Parallel Programming of an Arbitrary Feedforward Photonic Network

Sunil Pai, *Member, IEEE*, Ian A. D. Williamson, *Member, IEEE*, Tyler W. Hughes, Momchil Minkov, Olav Solgaard, *Fellow, IEEE*, Shanhui Fan, *Fellow, IEEE*, and David A. B. Miller, *Fellow, IEEE*

***Abstract*—Reconfigurable photonic mesh networks of tunable beamsplitter nodes can linearly transform N-dimensional vectors representing input modal amplitudes of light for applications such as energy-efficient machine learning hardware, quantum information processing, and mode demultiplexing. Such photonic meshes are typically programmed and/or calibrated by tuning or characterizing each beam splitter one-by-one, which can be time-consuming and can limit scaling to larger meshes. Here we introduce a graph-topological approach that defines the general class of feedforward networks and identifies columns of non-interacting nodes that can be adjusted simultaneously. By virtue of this approach, we can calculate the necessary input vectors to program entire columns of nodes in parallel by simultaneously nullifying the power in one output of each node via optoelectronic feedback onto adjustable phase shifters or couplers. This parallel nullification approach is robust to fabrication errors, requiring no prior knowledge or calibration of node parameters and reducing programming time by a factor of order N to being proportional to the optical depth (number of node columns). As a demonstration, we simulate our programming protocol on a feedforward optical neural network model trained to classify handwritten digit images from the MNIST dataset with up to 98% validation accuracy.**

***Index Terms*—Programmable optical networks, graph theory, optical neural networks, neuromorphic computing, machine learning.**

I. Introduction

FEEDFORWARD networks of tunable beamsplitters, typically implemented as meshes of Mach-Zehnder interferometers (MZIs), can perform linear operations on sets or "vectors" of N optical inputs in N single-mode waveguides [1]–[3]. With advances in photonic integration, these networks have found a wide range of classical and quantum photonic applications where the mesh is configured to implement some specific linear transform or matrix. Some applications, like mode unscrambling in optical communications [4], favor a self-configuring approach [2], [5] in which the mesh sets itself up in real time to implement the matrix that undoes the mixing. Other applications, including photonic neural networks [6], universal linear quantum computing [3], and photon random walks [7], may need to have the mesh implement some specific matrix that is calculated externally. These applications promise fast and energy-efficient matrix multiplication or analog computation via the physical process of light propagating through repeated programmable tunable beamsplitter (MZI) units that we will now more generally refer to as "nodes." For such applications, we will demonstrate how these nodes connected in an arbitrary feedforward network (e.g., the simple grid network of Fig. 1) can be efficiently programmed in a robust manner to implement a desired matrix operator.

Though it is straightforward to implement a matrix in such a mesh by tuning phase shifts, any fixed fabrication of such settings is challenging for large meshes due to the precise settings required [8]. For example, these errors limit the classification accuracy of optical neural networks [9] and prevent scaling up the number of components in quantum linear optical circuits [3], [8]. We therefore prefer reconfigurable beamsplitter nodes in such networks and corresponding setup algorithms that can directly program desired matrices.

Such setup algorithms also let us fully calibrate the nodes; finding the voltage drive settings for any programmed node subsequently allows us to interpolate among those settings to implement desired matrices by applying appropriate voltages directly. Although each node can be individually calibrated in some specific architectures [3], [10], this approach is slow for large-scale meshes and must be repeated if components experience environmental drift. Here, we show an approach that can greatly reduce the time required for such setup and/or calibration processes and also generalizes to any feedforward mesh (i.e., where light propagates unidirectionally). By exploiting a graph-topological approach, we identify which nodes can be programmed simultaneously and provide the necessary parallelized algorithm to reduce the setup time by a factor of order N. One general setup approach that does not require prior knowledge of each node's parameters is based on what could be called "nullification." By specific choices of N input modes (amplitude and phase) sent into N input waveguides, corresponding nodes can be programmed by nullifying power at one of their outputs by

Manuscript received December 12, 2019; revised May 6, 2020; accepted May 21, 2020. Date of publication May 28, 2020; date of current version July 24, 2020. This work was supported in part by the Air Force Office of Scientific Research and in part by the specifically for the Center for Energy-Efficient 3D Neuromorphic Nanocomputing (CEE3N^2) and a MURI program, under Grants FA9550-18-1-0186 and FA9550-17-1-0002, respectively. *(Corresponding author: Sunil Pai.)*

Sunil Pai, Ian A. D. Williamson, Momchil Minkov, Olav Solgaard, Shanhui Fan, and David A. B. Miller are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: sunilpai@stanford.edu; ian.williamson@utexas.edu; mminkov@stanford.edu; solgaard@stanford.edu; shanhui@stanford.edu; dabm@stanford.edu).

Tyler W. Hughes is with the Department of Applied Physics, Stanford University, Stanford, CA 94305 USA (e-mail: twhughes@stanford.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTQE.2020.2997849

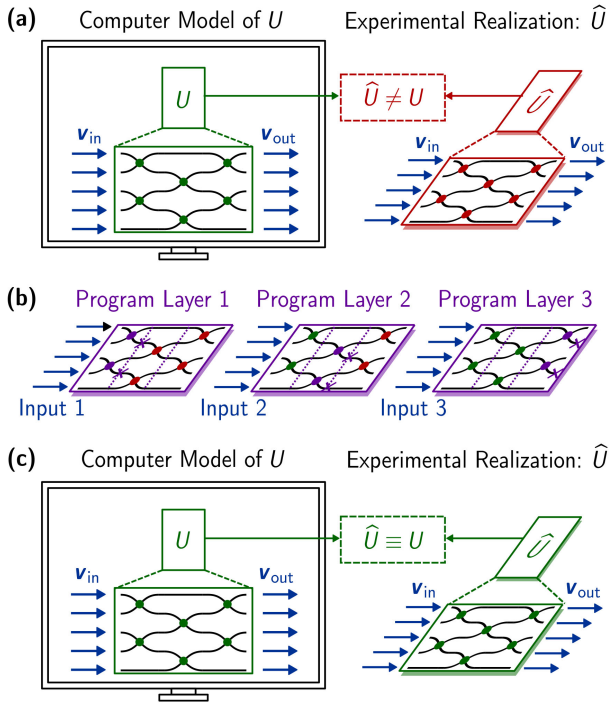

Fig. 1. (a) Prior to programming the mesh to desired physical operator $\widehat{U}$ (shown as unprogrammed in red), we have our corresponding calculated mesh model U (connectivity and tunable parameters) stored in the computer. (b) From U, we calculate a set of nullification vectors (the nullification set), and then send in corresponding physical mode vectors from this set sequentially (Input 1, Input 2 and Input 3). We tune the beamsplitter parameters of the corresponding column (denoted by purple dots) by nullifying power at all sampling photodetectors (denoted by purple crosses) in the column in parallel. (c) After this programming, we are guaranteed that our experimental realization and computer model match up to an output phase reference, i.e. $\widehat{U} \equiv U$ (denoted as programmed in green).

optoelectronic feedback control [2], [4], [11], [12]. For example, networks made from one or more diagonal sets of nodes (such as triangular networks) [2], [5] are programmed to implement arbitrary $N \times N$ unitary matrices using nullification. These techniques generally require that photodetectors or monitors be present in the mesh itself, which can have important advantages over global optimization and calibration approaches that rely on output detectors [13].

More generally, other feedforward networks, such as rectangular grids (Clements scheme [14]), may not support self-configuration, so some separate design calculation must be performed to calculate the desired parameters of each node. Nonetheless, with the knowledge of the desired node parameters, such networks can be progressively configured using the reversed local light interference method (RELLIM) nullification approach proposed in Ref. [11]. A key question is whether we can minimize the total time required to program or calibrate the network. The self-configuring algorithm for diagonal or triangular meshes and the RELLIM algorithm as originally conceived [11] give prescriptions for setting up the nodes sequentially (i.e., one-at-a-time) and thus require a number of steps equal to the number of nodes.

In this paper, we propose a framework that arranges any given feedforward network into columns of nodes that can be programmed simultaneously, with just one "nullification set" input vector for the entire column, rather than programming one node at a time with possibly different input vectors for each such node. The resulting "parallel nullification" or parallel RELLIM ("PRELLIM") protocol uses up to $(N/2)$-times fewer calibration steps and input vectors than RELLIM, where N is the number of input modes to the system. Our protocol ultimately enables efficient, robust, flexible, and scalable calibration (with time complexity of the number of columns in the device) of an *arbitrary* feedforward photonic mesh architecture. Example such architectures include triangular [1], [2] and rectangular [14] grid networks (capable of implementing arbitrary unitary matrices) and butterfly [8], [9] networks (capable of implementing any permutation or DFT unitary matrices).

We outline a typical scenario that benefits from parallel nullification in Fig. 1(a), where a model of an optical network stored in a digital computer (e.g. a CPU-trained optical neural network) must be programmed into a photonic circuit. The model consists of the network topology (node connection patterns) and tunable node parameters used to calculate the nullification set vectors for calibration. The parallel nullification procedure shown in Fig. 1(b), consists of calibrating the mesh one column at a time using the nullification set and tuning all nodes within each column in parallel until their bottom outputs are all nullified (i.e. transmit zero power). In Fig. 1(c), we show that after parallel nullification is applied to all columns, our device matches the computer model as accurately as physically possible.

The remainder of this paper is organized as follows. In Section II, we lay out the foundations of our graph-topological framework used to formally define a general feedforward photonic mesh. In Section III, we propose a parallel nullification protocol and demonstrate how our protocol can deploy machine learning models on optical neural networks in Section IV. We then more generally discuss robustness of parallel nullification to systematic errors in Section VI.

II. Feedforward Photonic Network

In this section, we introduce some required mathematical and graph-topological terminology and concepts for feedforward photonic networks. For any such network with N input and output waveguides, there is a linear device operator [15], [16] or matrix U relating the input and output waveguide amplitudes for monochromatic light at steady state. Because we are considering feedforward networks, by choice we consider only the forward amplitudes in all waveguides. With a set of N amplitudes in the input waveguides, represented by the input mode vector $\boldsymbol{v}_{\text{in}} \in \mathbb{C}^N$ (the nth element stores the amplitude and phase of the input mode in waveguide n), and a corresponding vector of output mode amplitudes $\boldsymbol{v}_{\text{out}}$, then $\boldsymbol{v}_{\text{out}} = U\boldsymbol{v}_{\text{in}}$ as shown in Fig. 1. Since we presume no backwards waves, U can be considered to be an $N \times N$ transmission matrix. Each individual tunable beamsplitter node (depicted by dots in Fig. 1) can similarly be described by a 2×2 transmission matrix T that describes how the light in its two input waveguide ports is distributed across its two output ports.

Based on the arguments of this section, we always arrive at a compact definition for U in terms of node columns that can

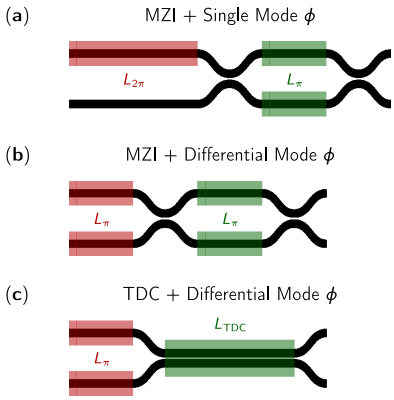

Fig. 2. Options for tunable beamsplitter node $T(\theta,\phi)$, where θ or S_{A} is controlled by the green block and ϕ or S_{P} is controlled by the red block. (a) the usual non-compact configuration; (b) the compact differential-mode MZI; (c) the compact TDC.

each be programmed in a single step as suggested by Fig. 1. For example, the triangular grid mesh [1] can be programmed in 2 N steps (as opposed to the typical $N(N-1)/2$ steps), the rectangular grid mesh [14] in N steps, and the butterfly mesh [8] in $\log N$ steps.

基于本节的论点，我们总是根据节点列得出一个紧凑的定义，每个节点列都可以在一个步骤中编程，如图1所示。例如，三角网格网格[1]可编程为2 N步（与典型的N（N 1）/2步相反），矩形网格网格网格[14]为N步，蝶形网格[8]为对数N步。

Each node of the feedforward network is a 2×2 tunable beamsplitter whose requirement is to be able to arbitrarily redistribute light. This is accomplished by concatenating a phase shifter ($S_{\mathrm{P}}(\phi)$) to a tunable coupler ($S_{\mathrm{A}}(\theta)$) resulting in the transmission matrix T:

$$T(\theta,\phi)=S_{\mathrm{A}}(\theta)S_{\mathrm{P}}(\phi)\equiv i\begin{bmatrix} e^{i\phi}\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \\ e^{i\phi}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\end{bmatrix}, \tag{1}$$

where $\theta\in[0,\pi]$ and $\phi\in[0,2\pi)$. By notation "$\equiv$," we note that there are many equivalent constructions of T which have mathematically different, but functionally equivalent, representations including tunable directional couplers (TDCs) and Mach-Zehnder interferometers (MZIs) as shown in Fig. 2.

The device programming is agnostic of the exact modulation schemes for S_{P} and S_{A}, as long as S_{P} corresponds to a controllable phase difference between the two input waveguides (so a differential phase) and S_{A} covers the full range of transmissivities (bar state or $\theta=\pi$ to cross state or $\theta=0$). The simplest general statement is that, for a 2×2 MZI node, we need two phase shifters, one of which must be on one waveguide arm inside the MZI to control the split ratio. The second phase shifter can be on any input or output waveguide [1] or on the other waveguide arm inside the MZI [2]. Though the latter is well-suited for self-configuration and setting up input vectors into the device, the protocol for parallel nullification may be less straightforward since this symmetric configuration does not have an "input phase equalizer" as do nodes of the form in Eq. 1 and thus cannot independently redistribute light without being cascaded with another node. For simplicity in programming the device, we primarily deal with configurations where the second phase shifter is on the *input* waveguide as in Eq. (1).

A commonly proposed alternative to the MZI for the split ratio modulation (S_{A} in Eq. (1)) is the tunable directional coupler (TDC), which can achieve any split ratio by simply tuning the coupling region of a directional coupler shown in Fig. 2(c). The transmissivity varies as $t=\cos^2(\kappa L_{\mathrm{TDC}})$ with $\theta=\kappa L_{\mathrm{TDC}}$ and κ the tunable coupling constant from coupled mode theory. Phase matched modes should always allow for the full range of transmissivities as long as the range of κL_{TDC} is large enough. One proposal that follows this scheme is the MEMS phase shifter of Ref. [17], which tunes the coupling gap electromechanically. Phase *mismatch* (tuning one waveguide instead of both waveguides in the coupling region with a dual drive directional coupler [18]) can also be used as a mechanism for controlling the split ratio provided the coupler can be set to a symmetric cross state configuration.

An alternative control scheme for phase shift operator S_{P} in Eq. (1) is a "differential mode" phase shifter scheme shown in Fig. 2(b, c). To make meshes more compact, it is functionally equivalent to have phase shifters in both the top and bottom waveguides that can reach a maximum of π phase shift. Tuning ϕ in the range of $[0,\pi)$ would then consist of tuning the top phase shifter from steady state, whereas tuning ϕ in the range of $[\pi,2\pi)$ would consist of tuning the bottom phase shifter from steady state. Of course, the tradeoff of this more compact scheme is increased complexity in the number of electrical contacts and the logic of the programming protocol.

In an N-port device, to represent the effect of one 2×2 element we may embed the 2×2 transmission matrix T along the diagonal of an otherwise $N\times N$ identity matrix. Formally, this would allow the resulting embedded operation $T_N^{[m]}$ to operate between modes in waveguides $2m-1$ and $2m$ (a Givens rotation) as follows:

$$T_N^{[m]} := \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} 2m-1 & & & 2m & & \end{array} \\ \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & T_{11} & T_{12} & \cdots & 0 \\ 0 & \cdots & T_{21} & T_{22} & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \end{array}\!\!\begin{array}{c} \\ \\ 2m-1 \\ 2m \\ \\ \\ \end{array}. \tag{2}$$

B. Graph-Topological Framework

We typically make photonic mesh networks in a grid-like manner [1], [2], [14] for efficiency and compactness in fabrication. This also can allow equal path lengths if we want to make the interference relatively insensitive to wavelength changes. Configuring the nodes of the network takes much longer than the time for light to propagate through the network. So, for the purposes of analyzing more general feedforward networks, we can consider monochromatic, continuous-wave light such that actual optical distances and geometrical arrangements are unimportant for calibration and programming of the network.

In contrast, the "network topology" (i.e., connectivity independent of lengths of inter-node links) *is* important for defining

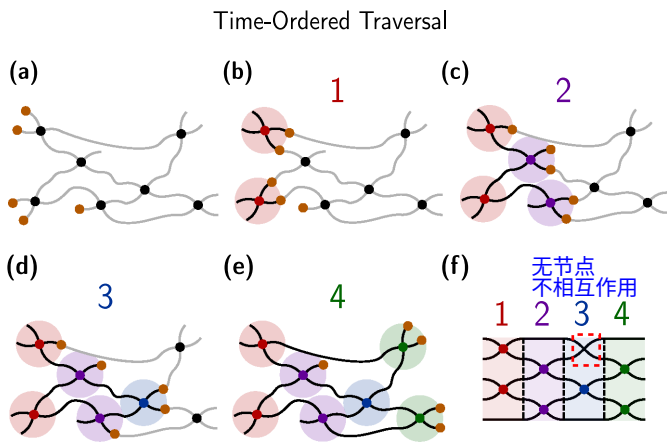

Fig. 3. From "left" to "right," we mathematically propagate a 5-mode vector of amplitudes (time-ordered waveguide traversal of orange dots) on an arbitrary $N = 5$ feedforward mesh (a) over panels (b)–(e) to generate the final compact, "layered" configuration of this mesh in panel (f). The numbers indicate the time step in which the nodes (shaded by a colored circle) are being traversed (i.e., simultaneously configured), and these numbers ultimately correspond to the columns (or time steps) to which nodes traversed at that step belong. The rule in the traversal is that two orange dots need to be at the input of a given node before that node can be traversed (i.e., the orange dots advance past the node) for that time step. (Note at the top of column 3 we have two waveguides crossing; there is no node at this crossing, and the waveguides cross without any interaction.)

遍历的规则是，两个橙色的点必须在给定节点的输入处，然后才能遍历该节点(即，橙色的点在该节点之前)。

valid feedforward configurations and the required inputs and node sequence by which we can progressively program or calibrate the network. We could mathematically view the inter-node links of the network as made of flexible (and stretchable) fibers between nodes that can move in space as long as the network topology is not changed. As such, we represent the notion of light moving "forward" or left-to-right (independent of the physical node locations in space) as moving along a given fiber away from network inputs or node outputs, which we define to be on the "left," and towards network outputs or node inputs, which we define to be on the "right." We will ultimately examine grouping nodes into "columns" to establish which nodes can be configured simultaneously or within the same propagation time step.

In such a graph-theoretic view, we can propose a sequential constructive definition that generates any arbitrary feedforward mesh network. As represented in the example in Fig. 3(a), we start with $N = 5$ input waveguides or fibers on the physical left (though this positioning is arbitrary). In constructing the network, we perform node addition by interfering any chosen pair of waveguides or fibers at the inputs of a node to generate outputs from that node. We assume light flows forward, so at any subsequent (further to the right) node addition, we interfere any two waveguides that exit earlier (further to the left) nodes. In this way, given a sufficient number of nodes and arbitrary choice of waveguides to couple, we can generate any feedforward mesh architecture. Furthermore, rather than adding a single node at a time in our construction, we can add at most $M := \lfloor N/2 \rfloor$ nodes at a time (where $\lfloor x \rfloor$ represents the largest integer less than or equal to x) to account for the largest number of waveguides that can be simultaneously coupled. The resulting architecture is shown in Fig. 3(f) where we maintain the feedforward property that propagation along the $N = 5$ waveguides always crosses the dotted vertical lines from left-to-right. This results in what we will define as the most "compact" representation, resembling more closely some of the commonly known rectangular and triangular feedforward architectures [1], [14].

It can be helpful to find this compact representation of Fig. 3(f) starting from the more arbitrary layout of Fig. 3(a) using a "topological sort" or homomorphic transformation. To do this, we propose a time-ordered "traversal" where we hypothetically 假想地 insert a mode at every input to the mesh and allow the resulting mode vector mathematically to propagate progressively (i.e., to traverse) through the device. At every step of the traversal, shown in Fig. 3(a)–(e), we allow the mode vector amplitudes to pass through and be transformed by the nodes in the network. Since the transmission matrix for each node in Eq. 1 interferes two inputs, two modes need to be input into a node to traverse that node (advance to the outputs of that node). Via this procedure, we equivalently construct sets of nodes (in circles of the same color of Fig. 3) that all connect only to previous (already traversed) nodes. We can then "compactify" the network into the numbered vertical columns as in Fig. 3(f) corresponding to the time steps (colored and indexed 1 to 4) in which nodes are traversed. Since the nodes in these columns are not connected to one another, we are free to configure these nodes in any order, including configuring them all simultaneously (i.e., in the same time step) assuming the preceding nodes are configured. Using this graph-topological approach, we can always generate the most compact device (in terms of number of node columns or "optical depth") possible for any feedforward architecture by lining up the nodes according to their time step as in Fig. 3(f).

Such a compactified version is the network with the lowest optical depth representation of the mesh, and nodes in each time step (or column) must be independent since there cannot be a valid path between them without contradicting our traversal algorithm; hence they can to be tuned in parallel. This column-wise or timestep labelling is a specific case of the well-known Dijkstra's algorithm [19] to find the maximum longest path (over all input source nodes) in a directed acyclic graph (feedforward mesh) using breadth-first (time-order) search through the mesh.

Each node belongs to exactly one column, but if our traversal algorithm finds multiple column assignments for that node (which would require revisiting that node), then there would be a cycle (or cyclic connection) in the graph. Therefore, by applying this algorithm, we can formally identify cyclic connections in any mesh architecture (as in the lattice meshes of Refs. [20] and [21]) that would disqualify such a mesh as a feedforward architecture. Though those meshes with cyclic connections can do many tasks not possible with feedforward architectures, the physical topology of nodes in such cyclic meshes is fundamentally different from the feedforward approaches we discuss here. As a result, such cyclic nodes must be tuned individually or using some global optimization approach [21] and may be susceptible to back-reflections during programming or calibration.

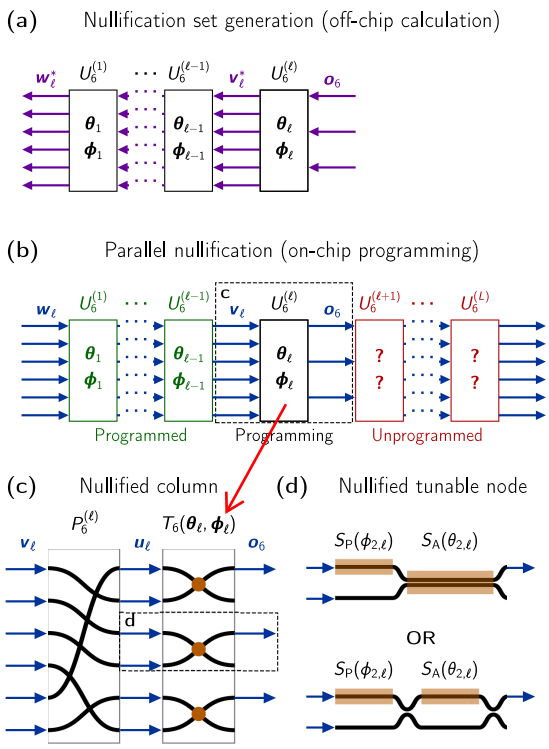

Fig. 4. We present the parallel nullification protocol for any feedforward photonic mesh network, here for an example with $N = 6$ waveguides, with blue arrows indicating physical implementation and purple arrows indicating offline calculations. (a) Nullification set generation for column ℓ in a mesh using RELLIM [11]. (b) Parallel nullification of column ℓ using nullification set vector $\boldsymbol{w}_\ell$ assumes columns $1, 2, \ldots \ell - 1$ have already been programmed. (c) Parallel nullification at column ℓ tunes $\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell$ in parallel via independent optoelectronic feedback optimizations until all bottom ports are nullified. (d) Nullification requires phase equalization (S_P) and split ratio modulation (S_A).

C. *Transmission Matrix Representation* 传输矩阵表示

Although our construction can generally be used to define any feedforward mesh, we need an equivalent transmission matrix that allows for straightforward simulation and calibration of such devices. Once we have arrived at the compact representation of Fig. 3(f), we can define each column entirely using the general definition of Eq. 3, later depicted in Fig. 4(c) and explicitly for example architectures in Fig. 5. For each new column, we reroute output waveguides of the previous column and connect them appropriately to simultaneously-acting nodes of the new column using the permutation matrix $P_N^{(\ell)}$ followed by the block-diagonal matrix $T_N(\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell)$ (the product of up to $M = \lfloor N/2 \rfloor$ nodes given by Eq. 2):

$$T_N(\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell) := \prod_{m=1}^{M} T_N^{[m]}(\theta_{m,\ell}, \phi_{m,\ell})$$
$$U_N^{(\ell)} := T_N(\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell) P_N^{(\ell)}, \tag{3}$$

where phase parameters are $\boldsymbol{\theta}_\ell = (\theta_{1,\ell}, \ldots \theta_{m,\ell}, \ldots \theta_{M,\ell})$ and $\boldsymbol{\phi}_\ell = (\phi_{1,\ell}, \ldots \phi_{m,\ell}, \ldots \phi_{M,\ell})$. Assuming $M_\ell \leq M$ nodes in the column, then $P_N^{(\ell)}$ is defined such that we can add synthetic bar state beamsplitter nodes for all $m > M_\ell$, i.e. $\theta_{m,\ell} = \phi_{m,\ell} = \pi$ for any waveguides that do not interact in that column. While all waveguides in a column can technically be interfered at nodes for even N, for odd N there will always be a single remaining waveguide that is not connected to a node in that column. However, this ultimately does not change the definition in Eq. (3) since we can always choose this to be the Nth waveguide by appropriate choice of $P_N^{(\ell)}$.

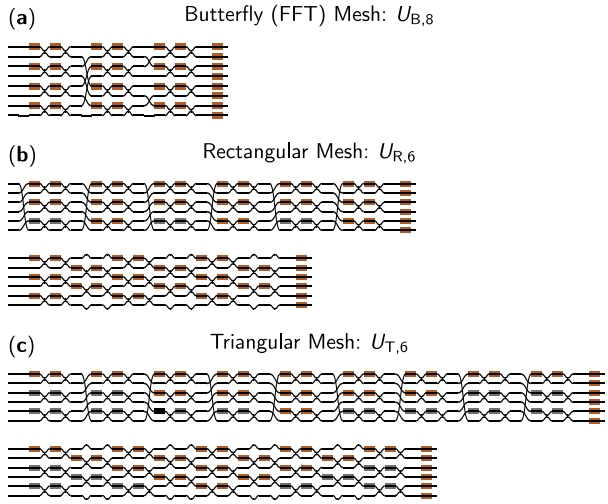

Fig. 5. For illustrative purposes, we show example mesh diagrams for (a) butterfly, (b) rectangular, and (c) triangular meshes. For (b)–(c), we show (top) how the device is modelled and (bottom) how it is physically implemented (the "grid" designs). In all mesh diagrams, orange phase shifters represent tunable phase shifters $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\gamma}$. Gray phase shifters represent bar state MZIs ($\theta = \phi = \pi$).

为了便于说明，我们展示了（a）蝶形网格、（b）矩形网格和（c）三角形网格的网格图示例。对于（b）（c），我们展示了（顶部）设备是如何建模的，（底部）是如何物理实现的（网格设计）。在所有的网格图中，橙色移相器代表可调移相器θ，ϕ，γ。灰色移相器代表棒态MZIs（$\theta=\phi=\pi$）。

To complete meshes for which we desire a fully arbitrary unitary transformation, we may need a further set of phase shifters on the output nodes of the mesh that effectively set the "reference phases" of the rows of the implemented matrix (represented by the diagonal unitary matrix $\Gamma_N(\boldsymbol{\gamma})$) [2]. (Such phase shifters can also be at the input if the external phase shift of each node is applied *after* the tunable coupler, effectively a mirror image of our current definition.) Furthermore, we include a final permutation P_N before or after those final phase shifters, allowing us to arbitrarily rearrange the rows of the matrix at the end.

Performing our transmission matrix construction for L columns, we arrive at an expression for an arbitrary feedforward mesh:

$$U_N(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\gamma}) := D_N \prod_{\ell=1}^{L} U_N^{(\ell)}$$
$$D_N := \Gamma_N(\boldsymbol{\gamma}) P_N, \tag{4}$$

where the matrices $\boldsymbol{\theta} := \{\theta_{m,\ell}\}, \boldsymbol{\phi} := \{\phi_{m,\ell}\}$ and vector $\boldsymbol{\gamma} := \{\gamma_n\}$ represent the full set of mesh parameters with ranges $\theta_{m,\ell} \in [0, \pi]$ and $\phi_{m,\ell}, \gamma_n \in [0, 2\pi)$. Any feedforward architecture or "graph topology" is entirely defined by these permutation matrices $\{P_N^{(\ell)}\}$ and P_N (representing the choices of waveguides to interfere), which we explicitly define for commonly proposed architectures (e.g., triangular, rectangular, butterfly) as in Fig. 5.

在图5中，我们为常用的架构(例如，三角形，矩形，蝶形)明确定义。

D. Feedforward Mesh Examples

We can apply the general definition of a feedforward mesh to commonly studied architectures shown in Fig. 5, which primarily involve appropriately defining $P_N^{(\ell)}$. Grid architectures include the rectangular or Clements mesh [14] and the triangular or Reck mesh [1], which are both universal unitary photonic mesh architectures. For a rectangular mesh, each of the $L = N$ columns has $M_\ell = M - 1 + \ell (\mathrm{mod}\ 2)$ and $P_N^{(\ell)}$ defined as an upward circular shift for odd ℓ and a downward circular shift for even ℓ. For a triangular mesh, each of the $L = 2\,N - 3$ columns has $M_\ell = \lceil \frac{\min(\ell, 2\,N-2-\ell)}{2} \rceil$ with the same $P_N^{(\ell)}$ as the rectangular mesh. The fact that the triangular mesh has the same $P_N^{(\ell)}$ as the rectangular mesh allows it to be "embeddable" within a rectangular mesh. This embedding may be necessary to (for example) equalize the loss for the final implemented operator as depicted in Fig. 5.

The butterfly architecture is an example of an alternative feedforward architecture that can be tuned using parallel nullification. Such architectures are designed to be compact, robust, and fault-tolerant alternatives to universal meshes [8], [9]. This means that implementing reconfigurable architectures of this form is potentially more scalable (to the feature size or number of inputs and outputs) and can be useful for machine learning approaches even though it cannot actually implement any arbitrary unitary operator. The operations that the butterfly mesh *can* implement, however, is the discrete Fourier transform (DFT) operator and (when concatenated with its mirror image to form the Benes network) any permutation operator.

Even meshes that are not explicitly feedforward-only [20], [22], but are meant for more general-purpose approaches, can in theory implement the rectangular or triangular grid architectures. With some additional characterization, it may be possible to program or calibrate such general architectures in the lab setting using our method.

III. Parallel Nullification

Now that we have defined a column-wise feedforward photonic mesh, we present parallel nullification as summarized in Fig. 4. The programming algorithm consists of an off-chip calculation of a sequence of inputs (or "nullification set") as in Fig. 4(a) followed by parallel nullification of columns from left-to-right of the mesh network as in Fig. 4(b).

In a realistic setting, we do not have direct access to the input of each column, but rather the inputs to the overall device. We therefore need the overall input vector assigned to each column ℓ that leads to the desired nullified output for that column. In both the RELLIM protocol [11] and our parallel nullification approach introduced here, the nullification set calculation works due to the reciprocity of feedforward meshes. In particular, we calculate a vector of complex amplitudes that would emerge from shining the desired nullified output backwards to the input from any column ℓ through a correctly programmed mesh. The nullification set for RELLIM results from mathematically propagating light backwards from the desired output of a *single* nullified node [11]. In contrast, the nullification set for parallel nullification results from mathematically propagating light backwards from the desired output of an entire *column* of nullified nodes, as in Fig. 4(a).

When it is time to actually program the columns on the physical mesh, we physically send the phase-conjugate (i.e., complex conjugate) of this result, which we can now call the nullification set vector $\boldsymbol{w}_\ell$, into the mesh inputs as in RELLIM [11]. As long as all the preceding mesh columns are set correctly, reciprocity ensures this vector can be used to correctly program the corresponding column ℓ to the desired mesh settings as shown in Fig. 4(b)–(d) by physically nullifying the appropriate outputs of that column.

In this section, we first formalize the nullification set calculation which is performed separately on a traditional computer. We then discuss the mathematics and physical procedures behind parallel nullification of each column and the overall programming algorithm that sets the physical parameters of the device (which we will refer to as $\boldsymbol{\alpha}, \boldsymbol{\beta}$) to the desired settings ($\boldsymbol{\theta}, \boldsymbol{\phi}$ respectively).

A. Nullification Set Calculation

For parallel nullification, there exist many valid calculations of nullification sets; we could choose to nullify either the top or the bottom port at any given node, and the value of the (non-zero) power at the non-nullified port does not matter. For definiteness, we will consider a simple valid target vector $\boldsymbol{o}_N = (1, 0, 1, 0, \cdots)$ or more formally:

$$\boldsymbol{o}_N := \sum_{m=1}^{M} \boldsymbol{e}_{2m-1}, \tag{5}$$

where $\boldsymbol{e}_{2m-1}$ represents the $(2\,m-1)$th standard Euclidean basis vector in $\mathbb{C}^N$, or equivalently, unit power in node output port $2m-1$.

We calculate the nullification set vector $\boldsymbol{w}_\ell$ for each column ℓ as depicted in Fig. 4(a):

$$\boldsymbol{w}_\ell^* := \prod_{\ell'=1}^{\ell} \left(U_N^{(\ell')} \right)^T \boldsymbol{o}_N. \tag{6}$$

Since we have already programmed columns $1 \to \ell - 1$, sending in $\boldsymbol{w}_\ell$ to the device yields correct values for $\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell$ after parallel nullification of column ℓ.

We can compute the entire nullification set $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots \boldsymbol{w}_L\}$ off-chip in $O(N \cdot L^2)$ time assuming all $\boldsymbol{\theta}, \boldsymbol{\phi}$ are known since each $\boldsymbol{w}_\ell$ results from reverse propagating $\boldsymbol{o}_N$ through ℓ columns. For example, we can calculate the nullification set in $O(N^3)$ time for the rectangular or triangular grid meshes. Code for calculating the nullification set (Eq. 6) for any feedforward mesh is provided in our Python software module neurophox [23], further discussed in the Appendix. [1] The nullification set results calculated for rectangular grid networks [14] are also shown in Fig. 6 and the Appendix.

[1] [Online]. Available: https://github.com/solgaardlab/neurophox.

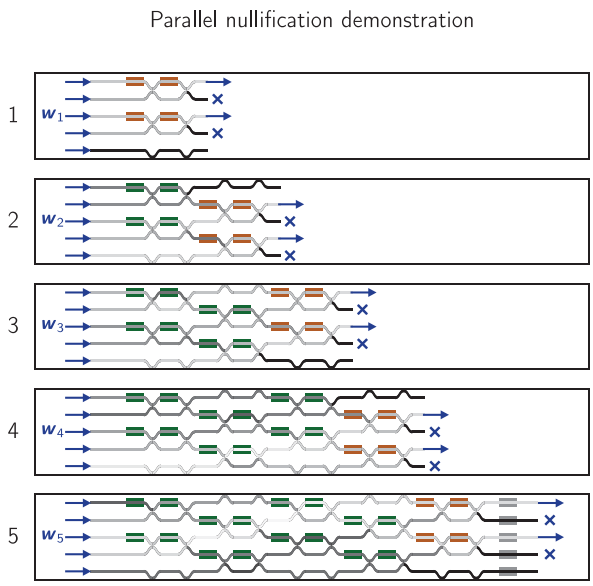

Fig. 6. Parallel nullification simulated on an $N = 5$ feedforward rectangular grid mesh in https://github.com/solgaardlab/neurophox. The relative field magnitudes are represented by grayscale values, the $\theta_{m,\ell}, \phi_{m,\ell}$ phase shifts are green and the γ_n are gray. At each step denoted on the left, we tune each column ℓ (depicted in orange) in parallel given input $\boldsymbol{w}_\ell$. This is accomplished progressively, with nullification indicated by the black-colored waveguides and blue crosses, from left to right until the full matrix is tuned.

N=5前馈矩形网格的并行零化模拟https://github.com/solgaardlab/neuropox。相对场的大小用灰度值表示，θm，φm，相移为绿色，γn为灰色。

B. Nullification

After calculating our nullification set off-chip, we program the physical device. Before programming a given node m in column ℓ, the settings $\alpha_{m,\ell}$ and $\beta_{m,\ell}$ will be different from desired settings $\theta_{m,\ell}$ and $\phi_{m,\ell}$ respectively. Given nullification set input $\boldsymbol{w}_\ell$ (presuming all preceding columns of the mesh are already set correctly), nullification can be achieved by two independent steps to nullify the bottom port (port 2 m) [2], [5] as proven explicitly in the Appendix:

1) Sweep $\beta_{m,\ell}$ (i.e., adjust the relative phase of the node inputs) until bottom port power is *minimized*.
2) Sweep $\alpha_{m,\ell}$ (i.e., adjust the node split ratio) until bottom port is *nullified*, as shown in Fig. 4(d).

We now give an explicit proof of this two-step nullification protocol [2]. We are given $\boldsymbol{x} = T(\alpha, \beta)\boldsymbol{u}$, i.e. an input vector $\boldsymbol{u} = (u_1, u_2)$ to a node T, yielding output vector $\boldsymbol{x} = (x_1, x_2)$. We want to show that minimizing bottom port power ($|x_2|^2$) with respect to β gives $\beta = \phi$ regardless of the value of α. Then, it is clear that if we sweep α to nullify power ($|x_2|^2 = 0$), we must have $\alpha = \theta$.

$$
\begin{aligned}
x_2 &= e^{i\beta}\cos\frac{\alpha}{2}u_1 - \sin\frac{\alpha}{2}u_2 \\
|x_2|^2 &= \cos^2\frac{\alpha}{2}|u_1|^2 + \sin^2\frac{\alpha}{2}|u_2|^2 \\
&\quad - 2\cos\frac{\alpha}{2}\sin\frac{\alpha}{2}|u_1||u_2|\mathrm{Re}(e^{i\arg(u_1)}e^{-i\arg(u_2)}e^{i\beta}) \\
\beta^{\mathrm{opt}} &:= \min_{\beta\in[0,2\pi)} |x_2|^2 = -\arg\left(\frac{u_1}{u_2}\right) = \phi, \qquad (7)
\end{aligned}
$$

where we allow any $\alpha \in [0, \pi]$, i.e. $\cos\frac{\alpha}{2}\sin\frac{\alpha}{2} \geq 0$. Now that we have optimized β, we optimize α to completely nullify $|x_2|^2$:

$$
\begin{aligned}
|x_2|^2 &= \left(\cos\frac{\alpha}{2}|u_1| - \sin\frac{\alpha}{2}|u_2|\right)^2 \stackrel{?}{=} 0 \\
\alpha^{\mathrm{opt}} &= 2\arctan\left|\frac{u_1}{u_2}\right| = \theta. \qquad (8)
\end{aligned}
$$

Now, let us extend this to the column case where we perform this procedure over many nodes in parallel. Given the permuted mode pair entering from the previous column $u_{2\,m-1,\ell}, u_{2\,m,\ell}$, this straightforward two-step optimization exactly adjusts settings $\alpha_{m,\ell}, \beta_{m,\ell}$ to be the desired $\theta_{m,\ell}, \phi_{m,\ell}$:

$$
\begin{aligned}
\alpha^{\mathrm{opt}}_{m,\ell} &= 2\arctan\left|\frac{u_{2m-1,\ell}}{u_{2\,m,\ell}}\right| = \theta_{m,\ell} \\
\beta^{\mathrm{opt}}_{m,\ell} &= -\arg\left(\frac{u_{2m-1,\ell}}{u_{2\,m,\ell}}\right) = \phi_{m,\ell}. \qquad (9)
\end{aligned}
$$

Parallel nullification of all nodes in column ℓ (i.e. $T_N(\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell)$) can be achieved because, as discussed in Section II, the choice of nodes assigned to column ℓ ensures all such optimizations are independent (i.e., do not influence each other). Nullification can be accomplished physically by sampling and measuring a small fraction of the power in the bottom output port. Nullification is then achieved through local feedback loops and is accomplished once zero power is measured. This nullification procedure has been experimentally demonstrated previously with noninvasive CLIPP detectors [4], [24], [25] that are effectively low-loss because they rely on light already absorbed in background loss processes in the waveguide.

C. Programming algorithm

The parallel nullification programming algorithm proceeds formally as in Algorithm 1, which we simulate in https://github.com/solgaardlab/neurophox [23]. If we are configuring the actual physical network, the entire FORWARDPROPAGATE method is a *physical process* in which we generate an actual vector of optical inputs $\boldsymbol{w}_\ell$, and propagate them through the mesh to physically generate the vector $\boldsymbol{v}_\ell$ at the node outputs in layer $\ell - 1$, which are permuted by $P_N^{(\ell)}$ to produce the vector we nullify in Eq. 9, $\boldsymbol{u}_\ell$. The vector $\boldsymbol{v}_\ell$ is the propagated fields of calculated inputs $\boldsymbol{w}_\ell$ to the output of column $\ell - 1$ as depicted in Fig. 4(b):

$$
\boldsymbol{v}_\ell := \prod_{\ell'=1}^{\ell-1} \widehat{U}_N^{(\ell-\ell')}\boldsymbol{w}_\ell, \qquad (10)
$$

where we use $\widehat{U}_N^{(\ell)}$ to mathematically represent the transmission matrices describing already-programmed physical columns of nodes (correctly set to column parameters $\boldsymbol{\theta}_\ell, \boldsymbol{\phi}_\ell$). The final (parallel) for-loop of Alg. 1 represents a physical parallel nullification of output powers using optoelectronic feedback on each column in order from $\ell = 1$ to L.

The inputs to Algorithm 1 are the desired settings for the feedforward mesh architecture which are used to first generate the nullification set $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots \boldsymbol{w}_L\}$. Algorithm 1 ultimately

Algorithm 1: Parallel Nullification.

function NULLIFICATIONVECTOR($\boldsymbol{\theta}$, $\boldsymbol{\phi}$, ℓ)
  $\boldsymbol{w}_\ell^* \leftarrow \boldsymbol{o}_N$
  for $\ell' \in [1, 2, \ldots, \ell]$ **do**
    $\boldsymbol{w}_\ell^* \leftarrow (U_N^{(\ell-\ell')})^T \boldsymbol{w}_\ell^*$ ▷ Eq. (6)
  end for
  return $\boldsymbol{w}_\ell$
end Function

function FORWARDPROPAGATE($\boldsymbol{w}_\ell$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, ℓ)
  $\boldsymbol{v}_\ell \leftarrow \boldsymbol{w}_\ell$
  for $\ell' \in [1, \ldots, \ell-1]$ **do**
    $\boldsymbol{v}_\ell \leftarrow \widehat{U}_N^{(\ell')} \boldsymbol{v}_\ell$ ▷ Eq. (10)
  end for
  return $\boldsymbol{v}_\ell$
end Function

procedure PARALLELNULLIFICATION ($\boldsymbol{\theta}$, $\boldsymbol{\phi}$)
  for $\ell \in [1, 2, \ldots, L]$ **do** ▷Off-chip
    $\boldsymbol{w}_\ell \leftarrow$ NULLIFICATIONVECTOR($\boldsymbol{\theta}$, $\boldsymbol{\phi}$, ℓ)
  end for
  for $\ell \in [1, 2, \ldots, L]$ **do** ▷On-chip
    $\boldsymbol{v}_\ell \leftarrow$ FORWARDPROPAGATE($\boldsymbol{w}_\ell$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, ℓ)
    $\boldsymbol{u}_\ell \leftarrow P_N^{(\ell)} \boldsymbol{v}_\ell$
    for $m \in [1, 2, \ldots, M_\ell]$ **do** ▷In parallel
      $\alpha_{m,\ell} \leftarrow \theta_{m,\ell}$
      $\beta_{m,\ell} \leftarrow \phi_{m,\ell}$ ▷ Eq. (9)
    end for
  end for
end procedure

results in setting the physical mesh parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$ to the desired $\boldsymbol{\theta}, \boldsymbol{\phi}$, for which a full testing suite is provided in neurophox [23]. As shown in Fig. 4(a), each nullification set vector $\boldsymbol{w}_\ell$ has the necessary information from past columns to tune all devices in column ℓ in parallel. We demonstrate the parallel nullification algorithm from Fig. 4(b) for a rectangular grid mesh network in the Appendix.

While we have ignored adjusting the final output phase shifts ($\boldsymbol{\gamma}$ from Section II) in Algorithm 1, we emphasize that such phase shifts merely serve to define an "output phase reference," which may be unimportant in some specific applications where only the power magnitude should be measured but is anyway straightforward to adjust after parallel nullification of all columns.

As shown in Fig. 6, a physical rectangular grid mesh (note the homomorphic transformation from Fig. 5(b)) can be progressively tuned to implement any unitary matrix by performing parallel nullification protocol.

IV. Optical Neural Networks

A particularly important application of parallel programming procedures is programming an optical neural network (ONN) device for an inference task. This neural network model, shown

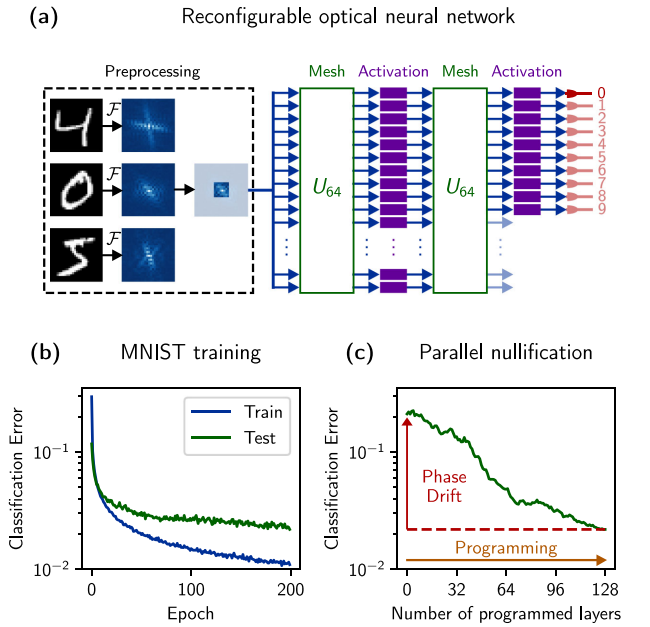

Fig. 7. (a) MNIST data is preprocessed and fed as input modes into the two-layer reconfigurable neural network of Ref. [26], and the light in the network is directed mostly towards the highlighted port corresponding to the correct label once trained. (b) Train and test accuracies for MNIST task for Adam gradient descent optimization over 200 epochs. (c) Parallel nullification corrects significant drift in phase shifter values (represented by Gaussian noise with standard deviation $\sigma_\theta = \sigma_\phi = 0.05$) and improves MNIST test accuracy.

solving a handwritten digit image classification task in Fig. 7 for the digit "0," consists simply of two alternating matrix (represented by feedforward meshes) and activation layers (represented elementwise nonlinear functions or activations). The model is first trained on a separate computer to solve a variety of non-trivial machine learning tasks, and as in Fig. 1, is then programmed onto a physical device.

This physical device may be realized as a hybrid ONN consisting of a matrix multiplication accelerator for each layer (as in Ref. [6], with nonlinearity layers implemented on the computer) or a fully integrated all-optical neural network (as in [26], with nonlinearities implemented on the device). We can perform the calibration in any fully-integrated ONN that allows us to effectively "turn off" the nonlinearities. This is possible in the scheme of Ref. [26] (which amplifies tapped power to modulate an MZI) by simply setting the MZI to a fixed setting. The parallel programming routine can then continue across the nonlinearity activation layer to the first column of the following layer as if they are straight waveguides.

To demonstrate the usefulness of accurate calibration, we train a unitary neural network architecture as in Fig. 7 similar to that of Ref. [26] with $N = 64$ meshes that can correctly classify 98% of the validation dataset for the MNIST handwritten digit classification task. We then show that if the device experiences drift in operation, parallel nullification can be used to correct the errors and restore the original set of phase shift parameters by correctly programming the linear layers as shown in Fig. 7(c).

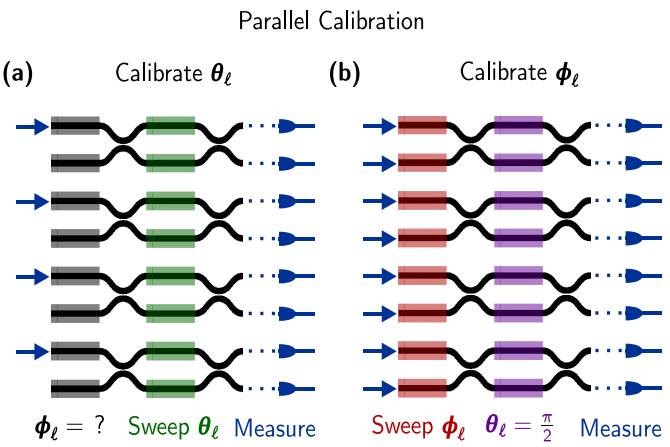

Fig. 8. Parallel calibration of any node column proceeds in two steps (where all blue arrows in the figure indicate a mode with the same amplitude). In (a), we send in the appropriate input (such that $\boldsymbol{u}_\ell = (1, 0, 1, 0, \ldots)$) to calibrate $\boldsymbol{\theta}_\ell$ regardless of how $\boldsymbol{\phi}_\ell$ is calibrated. In (b), we send in the appropriate input (such that $\boldsymbol{u}_\ell = (1, 1, 1, 1, \ldots)$) to calibrate $\boldsymbol{\phi}_\ell$ given the setting $\boldsymbol{\theta}_\ell = \pi/2$, which we know after calibrating $\boldsymbol{\theta}_\ell$.

V. Parallel Calibration

For systems that need to be re-calibrated often, it may be convenient to have a fast method to generate calibration curves for the voltage drive of each tunable element in the device [3], [7], [27]. Like parallel nullification, our "parallel calibration" protocol proceeds from left-to-right and can generate these transmission curves in parallel for all nodes within a given column.

In some grid meshes, all nodes can be systematically tuned to cross state (e.g., in the programmable nanophotonic processor (PNP) [7] or self-configuring triangular grid network [28]). In such schemes, power maximization at the appropriate output detectors can be used to remove the need for embedded detectors (i.e., use output detectors for the entire calibration procedure).

Other feedforward schemes, however, require embedding photodetectors for parallel nullification, which may as well be used for parallel calibration. To calibrate $\boldsymbol{\theta}_\ell$, we find the necessary input vector so that the bottom input port of all nodes in column ℓ are nullified (i.e., input power of $(1, 0)$ to all nodes in the column) while setting all previously calibrated layers to bar state. We then measure resulting output transmissivities as we sweep all $\boldsymbol{\theta}_\ell$ simultaneously from the minimum to maximum allowable voltage drive settings, as shown in Fig. 8(a). To calibrate $\boldsymbol{\phi}_\ell$, we input the uniform power in all mesh inputs such that all nodes in column ℓ have equal power in both input ports (i.e., input power of $(1, 1)$ to all nodes in the column). Assuming we have already calibrated $\boldsymbol{\theta}_\ell$, we set all tunable $\boldsymbol{\theta}_\ell = \pi/2$. Now, we can calibrate $\boldsymbol{\phi}_\ell$ just as we calibrated $\boldsymbol{\theta}_\ell$ as shown in Fig. 8(b), similar to a parallelized version of the meta-MZI scheme [7], [10]. This calibration approach works for any of the node configurations in Fig. 2. As outlined in Ref. [7], we can calibrate the phase given the transmissivity T using the relation $T = \sin^2(\theta(v) + \theta_0)$, where θ_0 is the reference phase and $\theta(v)$ is the calibration curve given a sweep over all possible voltage settings for v. Once the sweep is finished, we define our calibration by storing the full phase shift-voltage lookup table or a cubic model fit to that curve (3 parameters per tunable element or 6 parameters per node) [3].

Calibrating $\boldsymbol{\theta}_\ell$ then $\boldsymbol{\phi}_\ell$ for each layer of the mesh from left-to-right results in a fully calibrated device, irrespective of the feedforward architecture. Once calibrated, any reachable unitary operator on the device can be implemented by simply setting desired voltages based on the measured transmission curves. Our parallel calibration procedure would need to be repeated once the nodes in the device experience calibration drift due to fatigue or environmental perturbation, which can vary greatly depending on the modulation type. This calibration drift might be diagnosed efficiently by sending in nullification set input vectors (as defined in Eq. 6) to identify errors within each column of the feedforward network denoted by unnullified nodes for the corresponding column. This calibration can also be used to initialize parallel nullification or calculate the final updates for a procedure like *in situ* backpropagation [29].

Parallel calibration, like parallel nullification, generally gives us up to $O(N)$ speedup over iterative calibration and is also generally applicable to any feedforward mesh. It is worth noting also that the parallel calibration protocol might be transferrable to existing technologies such as the PNP grid architecture, with some minor modifications of the protocol discussed in the supplementary of Ref. [7]. Such grid architectures are already arranged in node columns and thus can be calibrated efficiently by following the steps of Fig. 8, potentially without embedded detectors.

VI. Error Considerations

We further discuss the robustness of parallel nullification to sources of static and dynamic errors that arise during fabrication and operation of feedforward mesh networks.

A. Phase Errors

The parallel nullification step in Eq. (9) is agnostic to any static phase shifts that may accumulate at each column due to path length variations, which in other cases (e.g., non-reconfigurable systems) result in phase errors. Specifically, parallel nullification implicitly sets the reference by which phases in the device are measured [11], so the nullification set calculation of Eq. (6) gives the correct inputs for parallel nullification regardless of how the node is controlled. A correctly programmed mesh can be achieved using a TDC- or MZI-based tunable beamsplitter as depicted in Figs. 2 and 4(d).

B. Split Ratio Error

Parallel nullification can correct split ratio errors similarly to how phase errors are corrected, but in some cases, the split ratio range can be limited at each node of the photonic mesh (e.g., due to imperfect 50/50 beamsplitters in typical MZIs [28]). This limited range problem can be avoided entirely using TDCs or double-MZIs [28] rather than single MZIs at each node. As discussed in Section II, we can equivalently consider θ as the "tunable coupling constant" for the TDC. We typically ensure by making the device suitably long (e.g., cross state coupling

length) such that the full split ratio range is contained between the minimum and maximum extent of the tunable coupling constant (i.e., such that $\theta \in [0, \pi]$).

C. Thermal Crosstalk

Thermal crosstalk between device elements can occur whenever there is significant heat generated in the phase-shifting process, such as in thermal phase shifters. We assume that thermal crosstalk is very small between columns and only occurs within each column so that calibrations of past columns are not affected by those of future columns. As this thermal crosstalk increases, the parallel nullification of each column takes longer because the optimizations within each column are no longer independent of each other (e.g., the settings of node m would be affected by the settings of nodes $m-1$ and $m+1$).

Phase shifter technologies that have little to no crosstalk (such as MEMS phase shifters [30]) would be faster to program because nullifications within the column would be truly independent. In Section II, we already propose node configurations with at most π phase shifts (rather than 2π), requiring smaller temperature variations per waveguide length and limiting thermal crosstalk compared to conventional designs.

D. Loss

Parallel nullification is capable of programming loss-balanced architectures. A loss-balanced architecture is achieved if all the waveguide path lengths and bends are equal (assuming uniform waveguide scattering loss) and the phase shifters are lossless (i.e., changing a phase shift does not increase or decrease loss incurred by that phase shift). If all modes encounter a loss μ_ℓ at each column ℓ, then it is straightforward to show the column can be programmed to implement $\mu_\ell U_N^{(\ell)}$ using parallel nullification. From Eq. (4), the overall network implements μU_N, where μ is ideally a "global loss" equal to the product of all column-wise losses, i.e. $\mu = \prod_\ell \mu_\ell$ [31]. Loss-balanced grid architectures (such as rectangular grid meshes [14]) and other symmetric architectures such as the butterfly (FFT) architecture [8] can be fabricated to fit this criterion. Other architectures (e.g., triangular meshes [2]) can include "dummy" elements to achieve the same balance. In the case of optical neural networks, some additional calibration of the nonlinear elements may also be required in the presence of loss, which may benefit from reprogrammability of such elements [26].

If the feedforward mesh (or specifically a column of the mesh) suffers from "loss imbalance," then different amounts of light are lost from each output as light propagates. In this case, we might need to readjust the nullification set (e.g., by adjusting the computer model of the mesh to account for lossy mesh columns) to more accurately program in the desired operator, which is a direction that should be further explored.

VII. Discussion and Conclusion

We derive a graph-topological property for any reconfigurable feedforward photonic network of tunable beamsplitter nodes that allows efficient programming ("parallel nullification") of node columns that are not affected by each other and thus can be tuned simultaneously. With a model of the device stored in a computer, we find a set of vector inputs to the device (the "nullification set") and for each column, nullify the bottom power of all tunable beamsplitters in parallel using the corresponding nullification vector. The nullification set can be internally generated given a single-mode input by appending the optical setup machine of Ref. [11] to the mesh.

Parallel nullification can quickly diagnose and error-correct phase drifts of a reconfigurable photonic device, demonstrated in the context of reconfigurable optical neural networks in Section IV and the Appendix. As nullification set inputs are sent in order from $\ell = 1$ to $\ell = L$, error appears as non-nullified power at the bottom output ports of the problematic column. This error may be corrected by nullifying these ports so that further debugging of the photonic circuit can be performed.

We have additionally shown that it is possible to apply a similar model to "parallel calibration," which allows us to define phase-voltage relationships. Our parallel calibration protocol is similar in principle to current calibration protocols [3], [6], [7], [10], but with notable differences (e.g., increased efficiency via parallelization) and simplifications. Such calibration can elucidate phase shift-voltage relationships, which are required for initialization of the network or *in situ* backpropagation [29] for training optical neural networks.

Our approach is similar to the RELLIM approach of Ref. [11], where each input tunes a *single* node, since it relies on the reciprocity of linear optical networks. However, each input in parallel nullification tunes an entire *column* of nodes simultaneously based on the feedforward mesh definition proposed in Eq. 4. Our approach can be robust for feedforward meshes with phase shift crosstalk (i.e., thermal crosstalk [6]) and beamsplitter fabrication errors. Additionally, parallel nullification is currently the most efficient protocol for programming or calibrating a feedforward photonic mesh. In particular, where L is the number of device columns and N is the number of input modes, our parallel nullification protocol requires just L input vectors and programming steps, resulting in up to $(N/2)$-times speedup over existing component-wise calibration approaches.

Appendix
Neurophox: Open Source Software

In our simulation framework neurophox, we provide our general definition of feedforward mesh architectures and the reconfigurable neural network model from the main text. This is the first time to our knowledge that feedforward meshes have been defined in this way, and it allows for a greatly simplified framework for defining and simulating mesh architectures. In neurophox, we provide Python code to calculate the nullification set and simulate parallel nullification on a physical chip using this nullification set.

We also provide the code to train fully optical feedforward neural networks on the MNIST dataset, a popular standard in machine learning models consisting of 28×28 images of handwritten decimal digits from 0 to 9. We use GPU-accelerated automatic differentiation in tensorflow [32] to train a two-layer

ONN (shown in Fig. 7(a)) consisting of two $N = 64$ rectangular grid meshes followed by ReLU-like optical nonlinearity or activation layers [26] to classify each image to the appropriate digit label. Reproducible code and instructions are given in the repository at [https://github.com/solgaardlab/neurophox-notebooks]. Our training examples consist of low-frequency FFT features preprocessed from each image that are input into the device, and our ultimate goal is to direct the light into port $n + 1$ labelled by digit n, as demonstrated in Fig. 7(a) for digit 0.

Reprogrammable electro-optic nonlinearities [26] (or generally any ReLU-like optical nonlinearities that can be tuned to operate in a linear regime) allow multi-layer, "deep" ONNs to be programmed or calibrated using our parallel approaches. Crucially, this means that it is possible to calculate inputs to the entire ONN (rather than each layer) and sequentially program the columns of all mesh networks in the overall device to program any desired operator of choice. We now demonstrate the use of this protocol for correcting significant drifts in our specific simulated ONN, which is evident since it is used for programming the network in the first place.

As in Ref. [26], we preprocess each image by applying a Fourier transform and discrete low-pass filter, since such a task is potentially feasible in the optical domain via Fourier optics. In our low-pass feature selection, we pick the center 8×8 block of pixels since most of the useful data is within this low-frequency band, giving us a total of 64 features. Unlike in Ref. [26], we retain some of the redundant Fourier features in this window since it slightly boosts neural network performance, and we use a mean square error loss instead of a categorical cross entropy loss due to slight performance improvement in the former.

Our prediction consists of dropping the final 54 outputs of the second neural network layer, and squaring the amplitudes of the remaining 10 outputs (equivalent to measuring power at photodetectors placed at those outputs). Given the dth data sample $(\boldsymbol{x}_d, \boldsymbol{y}_d)$ (pair of input feature vector and one-hot label vector), our cost function $\mathcal{L}$ is the mean square error

$$\mathcal{L} = \sum_{d=1}^{D} \left\| \frac{f(\boldsymbol{x}_d)}{\|f(\boldsymbol{x}_d)\|} - \boldsymbol{y}_d \right\|^2, \tag{11}$$

where $f(\boldsymbol{x}_d)$ represents the raw output powers of the neural network given input $\boldsymbol{x}_d$. In practice, our classification will always correspond to the port in which the highest output power is measured, ideally guiding input mode vector $\boldsymbol{x}_d$ to the output port corresponding to $\boldsymbol{y}_d$. Our model achieves a final train accuracy of 98.9% and a final test accuracy of 97.8%. Slightly worse training performance was found using a categorical cross entropy loss.

The training of the ONN parameters is shown in Fig. 7(b), achieving 98.9% accuracy on training data (60000 training examples) and 97.8% accuracy on hold-out evaluation data (10000 testing examples). In our simulated environment, we use parallel nullification to correct phase drift $\delta\theta_{m,\ell}, \delta\phi_{m,\ell} \sim \mathcal{N}(0, \sigma^2)$ for $\sigma = 0.05$ in our MNIST-trained network as shown in Fig. 7(c).

We train the same two-layer neural network for different sizes of $N = \{36, 64, 144\}$, achieving test accuracies of 96.6%, 97.8%, 98.1% respectively, though performance varies slightly from run to run. We note that the study in Ref. [9] studies the MNIST task on larger simulated ONNs and accomplishes a similar test accuracy (97.8%). Aside from training significantly fewer parameters than in Ref. [9], we use unitary rather than general linear layers, we use mean square error rather than categorical cross entropy loss, and our feature selection is different (low-frequency Fourier features rather than all raw features).

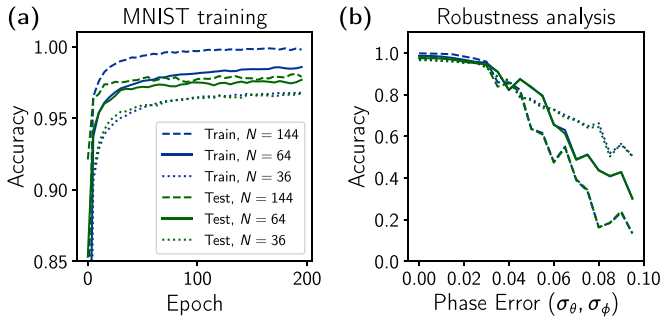

Fig. 9. (a) MNIST training results for $N = 36, 64, 144$ for the two-layer neural network simulated in the main text. Significant overfitting is observed when $N = 144$ due to lack of regularization. (b) Accuracy robustness analysis for the ONN for $N = 36, 64, 144$ for phase errors ranging from $\sigma = 0$ to $\sigma = 0.1$.

Finally, we perform a robustness analysis of imperfections in neural network performance (as in Ref. [9]). The training curves and robustness analysis for different phase errors (Gaussian phase errors of the form $\mathcal{N}(0, \sigma^2)$) are shown in Figure 9, where we find (for the model we trained on the computer) that phase errors above $\sigma = 0.02$ begin to significantly affect performance. Of course, once parallel nullification is applied, any such errors can be corrected regardless of the exact calibration model for the individual phase shifters. Note that in the case $N = 144$, significant overfitting is present, though adding dropout (i.e., zeroing-out power in some of the ports) after the first ONN layer might be one method to "regularize" the physical ONN and therefore reduce this overfitting.

Parallel nullification is therefore a promising option for realizing machine learning models on reconfigurable devices [6], [31]. Such devices provide strictly more generality and flexibility over non-reconfigurable ONNs which can only implement one model (specified pre-fabrication [9]) and furthermore cannot be dynamically error-corrected post-fabrication.

We additionally briefly discuss interesting structure in the nullification set for a random unitary matrix implemented on a rectangular mesh.

For a rectangular mesh, the implementation of these Haar random unitary matrices (i.e. matrices with roughly uniform-random magnitude elements) for large N involves low reflectivity (cross state) in the center of the mesh and random reflectivity on the boundary [23], [33]. The nullification vectors of a Haar random matrix lie somewhere between those for a mesh of only bar state nodes (the "bar/identity" label) and those of a mesh of only cross state nodes (the "cross/flip" label) in Fig. 10. As expected, the nullification set for the cross/flip mesh has an antisymmetric configuration versus the more symmetric configuration of the nullification set for the bar/identity mesh in Fig. 10. We note that the vectors $\boldsymbol{w}_\ell$ are not generally mutually

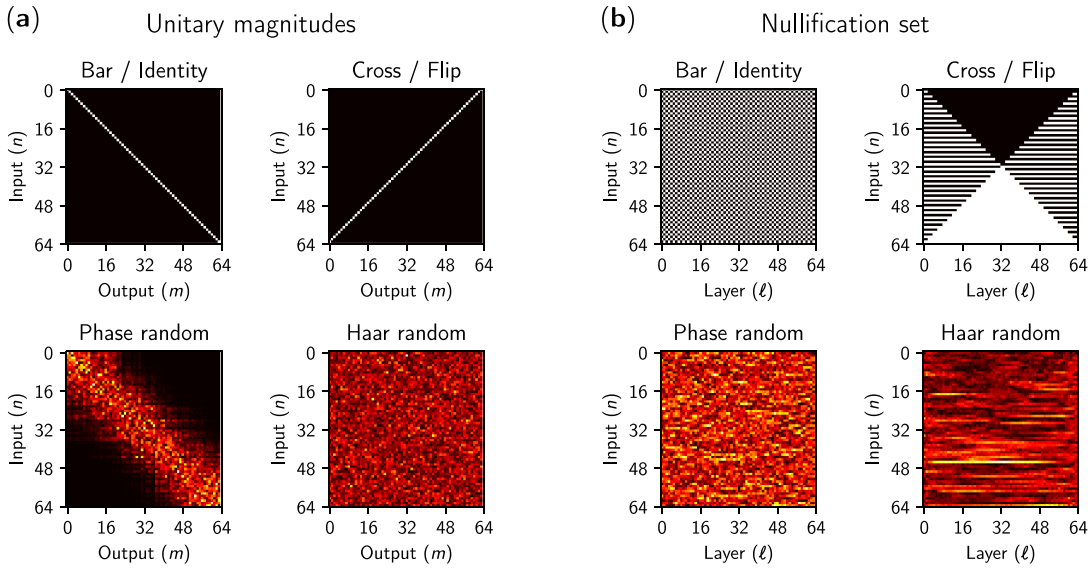

Fig. 10. (a) Magnitudes of the unitary matrix elements for a rectangular mesh for size $N = 64$ given bar state, cross state, uniformly random phase ("phase random"), and random unitary ("Haar random") phase settings. (b) Power magnitudes of the nullification set $\boldsymbol{w}_\ell$ for a rectangular mesh for size $N = 64$ given bar state, cross state, uniformly random phase, and Haar random phase settings.

orthogonal, so the nullification set also does not form a unitary matrix if arranged side by side.

The nullification set has a pattern for random matrices shown in Fig. 10(b) that lie somewhere between the phase random and cross/flip patterns. Furthermore, the nullification set for a mesh with phase settings that are uniformly set from $[0, 2\pi)$ (which leads to the "banded unitary" in Fig. 10(a)) looks more random (i.e. less structured) than the nullification set for the Haar-random phase settings (which leads to the truly random unitary in Fig. 10(a)). This is due to the nonlinear relationship between the transmission amplitude of the individual nodes and the final output magnitudes of the overall device, which ultimately necessitate that most of the nodes are biased towards cross state [23], [33].

ACKNOWLEDGMENT

The authors would like to thank Nathnael Abebe, Ben Bartlett, and Rebecca L Hwang for useful discussions.

REFERENCES

[1] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, "Experimental realization of any discrete unitary operator," *Phys. Rev. Lett.*, vol. 73, no. 1, pp. 58–61, 1994.

[2] D. A. B. Miller, "Self-configuring universal linear optical component [Invited]," *Photon. Res.*, vol. 1, no. 1, 2013, Art. no. 1. [Online]. Available: https://www.osapublishing.org/prj/abstract.cfm?uri=prj-1-1-1

[3] J. Carolan *et al.*, "Universal linear optics," *Science*, vol. 349, no. 6249, pp. 711–716, 2015.

[4] A. Annoni *et al.*, "Unscrambling light–Automatically undoing strong mixing between modes," *Light: Sci. Appl.*, vol. 6, no. 12, pp. e17110–e17110, 2017.

[5] D. A. B. Miller, "Self-aligning universal beam coupler," *Opt. Express*, vol. 21, no. 5, Mar. 2013, Art. no. 6360. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=oe-21-5-6360

[6] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photon.*, vol. 11, no. 7, pp. 441–446, Jul. 2017. [Online]. Available: http://www.nature.com/articles/nphoton.2017.93

[7] N. C. Harris *et al.*, "Quantum transport simulations in a programmable nanophotonic processor," *Nature Photon.*, vol. 11, no. 7, pp. 447–452, 2017.

[8] F. Flamini, N. Spagnolo, N. Viggianiello, A. Crespi, R. Osellame, and F. Sciarrino, "Benchmarking integrated linear-optical architectures for quantum information processing," *Scientific Rep.*, vol. 7, no. 1, 2017, Art. no. 15133. [Online]. Available: http://www.nature.com/articles/s41598-017-15174-2

[9] M. Y.-S. Fang, S. Manipatruni, C. Wierzynski, A. Khosrowshahi, and M. R. DeWeese, "Design of optical neural networks with component imprecisions," *Opt. Express.*, vol. 27, no. 10, 2019, Art. no. 14009. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=oe-27-10-14009

[10] J. Mower, N. C. Harris, G. R. Steinbrecher, Y. Lahini, and D. Englund, "High-fidelity quantum state evolution in imperfect photonic integrated circuits," *Phys. Rev. A*, vol. 92, no. 3, 2015, Art. no. 032322. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.92.032322

[11] D. A. B. Miller, "Setting up meshes of interferometers reversed local light interference method," *Opt. Express.*, vol. 25, no. 23, 2017, Art. no. 29233. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=oe-25-23-29233

[12] T. W. Hughes, R. J. England, and S. Fan, "Reconfigurable photonic circuit for controlled power delivery to laser-driven accelerators on a chip," *Phys. Rev. Appl.*, vol. 11, no. 6, 2019, Art. no. 064014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.11.064014

[13] K. Choutagunta, I. Roberts, D. A. Miller, and J. M. Kahn, "Adapting Mach-zehnder mesh equalizers in direct-detection mode-division-multiplexed links," *J. Lightw. Technol.*, vol. 38, no. 4, pp. 723–735, Feb. 2020.

[14] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "An optimal design for universal multiport interferometers," *Optica*, no. 2, pp. 1–8, 2016. [Online]. Available: http://arxiv.org/abs/1603.08788

[15] D. A. Miller, "All linear optical devices are mode converters," *Opt. Express.*, vol. 20, no. 21, 2012, Art. no. 23985.

[16] D. A. B. Miller, "Waves, modes, communications and optics," *arXiv:1904.05427*, Apr. 2019.

[17] D. A. Miller, "Phase shifting by mechanical movement," U.S. Patent 10338319, 2019. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u = PN.&OS=PN/10338319&RS=PN/10338319

[18] D. Pérez-López, A. M. Gutierrez, E. Sánchez, P. DasMahapatra, and J. Capmany, "Integrated photonic tunable basic units using dual-drive directional couplers," *Opt. Express.*, vol. 27, no. 26, 2019, Art. no. 38071.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://link.springer.com/10.1007/BF01386390

[20] D. Pérez *et al.*, "Multipurpose silicon photonics signal processor core," *Nature Commun.*, vol. 8, 2017, Art. no. 636.
[21] D. Pérez and J. Capmany, "Scalable analysis for arbitrary photonic integrated waveguide meshes," *Optica*, vol. 6, no. 1, 2019, Art. no. 19. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=optica-6-1-19
[22] D. Perez, I. Gasulla, J. Capmany, and R. A. Soref, "Hexagonal waveguide mesh design for universal multiport interferometers," in *Proc. IEEE Photon. Conf.*, 2017, pp. 285–286.
[23] S. Pai, B. Bartlett, O. Solgaard, and D. A. B. Miller, "Matrix optimization on universal unitary photonic devices," *Phys. Rev. Appl.*, vol. 11, no. 6, 2019, Art. no. 064044. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.11.064044
[24] S. Grillanda *et al.*, "Non-invasive monitoring and control in silicon photonics using CMOS integrated electronics," *Optica*, vol. 1, no. 3, 2014, Art. no. 129. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=optica-1-3-129
[25] F. Morichetti *et al.*, "Non-invasive on-chip light observation by contactless waveguide conductivity monitoring," *IEEE J. Sel. Topics Quantum. Electron.*, vol. 20, no. 4, Jul./Aug. 2014, Art. no. 8201710. [Online]. Available: http://ieeexplore.ieee.org/document/6712121/
[26] I. A. D. Williamson, T. W. Hughes, M. Minkov, B. Bartlett, S. Pai, and S. Fan, "Reprogrammable electro-optic nonlinear activation functions for optical neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, Jan./Feb. 2020, Art. no. 7700412. [Online]. Available: https://ieeexplore.ieee.org/document/8769881/
[27] C. Taballione *et al.*, "8 × 8 programmable quantum photonic processor based on silicon nitride waveguides," in *Proc. Frontiers Optics/Laser Sci.*, Washington, DC, USA: OSA, 9 2018, Paper JTu3A.58. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=FiO-2018-JTu3A.58
[28] D. A. B. Miller, "Perfect optics with imperfect components," *Optica*, vol. 2, no. 8, 2015, Art. no. 747. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=optica-2-8-747
[29] T. W. Hughes, M. Minkov, Y. Shi, and S. Fan, "Training of photonic neural networks through in situ backpropagation and gradient measurement," *Optica*, vol. 5, no. 7, 2018, Art. no. 864. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=optica-5-7-864
[30] P. Edinger, C. Errando-Herranz, and K. Gylfason, "Low-loss MEMS phase shifter for large scale reconfigurable silicon photonics," in *Proc. 32nd IEEE Int. Conf. Micro Electro Mech. Syst.*, 2019, pp. 919–921.
[31] N. C. Harris *et al.*, "Linear programmable nanophotonic processors," *Optica*, vol. 5, no. 12, 2018, Art. no. 1623. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=optica-5-12-1623
[32] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. Operat. Syst. Des. Implementation*, Savannah, GA, USA, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentatio n/abadi
[33] N. J. Russell, L. Chakhmakhchyan, J. L. O'Brien, and A. Laing, "Direct dialling of Haar random unitary matrices," *New J. Phys.*, vol. 19, no. 3, 2017, Art. no. 033007.

Sunil Pai (Member, IEEE) received the B.S. degree in physics and the M.S. degree in computer science from Stanford University in 2015 and 2016, respectively. He is currently a Ph.D. Student in electrical engineering at Stanford University. His research interests include machine learning, photonics, and quantum optics.

Ian A. D. Williamson (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from The University of Texas at Austin, Austin, TX in 2010, 2014, and 2017, respectively. He is currently a Postdoctoral Research Fellow with the Department of Electrical Engineering and the Edward L. Ginzton Laboratory, Stanford University, Stanford, CA, USA. His current research interests include applications of optics and acoustics for information processing, analog computing, and machine learning. He is also interested in applications of automatic differentiation and adjoint variable methods for inverse problems and physics-constrained optimization. He was the recipient of the Elmer L. Hixson Endowed Graduate Fellowship in Acoustics Engineering in 2014.

Tyler W. Hughes received the B.S. degree in physics from the University of Michigan, Ann Arbor in 2013, the M.S. degree in 2016 and Ph.D. degree in 2019 from the Department of Applied Physics at Stanford University. He was a Research Assistant at the National University of Singapore's Centre for Quantum Technologies. His research interests include electromagnetic inverse design, photonic hardware platforms for machine learning, and laser-driven particle accelerators on a chip.

Momchil Minkov received the B.S. degree in physics from Jacobs University Bremen, Germany, in 2010, and the M.S. and Ph.D. degrees in physics from the Swiss National Polytechnic Institute in Lausanne, Switzerland, in 2012 and 2016, respectively. He is currently a Postdoctoral Research Associate with Stanford University, under a fellowship from the Swiss National Science Foundation. He has coauthored more than 30 peer-reviewed articles, and his research interests include photonic crystals and their applications, topological photonics, and machine learning with optics and photonics.

Olav Solgaard (Fellow, IEEE) received the Ph.D. degree from Stanford University in 1992. His doctoral dissertation: "Integrated Semiconductor Light Modulators for Fiber-optic and Display Applications" was the basis for the establishment of a Silicon Valley firm Silicon Light Machines (SLM), cofounded by Dr. Solgaard in 1994. From 1992 to 1995 he carried out research on optical MEMS as a Postdoctoral Fellow at the University of California, Berkeley, and in 1995, he joined the Electrical Engineering Faculty of the University of California, Davis. His work at UC Davis led to the invention of the multi-wavelength, fiber-optical switch, which has been developed into commercial products by several companies. In 1999 he joined Stanford University where he is now a Professor of electrical engineering and the Director of graduate studies in the Department of Electrical Engineering. Professor Solgaards research interests include optical MEMS, photonic crystals, optical sensors, microendoscopy, atomic force microscopy, and solar energy conversion. He has authored more than 370 technical publications and holds 75 patents. Professor Solgaard is a Fellow of the IEEE, the Optical Society of America, the Royal Norwegian Society of Sciences and Letters, and the Norwegian Academy of Technological Sciences.

Shanhui Fan (Fellow, IEEE) received the Ph.D. degree in theoretical condensed matter physics from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 1997. He is currently a Professor in electrical engineering, a Professor in applied physics (by courtesy), a Senior Fellow with the Precourt Institute of Energy,and the Director of the Edward L. Ginzton Laboratory, Stanford University, Stanford, CA, USA. He was a Research Scientist with the Research Laboratory of Electronics, MIT. He has authored or coauthored more than 400 refereed journal articles that were cited more than 35,000 times according to the Web of Sciences, has given more than 300 invited talks, and was granted 62 U.S. patents. His research interests include fundamental studies of solid state and photonic structures and devices, especially photonic crystals, plasmonics, and metamaterials, and their applications in information and energy technology. Dr. Fan was the recipient of the National Science Foundation Career Award in 2002, the David and Lucile Packard Fellowship in Science and Engineering in 2003, the National Academy of Sciences Award for Initiative in Research in 2007, the Adolph Lomb Medal from the Optical Society of America in 2007, and the Vannevar Bush Faculty Fellowship in 2017. He has been listed as a highly cited Researcher in physics by Thomson Reuters since 2015. He is a Fellow of the American Physical Society, the Optical Society of America, and the SPIE.

David A. B. Miller (Fellow, IEEE) received the B.Sc. degree from St. Andrews University, the Ph.D. degree in physics from Heriot-Watt University in 1979, and was with Bell Laboratories from 1981 to 1996, as a Department Head from 1987. He is currently the W. M. Keck Professor of electrical engineering at Stanford University. His interests include nanophotonics, quantum-well optoelectronics, and optics in information sensing, interconnects, and processing. He has published more than 270 scientific papers, a quantum mechanics text, and 75 patents, and has a Google h-index of over 100. He has taught open online quantum mechanics classes to over 40,000 students. He was President of IEEE LEOS (now Photonics Society) in 1995. He has received several awards, is a Fellow of APS, OSA, IEEE, the Electromagnetics Academy, the Royal Society of London and the Royal Society of Edinburgh, holds two Honorary Doctorates, and is a Member of the US National Academies of Sciences and of Engineering.