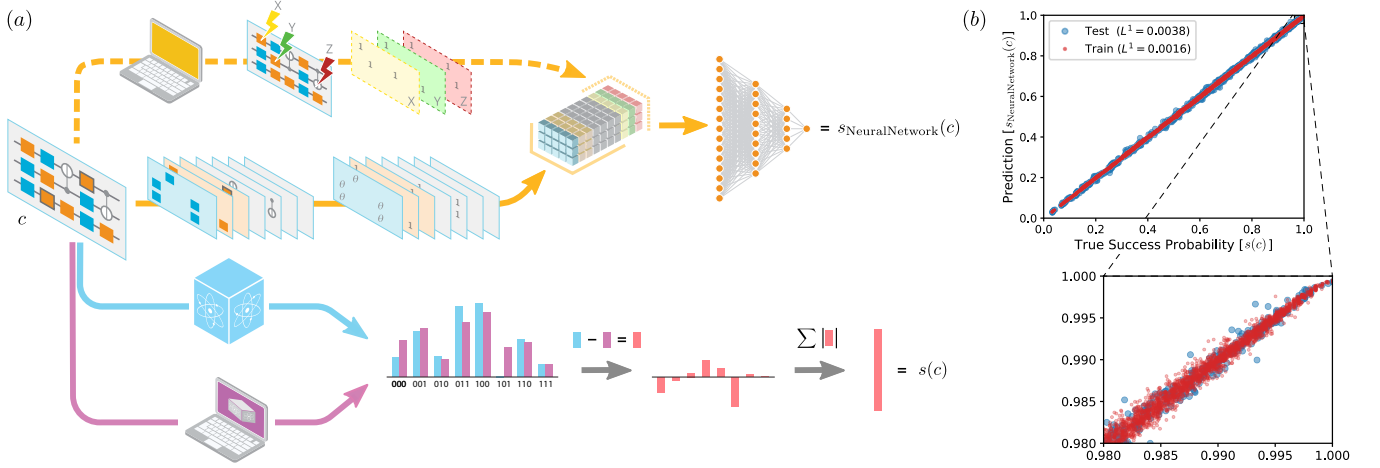


FIG. 1. **Modelling a quantum computer’s capability using neural networks.** (a) Many quantum circuits (c) are intended to sample from some distribution (purple histogram), which, in principle, can be calculated by simulating c on a classical computer (purple arrow). However, when c is run on a real quantum processor (blue arrow), hardware errors mean that we sample from a different distribution (blue histogram). The difference between the ideal and actual distributions (red histogram) encodes how well the processor ran c , which can be summarized by, e.g., TVD (red bar). We can quantify how well a processor can run any circuit using a *capability function* (s) that maps any circuit c to how well the processor runs c [$s(c)$]. In this work, we aim to construct a model for $s(c)$ using classical artificial neural networks (orange arrows). We input circuits into convolutional neural networks (CNNs) by representing them as three-dimensional tensors that encode which gates are applied in each layer of a circuit (lower orange arrow) and some information about what kinds of errors a circuit is sensitive to (upper orange arrow). In this work, we also define capability functions for circuits that ideally create a quantum state (i.e., they can end with any measurement), or implement a unitary (i.e., they also have an unspecified input), which is not denoted here. (b) The predictions of a CNN trained on random circuits from a hypothetical 5-qubit processor subject to strongly biased local stochastic Pauli errors (see Section IV for details). This demonstrates that CNNs can accurately model $s(c)$ for a relatively simple but ubiquitous family of errors. To train these CNNs we must gather training data (predictions on training data are shown in red), i.e., a set of circuits each labelled with $s(c)$, and we explain how to do this efficiently in this paper.

and to denote the perfect action of any circuit c by $\gamma(c)$, i.e., for a QIQO circuit

$$\gamma(c) = \mathcal{U}(c) = \mathcal{U}(l_d) \cdots \mathcal{U}(l_2) \mathcal{U}(l_1). \quad (4)$$

A QIQO circuit can be embedded within other quantum circuits—i.e., any w -qubit quantum state can be input into the circuit, and any map can be applied to its output w -qubit quantum states. However we may intend to apply a circuit to a fixed input state, followed by a fixed basis measurement. This is important here, because a circuit’s intended use impacts the appropriate metric for quantifying how well a processor can run that circuit. A circuit’s intended use can be formalized by specifying the intended input space for a circuit as well as the intended set of maps on its outputs. QIQO circuits map quantum states to quantum states. We specify two other important choices for input/output spaces by defining standard-input quantum-output (SIQO) and standard-input classical-output (SICO) circuits [4]. A w -qubit SIQO circuit $c = l_d \cdots l_1 l_{\text{init}}$ is a QIQO circuit ($l_d \cdots l_1$) with the addition of an initial instruction (l_{init}) to initialize each of w qubits in the $|0\rangle$ state. Its perfect action [$\gamma(c)$] creates the pure w -qubit quantum state $|\psi(c)\rangle\langle\psi(c)|$ given by:

$$\gamma(c) = |\psi(c)\rangle\langle\psi(c)| = \mathcal{U}(l_d \cdots l_1)[|0\rangle\langle 0|^{\otimes w}]. \quad (5)$$

A w -qubit SICO circuit $c = l_{\text{readout}} l_d \cdots l_1 l_{\text{init}}$ is a SIQO circuit ($l_d \cdots l_1 l_{\text{init}}$) with the addition of a final instruction (l_{readout}) to

measure all of the qubits in the computational basis. Its perfect action is to draw a sample from a probability distribution $\gamma(c) = \mathbf{P}(c)$ over length- w bit strings x , whereby the probability to obtain the bit string x (denoted $\mathbf{P}(c)[x]$) is given by

$$\mathbf{P}(c)[x] = \text{Tr}\left(|x\rangle\langle x| \mathcal{U}(l_d \cdots l_1)[|0\rangle\langle 0|^{\otimes w}]\right). \quad (6)$$

The three kinds of circuits we consider are summarized in Table I. All three of these circuit families are part of a broader class of circuits, with mixed quantum-classical inputs and outputs (MIMO), which we do not consider further.

For the purposes of this paper, operations across multiple layers in a circuit must *not* be combined (compiled) together by implementing a physical operation that enacts their composite unitary. This is accomplished by adding in “barriers” between circuit layers. These barriers between circuit layers are often used in benchmarking and characterization methods [2–4, 7], and it simplifies our prediction task. This is because the inclusion of barriers within circuits removes the need to learn the behaviour of any classical algorithms used to compile together circuit layers [41], which can be arbitrarily complex.

B. Modelling an imperfect quantum circuit

The definition of capability functions (below) uses a mathematical model [$\tilde{\gamma}$] for a processor’s imperfect implementation

of a circuit, and we now introduce this model. Our mathematical model $\tilde{\gamma}$ does not rely on the most widely-used assumptions about a processor’s errors (e.g., Markovianity). We avoid encoding those assumptions into our definition of capability functions because methods for learning capabilities have the potential to be accurate even when those assumptions are violated. Our model $\tilde{\gamma}$ assumes that a processor’s imperfect implementation of a circuit c depends on c and possibly some auxiliary classical observable “context” variable[s] (e.g., time) from some state space \mathbb{A} . (No quantum degrees of freedom are allowed.) That is, we use a function $\tilde{\gamma}(c, a)$ to represent the imperfect implementation of $c \in \mathbb{C}$ with context $a \in \mathbb{A}$. Specifically (as summarized in Table I):

1. $\tilde{\gamma}(c, a)$ is an unknown probability distribution over w -bit strings $[\tilde{P}(c, a)]$ if c is a SICO circuit,
2. $\tilde{\gamma}(c, a)$ is an unknown w -qubit quantum state $[\rho(c, a)]$ if c is a SIQO circuit, and
3. $\tilde{\gamma}(c, a)$ is an unknown w -qubit completely positive and trace preserving (CPTP) map $[\Lambda(c, a)]$ if c is a QIQO circuit.

This framework encompasses all Markovian errors (as defined in Appendix B and Ref. [7]), as well as a wide variety of (but not all) non-Markovian errors. Because $\tilde{\gamma}$ is a general function from circuits to distributions, states, or CPTP maps, this framework can represent the effect of complex error processes within circuits, including: gate errors that increase over the course of a circuit (caused by, e.g., heating in ion-traps); gate error processes that depend on what gates were applied earlier in a circuit (known as serial context dependence); and general crosstalk errors [11]. Because $\tilde{\gamma}$ also depends on observable context variable[s] $a \in \mathbb{A}$, this framework can model the effects of many non-Markovian errors, such as time-varying error processes [12–14, 20] like slow drift (by letting a include wall-clock time, or the time since the last calibration), or the impact of measurable control or environmental parameters.

C. A quantum computer’s capability function

We now define capability functions, which we aim to learn approximations to. Capability functions are intended to quantify how close a processor’s implementation of each circuit c , within some circuit set \mathbb{C} , is to c ’s ideal action $[\gamma(c)]$. A capability function is defined using (1) a set \mathbb{C} of circuits, (2) our mathematical model $\tilde{\gamma}$ for a processor’s imperfect implementation of a circuit, and (3) a function ϵ that quantifies the difference between the perfect and imperfect implementations of a circuit. The capability function is a map from circuits (\mathbb{C}), and any observable context variables (\mathbb{A}) on which $\tilde{\gamma}$ depends, to $\epsilon[\gamma(c), \tilde{\gamma}(c, a)]$. Specifically, the capability function for metric ϵ is

$$s_\epsilon(c, a) = \epsilon[\gamma(c), \tilde{\gamma}(c, a)]. \quad (7)$$

There are many well-motivated choices for ϵ , including: the TVD, cross-entropy, or Hellinger (classical) fidelity [for SICO

Circuit Type	Action	$\gamma(c)$	$\tilde{\gamma}(c, a)$
QIQO	Applies a quantum process	$\mathcal{U}(c)$	$\Lambda(c, a)$
SIQO	Creates a quantum state	$ \psi(c)\rangle\langle\psi(c) $	$\rho(c, a)$
SICO	Samples from a distribution	$P(c)$	$\tilde{P}(c, a)$

TABLE I. A summary of the three types of quantum circuit considered herein, their action (the kind of process they produce), and the mathematical objects and the corresponding notation we use to represent their perfect $[\gamma(c)]$ and imperfect $[\tilde{\gamma}(c, a)]$ implementations. See Section II for the definition of each element in this table.

circuits, where $\gamma(c)$ and $\tilde{\gamma}(c, a)$ are probability distributions]; the trace distance or quantum state fidelity [for SIQO circuits, where $\gamma(c)$ and $\tilde{\gamma}(c, a)$ are quantum states]; or the diamond distance [42], total error [15], or process fidelity [43] [for QIQO circuits, where $\gamma(c)$ and $\tilde{\gamma}(c, a)$ are CPTP maps]. Each metric has a different interpretation—e.g., diamond distance is a form of worst-case error—and the ideal metric with which to define s_ϵ will depend on the intended uses for a model for s_ϵ .

D. Evaluating a capability function

To directly learn an approximation to s_ϵ we require labelled training data, i.e., a dataset consisting of a set of circuits $\{c\}$ and (when relevant) contexts $\{a\}$ with each (c, a) paired with an estimate of $s_\epsilon(c, a)$. Creating such a dataset requires a method for estimating $s_\epsilon(c, a)$. We now discuss whether and how this estimation can be done efficiently. Note that evaluating $s_\epsilon(c, a)$ is closely related to the well-known problem of verifying the correctness of the results of a quantum computation.

In principle, a capability function $s_\epsilon(c, a)$ can be estimated to any desired precision for any circuit $c \in \mathbb{C}$ and any controllable context $a \in \mathbb{A}$ using a tomographic method. This method consists of (1) running experiments that enable the estimation of $\tilde{\gamma}(c, a)$, and then estimating $\tilde{\gamma}(c, a)$ from the data, (2) computing $\gamma(c)$ by simulating c on a classical computer, and then (3) computing $\epsilon[\gamma(c), \tilde{\gamma}(c, a)]$. For example, if c is a SICO circuit then $\tilde{\gamma}(c, a)$ is the probability distribution that a sample is drawn from in each execution of c in context a . In principle, this can be estimated simply by running c many times, in context a . Similarly, if c is a QIQO or SIQO circuit, then $\tilde{\gamma}(c)$ is a quantum process or quantum state, respectively, which can be estimated to any desired precision using process or state tomography [7], respectively (or, to avoid known inconsistencies in state and process tomography, using GST [7]). This procedures consist of embedding c within a variety of circuits and then inferring $\tilde{\gamma}(c, a)$ from the data. However, for a general circuit c , the tomographic method for evaluating $s_\epsilon(c, a)$ is well-known to be inefficient. In general, this method for evaluating $s_\epsilon(c, a)$ requires classical computations (to simulate c) that are exponentially expensive in the number of qubits (n) on which c acts, and a number of circuit executions that is also exponentially large in n . So capability learning based on a tomographic approach to estimating $s_\epsilon(c, a)$ is inefficient, in

general.

Direct and efficient methods for estimating the value of a capability function $s_\epsilon(c, a)$ are critical for direct learning of capability functions. The circumstances under which $s_\epsilon(c, a)$ can be efficiently estimated is an interesting open question, i.e., for which ϵ and under what assumptions about a processor’s errors [which includes assumptions about $\tilde{\gamma}(c, a)$] is there an efficient method for estimating $s_\epsilon(c, a)$? However, there is a well-motivated choice for ϵ for which an efficient method for estimating $s_\epsilon(c, a)$ is known: process fidelity. Process fidelity (F) is defined as [44]

$$F[\gamma(c), \tilde{\gamma}(c, a)] = \langle \varphi | \gamma^\dagger(c) \tilde{\gamma}(c, a) | \varphi \rangle \langle \varphi | \varphi \rangle, \quad (8)$$

where φ is any maximally entangled state in a doubled Hilbert space [43]. Almost any circuit’s process fidelity can be efficiently estimated using mirror circuit fidelity estimation (MCFE) [45] (under certain assumptions about the underlying error processes, detailed in Ref. [45]). Because a capability function defined by process fidelity can be evaluated using a method that is efficient in the number of qubits n , it is feasible that an approximation to s_F can be learned even in the many-qubit setting, where approximate capability function models will be most useful.

E. The capability function for definite outcome circuits

The capability function defined by process fidelity (s_F) is well-motivated, and learning an approximation to s_F is feasible, in the sense that training data can be efficiently obtained. However, in this work we apply neural networks to a different but related problem. Instead we consider the problem of learning a capability function for *definite outcome circuits* (defined below), which are a subclass of SICO circuits—that is, we consider s defined over a circuit set \mathbb{C} containing only definite outcome circuits. In Appendix C we explain why we choose to address this problem, rather than process fidelity learning, and why we conjecture that a neural network method that can accurately model s for definite outcome circuits will, with minor adaptations, be able to accurately model s_F when trained on circuit process fidelities.

A SICO circuit c is a definite outcome circuit if and only if its error-free output distribution $\gamma(c)$ has support only on a single “success” bit string $x_s(c)$, i.e.,

$$\gamma(c)[x_s(c)] = 1. \quad (9)$$

For definite outcome circuits, the probability that a circuit c outputs its success bit string $x_s(c)$ is the single natural choice for ϵ [46]. Therefore, the unique well-motivated definition for the capability function of definite outcome circuits is simply

$$s(c, a) = \tilde{\gamma}(c, a)[x_s(c)], \quad (10)$$

which is the circuit’s “success probability”. The success probability of a definite outcome circuit c can be efficiently estimated whenever the success bit string $x_s(c)$ can be efficiently

computed on a classical computer. In particular we can estimate $s(c, a)$ [denoted $\hat{s}(c, a)$] from N_{shots} executions of c in context a as

$$\hat{s}(c, a) = \frac{N_{x_s(c)}}{N_{\text{shots}}} \quad (11)$$

where $N_{x_s(c)}$ is the number of times the bit string $x_s(c)$ is output from the circuit.

Almost any circuit c can be turned into a definite outcome “mirror circuit” $m(c)$ [3–6] for which $x_s[m(c)]$ can be efficiently computed, and all the circuit sets used in our numerical experiments—i.e., the applications of our CNNs to simulated or experimental data—contain only mirror circuits $m(c)$. Circuit mirroring is a motion-reversal circuit or Loschmidt echo (i.e., following c by its inverse) that is modified to prevent the systematic cancellation of errors that can occur within standard motion-reversal circuits.

III. PREDICTING CAPABILITIES USING CONVOLUTIONAL NEURAL NETWORKS

In this section we introduce our method for predicting circuit success probabilities using convolutional neural networks (CNNs). In this work we aim to predict the success probabilities $[s(c)]$ of definite outcome circuits, run on a specific quantum processor, and we consider no context information (i.e., \mathbb{A} from Section II is trivial). So, the input of our neural networks is a definite outcome circuit c , and the output is a prediction for the success probability $s(c)$ that would be observed if c were run on the processor that we are modelling. To address this prediction problem using neural networks we must choose: (1) the set of quantum circuits whose success probabilities we aim to predict (see Section III A); (2) a representation for the circuits that enables inputting them into a neural network (see Section III B); (3) a structure for the neural networks (see Section III C); (4) methods for training the parameters [i.e., weights] of the neural network (see Section III D, and tuning its hyperparameters (see Section III E); and (5) methods for evaluating the final model’s performance (see Section III F).

A. Circuit selection

Training, hyperparameter tuning, and evaluation of our neural networks requires training, validation, and test datasets. These datasets each consist of circuits $\{c\}$ that are each labelled with an estimate $[\hat{s}(c)]$ of that circuit’s success probability $[s(c)]$. We denote these datasets by D_{train} , D_{validate} , and D_{test} herein. Here we explain how we select the training, validation, and test circuit sets (in Sections II E–II D we explained how we estimate $s(c)$ for any circuit c). Each of these circuit sets is sampled from a set \mathbb{C} defining the set of all possible circuits (the set \mathbb{C} used in our examples is introduced below).

We construct training, validation, and (in-distribution) test circuit sets using a parameterized distribution $p(\alpha) : \mathbb{C} \rightarrow \mathbb{R}$.

First a circuit set D_{combined} is constructed by (1) either systematically varying the distribution’s parameters α (corresponding, e.g., to the circuits’ depths) or randomly sampling the distributions parameters, and (2) drawing independent samples from $p(\alpha)$ for each selected parameter value α . The sampled circuit set D_{combined} is then randomly partitioned into training, validation, and (in-distribution) test circuit sets. This is consistent with standard practices in machine learning.

Evaluating a model’s prediction accuracy on test circuits drawn from the same distribution as the training circuits corresponds to standard practice, and neural network models often do not generalize well to data drawn from a different distribution. However, the utility of a neural network model for $s(c)$ will depend on its ability to accurately predict the performance of those circuits that are of most interest to a user of this model for $s(c)$ (e.g., perhaps only circuits that implement a particular algorithm are of interest), and the relevant circuit set[s] might not be known at the time of model training (therefore preventing the relevant circuit set[s] from being used to define p). In Sections VI A we explore whether CNN models for $s(c)$ generalize to additional test data (D_{test}), containing circuits drawn from a distribution p' that differs from p . This is an example of what is known as *out-of-distribution* prediction or generalization.

Specific circuit sets are sampled from the set of all possible circuits (\mathbb{C}) and we now specify how \mathbb{C} is defined in this work. We consider circuit sets for an n -qubit processor (we label the qubits by $\mathbb{Q} = \{1, \dots, n\}$) with a set of logic layers that is specified by a directed connectivity graph (G) over the qubits \mathbb{Q} , a two-qubit gate set \mathbb{G}_2 , and a one-qubit gate set \mathbb{G}_1 . We consider all $w \leq n$ qubit (SICO) circuits, over a w -qubit subset of \mathbb{Q} (\mathbb{Q}_w), consisting of layers that contain parallel applications of gates from \mathbb{G}_1 and two-qubit gates from \mathbb{G}_2 that respect the processor’s connectivity graph. Our circuit encoding (see Section III B) assumes a connectivity graph that can be embedded in a square grid (however, extensions to other connectivity graphs are simple). Our circuit encoding assumes a single two-qubit gate, and in all our numerical examples

$$\mathbb{G}_2 = \{\text{CNOT}\}. \quad (12)$$

The circuit encoding assumes a single-qubit gate set in which each gate can (but need not be) parameterized by a continuous variable. For all our simulated datasets

$$\mathbb{G}_1 = \{Z(\theta), X_{\pi/2}, X_{-\pi/2}\}, \quad (13)$$

where $Z(\theta)$ is a Z rotation by θ , i.e., $U[Z(\theta)] = \exp(-i\theta Z/2)$, and $X_{\pi/2}$ and $X_{-\pi/2}$ are X rotations by $\pi/2$ and $-\pi/2$, respectively. For all datasets from cloud-access quantum computers

$$\mathbb{G}_1 = \mathbb{C}_1, \quad (14)$$

where \mathbb{C}_1 is the set of 24 single-qubit Clifford gates. Finally, our current circuit encoding requires that every gate in every circuit is a Clifford gate [so, for the gate set of Eq. (13), this means $\theta \in \{-\pi/2, 0, \pi/2, \pi\}$]. The restriction to Clifford circuits is necessary for an *optional* part of our circuit encoding that we conjecture can be adapted to general circuits (see the discussion in Section III B and Section VI C).

Throughout this paper our training, validation and test circuit sets are sampled from two families of parameterized distributions over circuits: randomized mirror circuit [3–6] and periodic mirror circuits [3] (see Fig. 1 of Ref. [3] for a diagrammatic representation of these circuits, and the supplemental material therein for comprehensive definitions). Depth d randomized mirror circuits consist of $d/2$ independently sampled layers of gates, followed by $d/2$ layers consisting of the inverse of that circuit (with some added randomization to prevent systematic error cancellation). In contrast, periodic mirror circuits consist of repeating the same short (randomly sampled) sequence of gates many times, followed by the inverse circuit (again with some added randomization to prevent systematic error cancellation). Randomized mirror circuits are highly disordered—and they are similar in nature to the random circuits used in many benchmarking methods (e.g., [1, 2, 10])—whereas periodic mirror circuits can amplify coherent errors. Each distribution is parameterized by: (1) circuit width $[w]$; (2) circuit depth $[d]$; (3) the subset $\mathbb{Q}_w \subset \{1, \dots, n\}$ of w connected qubits that the circuit runs on; and (4) expected two-qubit gate density $[\xi]$. We aim to model $s(c)$ for variable w , d and \mathbb{Q}_w , so we either systematically vary these three parameters or sample them randomly.

B. Circuit encoding

To learn an approximation to a capability function $s(c)$ we must choose a mathematical representation $I(c)$ for the circuits that can be input into the chosen neural network architecture. The neural network is tasked with approximating $s(c)$ given $I(c)$, so the complexity of its learning task depends on the choice for this representation. The learning problem is easier if we use a representation of the circuits that makes it easy for the chosen neural network architecture to extract features of circuits that are highly predictive of $s(c)$ [47]. We represent a $w \times d$ circuit for an n -qubit processor (where $w \leq n$) using a $n \times d$ image with multiple “color channels”, as illustrated in Fig. 1. That is, we represent the circuit c by an $n \times d \times h$ tensor $I(c)$ where h is the number of channels. The (i, j) “pixel” of the image $[I_{ij}(c)]$, meaning the vector

$$I_{ij}(c) = \left(I_{ij1}[c], I_{ij2}[c], \dots, I_{ijh}[c] \right)^T, \quad (15)$$

stores information about what happens to qubit i in layer j of the circuit. So this encoding preserves the locality of consecutive layers in the circuit. However, it does *not* encode information about the spatial arrangement of the physical qubits (our labelling of the qubits is arbitrary). The channels are split into two kinds of channel: gate channels and error sensitivity channels, introduced in turn below.

The gate channels are used to encode which gate is being applied to each qubit in each layer, using a modified one-hot encoding. Our encoding uses $h_{\text{gate}} = |\mathbb{G}_1| + 4$ gate channels: one for each single-qubit gate in \mathbb{G}_1 and four channels to encode CNOT gates [48]. We use four channels for the CNOT gates as each qubit has at most four neighbours in the connectivity graph, so four channels is sufficient for a lossless encoding of the CNOT gates. The channels correspond to a CNOT

gate with a neighbour that is to the left, to the right, above, or below the qubit in question [which is qubit i for pixel $I_{ij}(c)$]. If in layer j of circuit c the gate G is applied to qubit i and G is encoded in channel k then $I_{ijk}(c) = v(G)$, with the value in all other channels set to zero (as in one-hot encoding). If G is a single-qubit parameterized gate then $v(G) = \theta$ where θ is the gate’s parameter value [so, e.g., $v(G) = \theta$ for a $Z(\theta)$ gate], and if G is a single-qubit gate with no parameters then $v(G) = 1$. If G is a CNOT gate then $v(G) = 1$ ($v(G) = -1$) if qubit i is the control (target) qubit. Therefore, the value $v(G)$ stored in a channel includes any information about the identity of the gate that is being applied that is not encoded into the channel index.

Machine learning techniques applied to physics problem are often more accurate if known physics is encoded into the methods [36] (known as physics-informed machine learning). We implement a simple form of physics-informed machine learning by encoding into $I(c)$ some information about what errors the circuit c is sensitive to. This is the role of our *error sensitivity channels*. The error sensitivity channels are used to encode, into pixel $I_{ij}(c)$, information about which kinds of errors qubit i is sensitive to when layer j of c is applied. In our encoding, we utilize three error sensitivity channels h_X , h_Y and h_Z . For $P \in \{X, Y, Z\}$, h_P encodes whether the single-qubit Pauli error P on qubit i at layer j would transform the qubits into an orthogonal state when applied to the ideal state of the system after layer j . Specifically, letting

$$|\psi_j\rangle = U(l_j) \cdots U(l_1)|0\rangle^{\otimes n}, \quad (16)$$

for the circuit $c = l_d \dots l_1$, then

$$I(c)_{ijh_P} = 1 - |\langle \psi_j | P_i | \psi_j \rangle|. \quad (17)$$

Thus, $I(c)_{ijh_P} = 0$ if $|\psi_j\rangle$ is an eigenstate of P_i and otherwise $I(c)_{ijh_P} = 1$. Here P_i denotes Pauli P operator on qubit i (tensored with an identity on all other qubits).

A complete description of ψ_j could be encoded into $I(c)$ using $O(n)$ bits (at each pixel) that specify a set of generators for ψ_j ’s stabilizer group. This is because ψ_j is a stabilizer state, as we consider only Clifford circuits herein. We do not do so for two reasons. First, our encoding provides a CNN with easy access to information about the impact of a particular important kind of errors—local stochastic Pauli errors—which is not easily extracted from an arbitrarily chosen set of generators for ψ_j ’s stabilizer group. Second, we conjecture that limited error sensitivity information similar to that encoded here can be obtained even for general, non-Clifford circuits, whereas encoding a complete but efficient description of the state ψ_j does not generalize to arbitrary non-Clifford circuits.

C. Convolutional neural networks

In this work we use CNNs, which accomplish a regression or classification task by learning convolutional filters that extract predictive features out of a dataset [33–35], to model $s(c)$. Here, we introduce the particular CNNs that we used in this work, we explain why CNNs are a promising approach

to modelling $s(c)$, and we highlight some limitations of this approach to modelling $s(c)$. See Goodfellow *et al.* [35] for a detailed introduction to CNNs.

Our aim is to learn an approximation to the mapping from circuits c , encoded into $n \times d \times h$ images $I(c)$, to $s(c) \in [0, 1]$. Therefore, our CNNs are functions

$$\text{CNN} : \mathbb{R}^{n \times d \times h} \rightarrow [0, 1]. \quad (18)$$

Our CNNs are built out of three kinds of layers: convolutional layers (**conv**), pooling layers (**pool**), and dense layers (**dense**). These CNNs consist of interleaved convolutional and pooling layers, followed by a sequence of dense layers, as illustrated in Fig. 2.

Convolutional layers create *feature maps* by convolving an input image with learnable kernels. A convolutional layer for an input image $I^{(\text{in})} \in \mathbb{R}^{n_{\text{in}} \times d_{\text{in}} \times h_{\text{in}}}$ is specified by an activation function f , a kernel shape (k_w, k_d) , and h_{out} convolutional filters $\mathbb{R}^{n_{\text{in}} \times d_{\text{in}} \times h_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{in}} \times d_{\text{in}}}$ containing learnable parameters. The convolutional layer is a map

$$\text{conv} : \mathbb{R}^{n_{\text{in}} \times d_{\text{in}} \times h_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{in}} \times d_{\text{in}} \times h_{\text{out}}} \quad (19)$$

constructed by “stacking” the output of the h_{out} filters. Each convolutional filter is defined by a learnable kernel $K^{(h)} \in \mathbb{R}^{k_w \times k_d \times h_{\text{in}}}$ and a learnable bias $b_h \in \mathbb{R}$. The filter turns the three-dimensional input image $I^{(\text{in})}$ into a two-dimensional feature map, by convolving the kernel with $I^{(\text{in})}$ —i.e., by sliding the kernel across the image and, at each location, taking the inner product of the kernel with the local image patch—then adding the bias (b_h) to each pixel in the resultant two-dimensional image, and finally applying the activation function (f). So the three-dimensional image output by the convolutional layer [$I^{(\text{out})}$] is given by

$$I_{ijh}^{(\text{out})} = f \left(\sum_{i'=-\lfloor k_w/2 \rfloor}^{\lfloor k_w/2 \rfloor} \sum_{j'=-\lfloor k_d/2 \rfloor}^{\lfloor k_d/2 \rfloor} \sum_{h'=1}^{h_{\text{in}}} K_{i'j'h'}^{(h)} I_{(i+i'),(j+j'),h'}^{(\text{in})} + b_h \right), \quad (20)$$

for $i = 1, \dots, n_{\text{in}}$, $j = 1, \dots, d_{\text{in}}$, and $h = 1, \dots, h_{\text{out}}$. Note the edges of the input image are padded with zeros, e.g., by definition $I_{-1jh}^{(\text{in})} = 0$. In our work, the activation function $f : \mathbb{R} \rightarrow \mathbb{R}$ is the rectified linear unit (ReLU):

$$\text{ReLU}(x) = \max(0, x). \quad (21)$$

Convolutional layers are useful for extracting predictive features from quantum circuits $I(c)$ because they can create feature maps that identify the locations (and the number of instances) of specific patterns of gates in c . This is relevant to predicting $s(c)$ because particular patterns of gates can increase the failure rate of circuits [4, 11]. For example, error rates can increase when a particular gate is repeated multiple times in a row (known as serial context dependence) or when certain gates are applied in parallel (known as parallel context dependence or crosstalk) [11]. Convolutional filters exist that, e.g., find all instances of sequential applications of the same gate in $I(c)$ (see Section VD and Appendix F). Furthermore, in Appendix D we present convolutional filters that extract feature maps from our circuit encoding that are sufficient to

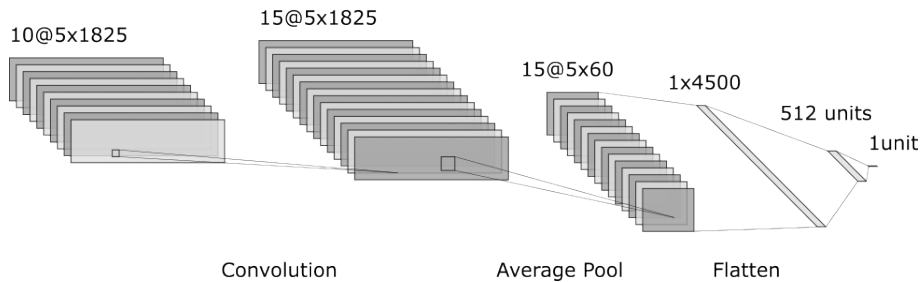


FIG. 2. **A convolutional neural network for predicting circuit success probabilities.** In this work we explore using CNNs to predict a circuit c 's success probability $s(c)$ when run on a particular quantum computer. Here we provide an example of a CNN architecture that we use for modelling $s(c)$. The input to our CNNs is an image representation $I(c)$ of a circuit c , where $I(c) \in \mathbb{R}^{n \times d \times h}$ and n is the number of qubits (c 's width), d is c 's depth, and h is the number of channels used in the encoding. In this example, $n = 5$, $d = 1825$ and $h = 10$. Our CNNs consist of one or more layers of convolutional filters (here there is one convolutional layer, with 15 kernels of shape $(1, 4)$), interspersed with dimension-reducing average pooling layers (here there is one pooling layer, which averages across the depth dimension), followed by a multi-layer perceptron consisting of dense layers of neurons (here there are two dense layers). The final dense layer contains a single neuron with a sigmoid activation function, so that the output—the model's prediction for $s(c)$ —is within $[0, 1]$. Convolutional and dense layers contain parameters (weights and biases) that are learned in the training process: the example shown here contains 2,305,640 parameters. The structure of the CNN is selected using hyperparameter tuning. The example shown here is the CNN used for the local Pauli stochastic error model trained on $N_{\text{circuits}} = 14940$ and $N_{\text{shots}} = \infty$ (see Fig. 3). The image was generated using [49].

approximate $s(c)$ under a local stochastic Pauli error model (this family of parameterized models is defined in Section V), and local stochastic Pauli errors constitute a significant proportion of the total error in many systems.

In our networks, each convolutional layer is followed by a pooling layer (see Fig. 2)

$$\text{pool} : \mathbb{R}^{n_{\text{in}} \times d_{\text{in}} \times h_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{out}} \times m_{\text{out}} \times h_{\text{out}}}, \quad (22)$$

where $n_{\text{out}} \leq n_{\text{in}}$ and $d_{\text{out}} \leq d_{\text{in}}$. Pooling layers reduce the size of an image, by partitioning the image into distinct (p_w, p_d) -shaped segments and, in each channel h , replacing each such segment with the maximum or average value within that segment. Pooling layers, which contain no learnable parameters, are included in the networks for dimensionality reduction [50].

The dense layers of a CNN (see Fig. 2) are used to process the final feature maps in order to make a prediction. Each dense layer is a map

$$\text{dense} : \mathbb{R}^{n_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{out}}} \quad (23)$$

that consists of n_{out} artificial neurons $\{u_i\}_{i=1}^{n_{\text{out}}}$ and a non-linear activation function $f : \mathbb{R} \rightarrow \mathbb{R}$ (we again used ReLU, except for the final layer). Each neuron $u : \mathbb{R}^{n_{\text{in}}} \rightarrow \mathbb{R}$ is defined by a learnable weight vector, $w \in \mathbb{R}^{n_{\text{in}}}$, and a learnable bias, b . The mapping defined by a neuron is $u(v) = f(w^T v + b)$, and the output of the layer is $v' = (u_1(v), \dots, u_{n_{\text{out}}}(v))^T$. Our networks terminate with a dense layer containing a single neuron with a sigmoid activation function. This guarantees that our network's output is in $[0, 1]$, and thus represents a probability.

D. Network training

CNNs are trained to approximate a function by iteratively modifying all of their learnable parameters, to improve the

model's predictions on training data. We optimize a network's weights using the Adam optimization algorithm, a gradient-based optimization method [51]. A round of training consists of: (1) evaluating the network's predictions on D_{train} using a loss function and (2) updating the network's weights to minimize the training loss. This process is repeated for some number of *epochs* (the number of training epochs is a hyperparameter, as discussed below).

The loss function that we use is the average binary cross-entropy (BCE). The average BCE of a model \mathcal{M} 's predictions $s_{\mathcal{M}} = \{s_{\mathcal{M}}(c)\}$ to a set of observations $\hat{s} = \{\hat{s}(c)\}$ is

$$H(\hat{s}, s_{\mathcal{M}}) = -\frac{1}{N_{\text{circuits}}} \sum_c s_{\mathcal{M}}(c) \log[\hat{s}(c)],$$

where N_{circuits} is the number of circuits in the dataset. A circuit's success probability $s(c)$ is estimated using $\hat{s}(c) = N_{s(c)} / N_{\text{shots}}$ [see Eq. (11)] where N_{shots} is the number of times each circuit is repeated. Whenever N_{shots} is finite and equal for all the circuits in a dataset then $H(\hat{s}, s_{\mathcal{M}})$ is equal to the log-likelihood of the data given the model's predictions multiplied by a multiplicative factor (of $-N_{\text{shots}} N_{\text{circuits}}$). These conditions are satisfied for many of our datasets, and in those cases minimizing the average BCE is equivalent to maximizing the likelihood of the model given the data.

E. Hyperparameter tuning

A CNN's weights and biases are optimized when training the network, but there are many other parameters that can also affect a CNN's prediction accuracy. Such parameters are called *hyperparameters*. Hyperparameters include (but are not limited to): (i) the number of convolutional, pooling, and dense layers, (ii) the number of neurons in each dense layer, (iii) the shape of each kernel in the convolutional layers, and

(iv) the number of training epochs. Optimal values for hyperparameters are searched for by hyperparameter tuning.

Hyperparameter tuning consists of searching over a space of candidate values for the hyperparameters (we used Bayesian optimization [52, 53]). At each step in the search process, a CNN is created with the candidate hyperparameter values and trained using the training dataset (D_{train}). Each model is evaluated on the validation dataset (D_{validate}), and the hyperparameters with the smallest loss on D_{validate} are chosen. A separate validation set is used because it mitigates the effects of over-fitting each CNN’s weights to the training data. After hyperparameter tuning is complete, a new CNN is created using the hyperparameters associated with the lowest loss on D_{validate} . This network is trained on $D_{\text{train}} \cup D_{\text{validate}}$ (this is standard practice in machine learning). For expediency, we often refer to the combined training and validation dataset as the “training” data when no confusion will arise.

The hyperparameter spaces used in our numerical experiments are provided in the supplementary data and code. Our hyperparameter optimizations included varying the shape of the convolutional kernels, as different kernel shapes (k_w, k_d) can create feature maps that extract circuit features that are relevant for different kinds of error. We varied the width of the kernels, as width- k_w kernels jointly analyze the gates applied to k_w qubits, so they can extract circuit features that are relevant for modelling the effects of k_w -qubit crosstalk. As our encoding does not preserve the spatial locality of qubits, for few qubit processors (small n) we include kernels of width up to $k_w = n$. We varied the length (k_d) of the kernels, as length- k_d kernels jointly analyze k_d circuit layers, so they can extract circuit features that are relevant for modelling the effects of errors that depend on k_d sequential layers (e.g., serial context dependence).

F. Evaluating model performance

To evaluate the performance of a model we quantify its prediction accuracy on one or more test datasets (D_{test}), which were not used during training or hyperparameter tuning. We used three complimentary figures of merit to evaluate the performance of a model on test data: Kullback-Leibler (KL) divergence, the mean absolute error (L^1 error), and the Pearson correlation coefficient (r). KL divergence is defined by

$$d_{\text{KL}}(\hat{s}, s_{\mathcal{M}}) = H(\hat{s}, s_{\mathcal{M}}) - H(\hat{s}),$$

where $H(\hat{s}) = H(\hat{s}, \hat{s})$ is the entropy of \hat{s} . We use KL divergence as a figure of merit in part because the mean $H(\hat{s}, s_{\mathcal{M}})$ is the loss function in the training. The KL divergence removes the entropy of the dataset from $H(\hat{s}, s_{\mathcal{M}})$, facilitating easier comparisons between a model’s performance on datasets with different entropies. The mean absolute error (or L^1 error) defined by

$$d_{L^1}(\hat{s}, s_{\mathcal{M}}) = \frac{1}{N_{\text{circuits}}} \sum_c |s_{\mathcal{M}}(c) - \hat{s}(c)|,$$

and the Pearson correlation coefficient (r) were chosen due to their straightforward interpretations, and to allow comparison

to other work. Note, however, that $r = 1$ does not imply perfect prediction accuracy. This is because r quantifies the linear correlation between a model’s predictions and the data.

G. Predicting capabilities using error rates models

Neural network models for $s(c)$ will be useful if their predictions are sufficiently accurate. A particular task may require a model [$s_{\mathcal{M}}(c)$] for $s(c)$ that achieves a certain accuracy threshold (e.g., 1% or less absolute error on every circuit within some circuit set), and whether a particular neural network model for $s(c)$ satisfies such a criteria can be judged given that task. But a neural network model for $s(c)$ is also only useful if its prediction accuracy is at least as good as other available and equally convenient (e.g., as fast to query) models for $s(c)$. This can be assessed without a particular use-case for $s_{\mathcal{M}}(c)$. Herein, we compare the predictions of our CNNs to that of an *error rates model* (ERM) [3], which we now introduce.

An ERM is a parameterized error model for a processor that consists of modelling each of a processor’s logic operation by an error rate (ϵ). The model’s prediction for $s(c)$ approximately corresponds to multiplying together the success rates $(1 - \epsilon)$ for every logic operation in c . An ERM’s parameters consist of one-qubit gate error rates $\{\epsilon(G, i)\}_{i \in \mathbb{Q}, G \in \mathbb{G}_1}$, two-qubit gate error rates $\{\epsilon(G, i, j)\}_{(i, j) \in \mathbb{B}, G \in \mathbb{G}_2}$ where \mathbb{B} is the set of all connected pairs of qubits, and readout error rates $\{\epsilon(i)\}_{i \in \mathbb{Q}}$. An ERM’s prediction for $s(c)$ is approximately given by

$$s(c) = \prod_{i \in \mathbb{Q}_w} (1 - \epsilon(i)) \prod_{g \in c} (1 - \epsilon(g)), \quad (24)$$

where \mathbb{Q}_w is the set of qubits on which c acts, and $g \in c$ runs over all the gates (labelled with the qubits on which they act) in c . The exact formula for predicting $s(c)$ from an ERM is obtained by using the error rates $\{\epsilon\}$ to construct a global depolarization model [54] [and it differs from Eq. (24) only by $O(1/2^n)$ factors]. That formula can be found in Ref. [3] (see Eqs. (54)-(55) in the supplemental material of Ref. [3]).

ERMs are useful models against which to compare our CNNs because an ERM’s parameters can be efficiently estimated from data (for any number of qubits n), ERMs are fast to query, and (unlike CNNs) ERMs have interpretable parameters. ERMs have these properties because (1) they contain at most $O(n^2)$ parameters that must be learned from data [and, if the qubit’s connectivity is a planar graph then an ERM contains only $O(n)$ parameters], and (2) the prediction for $s(c)$ can be quickly evaluated for any circuit c . For these reasons, an ERM is arguably preferable to a neural network model for $s(c)$ if the two models have equal prediction accuracy. Throughout this work, we fit an ERM to the same data used to train a neural network (we use maximum likelihood estimation to fit ERMs).

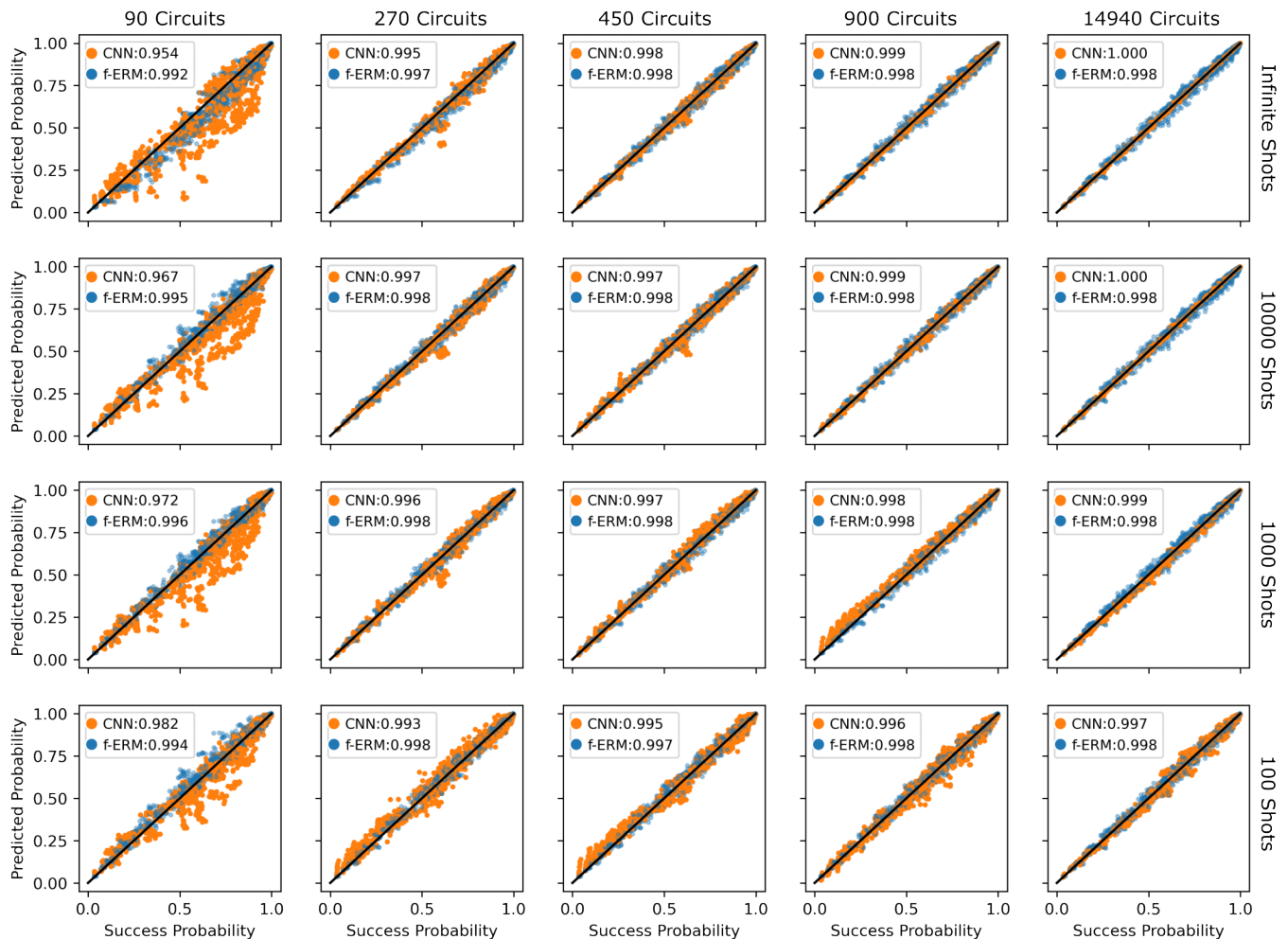


FIG. 3. **Modelling the effect of stochastic errors with CNNs.** The prediction accuracy of CNNs trained on simulated data from few-qubit random circuits with a stochastic Pauli errors model. We constructed a circuit set consisting of 16600 randomized mirror circuits on 1-5 qubits, split it into training, validation and testing circuits (a 70%, 20%, 10% split), simulated each circuit under a single biased stochastic Pauli errors model to compute $s(c)$, and then trained and tuned CNNs on sub-sampled datasets of varying quality—constructed by independently varying the training dataset size (N_{circuits}) and the shot count (N_{shots}), i.e., the number of repetitions of each circuit c used to estimate $s(c)$. We also fit an ERM (error rates model) to the same training data, for comparison. Each subplot shows the true success probabilities $s(c)$ versus the predictions of the CNN (orange) and the fit ERM (f-ERM, blue) evaluated on the full set of 1660 test circuits, for a single (nested) training dataset instance at each (combined validation and) training dataset size and shot count. We observe that the CNN’s predictions improve as the dataset size increases and shot noise decreases. Legends show each model’s correlation coefficient r (note that $r = 1$ implies perfect linear correlation, not perfect predictions). The improvement in the CNN’s accuracy with improving dataset quality is quantified in Fig. 4.

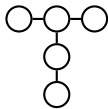
IV. PREDICTING CAPABILITIES WITH FEW QUBITS AND STOCHASTIC ERRORS

Accurate modelling of capability functions for real quantum processors using neural networks will require an architecture that can (1) model the effects of common kinds of errors, and (2) be trained with feasible amounts of data. In this and the following two sections, we use data from simulations of noisy quantum computers to investigate the circumstances under which CNNs can accurately predict $s(c)$. Markovian stochastic Pauli errors—such as uniform depolarization or dephasing—are ubiquitous in experimental quantum computing systems and their effects are relatively simple

to model. So we first study whether CNNs can model $s(c)$ in the presence of Markovian stochastic Pauli errors. In this section we show that CNNs can learn to accurately predict the success probabilities of few-qubit random circuits ($n = 5$) that are subject to local stochastic Pauli errors. We demonstrate that, given sufficient data, a CNN will outperform an ERM fit to the same data. We explore the impact of dataset size, and we find that (1) increasing the training dataset size improves the CNNs predictions, and (2) CNNs perform well even with fairly small training datasets (e.g., ~ 1000 circuits).

A. Error models and datasets

We constructed a dataset consisting of 16600 randomized mirror circuits (see Section III A), for a 5-qubit device with a “T” topology:



The circuits varied in width from 1 to 5, with a circuit of width w designed for and applied to a randomly chosen set of w connected qubits. The circuits varied in depth from 3 to 1825 layers [55].

We simulated the circuits under a single error model, consisting of local stochastic Pauli errors that are maximally biased: for each gate on each qubit, only one of X , Y or Z errors occurs with non-zero probability. The exact error model was randomly selected, i.e., which error can occur for a particular gate and qubit, and the rate of that error, was chosen at random (see Appendix E for the selection protocol). We chose to simulate *biased* errors because larger bias makes the task of modelling $s(c)$ harder in the following sense: when Pauli stochastic errors are biased the success probability of a circuit $s(c)$ not only depends upon the number of times each gate appears in a circuit, but also on the state of the qubits when that gate is applied (e.g., a Z error has no impact on a qubit in a Z eigenstate). The one-qubit and two-qubit error rates were selected to ensure a wide distribution of success probabilities $s(c)$. Each sampled circuit c was simulated under the selected error model to compute its exact success probability $s(c)$, resulting in the dataset $D = \{(c, s(c))\}$. This dataset was then randomly partitioned into training, validation, and testing subsets—with a split of 70%, 20%, 10%, respectively.

To explore how model performance depends on the amount of training data, and on the number of times each circuit was run, we used D to create datasets with fewer total circuits, and with estimates of each $s(c)$ calculated from a finite number of repetitions (N_{shots}) of each circuit c [56]. We created 11 instances of circuit sets containing 100 circuits (i.e., 70 training, 20 validation, and 10 test circuits) and 5 instances of circuit sets containing 300, 500, and 1000 circuits, by sub-sampling from our 16600 circuits [57]. This results in datasets in which the number of combined training and validation circuits (N_{circuits}) is equal to 90, 270, 450, 900, and 14940. For each dataset size, we created datasets with $N_{\text{shots}} = 100, 1000, 10000$, as well as datasets without shot noise (denoted by $N_{\text{shots}} = \infty$), i.e., datasets containing $s(c)$ rather than estimates for $s(c)$.

B. Results

For each dataset, we trained a CNN and fit an ERM on the same data. The prediction accuracy for the trained CNNs and the fit ERMs (f-ERMs) on the test data are summarized in Figs. 3 and 4. Each CNN’s hyperparameters were tuned using the procedure described in Section III E. For every dataset

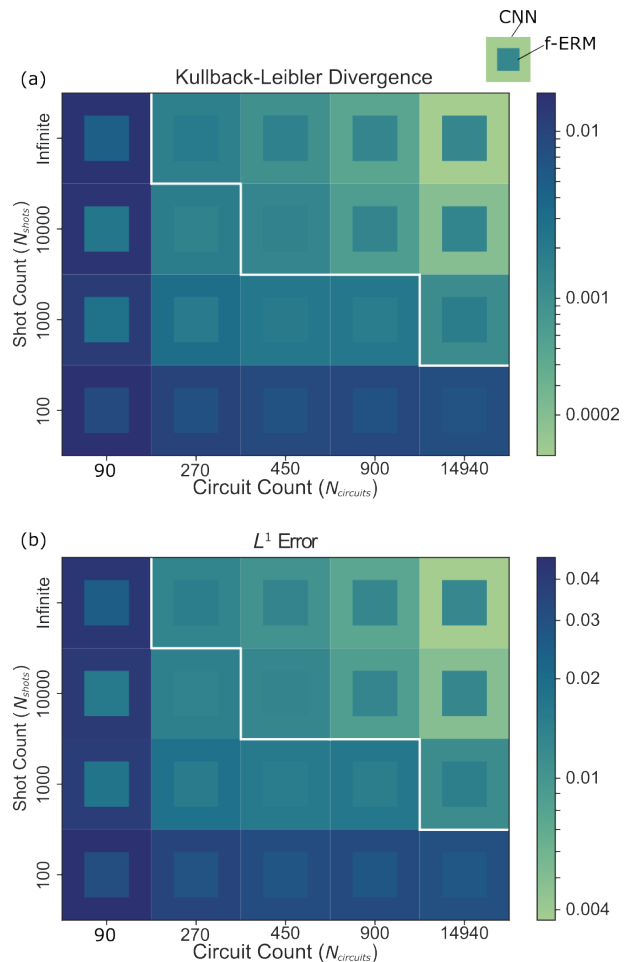


FIG. 4. **Modelling the effect of stochastic errors with CNNs (cont.)** Quantifying the prediction error of CNNs and f-ERMs (fit error rates models) trained on simulated data from few-qubit random circuits with a stochastic Pauli errors model (see the caption of Fig. 3 for details). (a) The KL divergence and (b) the L^1 error for the CNN’s predictions (outer squares) and f-ERM’s predictions (inner squares), averaged over multiple sub-sampled datasets of each size. The CNN’s prediction accuracy surpasses that of the f-ERM in the region above the white line. In contrast with the f-ERM, we observe that the accuracy of the CNNs continues to increase up to the largest dataset size we used.

partition, we evaluated the performance of both the CNN and the f-ERM on the full set of test circuits (1660 circuits) using the true success probabilities $s(c)$ (i.e., the test data for $N_{\text{circuits}} = 14940$ and $N_{\text{shots}} = \infty$). Fig. 3 shows the predictions of trained CNNs and f-ERMs for a single instance of a dataset of each size and shot count. We observe that the prediction accuracy of the CNNs generally increases with both increased dataset size (increasing N_{circuits}) and reduced shot noise (increasing N_{shots}). The CNN outperforms the f-ERM for sufficiently large N_{shots} and N_{circuits} . An arguably necessary condition for a neural network model for $s(c)$ to be useful is that its prediction accuracy is better than a f-ERM (fitting ERMs is efficient and scalable, and an ERM’s parameters can be interpreted), and we observe that our CNNs are satisfy-

ing this necessary criteria for sufficiently high-quality training datasets.

To quantify the accuracy of each model’s predictions, in Figs. 4 (a-b) we show the mean KL divergence (d_{KL}) and mean L^1 error (d_{L^1}) for each dataset size and shot count, averaged over the multiple dataset instances (at each value for N_{circuits} and N_{shots}). The outer and inner squares show the prediction error for the CNN and f-ERM, respectively, as a function of N_{circuits} and N_{shots} . The CNN’s prediction error decreases with increasing training set size and shot count, as quantified by both the KL divergence and the L^1 error. For example, $d_{L^1} = 0.047$ and $d_{\text{KL}} = 0.017$ averaged over the datasets with $N_{\text{circuits}} = 90$ and $N_{\text{shots}} = 100$, whereas $d_{L^1} = 0.08$ and $d_{\text{KL}} = 0.0006$ averaged over the datasets with $N_{\text{circuits}} = 900$ and $N_{\text{shots}} = 10000$. A moderately accurate ERM can be obtained by fitting to few data (see the blue data in the first column of Fig. 3). This is because (1) the ERM is a few-parameter model (in this case it has 28 parameters [58]), and (2) the f-ERM captures significant aspects of the true, data-generating process (the f-ERM fails to capture the bias in the errors, but it does capture the average error in each gate when applied to a random input state). In particular, (when $n \gg 1$) the success rate $s(c)$ of a typical randomized mirror circuit c under a stochastic Pauli error model is well-approximated (although not exactly modelled) by multiplying the probability of a gate causing no error over all the gates in a circuit. The f-ERM therefore significantly outperforms the CNN in the small dataset regime.

We find that the CNNs are more accurate than the ERMs when the datasets are moderately sized and have moderately low shot noise (see Fig. 4 for the precise boundaries). An ERM cannot exactly represent the true data-generating process (a biased Pauli stochastic error model), so its performance is intrinsically limited even in the large dataset limit. These results imply that CNNs are able to learn features of circuits that are more predictive of circuit success probabilities than those encoded into an ERM (gate and readout error rates). The accuracy of the CNN models continues to increase up to the largest dataset size we used (N_{circuits} in the combined training and validation sets). However, the prediction error will converge to a non-zero value as $N_{\text{circuits}} \rightarrow \infty$ if the CNN’s ansatz does not contain the exact $s(c)$ function (or if the optimizer cannot find this function).

V. PREDICTING CAPABILITIES WITH MANY QUBITS AND NON-MARKOVIAN ERRORS

Neural network approaches to modelling a quantum computer’s capability $s(c)$ are appealing because it is plausible that they can circumvent some important limitations of conventional approaches to modelling $s(c)$. The conventional approach to predicting the success probability $s(c)$ of some circuit c is to learn the parameters of a parameterized error model (e.g., process matrices with unknown entries) and to then predict $s(c)$ by simulating the circuit c under this learned error model (e.g., by multiplying together the error model’s process matrices). This approach has two important limitations:

(1) a parameterized model evidently cannot account for effects that are outside of its model, and (2) it is often infeasible to compute $s(c)$ via simulation of c under the learned error model, beyond the few-qubit regime. In this section, we explore whether neural network approaches to approximating $s(c)$ can avoid these two limitations. We investigate whether CNNs can accurately model $s(c)$ in (1) the many-qubit regime and (2) the presence of errors that cannot be described by the most common kinds of parameterized error models (i.e., error models that are restrictions on the maximal Markovian model, as defined in Appendix B). Using simulated data, below we show that CNNs can learn to accurately predict the success probabilities $s(c)$ of many-qubit random circuits ($n = 49$) that are subject to stochastic Pauli errors, and that accurate predictions are still possible with the addition of *non-Markovian* stochastic errors.

A. Datasets

One of the aims in this section is to explore whether CNNs can accurately predict $s(c)$ outside of the few-qubit setting. We therefore focus on a hypothetical 49-qubit system, with a 7×7 grid connectivity. We again consider the task of predicting $s(c)$ for randomized mirror circuits. We created a circuit set containing 10000 randomized mirror circuits, for our hypothetical 49-qubit device, with a training-validation-test split of 37.5%, 12.5%, and 50% (we limited the number of training and validation circuits to 5000 to speed up the model training and hyperparameter tuning). The circuit widths ranged from 1 to 49 qubits, and a w -qubit circuit is designed for a randomly selected set of w connected qubits. The circuit depths ranged from 4 to 272 layers.

B. Predicting many-qubit circuits

First we explore whether CNNs can predict circuit success probabilities in the many-qubit setting. To address this question we generated a many-qubit dataset by simulating the 49-qubit circuit set (described above) under a local stochastic Pauli error model with randomly selected error rates (the model parameters). We used this kind of error model for two reasons: (1) it enables direct comparisons with the performance of CNNs trained on the 5-qubit data presented in Fig. 3, and (2) it isolates the problem of predicting $s(c)$ for larger circuits c from the problem of predicting more complex kinds of errors, e.g., non-Markovian stochastic errors (see Sections VC and VD) or coherent errors (see Section VIB). In this error model, each gate on each qubit is assigned independent error rates for the three possible Pauli errors, resulting in a data-generating model that is described by 1498 parameters [59] (the specific error model used is provided in the supplementary data and code [60]). We can denote the parameters of this model by $\epsilon_P(g, i)$ where P indexes the Pauli error (X , Y , or Z), g denotes the gate (a single-qubit gate or CNOT, implicitly index by the qubit[s] on which it acts), and i denotes one of the qubits that g can act on. We computed $s(c)$

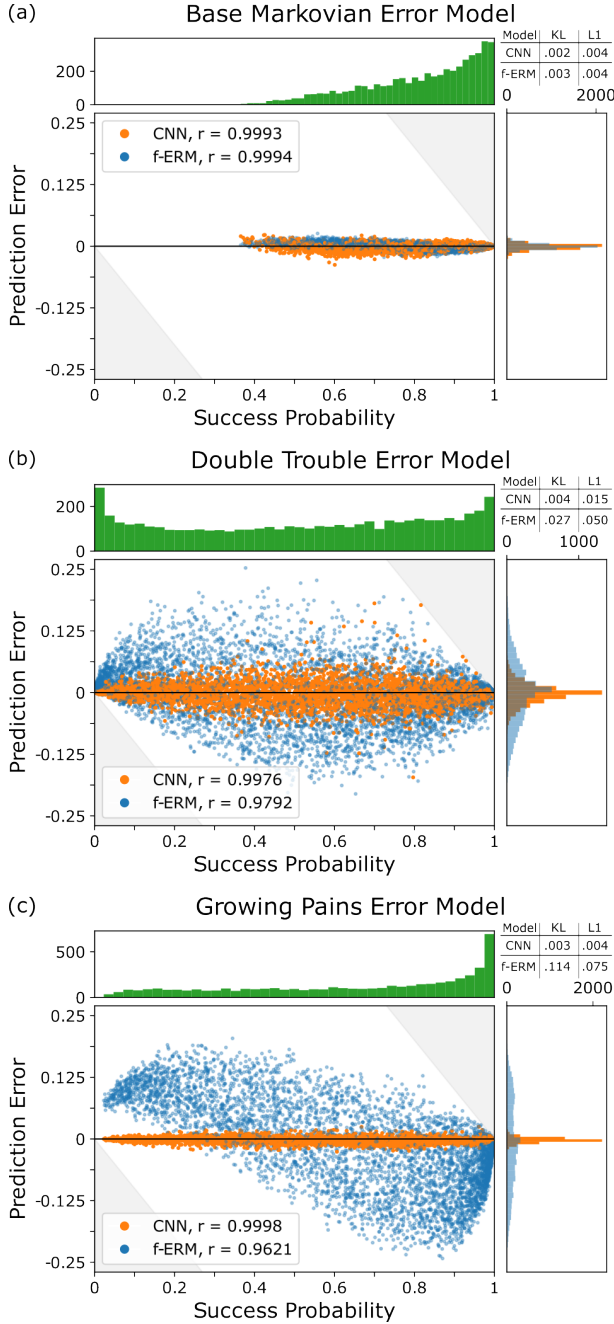


FIG. 5. **Predictions for non-Markovian errors.** The prediction accuracy of CNNs trained on simulated data from random circuits for a hypothetical 49-qubit system with three different models. (a) A Pauli stochastic error model, with gate- and qubit-dependent error rates. This model forms the basis for two non-Markovian models in which (b) a two-qubit gate’s error rate increases if it is preceded by a two-qubit gate on either of its qubits, and (c) gate errors increase over the duration of a circuit. Each main plot within (a-c) shows $s(c)$ versus the prediction error $[\delta(c)]$ defined in Eq. (25) on test data, for a CNN and an ERM fit to the same dataset (f-ERM). The CNN and f-ERM have similar prediction error for the Markovian model, but the CNN significantly outperforms the f-ERM for both non-Markovian models. This is summarized by the $\delta(c)$ histograms [lower right, (a-c)] as well as the KL divergence and L^1 error for each model [upper right, (a-c)]. Each subfigure also contains a histogram of the test data [upper left, (a-c)].

using a simulator that uses the approximation that two or more stochastic errors never cancel (see the supplemental data and code for details [60]). This approximation enables fast simulation, and it is a very good approximation for random circuits with low gate error rates [61] as is the case here. We trained a CNN and fit an ERM using this dataset. For all datasets presented in this section, the CNN’s hyperparameters were tuned using the procedure described in Section III E (the hyperparameter space that we used is provided in the supplementary data and code [60]).

Figure 5 (a) shows the prediction error $\delta(c)$ on the test data for both the CNN and the f-ERM. The prediction error is simply

$$\delta(c) = \hat{s}(c) - s_M(c), \quad (25)$$

where $s_M(c)$ is the model’s prediction. The CNN’s prediction error is small ($d_{L^1} = 4.34 \times 10^{-3}$), demonstrating that CNNs can accurately predict success probabilities for many-qubit circuits and that the CNN can be trained using data from a practically feasible number of circuits (5000 circuits). This CNN’s prediction error is similar to that observed when we trained a CNN on simulated data from a hypothetical 5-qubit system with a similar error model (see Fig. 3), while the number of parameters required to describe the true data-generating process has increased to around 1498. However, note that the f-ERM exhibits nearly the same prediction error as the CNN ($d_{L^1} = 4.18 \times 10^{-3}$), demonstrating that this prediction problem is relatively easy. Furthermore, note that these results demonstrate that a CNN can be successfully trained on many-qubit circuits to predict $s(c)$ for one category of errors—local stochastic Pauli errors—but they do not imply that CNNs will be able to accurately model the effects of more complex kinds of errors that plague many-qubit quantum computing systems, such as crosstalk.

C. Non-Markovianity: temporal context dependence

We now investigate whether CNNs can accurately predict circuit success probabilities in the presence of effects that cannot be modelled by conventional parameterized models for quantum computers (including ERMs). Conventional parameterized models are constructed by placing restrictions on the maximal Markovian model (see Appendix B), so they cannot model non-Markovian errors. We conjecture that neural network methods for modelling $s(c)$ can learn good approximations to $s(c)$ even in the presence of many kinds of non-Markovian errors. To test this conjecture, we created a dataset for each of two different non-Markovian models. Each of these models is a modification to the Markovian error model described above, in Section V B.

One kind of non-Markovianity is gates that get worse over the course of a circuit (e.g., this can occur in ion-trap systems due to heating). To investigate whether CNNs can accurately model $s(c)$ in the presence of this kind of error, we created a simple model (“Growing Pains”) where the error rate of each gate monotonically increases as a function of layer depth. In particular, for each error rate $\epsilon_P(g, i)$ in the Markovian model

(see above), we replace the static error rate with an error rate that is indexed by the layer in the circuit in which the gate occurs:

$$\epsilon_{\text{GP},P}(g, i, l) = \frac{2\epsilon_P(g, i) + \epsilon_P^{\text{max}}(g, i)[e^{l\tau} - 1]}{e^{l\tau} + 1}, \quad (26)$$

where l denotes the layer index. Here $\epsilon_P(g, i)$ is the rate of the Pauli error $P = X, Y, Z$ when the gate at circuit location (i, j) is applied to qubit i in the base Markovian error model [note that $\epsilon_{\text{GP},P}(g, i, 0) = \epsilon_P(g, i)$], $\epsilon_P^{\text{max}}(g, i)$ is a parameter that specifies the error rate at $l \rightarrow \infty$ [we choose $\epsilon_P^{\text{max}}(g, i) = 9\epsilon_P(g, i)$], and τ is a parameter that controls the rate of increase in the error rates (we choose $\tau = 1/350$) [62]. We created a dataset for the Growing Pains error model, by simulating the set of 10000 circuits described in Section V A to compute each circuit's $s(c)$, and we tuned and trained a CNN on this data (and fit an ERM).

Figure 5 (c) shows the prediction error for the CNN and f-ERM, on the test data for the Growing Pains error model. The CNN's prediction error is low ($d_{\text{KL}} = 2.65 \times 10^{-3}$ and $d_{L^1} = 4.22 \times 10^{-3}$), and it is similar to the CNN's prediction error on the Markovian model ($d_{\text{KL}} = 1.82 \times 10^{-3}$ and $d_{L^1} = 4.34 \times 10^{-3}$). We find that 95% of the CNN's predictions fall within ± 0.012 of $s(c)$, with consistently good performance across circuits of different depths, widths, and values for $s(c)$. Convolutional layers are translationally invariant—each convolutional filter does not distinguish between the same gate pattern that appears near the start of circuit and near the end of a circuit—but the data generating model is not. However, some circuit location information is preserved by the convolutional layers of the network (the convolutional portion of the network outputs an image). Therefore, the subsequent dense network can learn weights that enable the complete network to model the effect of increasing gate error rates.

The CNN vastly outperforms the f-ERM—the L^1 error for the f-ERM ($d_{L^1} = 0.075$) is almost twenty times larger than the L^1 error for the CNN trained on the same data. No conventional Markovian error model, including an ERM, can describe gates whose performance gets worse over the duration of a circuit. So, when fit to this dataset (which contains circuits of various depths, making this non-Markovianity visible), the best-fit error rates produce a f-ERM with over-optimistic predictions for deep circuits and over-pessimistic predictions for shallow circuits. This is the cause of the bias in the predictions of the f-ERM for high and low success probabilities circuits seen in Fig. 5 (b) [$s(c)$ is anti-correlated with circuit depth].

D. Non-Markovianity: serial context dependence

We now apply CNNs to learn a capability function in the presence of another kind of non-Markovian error: *serial context dependence*. In an error model with serial context dependence, a gate's error process depends on the gates that precede or follow it. To investigate whether CNNs can model $s(c)$ in the presence of serial context dependence, we constructed a simple model (“Double Trouble”) for this kind of error. We

modified our Markovian error model so that a qubit's error rates for a two-qubit gate are increased if that qubit is acted on by another two-qubit gate in the preceding layer. Specifically, a gate g 's rate of Pauli P errors on qubit i in layer l of circuit c is given by

$$\epsilon_{\text{DT},P}(g, i, l) = 1 - [1 - \epsilon_P(g, i)](1 - \epsilon_{\text{add}})^{N_{\text{CNOT}}(i,l)N_{\text{CNOT}}(i,l-1)}, \quad (27)$$

where $N_{\text{CNOT}}(i, l) = 1$ if location (i, l) in circuit c is a CNOT gate and otherwise $N_{\text{CNOT}}(i, l) = 0$. We choose $\epsilon_{\text{add}} = 0.005$, so if a qubit is involved in two consecutive CNOT gates, the error rate of the second CNOT gate increases by approximately 0.005.

Figure 5 (c) shows the prediction error for a CNN, trained and tuned on data from the Double Trouble error model, and a ERM fit to the same data (f-ERM). The CNN's prediction error on test data is significantly larger than it is for a CNN trained and tested on the base Markovian model (the L_1 error is approximately three times larger: $d_{L^1} = 0.015$ compared to $d_{L^1} = 4.34 \times 10^{-3}$). However, the CNN still significantly outperforms the f-ERM, as the CNN's L_1 error is approximately three times smaller ($d_{L^1} = 0.015$ compared to $d_{L^1} = .05$). We find that 95% percent of the CNN's predictions are within ± 0.045 of $s(c)$.

Convolutional layers can pick out localized pattern of gates, by learning convolutional filters (and biases) that return a non-zero value if and only if that pattern of gates appears. Sequential CNOT gates can therefore be identified by a convolutional filter applied directly to the input image representation of a circuit (i.e., a convolutional filter in the first convolutional layer). In particular, there exists a set of four convolutional filters that enable a convolutional layer to identify all instances of sequential CNOT gates (these filters are provided in Appendix F). This suggests that the CNN training and hyperparameter tuning process that we have used is finding significantly sub-optimal convolutional filters in this case.

VI. CHALLENGES TO USEFUL NEURAL NETWORK MODELS FOR CAPABILITIES

In this section we explore some important challenges to creating useful neural network models for a quantum computer's capability $s(c)$. We explore the problem of predicting $s(c)$ for circuits sampled from a different distribution to the training data (Sections VI A); we highlight the difficulty of modelling the impact of coherent errors using neural networks (Section VI B); and we investigate the importance of including error sensitivity information within the neural network's input (Section VI C).

A. Generalizing to out-of-distribution circuits

Each practical application for a model of $s(c)$ will require that the model's predictions are accurate for some set of circuits of interest \mathbb{C} , and this circuit set will generally be application-specific. For example, one application for a model of $s(c)$ is finding a low-error compilation for some algorithm

\mathcal{A} . There are many circuits that implement an algorithm \mathcal{A} , and the primary aim in compilation is to find the circuit c in the set of all such circuits ($\mathbb{C}_{\mathcal{A}}$) that maximizes $s(c)$. Using a model for $s(c)$ to inform this compilation (e.g., to define a cost function to be used by an optimizer) requires that it is accurate for the circuit set $\mathbb{C}_{\mathcal{A}}$. The relevant set of circuits will vary between applications of our model for $s(c)$. For each circuit set \mathbb{C} of interest, a neural network can be trained on data from circuits sampled from \mathbb{C} . However, retraining a neural network is expensive (it requires new training data, and network training can be slow). Therefore, it is interesting to explore whether neural network models can accurately predict $s(c)$ for circuit sets \mathbb{C} that are sampled from a different distribution to the training data—a task that is often referred to as *out-of-distribution generalization* [63]. Below we present two examples of out-of-distribution generalization.

First, we consider the problem of predicting wide circuits (here $w > 25$ active qubits) using a CNN trained on data containing only narrow circuits ($w \leq 25$ active qubits). This is a simple example with which to explore out-of-distribution generalization, but it also has practical relevance. In particular, for some definitions for $s(c)$ (such as TVD) there is no known method for efficiently measuring $s(c)$ in an experiment if c is a wide and deep circuit (see Section II). We use the data simulated under the Growing Pains noise model (see Section V). We trained a CNN on only the narrow circuits in the training dataset (n-CNN), and we fit an ERM to the same data (nf-ERM). No automated hyperparameter tuning was performed, except to select the number of training epochs (see the supplemental data and code [60] for the architecture) [64]. In this error model, gate performance gets worse later in a circuit, but a gate’s error rate is independent of width. In particular, the parameters of the data generating error model (Growing Pains) can be learned from the $w \leq 25$ qubit circuits data. It is therefore plausible that a CNN trained on data from narrow circuits will generalize to accurately predict $s(c)$ for wide circuit (whereas we could not expect a model trained on few-qubit circuits data to accurately predict the success probabilities of many-qubit circuits if there are additional errors that only appear in those many-qubit circuits, e.g., many-qubit crosstalk effects [5]).

Figure 6 displays the n-CNN’s out-of-distribution predictions on the wide circuit test data (purple points) alongside the predictions of the CNN that was trained on all of the training data (a-CNN, orange points), which includes narrow and wide circuits. The n-CNN generalizes moderately well in the sense that the prediction error of n-CNN is reasonably small ($d_{L^1} = 0.02$). However, the prediction error for the n-CNN is approximately twenty times larger than for a-CNN ($d_{L^1} = 0.001$), so the relative increase in the prediction error when removing wide circuits from the training dataset is large. There is also a bias in the out-of-distribution network’s predictions: the n-CNN systematically underestimates $s(c)$ on the out-of-distribution circuits [i.e., typically $\delta(c) > 0$]. Interestingly, the n-CNN still achieves low prediction error on those test circuits with small $s(c)$, even though circuits with small $s(c)$ are underrepresented in the narrow circuit training data.

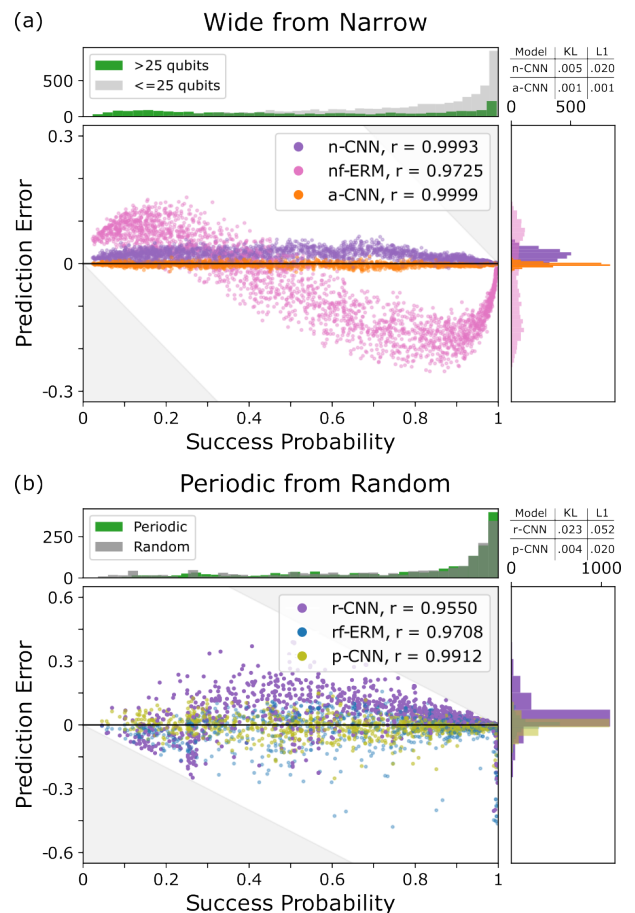


FIG. 6. Generalizing to out-of-distribution circuits. Two examples of generalizing a neural network to predict circuits that are drawn from a distribution that differs significantly from the training distribution. **(a)** The prediction accuracy [$\delta(c)$] of a CNN (n-CNN, purple) trained on narrow circuits ($w \leq 25$ qubits) evaluated on a test dataset containing wide circuits ($w > 25$ qubits). The data was simulated under an error model in which gate errors increase with circuit depth, so narrow circuits reveal all important aspects of the error model, meaning that accurate generalization to wider circuits is feasible. The prediction error $\delta(c)$ for n-CNN is much smaller than for an ERM fit to the same training data (nf-ERM, pink). However, $\delta(c)$ is larger for n-CNN than for a CNN trained on a dataset containing both narrow and wide circuits (a-CNN, orange). **(b)** The prediction error of a CNN (r-CNN, purple) trained on random circuits and evaluated on periodic circuits. The r-CNN has an L^1 error of $d_{L^1} = 0.005$ on test data drawn from the same distribution as the training data (random circuits), but it increases to $d_{L^1} = 0.052$ on periodic circuit data, resulting in worse predictions than provided by an ERM fit to the same training data (rf-ERM, blue). However, the CNN’s performance can be substantially improved by re-training the CNN on periodic circuit training data (p-CNN, yellow), while maintaining the same architecture that was found by hyperparameter tuning using the random circuits data.

Our second example of out-of-distribution generalization considers the problem of predicting $s(c)$ for circuits with different structures to those in the training dataset. In this work we have so far considered training CNNs on data from *random* circuits. In particular, we have used data from randomized mirror circuits (see Section II), and these circuits are de-

finer by a distribution that has support on all possible mirror circuits. However, a circuit sampled from this distribution is almost certainly highly disordered [3], e.g., it will almost certainly not contain repeated patterns of gates. It is not *a priori* clear whether a CNN trained only on highly disordered circuits (e.g., randomized mirror circuits) will generalize to accurately predict $s(c)$ for highly ordered circuits (e.g., periodic circuits, or algorithmic circuits). However, error propagation in disordered circuits (where errors are scrambled, and coherent errors add quadratically) [61] differs substantially from error propagation in highly ordered circuits (where errors can be amplified or echoed away, and coherent errors can add linearly) [7], suggesting that neural networks trained only on disordered circuits will generalize poorly to structured circuits.

To explore whether CNNs trained on random circuits generalize to ordered circuits we created a dataset of 5-qubit periodic mirror circuits and simulated them under the Markovian error model described and used throughout Section IV (our simulation used $N_{\text{shots}} = \infty$ shots). We used a CNN that was trained on data from $N_{\text{circuits}} = 14940$ random mirror circuits (also with $N_{\text{shots}} = \infty$) to predict $s(c)$ for these periodic mirror circuits (we denote this CNN by r-CNN). Figure 6 (b) shows r-CNN’s prediction error on the test dataset (i.e., the periodic mirror circuits). The prediction error for r-CNN is large and it is approximately ten times larger on this out-of-distribution test data ($d_{L_1} = 0.052$) than for in-distribution test data ($d_{L_1} \approx 0.004$).

To quantify the performance of the r-CNN we compare it to two alternative models: an ERM fit to the random circuit data (rf-ERM) and a CNN trained on periodic mirror circuit data (p-CNN). As shown in Fig. 6 (b), the rf-ERM outperforms the r-CNN on the out-of-distribution test data, even though CNNs trained on random circuit data have substantially lower prediction error than fit ERMs on in-distribution test data (see Fig. 4). However, if we retrain the CNN using the periodic mirror circuit training data, we obtain significantly better model performance (p-CNN in Fig. 6). These results suggest that CNNs learn features that encode how a specific error model interacts with the specific class of circuit used in the training. This is both a strength—as it enables CNNs to outperform ERMs, which cannot model the interaction between circuit structure and a specific error model—and a weakness—as it limits the prediction accuracy of CNNs on out-of-distribution circuits. We conjecture that two complementary approaches will improve a CNNs out-of-distribution prediction accuracy: (1) fine-tuning a trained neural network using a small amount of data for each circuit family of interest, e.g., by retraining only some of a network’s weights; (2) encoding known physics for how errors propagate through circuits within a neural network’s architecture and/or within the circuit encoding.

B. Prediction in the presence of coherent errors

Techniques for modelling capability functions will only be useful in practice if they can learn a good approximation to $s(c)$ in the presence of all the types of errors that real pro-

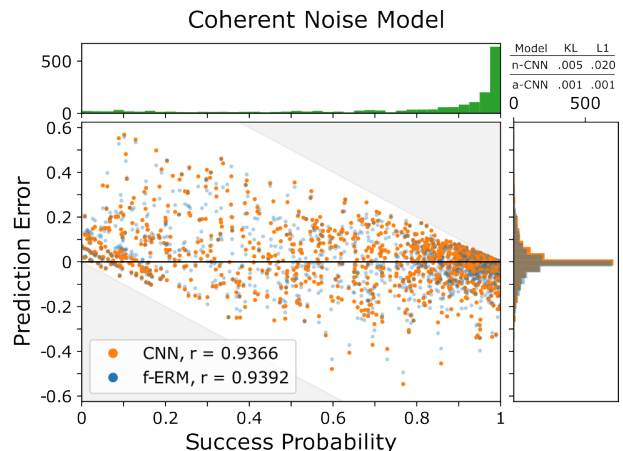


FIG. 7. **Inaccurate capability models when errors are purely coherent.** The prediction accuracy of a CNN and a f-ERM trained on randomized mirror circuit data ($n = 5$ qubits) generated from an error model with purely coherent (i.e., Hamiltonian) errors. We find that neither model accurately predicts the test data ($d_{L_1} \approx 0.06$ for both models). Predicting $s(c)$ in the presence of coherent errors is challenging because coherent errors can add or cancel across an entire circuit.

cessors commonly experience. In this paper so far, we have only considered modelling $s(c)$ in the presence of (Markovian and/or non-Markovian) stochastic Pauli errors, but real quantum processors also experience many other kinds of error (e.g., see Ref. [12]). Coherent errors are ubiquitous and their effect on $s(c)$ is particularly challenging to predict, because they can coherently add or cancel within circuits. We therefore investigated whether CNNs can accurately model $s(c)$ in the presence of coherent errors.

We simulated a set of 5-qubit randomized mirror circuits (the circuits of Section IV) under an error model consisting of purely coherent errors on each gate. We randomly sampled the error model, using the procedure provided in Appendix E. Figure 7 shows the prediction accuracy for a CNN (with tuned hyperparameters) and a f-ERM. The CNN has a slightly larger prediction error ($d_{L_1} = .06$) than the f-ERM ($d_{L_1} = .0599$), and neither model’s predictions are accurate. For both models, there are circuits within the test set for which the L_1 error is over 0.5, which is arguably a catastrophic prediction failure. Coherent errors are a significant proportion of the total error in many contemporary quantum computing systems (see, e.g., Refs. [12, 65]), and so CNNs’ inability to model $s(c)$ in the presence of coherent errors will limit the prediction accuracy of CNN models for $s(c)$ that are trained on experimental data (this is consistent with our results in Section VII).

Low prediction accuracy for CNN models of $s(c)$ in the presence of coherent errors can be explained as follows. The impact of coherent errors on a particular circuit c is difficult to predict because coherent errors can coherently add or cancel across the entire circuit. Whether two coherent errors at two circuit locations (i, j) and (i', j') coherently add or cancel strongly depends both on these errors (their magnitudes and directions) as well as the unitary evolution caused by the

circuit layers between j and j' . Furthermore, because coherent errors at any two circuit locations can cancel (or add), their combined effect likely cannot be modelled by a CNN that has convolutional filters that are much smaller than the circuit size. Exact classical modelling of the effect of coherent errors on a general circuit c likely requires simulating the unitary evolution of each circuit layer, and it is perhaps infeasible for a neural network to learn a good approximation to $s(c)$ when coherent errors dominate. However, it is possible that improvements to the circuit encoding (see below) or the neural network architecture may greatly improve the accuracy of neural network models for $s(c)$ in the presence of coherent errors, as we discuss briefly in Sections VIC and VIII. Furthermore, note that coherent errors can be converted into stochastic Pauli errors using randomized compiling [65, 66] or Pauli frame randomization [67].

C. The role of error sensitivities in capability learning

Our encoding $[I(c)]$ for a circuit c includes a limited amount of information about the sensitivity of that circuit to errors. In particular, pixel $I_{ij}(c)$ encodes information about whether a Pauli X , Y , or Z error on qubit i after layer j will change the state of the qubits (see Section III B). This information is stored in three “error sensitivity” channels, that correspond to these three kinds of error. We included these three channels in our encoding as we conjectured that they make it easier for a CNN to learn an accurate model for $s(c)$ in the presence of local stochastic Pauli errors (including non-Markovian local stochastic Pauli errors). This conjecture is supported by the observation that there exists a CNN (an architecture with specific weights) that is a good approximation to any local stochastic Pauli error model and that our construction for such a CNN (see Appendix D) uses the information in these error sensitivity channels.

To examine the importance of the error sensitivity channel, we trained a CNN on a dataset that removed the error sensitivity channels from our circuit encoding $I(c)$. We used the 5-qubit local Pauli stochastic errors dataset from Section IV (we used the $N_{\text{shots}} = \infty$ and $N_{\text{circuits}} = 14940$ dataset). We did not perform hyperparameter tuning (using instead the network architecture found when performing hyperparameter tuning with the error sensitivity channels included). Figure 8 shows the predictions of the CNN trained on datasets that do and do not include the error sensitivity channels. The performance of the CNN without access to the error sensitivity information is significantly worse ($d_{L1} = 0.011$ versus $d_{L1} = 0.004$), supporting our hypothesis that this error sensitivity information significantly reduces the difficulty of the learning problem.

Our results suggest that channels that encode a circuit’s sensitivity to local Pauli stochastic errors improve a CNN’s ability to model $s(c)$ in the presence of local Pauli stochastic errors. This suggests a promising path forward for accurate modelling of $s(c)$ in the presence of more general classes of error, including coherent errors: the inclusions of additional error sensitivity information in the circuit encoding. There are, however, significant challenges to this approach. First, the

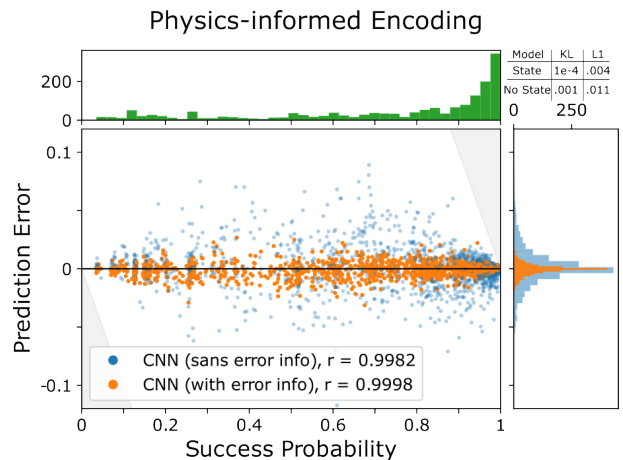


FIG. 8. **Error sensitivity information improves model accuracy.** The prediction error of CNNs trained with and without the error sensitivity channels—which we use to encode information about each qubit’s sensitivity to the three single-qubit Pauli errors at each circuit location—on randomized mirror circuit data ($n = 5$ qubits) simulated under a Pauli stochastic error model. We observe significantly better performance (the KL divergence is an order of magnitude smaller) for the CNN that has access to the error sensitivity channels. This suggests that error sensitivity information is important for accurate capability learning.

impact of some kinds of errors—including coherent errors—cannot be localised to each circuit location, because their overall effect depends on how they combine. Second, our current method for including error sensitivity information within $I(c)$ relies on the circuit containing only Clifford gates—but a useful model for $s(c)$ arguably also needs to predict $s(c)$ for non-Clifford circuits. We therefore suggest that an interesting open problem is the development of a representation of circuits that includes partial or approximate error sensitivity information for a broad range of errors in general circuits.

VII. DEMONSTRATION ON EXPERIMENTAL DATA

In this section we explore how accurately CNNs can model the capabilities of cloud-access quantum computing systems.

A. Datasets

We used data from an existing publicly-available dataset [68] that consists of success probabilities for varied width and depth randomized and periodic mirror circuits, run on various cloud-access quantum processors. We used data from seven different IBM Q processors: a 14-qubit system (IBM Q Melbourne), and six different 5-qubit systems. Each processor’s dataset was obtained by running 40 shape (w, d) randomized mirror circuits and 40 shape (w, d) periodic mirror circuits, with w and d varied systematically (w took every possible value, and d was exponential spaced). For each processor, all w -qubit circuits were run on the same set of w qubits. The

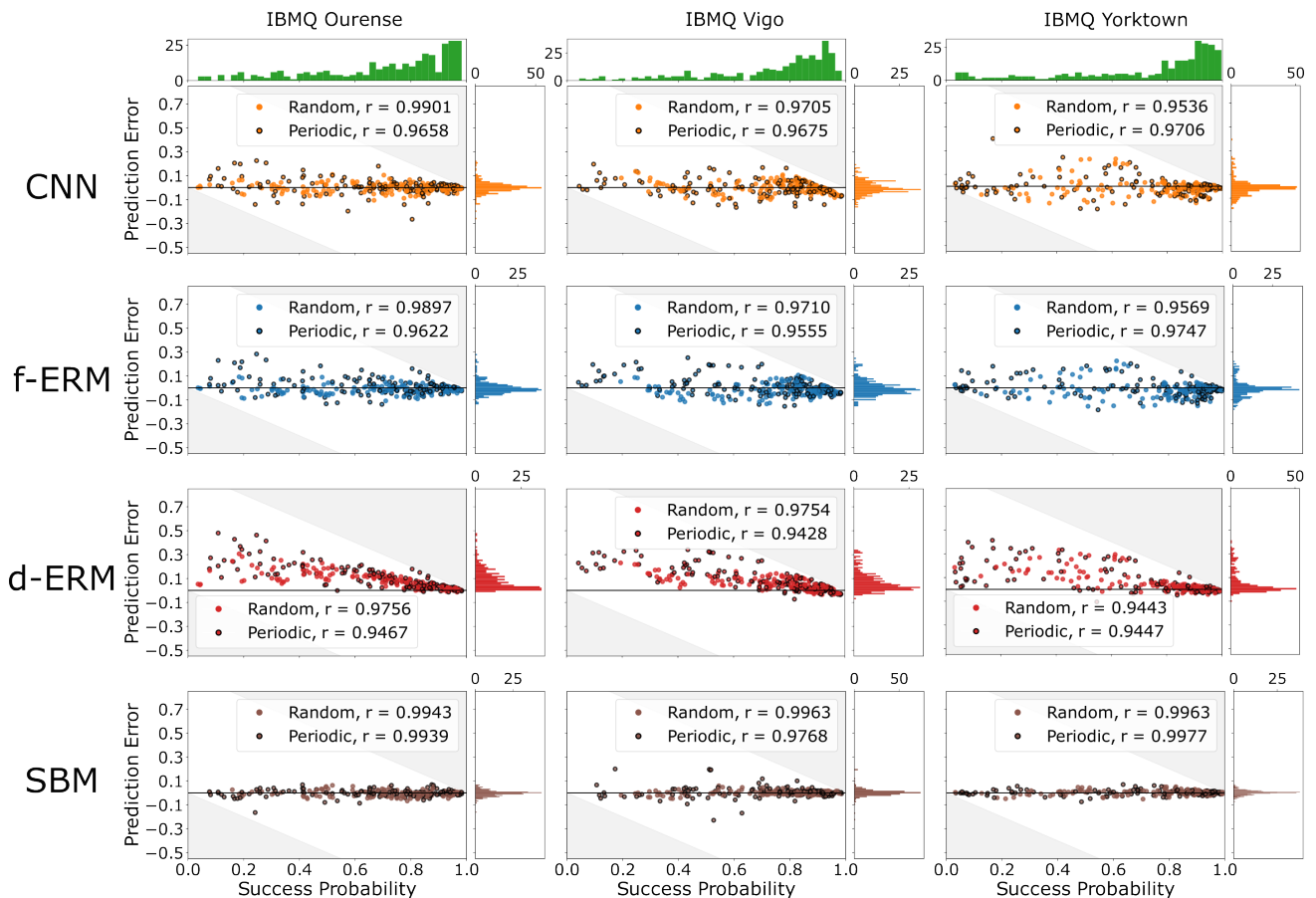


FIG. 9. **Predicting the capabilities of cloud-access IBM Q processors.** We trained CNNs to predict circuit success probabilities for seven cloud-access IBM Q processors (which are all 5-qubit or 14-qubit systems), using a dataset that consists of success probability estimates $\hat{s}(c)$ for approximately 3000 (5-qubit processors) or 5000 (14-qubit processors) periodic and randomized mirror circuits of varying widths and depths. We show the prediction error on the test data (for three processors) for the CNN (top row, orange), an ERM fit to the same training data (f-ERM, second row, blue), an ERM that uses the gate error rates provided by IBM (d-ERM, third row, red), and a “stability baseline model” (SBM) that quantifies the stability of the processor (fourth row, brown). The CNN and f-ERM’s prediction error are comparable and constitute moderate prediction accuracy, although the CNN outperforms the f-ERM in most cases (see Fig. 10). Prediction error is separated into periodic mirror circuits (black outline) and randomized mirror circuits (no outline), and for all but one processor we observe lower prediction error for the randomized mirror circuits (see r values reported in legends). Both the CNN and f-ERM are substantially more accurate than the d-ERM (the d-ERM’s error rates are known to miss important sources of error [5]). The SBM model (fourth row) is a baseline that quantifies how stable each circuit’s success probability is over time (see main text for details), and it is unlikely that a CNN (or another model) will outperform the SBM without including additional context information in the training data (e.g., timestamps). We observe that neither the CNN nor the ERMs achieve the prediction accuracy of the SBM, but for all but two processors the average L^1 error of the CNN is less than two times the average L^1 error of the SBM (see Fig. 10).

number of circuits in each dataset is approximately 3000 (for the 5-qubit systems) or approximately 5000 (for IBM Q Melbourne). Each set of circuits was run, in a randomized order, obtaining an estimate $\hat{s}(c)$ for each circuit’s success probability $s(c)$ from $N_{\text{shot}} = 1024$ executions of the circuit. The entire circuit list was then immediately run again, obtaining a second estimate $\hat{s}'(c)$ (with $N_{\text{shot}} = 1024$) for each circuit. We use the data from the first pass through the circuits (“pass 1”) for training and evaluating our CNNs, and the data from the second pass through the circuits (“pass 2”) to quantify stability. Further details of these datasets and experiments are presented in Ref. [3] (the datasets used here are referred to as “experiment 2” in the supplementary information therein

[69]). We randomly separated the circuits for each processor into training, validation, and test circuits, with an 80%, 10%, 10% split.

B. Results

We trained and tuned CNNs separately for each processor, using that processor’s training and validation datasets. Figure 9 (top row) shows the prediction accuracy of the CNNs on the test data for three of the seven processors, separated into periodic and randomized mirror circuits. Equivalent plots for the other four processors can be found in Appendix G (see

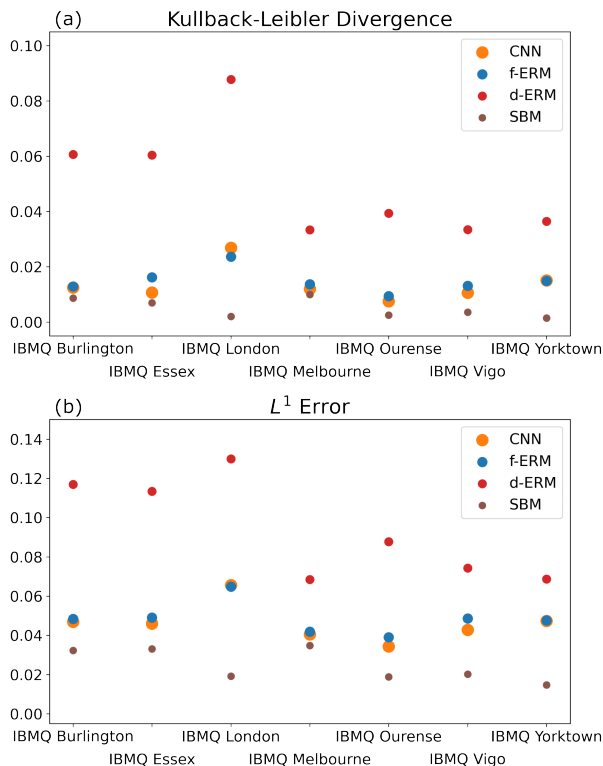


FIG. 10. **Prediction accuracy of capability models for cloud-accessible IBM Q processors.** The prediction accuracy on test data for CNNs trained on data from seven different cloud-access IBM Q processor, quantified by (a) KL divergence and (b) L^1 error. We also show the prediction accuracy for f-ERMs, d-ERMs, and SBMs. See Fig. 9 and Section VII for further details. (In all cases, error bars are too small to be visible.)

Figs. 11 and 12). We observe moderate prediction accuracy—the average L^1 error on the test data ranges from 0.04 to 0.08, and the KL divergence ranges from 0.01 to 0.09 (see Fig. 10). This prediction accuracy is similar to or better than that obtained in other recent work that applies neural networks to model $s(c)$ on cloud-access quantum processors [38, 40], but note that those papers apply neural networks to slightly different tasks and different circuit families, so meaningful quantitative comparisons are not possible. For all but one processor, the prediction error for the randomized mirror circuits is smaller than the prediction error on the periodic mirror circuits (compare the r values in the legends of Figs. 9, 11 and 12). The success probabilities of periodic circuits are typically harder to predict, due to the interactions between the (unknown) structure in a processor’s errors and the circuit’s structure. Note that we observed lower prediction accuracy on periodic circuits than on random circuits with simulated data (see Section VIA).

To explore how the accuracy of our CNNs compares to alternative models for $s(c)$, we compare their predictions to two classes of ERMs. One of these ERMs has parameters (i.e., error rates) obtained from the processor’s performance data provided by IBM Q [70], which we call the “device error rates model” (d-ERM). The CNN substantially outperforms the d-

ERM for every dataset, as shown in Fig. 9 (compare the top and third rows) and Fig. 10. This large improvement in the prediction accuracy compared to a d-ERM highlights the limitations of using parameterized error models obtained from only one- and two-qubit RB experiments (IBM Q’s performance data), as they miss many important sources of error in many-qubit circuits (e.g., crosstalk). As throughout this paper, we also compare each CNN’s predictions to an ERM whose parameters are also fit to the training data (f-ERMs). For most processors, the CNN’s prediction error is lower than the f-ERM, but it is only a slightly improvement (see Fig. 10, and compare the top and second row of Fig. 9). For one dataset (from IBM Q London), the f-ERM actually outperforms the CNN, even though a good approximation to the f-ERM exists within the CNN’s parameter space (see Appendix D). For all seven datasets, the CNN’s increase in accuracy over the f-ERM is too small to be of any practical significance. Like most contemporary quantum computing systems, IBM Q processors experience a mixture of coherent and stochastic errors, and the prediction accuracy of these CNNs is between what we observed in simulations with purely stochastic errors (Sections IV and V) and purely coherent errors (Section VIB). We therefore conjecture that a neural network method that can accurately predict $s(c)$ when coherent errors are a significant proportion of the total error budget is the primary advance required to obtain useful neural network models for $s(c)$.

Real quantum processors are not stable, i.e., their error processes vary with time [12, 13, 20], which implies that a processor’s capability function $s(c)$ depends on time t (and possibly other context variables—see Section II). Because our datasets do not include execution time in the data encoding, it is infeasible to learn the impact of any processes that vary over time scales that are longer than the time for running individual circuits (processes that varying within the execution time of a single circuit can be learned, as demonstrated above). Therefore the magnitude of a processor’s instability provides a lower bound on how accurate a model trained on that data can be. The magnitude of a processor’s instability can be quantified by using the estimate of $s(c)$ for each circuit from the second pass through all the circuits—denoted by $\hat{s}'(c)$ —as a prediction for $s(c)$ on the test data (obtained in pass 1). This prediction for $s(c)$, which we call the *stability baseline model* (SBM), is shown in the fourth row of Fig. 9. We observe that the CNNs (and the f-ERMs) do not achieve the prediction accuracy of the SBMs. However, for all but two processors the average L^1 error of the CNN is less than two times the average L^1 error of the SBM (see Fig. 10).

VIII. CONCLUSIONS

Understanding a quantum processor’s computational power requires knowledge of what quantum circuits it can run with low probability of error, and this can be formalized using the concept of a *capability function* $s(c)$. But modelling a quantum processor’s capability $s(c)$ is hard, as $s(c)$ depends on a processor’s unknown errors and how those errors interact with each circuit c . Parameterized error models can be used to pre-

dict $s(c)$, but simple, scalable error models fail to accurately model $s(c)$. More complex error models like process matrices are not scalable—yet they still often fail to model $s(c)$ accurately [19]. In this work we have investigated using neural networks to learn an approximation to $s(c)$. We have formalised this prediction problem [i.e., defining $s(c)$], and we have shown how to efficiently obtain training data for a particular definition of $s(c)$: process fidelity.

We have presented a CNN architecture and circuit encoding for modelling $s(c)$, and we have shown that these CNNs can accurately model $s(c)$ in the presence of Markovian or non-Markovian local Pauli stochastic noise. Interestingly, we have demonstrated that a CNN vastly outperforms an ERM (error rates model) on some simple non-Markovian error models, demonstrating the power of neural networks to model the impact of errors that are outside the standard framework of parameterized error models. However, a practical technique for modelling $s(c)$ must be able to learn a good approximation to $s(c)$ in the presence of all the kinds of error that real processors commonly experience, and this is not the case with our CNNs. We find that they cannot model $s(c)$ when coherent errors are dominant, and this severely limits the applicability of these networks to real systems. A promising approach to solving this problem is to use a circuit encoding (and a corresponding network architecture) that contains detailed error sensitivity information, generalizing the error sensitivity channels in our encoding. Our error sensitivity channels encode a circuit’s sensitivity to local Pauli stochastic errors, and they help our CNNs perform excellently on local Pauli stochastic error models. However, whether it is possible to efficiently summarize general error sensitivity information for general circuits is an open problem.

The CNN architecture we have employed is well-suited to our prediction problem in a number of ways, e.g., a convolutional filter jointly analyzes information from a temporal neighbourhood of a gate. But this architecture also has some features that likely limit its ability to accurately model $s(c)$. Firstly, spatial relationships between qubits are not explicitly encoded, yet it is likely to be of significance for predicting $s(c)$ due to the importance of effects like localized crosstalk errors. Second, each convolutional filter in a CNN is transitionally invariant, yet error processes in quantum computers are not: different qubits experience different errors and at different rates. So, although we have found that a CNN can accurately learn to predict $s(c)$ under error models that are not spatially invariant, it is not optimized for this task. We therefore conjecture that alternative neural network architectures—such as graph neural network—will substantially outperform CNNs on the capability learning problem. Indeed, a promising method for predicting $s(c)$ using a graph neural network was recently demonstrated by Wang *et al.* [38]. It is possible that a neural network and data encoding that combines error sensitivity information with graph neural networks—i.e., a physics-informed neural network model of $s(c)$ —could learn accurate models for $s(c)$ on real quantum processors, enabling fast and accurate predictions of a processor’s computational capabilities.

ACKNOWLEDGEMENTS

This material was funded in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Testbed Pathfinder Program, and by the Laboratory Directed Research and Development program at Sandia National Laboratories. T.P. acknowledges support from an Office of Advanced Scientific Computing Research Early Career Award. Sandia National Laboratories is a multi-program laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA-0003525. All statements of fact, opinion or conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of the U.S. Department of Energy, or the U.S. Government.

DATA AND CODE AVAILABILITY

Data, code, and models are available at Ref. [60]. All circuit sampling and error model simulations were performed using pyGSTi [71] or custom code that can be found at Ref. [60]. All neural networks were constructed and trained using TensorFlow with Keras [72, 73].

REFERENCES

- [1] Frank Arute, Kunal Arya, Ryan Babbush, *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature* **574**, 505 (2019).
- [2] Easwar Magesan, J. M. Gambetta, and Joseph Emerson, “Scalable and robust randomized benchmarking of quantum processes,” *Phys. Rev. Lett.* **106**, 180504 (2011).
- [3] Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout, “Measuring the capabilities of quantum computers,” *Nature Phys* **18**, 75 (2021).
- [4] Timothy Proctor, Stefan Seritan, Kenneth Rudinger, Erik Nielsen, Robin Blume-Kohout, and Kevin Young, “Scalable randomized benchmarking of quantum computers using mirror circuits,” *Phys. Rev. Lett.* **129**, 150502 (2022).
- [5] Jordan Hines, Marie Lu, Ravi K Naik, Akel Hashim, Jean-Loup Ville, Brad Mitchell, John Mark Kriekebaum, David I Santiago, Stefan Seritan, Erik Nielsen, Robin Blume-Kohout, Kevin Young, Irfan Siddiqi, Birgitta Whaley, and Timothy Proctor, “Demonstrating scalable randomized benchmarking of universal gate sets,” [arXiv:2207.07272](https://arxiv.org/abs/2207.07272).
- [6] Karl Mayer, Alex Hall, Thomas Gatterman, Si Khadir Halit, Kenny Lee, Justin Bohnet, Dan Gresh, Aaron Hankin, Kevin Gilmore, and John Gaebler, “Theory of mirror benchmarking and demonstration on a quantum computer,” [arXiv:2108.10431](https://arxiv.org/abs/2108.10431).
- [7] Erik Nielsen, John Gamble, Kenneth Rudinger, Travis Scholten, Kevin Young, and Robin Blume-Kohout, “Gate set tomography,” *Quantum* **5**, 557 (2021).
- [8] Robin Blume-Kohout, Marcus P. da Silva, Erik Nielsen, Timothy Proctor, Kenneth Rudinger, Mohan Sarovar, and Kevin

- Young, “A taxonomy of small markovian errors,” *PRX Quantum* **3**, 020335 (2022).
- [9] Robin Blume-Kohout and Kevin Young, “A volumetric framework for quantum computer benchmarks,” *Quantum* **4**, 362 (2020).
- [10] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta, “Validating quantum computers using randomized model circuits,” *Phys. Rev. A* **100**, 032328 (2019).
- [11] Mohan Sarovar, Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout, “Detecting crosstalk errors in quantum information processors,” *Quantum* **4**, 321 (2020).
- [12] S Mavadia, C L Edmunds, C Hempel, H Ball, F Roy, T M Stace, and M J Biercuk, “Experimental quantum verification in the presence of temporally correlated noise,” *npj Quantum Information* **4**, 7 (2018).
- [13] Timothy Proctor, Melissa Revelle, Erik Nielsen, Kenneth Rudinger, Daniel Lobser, Peter Maunz, Robin Blume-Kohout, and Kevin Young, “Detecting and tracking drift in quantum information processors,” *Nat. Commun.* **11**, 5396 (2020).
- [14] Jonas Bylander, Simon Gustavsson, Fei Yan, Fumiki Yoshihara, Khalil Harrabi, George Fitch, David G Cory, Yasunobu Nakamura, Jaw-Shen Tsai, and William D Oliver, “Noise spectroscopy through dynamical decoupling with a superconducting flux qubit,” *Nature Phys* **7**, 565 (2011).
- [15] Mateusz T Mądzik, Serwan Asaad, Akram Youssry, Benjamin Joecker, Kenneth M Rudinger, Erik Nielsen, Kevin C Young, Timothy J Proctor, Andrew D Baczewski, Arne Laucht, Vivien Schmitt, Fay E Hudson, Kohei M Itoh, Alexander M Jakob, Brett C Johnson, David N Jamieson, Andrew S Dzurak, Christopher Ferrie, Robin Blume-Kohout, and Andrea Morello, “Precision tomography of a three-qubit donor quantum processor in silicon,” *Nature* **601**, 348 (2022).
- [16] Robin Harper, Steven T Flammia, and Joel J Wallman, “Efficient learning of quantum noise,” *Nat. Phys.* **16**, 1184 (2020).
- [17] Alexander Erhard, Joel J Wallman, Lukas Postler, Michael Meth, Roman Stricker, Esteban A Martinez, Philipp Schindler, Thomas Monz, Joseph Emerson, and Rainer Blatt, “Characterizing large-scale quantum computers via cycle benchmarking,” *Nat. Commun.* **10**, 5347 (2019).
- [18] Daniel C Murphy and Kenneth R Brown, “Controlling error orientation to improve quantum algorithm success rates,” *Phys. Rev. A* **99**, 032318 (2019).
- [19] Robin Blume-Kohout, Kenneth Rudinger, Erik Nielsen, Timothy Proctor, and Kevin Young, “Wildcard error: Quantifying unmodeled errors in quantum processors,” [arXiv:2012.12231](https://arxiv.org/abs/2012.12231).
- [20] Mingxia Huo and Ying Li, “Self-consistent tomography of temporally correlated errors,” [arXiv:1811.02734](https://arxiv.org/abs/1811.02734).
- [21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, “Multilayer feedforward networks are universal approximators,” *Neural Networks* **2**, 359.
- [22] Mario Krenn, Jonas Landgraf, Thomas Foesel, and Florian Marquardt, “Artificial intelligence and machine learning for quantum technologies,” *Phys. Rev. A* **107**, 010101 (2023).
- [23] Valentin Gebhart, Raffaele Santagati, Antonio Gentile, Erik Gauger, David Craig, Natalia Ares, Leonardo Banchi, Florian Marquardt, Luca Pezzè, and Christian Bonato, “Learning quantum systems,” *Nat. Rev. Phys.* **5**, 141 (2023).
- [24] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta, “Reinforcement learning in different phases of quantum control,” *Phys. Rev. X* **8**, 031086 (2018).
- [25] Chunlin Chen, Daoyi Dong, Han-Xiong Li, Jian Chu, and Tzyh-Jong Tarn, “Fidelity-based probabilistic q-learning for control of quantum systems,” *IEEE Transactions on Neural Networks and Learning Systems* **25**, 920 (2014).
- [26] Thomas Fösel, Petru Tighineanu, Talitha Weiss, and Florian Marquardt, “Reinforcement learning with neural networks for quantum feedback,” *Phys. Rev. X* **8**, 031084 (2018).
- [27] Murphy Niu, Sergio Boixo, Vadim Smelyanskiy, and Hartmut Neven, “Universal quantum control through deep reinforcement learning,” *npj Quantum Inf* **5** (2019).
- [28] Lorenzo Moro, Matteo Paris, Marcello Restelli, and Enrico Prati, “Quantum compiling by deep reinforcement learning,” *Commun Phys* **4** (2021).
- [29] Juan Carrasquilla, Giacomo Torlai, Roger Melko, and Leandro Aolita, “Reconstruction quantum states with generative models,” *Nat Mach Intell* **1**, 155 (2021).
- [30] Tobias Schmale, Moritz Reh, and Martin Gärtner, “Efficient quantum state tomography with convolutional neural networks,” *npj Quantum Inf* **8**, 115 (2022).
- [31] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo, “Neural-network quantum state tomography,” *Nat Phys* **14**, 447 (2018).
- [32] Giuseppe Carleo and Matthias Troyer, “Solving the quantum many-body problem with artificial neural networks,” *Science* **355**, 602 (2017).
- [33] Keiron O’Shea and Ryan Nash, “An introduction to convolutional neural networks,” [arXiv:1511.08458](https://arxiv.org/abs/1511.08458).
- [34] Ronan Collobert and Jason Weston, “A unified architecture for natural language processing: Deep neural networks with multi-task learning,” in *Proceedings of the 25th International Conference on Machine Learning*, ICML ’08 (Association for Computing Machinery, New York, NY, USA, 2008) p. 160.
- [35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning* (MIT Press, 2016) <http://www.deeplearningbook.org>.
- [36] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang, “Physics-informed machine learning,” *Nat Rev Phys* **3**, 422 (2021).
- [37] Daniel Hothem, Kevin Young, Tommie Catanach, and Timothy Proctor, “Predicting circuit success rates with artificial neural networks,” (2021), presented at the APS March Meeting.
- [38] Hanrui Wang, Pengyu Liu, Jinglei Cheng, Zhiding Liang, Jiaqi Gu, Zirui Li, Yongshan Ding, Weiwan Jiang, Yiyu Shi, Xuehai Qian, David Pan, Frederic Chong, and Song Han, “Quest: Graph transformer for quantum circuit reliability estimation,” in *Proceedings of the 39th International Conference on Computer-Aided Design* (ACM Press, New York, New York, 2022).
- [39] Avi Vadali, Rutuja Kshirsagar, Prasanth Shyamsundar, and Gabriel Perdue, “Quantum circuit fidelity estimation using machine learning,” [arXiv:2212.00677](https://arxiv.org/abs/2212.00677).
- [40] Norhan Elsayed Amer, Walid Gomaa, Keiji Kimura, Kazunori Ueda, and Ahmed El-Mahdy, “On the learnability of quantum state fidelity,” *EPJ Quantum Technology* **9**, 31 (2022).
- [41] Note, however, that an individual circuit layer will necessarily be compiled into a processor’s native gates. This compilation can also be arbitrarily complex in general, but its complexity is limited in practice if we choose a layer set that closely corresponds to the native circuit layers of the processor in question.
- [42] Dorit Aharonov, Alexei Kitaev, and Noam Nisan, “Quantum circuits with mixed states,” in *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC ’98* (ACM Press, New York, New York, USA, 1998) p. 20.

- [43] Michael A Nielsen, “A simple formula for the average gate fidelity of a quantum dynamical operation,” *Phys. Lett. A* **303**, 249 (2002).
- [44] Process fidelity comes in two variants: average gate fidelity and entanglement fidelity, that are linearly related by a dimensionality factor [43], and herein we use entanglement fidelity.
- [45] Timothy Proctor, Stefan Seritan, Erik Nielsen, Kenneth Rudinger, Kevin Young, Robin Blume-Kohout, and Mohan Sarovar, “Establishing trust in quantum computations,” (), [arXiv:2204.07568](https://arxiv.org/abs/2204.07568).
- [46] All standard choices for ϵ with SICO circuits are, when applied only to definite outcome circuits, equivalent to the probability of the correct bit string. This includes the TVD and the Hellinger fidelity.
- [47] To see this, note that the learning problem is infeasible if we used an encoding that encrypts the circuits.
- [48] When we applied our techniques to data from cloud-access quantum computers we use a slightly different encoding, which can be found in the supplemental data and code.
- [49] Alex LeNail, “Nn-svg: Publication-ready neural network architecture schematics,” *Journal of Open Source Software* **4**, 747 (2019).
- [50] Reducing the size of the image reduces the size of the input to the first dense layer, therefore reducing the number of weights that must be learned for that layer.
- [51] Diederik Kingma and Jimmy Ba, “Adam: A method for stochastic optimization,” in *Proceedings of the 3rd International Conference for Learning Representations*, edited by Yoshua Bengio and Yann LeCun (Microtome, 2015).
- [52] J. Moćus, *Bayesian Approach to Global Optimization* (Springer Dordrecht, Berlin, Germany, 1989).
- [53] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *NIPS’12: Proceedings of the 25th International Conference on Neural Information Processing Systems*, Vol. 2, edited by F. Pereira, C.J.C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates Inc., Red Hook, NY, 2012).
- [54] Like most widely-used parameterized error models, ERMs can be constructed by placing restrictions on the maximal Markovian model (see Ref. [7] or Appendix B), which models a processor’s operations using general n -qubit CPTP maps.
- [55] Note that “depth” here and throughout means the number of circuit layers in the circuit, rather than the “benchmark depth” of the randomized mirror circuits (as defined in Ref. [3]), which is an alternative notion of the depth of a randomized mirror circuit that is adopted when using these circuits to estimate average layer error rates.
- [56] The exact value of $s(c)$ can be calculated when simulating an error model, but in experiments $s(c)$ can only ever be estimated from a finite number of runs of a circuit—each of which either returns the “success” bit string or does not.
- [57] For each of the 5 (alt. 11) instances the datasets are nested, e.g., the 500 circuits are sampled from the 1000 circuits, and we use the same training, validation, and testing partition.
- [58] These 28 parameters correspond to a readout error rate for each of the 5 qubits, $15 = 3 \times 5$ single-qubit gate error rates, and $8 = 4 \times 2$ CNOT gate error rates (there are four edges in the connectivity graph, and CNOTs can be applied in each direction on the edge).
- [59] There are 42 edges in the graph, there is a CNOT gate associated with each direction on each edge, and each CNOT gate is associated with 6 error rates. There are 3×49 different single qubit gates, each associated with 3 errors rates, and there are 49 readout error rates. This results in $42 \times 2 \times 6 + 4 \times 49 = 1498$ parameters in the model.
- [60] Daniel Hothem, Tommie Catanach, Kevin Young, and Timothy Proctor, <https://doi.org/10.5281/zenodo.7829489>, published: 2023-04-12.
- [61] Anthony M Polloreno, Arnaud Carignan-Dugas, Jordan Hines, Robin Blume-Kohout, Kevin Young, and Timothy Proctor, “A theory of direct randomized benchmarking,” [arXiv:2302.13853](https://arxiv.org/abs/2302.13853).
- [62] For our choice for the values of the parameters in Eq. (26), $\epsilon_{GP,P}(g, i, l)$ is approximately linear in l over the range of depths in our circuit set (d up to 272), with $\epsilon_{GP,P}(g, i, 272) \approx 4\epsilon_{GP,P}(g, i, 0)$.
- [63] Zheyang Shen, Jiashou Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui, “Towards out-of-distribution generalization: A survey,” [arXiv:2108.13624](https://arxiv.org/abs/2108.13624).
- [64] A validation set of narrow circuits was used.
- [65] Akel Hashim, Stefan Seritan, Timothy Proctor, Kenneth Rudinger, Noah Goss, Ravi K Naik, John Mark Kreikebaum, David I Santiago, and Irfan Siddiqi, “Benchmarking verified logic operations for fault tolerance,” [arXiv:2207.08786](https://arxiv.org/abs/2207.08786).
- [66] Joel J Wallman and Joseph Emerson, “Noise tailoring for scalable quantum computation via randomized compiling,” *Phys. Rev. A* **94**, 052325 (2016).
- [67] Emanuel Knill, “Quantum computing with realistically noisy devices,” *Nature* **434**, 39 (2005).
- [68] Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout, <https://doi.org/10.5281/zenodo.5197499> (), accessed: 2023-04-12.
- [69] Experiment 2 consisted of running the same benchmark on seven IBM Q processors and one Rigetti processor. Herein we do not present results for the Rigetti processor. This is because, at the time of those experiments, Rigetti did not provide up-to-date calibration data, so we cannot create an error rates model from that data.
- [70] We use the same method as described in Ref. [3] to turn IBM Q’s RB error rates into values for the parameters of an error rates model.
- [71] Erik Nielsen, Kenneth Rudinger, Timothy Proctor, Antonio Russo, Kevin Young, and Robin Blume-Kohout, “Probing quantum processor performance with pyGSTi,” *Quantum Sci. Technol.* **5**, 044002 (2020).
- [72] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” (2015), software available from tensorflow.org.
- [73] François Chollet *et al.*, “Keras,” <https://keras.io> (2015).
- [74] Ji Liu and Huiyang Zhou, “Reliability modeling of NISQ-era quantum computers,” in *2020 IEEE International Symposium on Workload Characterization (IISWC)* (2020) p. 94.

Appendix A: Previous Work

In this appendix, we contextualize our work by summarizing other efforts to model a quantum processor’s capability function using neural networks. First, we focus on attempts to model a circuit’s *probability of successful trial*, which can be thought of as a kind of fidelity estimation. Probability of successful trial (PST) is a generalization of a definite outcome circuit’s success probability to arbitrary circuits. For an arbitrary quantum circuit c , $\text{PST}(c)$ is defined as the success probability of the definite outcome circuit created by concatenating c with c^{-1} . PST is strongly correlated with state fidelity [38], although $\text{PST}(c)$ is not a reliable estimate of state fidelity (errors can echo away in Loschmidt echos). Two works have used neural networks to predict PST for arbitrary quantum circuits. Liu *et al.* [74] used shallow neural networks to predict the PST of both random and algorithmic circuits, while Wang *et al.* [38] used graph neural networks to perform the same task. Both of these works are complementary to ours, as Refs. [38, 74] use different neural network architectures. A consequence of this is that Refs. [38, 74] use circuit encodings that differ from our image-based encoding. Liu *et al.* represented each circuit as a tuple containing: (1) the width of the circuit; (2) the depth of the circuit; (3) the total number of each single-qubit and two-qubit gates in the circuit; (4) the number of measurements; and (5) a dictionary listing the target and control qubits for each two-qubit gate in the circuit. Wang *et al.* represented each circuit as a colored undirected graph, with colored vertices representing gates. Additional information, like one- and two-qubit gate error rates were also embedded in the graph.

To our knowledge, this paper is the first to use CNNs to predict circuit success probability, but other papers have used CNNs for state fidelity estimation. Amer *et al.* [40] used one-dimensional CNNs to predict the state fidelity between the output state of shallow (3 to 5 layers) 1, 3, and 5-qubit circuits run on IBM Q devices. Vadali *et al.* [39] similarly focused on shallow circuits, but instead chose to predict state fidelity using a 3-dimensional CNNs. They also analyzed wider circuits (up to 25 qubits), which limited their efforts to working with simulated local depolarization and crosstalk errors. In common with our work, Amer *et al.* and Vadali *et al.* use a modified version of one-hot encoding to encode circuits. However, the differences in the dimensions of the CNNs lead to different encoding schemes. Amer *et al.* flatten each circuit into a one-dimensional vector, while Vadali *et al.* maintain local connectivity information by assigning each gate a two-dimensional coordinate on a grid.

Scalability is a significant problem when using neural network to model state fidelity. Amer *et al.* avoid this problem by focusing on short, few-qubit circuits, although it is arguable that useful neural network models for a processor’s capability will need to be able to predict deep, many-qubit circuits. This allows Amer *et al.* to perform state tomography to gather their training and test data (state tomography on states produced by general n -qubit circuits is inefficient in n). Vadali *et al.* instead focus on Clifford and Clifford-reducible circuits (two classes of efficiently classically simulable circuits) simulated under

symmetric local depolarization noise with two-qubit gate ZZ-crosstalk. They then investigate how well their networks’ extend to generic circuits simulated under the same noise model. Our work differs from that of Amer *et al.* and Vadali *et al.* in several important ways. In this work we examine how robust our CNNs are (e.g., how well does a CNN model generalize to out-of-distribution circuits?) as well as how a CNN’s performance scales as a function of dataset size and quality. We also include error sensitivity information in the circuit encoding, we demonstrate the successful application of a neural network approach to modelling $s(c)$ with non-Markovian errors, and we show that CNNs fail to accurately model $s(c)$ in the presence of coherent noise.

Appendix B: Parameterized error models

Conventional approaches to modelling a capability function $s(c)$ do so indirectly using a parameterized error model. In this appendix we review the maximal Markovian error model. The most widely-used models for a quantum computer’s errors—including the ERMs used in this work—can be constructed by placing constraints on the maximal Markovian error model. In the maximal Markovian model each imperfect n -qubit layer of gates $l \in \mathbb{L}_n$ is modelled by a fixed but unknown CPTP map on n -qubits, a state preparation by a n -qubit density matrix ρ , and a measurement by an n -qubit POVM M . Each n -qubit CPTP map has $4^n(4^n - 1)$ parameters. So, the maximal Markovian model contains

$$N_p(n) = 4^n(4^n - 1)N_L + N_{\text{SPAM}} = O(16^n N_L) \quad (\text{B1})$$

parameters where N_L is the number of possible circuit layers and N_{SPAM} is the number of parameters in ρ and M .

In principle, the parameters of the maximal Markovian model can be learned from GST (up to a gauge freedom [7]), and the elements of the learned model can be composed to compute the model’s prediction for $s(c)$ for any circuit c [7]. But it is infeasible to estimate all $O(16^n N_L)$ of this model’s parameters when $n \gg 1$. Tractable models with a polynomial number of parameters can be obtained by placing restrictions on the parameters of this model. This is the basis for the existing and nascent scalable models for a quantum computer’s errors, including “low-weight” error models [8], restricted Pauli stochastic models [16], crosstalk-free models [11], and the ERMs we use in the main text.

Appendix C: The motivation for success probability learning

In this appendix we briefly expand on our explanation for why we focus on predicting the success probabilities of mirror circuits in this work. Methods that can accurately predict the success probabilities of mirror circuits are of little direct utility—no useful algorithms are mirror circuits, most quantum circuits are not definite outcome circuits, and we cannot *a priori* assume that a neural network trained on mirror circuits will generalize to other families of circuit. We have

chosen to consider the problem of modelling $s(c)$ for definite outcome circuits because it is a convenient setting in which to explore capability learning methods for the following two reasons. First, training data is easy to gather: estimating $s(c)$ with precision δ for a circuit c simply requires sampling from that circuit’s probability distribution (either in simulation, or on a real system) $O(1/\delta)$ times. In contrast, although mirror circuit fidelity estimation (MCFE) is efficient in n , MCFE requires running many circuits to estimate $s_F(c)$ [45]. Second, the problem of predicting circuit success probabilities retains many of the aspects of fidelity learning, and we conjecture that a neural network method that can accurately model $s(c)$ when trained on circuit success probabilities will, with only minor adaptations, be able to accurately model $s_F(c)$ when trained on circuit process fidelities. One reason for this conjecture is that MCFE estimates a circuit c ’s process fidelity by simple data processing on estimations of the success probabilities from a set of mirror circuits based on c .

Appendix D: CNNs can approximate local stochastic Pauli error models

In this appendix we provide CNN filters that approximate a local stochastic Pauli errors model, under the assumption of no parameterized gates. This is a proof that such a network exists. We are not claiming that a network with this structure and these weights is learned when we train a CNN on data from a local Pauli stochastic error model (and indeed it is not). For a stochastic Pauli errors model,

$$s(c) \approx \prod I(c)_{ijh_P}(1 - \epsilon_P(c_{ij})). \quad (\text{D1})$$

This approximate formula for $s(c)$ accounts for bias in the errors (i.e., that a particular Pauli operator does not change the state of the qubits if it is applied to a state that is an eigenstate of that Pauli operator), by making use of the error sensitivity channels. The approximation used here is that two Pauli errors, that occur at different circuit locations, never cancel, which is a very good approximation in random circuits when $n \gg 1$ [61].

To demonstrate that we can approximately reproduce Eq. (D1) using a CNN we will specify a convolution filter for every possible gate and qubit pair and every possible P . Assume that $I(c)$ has shape $w \times d \times n_{\text{channels}}$. For a gate G (encoded by a 1 in channel $\text{ch}(G)$) qubit i and Pauli P we define the following filter K , which we use with a bias of -1 : K is shape $w \times 1 \times n_{\text{channels}}$ and it contains zeros everywhere except on qubit i . For that qubit, it contains a 1 in the error sensitivity channel for P and $\epsilon_P(G, i)$ in the channel for the gate G . Consider the single convolutional layer that is a collection of these kernels. This produces an image whereby the elements are all zero except for $d \times w$ elements which are the $\epsilon_P(c_{ij})$. We can then approximate $s(c)$ simply by the sum of all the elements of this tensor. To obtain the (better) approximation to $s(c)$ stated in Eq. (D1) we need to take the product of $1 - \epsilon_P(c_{ij})$ —and a dense network can be trained to approximate the product of one minus each of its input.

Appendix E: Markovian error models and simulations

This appendix contains additional details on how we constructed each of the three simulated Markovian error models used in this paper. In order, they are: (i) the 5-qubit Markovian local Pauli stochastic error model from Section IV; (ii) the 5-qubit Markovian local Hamiltonian error model used in Section VI B; and (iii) the 49-qubit base Markovian local Pauli stochastic error model from Section V. We first explain how the 5-qubit Markovian local Pauli stochastic error model was constructed, before enumerating the alternations made to construct the Markovian local Hamiltonian model. We then conclude with a brief description of the 49-qubit base Markovian local Pauli stochastic error model.

The 5-qubit local Pauli stochastic error model was specified using the error generator formalism of Ref. [8]. It consists of operation-dependent errors randomly sampled according to a two-step process (i.e., there are independently sampled error rates for each gate and qubit[s] pair). First, an error rate was selected for each type of one- or two-qubit gate by uniformly sampling a value in $[0, 1]$ and multiplying it by a pre-determined maximum error rate. Then a single one-qubit (resp. two-qubit) stochastic error generator was uniformly sampled for each one-qubit (resp. two-qubit) gate. Each randomly sampled error generator was then assigned its gate’s error rate. Maximum one- and two-qubit gate error rates of .25% and 1% were chosen for the Pauli stochastic model. The exact error generators and rates are available in the Supplementary Materials.

The 5-qubit local Hamiltonian error model (see Section VI B) was also specified using the error generator formalism. Operation-dependent Hamiltonian error generators were uniformly sampled. Each randomly sampled error generator was assigned an error rate of 5%. The sampled error generators are available in the Supplementary Materials.

The 49-qubit base Markovian local Pauli stochastic error model (see Section V) was specified by directly sampling the rates of Pauli X , Y and Z error rates (this differs slightly from the parameters in the error generator formalism) for each gate and each qubit on which that gate acts. Unlike for the 5-qubit local Pauli stochastic error model, we did not use maximally biased errors, so each single-qubit gate and qubit pair is assigned three error rates (and each two-qubit gate and qubits pair is assigned six error rates). These error rates were uniformly sampled from $[0, .0001]$. Readout error rates of .0001 were also assigned to each qubit.

Appendix F: Convolutional filters for Double Trouble

In this appendix we state a set of four convolutional filters that enable the identification of all instances of sequential CNOT gates in a circuit. Consider a circuit c and assume that there is a CNOT gate in layer j that acts on qubit i (and some other qubit i'). Then, in our tensor encoding $I(c)$ of the circuit, $I(c)_{ijk} = \pm 1$ where k is one of the four channels used to encode CNOT gates. Therefore, there is also a CNOT gate on

qubit i in the layer before j (i.e., $j - 1$) if and only

$$\sum_{k \in \mathbb{C}_{\text{cnot}}} |I(c)_{i(j-1)k}| + |I(c)_{ijk}| = 2, \quad (\text{F1})$$

where the summation is over the four CNOT channels. There is a set of four $1 \times 2 \times n_{\text{channels}}$ convolutional kernels K and biases b whereby one of these four convolutional filters outputs a non-zero pixel if and only if this criteria is satisfied. These four filters correspond to $K_{1,1,k} = \pm 1/2$ and $K_{1,2,k} = \pm 1/2$ if k is a CNOT channel, with $K_{1,1,k} = K_{1,2,k} = 0$ otherwise, all with a bias of $b = -1/2$. The output $p(i, j)$ when a $1 \times 2 \times n_{\text{channels}}$ kernel is applied at location (i, j) is

$$p(i, j) = \text{ReLU} \left(\sum_{j'=0}^1 \sum_k I(c)_{i(j-j')k} K_{i(j-j')k} - b \right), \quad (\text{F2})$$

where ReLU is the ReLU activation function. Therefore if, e.g., $K_{1,1,k} = K_{1,2,k} = 1/2$ then $p(i, j) = 1/2$ if $\sum_{k \in \mathbb{C}_{\text{cnot}}} [I(c)_{i(j-1)k} + I(c)_{ijk}] = 2$ and otherwise $p(i, j) = 0$.

Appendix G: Additional Experimental Data

In the main text we presented plots of the prediction error for three of the seven IBM Q processors (see Fig. 9). In Figs. 11 and 12 of this appendix we present equivalent plots for the remaining four processors.

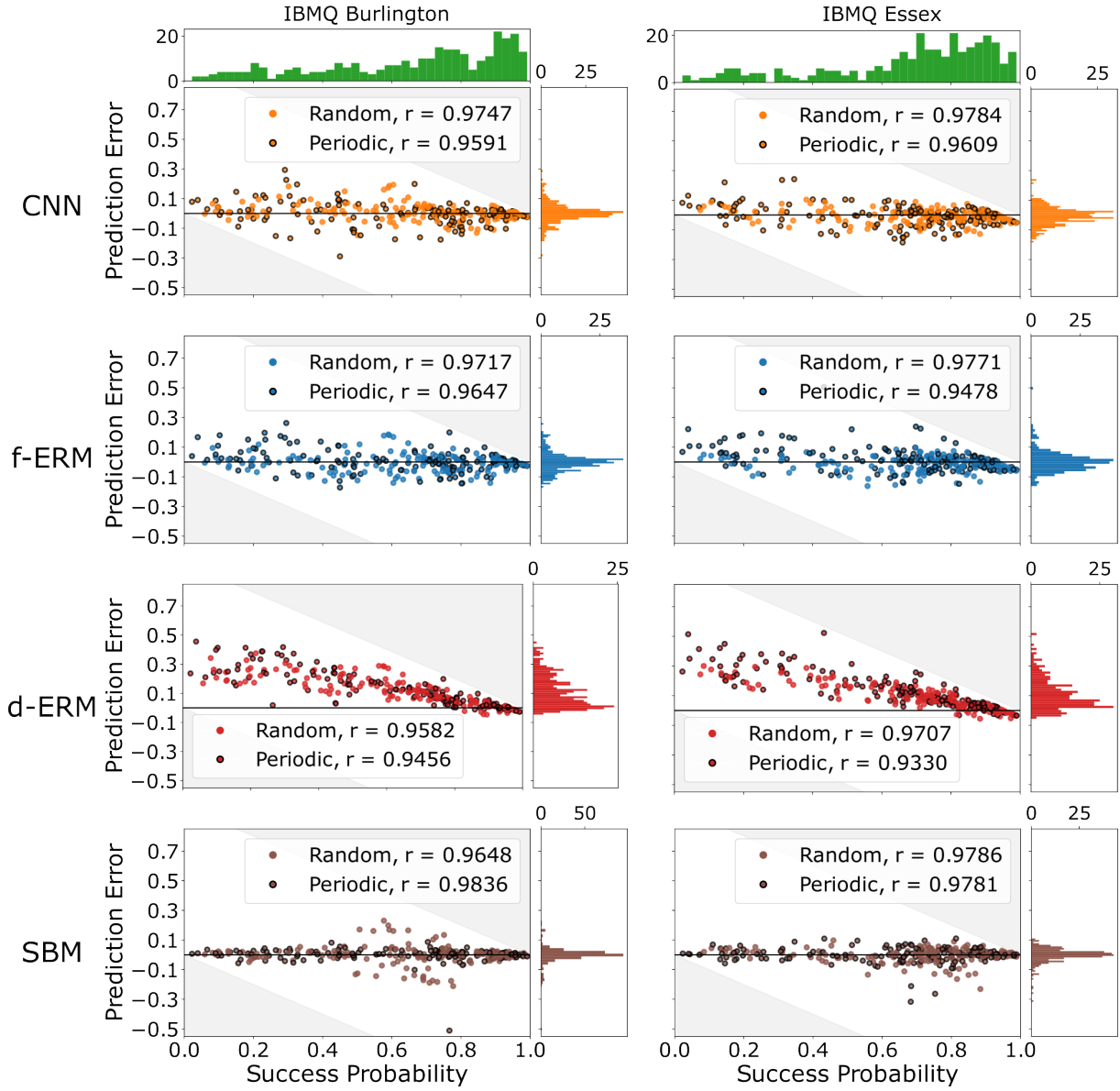


FIG. 11. **Predicting the capabilities of cloud-access IBM Q processors (continued).** Plots of the prediction errors for two of the four processors not shown in Fig. 9 of the main text. See the caption of Fig. 9 for details.

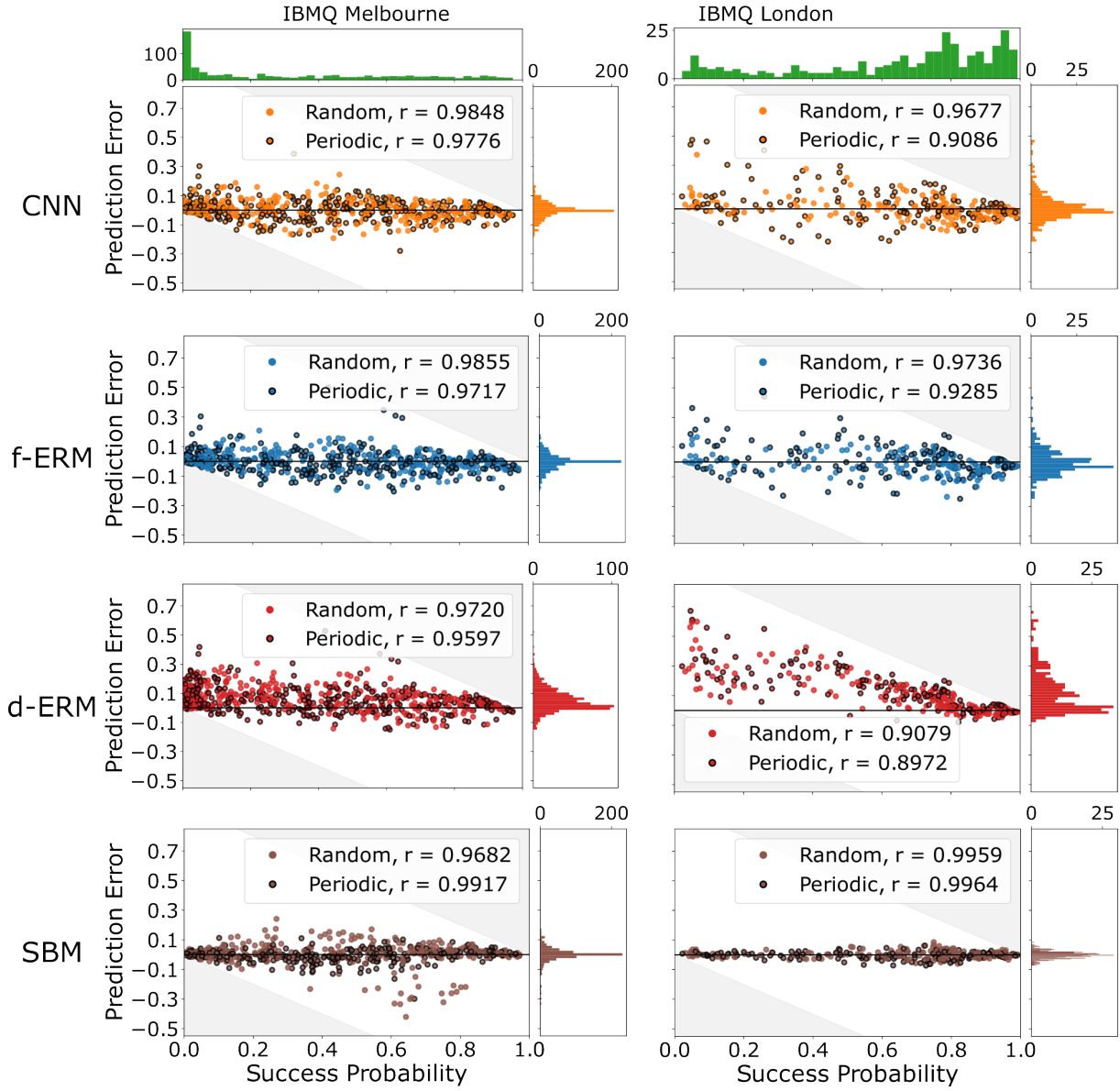


FIG. 12. **Predicting the capabilities of cloud-access IBM Q processors (continued).** Plots of the prediction errors for two of the four processors not shown in Fig. 9 of the main text. See the caption of Fig. 9 for details.