Memory-Efficient Learning for Large-Scale Computational Imaging

Michael Kellman , Kevin Zhang, Eric Markley, Jon Tamir , *Member, IEEE*, Emrah Bostan,
Michael Lustig, and Laura Waller

Abstract—**Critical aspects of computational imaging systems, such as experimental design and image priors, can be optimized through deep networks formed by the unrolled iterations of classical physics-based reconstructions. Termed physics-based networks, they incorporate both the known physics of the system via its forward model, and the power of deep learning via data-driven training. However, for realistic large-scale physics-based networks, computing gradients via backpropagation is infeasible due to the memory limitations of graphics processing units. In this work, we propose a memory-efficient learning procedure that exploits the reversibility of the network's layers to enable physics-based learning for large-scale computational imaging systems. We demonstrate our method on a compressed sensing example, as well as two large-scale real-world systems: 3D multi-channel magnetic resonance imaging and super-resolution optical microscopy.**

Index Terms—**Fourier ptychographic microscopy, image reconstruction, iterative optimization, magnetic resonance imaging, memory-efficient backpropagation, physics-based learning, unrolled networks.**

I. INTRODUCTION

COMPUTATIONAL imaging systems (e.g. tomographic systems, computational optics, magnetic resonance imaging (MRI)) jointly design software and hardware to retrieve information which is not traditionally accessible. Generally, such systems are characterized by how the information is encoded (forward process) and decoded (inverse problem) from the measurements. Recently, physics-based learning [1] has demonstrated the ability to directly optimize a computational imaging system's performance [2]–[14]. Physics-based learning takes advantage of both the known physics of the system's forward model process and the architecture of the decoder's iterative optimizer to build a differentiable neural network that is efficiently parameterized by only a limited number of learnable variables, thereby enabling training using less data [8], [9], [11], while still retaining the robustness and interpretability associated with conventional physics-based inverse problems. Specifically, physics-based networks (PbN) are constructed by unrolling the iterations of an image reconstruction algorithm (e.g. proximal gradient descent [15] or half quadratic splitting [16]), where the iterations of the optimizer form the layers of the network. Hence, the physical forward model is built into the architecture of the network. Commonly, standard signal prior models (e.g. total variation) or a function that enforces consistency (i.e. proximal operators) have been replaced by a learnable convolutional neural network [2], [4], [6], [11], [13], [17] to model the image priors. One can also learn the data capture scheme (i.e. experimental design) by making the system parameters that form the measurements learnable [8]–[10]).

Many computational imaging systems present a unique challenge for PbN implementation, due to the large size and dimensionality of variables that are decoded from the measurements. Training such a PbN relies on gradient-based updates computed using backpropagation (an implementation of reverse-mode differentiation [18]) for learning. As the quantity of decoded information grows, the memory required to perform backpropagation (via automatic differentiation) may exceed the memory capacity of the graphics processing unit (GPU).

Methods to save memory during backpropagation (forward recalculation, forward checkpointing, and reverse recalculation) trade off storage and computational complexity (i.e. the amount of memory and time required for each unrolled layer) [18]. Rather than storing the whole computational graph required for auto-differentiation in memory, these methods reform the graph on an on-demand basis. For a PbN with N layers, standard backpropagation stores the whole graph, achieving $\mathcal{O}(N)$ computational and storage complexity. Forward recalculation instead reforms unstored parts of the graph by reevaluating the operations of the network forward from the beginning. This

Michael Kellman, Kevin Zhang, Jon Tamir, Michael Lustig, and Laura Waller are with the Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA 94720 USA (e-mail: kellman@berkeley.edu; kevinzhang1@berkeley.edu; jtamir@utexas.edu; mlustig@berkeley.edu; waller@berkeley.edu).

Eric Markley is with the Bioengineering, UC Berkeley, Berkeley, CA 94720 USA (e-mail: emarkley@berkeley.edu).

Emrah Bostan is with the Informatics Institute, University of Amsterdam, 1012 WX Amsterdam, Netherlands (e-mail: emrah.bostan@gmail.com).

achieves $\mathcal{O}(1)$ storage complexity, but has $\mathcal{O}(N^2)$ computational complexity because layers of the graph are recomputed from the beginning of the network, while backpropagation requires access to the layers in reverse order. Forward checkpointing saves variables every K layers and forward-recalculates unstored layers of the graph from the closest stored variables (checkpoints), thus directly trading off computational, $\mathcal{O}(NK)$, and storage, $\mathcal{O}(N/K)$, complexity. Reverse recalculation provides a practical solution to beat the trade-off between storage vs. computational complexities by reforming unstored layers of the graph in reverse order from the output of the network (in the same order as required for backpropagation), yielding $\mathcal{O}(N)$ computational and $\mathcal{O}(1)$ storage complexities.

Here, we propose a memory-efficient learning procedure based on the concept of reverse recalculation and invertibility that will enable physics-based learning for general large-scale computational imaging systems. The main contributions of this work are:

1) *General memory-efficient learning procedure for physics-based networks.* We describe how to compute gradients for physics-based learning for networks formed from gradient and proximal update layers without any significant modifications to the architecture of the layers. These layers are seen in a large swath of image reconstruction algorithms, making physics-based learning more feasible for many large-scale physics-based networks. In this work, we focus on the commonly-used proximal gradient descent [15] and half quadratic splitting [16] algorithms.

2) *A hybrid reverse-recalculation and checkpointing scheme to ensure accuracy.* There are several common sources of error: accumulation due to numerical precision and the convergence of PbNs constructed from convex programs. Our hybrid scheme mitigates both issues with a small number of checkpoints.

3) *Demonstration of results for general computational imaging systems.* We learn the design for several large-scale computational imaging systems: 3D multi-channel compressed sensing MRI and super-resolution optical microscopy. In each of these applications, we are able to learn the computational imaging system's design using PbNs previously proposed in literature at a scale larger than was previously possible. These specific architectures were selected from already published works to demonstrate the broad class of physics-based network to which our method applies. We observe similar quality results to those works, but do not claim that they are the best performing. To the best of our knowledge, we demonstrate the first example of memory-efficient learning in the area of computational microscopy.

II. RELATED WORKS

Our work considers memory-efficient learning for a general class of physics-based networks to enable learning for large-scale computational imaging systems. Previous work can be classified under the following categories:

Learning for Computational Imaging: Methods can be categorized into *physics-based* and *physics-free* approaches. *Physics-based* [1]–[14] methods consider the inclusion of the forward model process and the structure of inverse problem optimization in the physics-based network. *Physics-free* approaches use a black box architecture (e.g. UNets [19]) to learn the decoder relationship without prior knowledge of the forward model. The former require fewer learnable parameters than the later, allowing them to be trained using less data. In addition, *physics-based* methods are more robust in experimental settings and inherit the interpretability associated with classic inverse problems. The recent work by Ongie *et al.* [14] has a comprehensive review of these methods.

Invertible Learning: Recently, invertible networks have been popularized to perform reverse-mode differentiation to save memory [17], [20]–[23] and model high-dimensional densities [24]–[30]. All based on the concept of reverse recalculation [18], these methods form networks from a sequence of invertible operations, thereby alleviating the need to store intermediate variables in memory for computing gradients using backpropagation. Our method relies on the same concept of reverse recalculation to perform memory-efficient learning [18], [21] for networks composed of gradient and proximal update layers, making physics-based learning feasible for a wider variety of large-scale physics-based networks.

The work by Putzky and Welling (2019) [17] is most similar to our work. It demonstrates memory-efficient learning using a modified recurrent inference machine [12] architecture that relies on the forward model and an invertible layer with orthogonal 1×1 convolutions for applications in MRI. With a similar goal in mind, our work presents a more general method that does not require any significant modification to the physics-based network for invertibility and includes many options for layers (i.e. gradient, proximal, least-squares update layers). This will make physics-based learning more feasible for users wanting to design large-scale computational imaging systems. The storage and computation required for our method and the work in [17] are further contrasted in Sec. VII.

Implicit Function Theorem: Memory-efficient differentiation can be performed using implicit function theorem (IFT) when the network minimizes an objective function. Specifically, IFT can compute gradients of a network that achieves a fixed point or an optimum by differentiating its optimality equations at that point with respect to the learnable parameters. It's usefulness has been demonstrated to differentiation through optimization problems via the Karush–Kuhn–Tucker conditions [31], for meta-learning [32], [33], to learn fixed-point methods [34], and to backpropagate through recurrent networks [35]. Physics-based networks are formed by optimization problems and thus could potentially benefit from these concepts. However, in many cases physics-based networks are stopped early prior to convergence to a fixed point due to limited computation or as a method of regularization. Using IFT also does not allow independent parameters to be learned for different layers of the network and variables associated with the optimizer itself (e.g. step size and
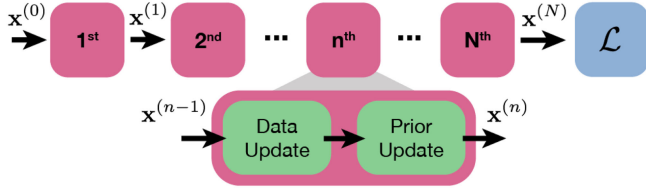
Fig. 1. Physics-based Networks (PbNs) are formed by unrolling the iterations of an image reconstruction optimization. Each layer contains one iteration, made up of a data consistency update and signal prior update. The PbN input is the reconstruction's initialization, $\mathbf{x}^{(0)}$, and the output is the reconstructed image from the Nth layer, which is fed into the learning loss, $\mathcal{L}$.

Algorithm 1: Proximal Gradient Descent.

Inputs $\mathbf{x}^{(0)}$-initialization, α-step size, N-maximum number of iterations
Output $\mathbf{x}^{(N)}$-final estimate of image
1: $n \leftarrow 1$
2: **for** $n < N$ **do**
3: $\quad \mathbf{z}^{(n)} \leftarrow \mathbf{x}^{(n-1)} - \alpha \nabla_{\mathbf{x}} \mathcal{D}(\mathbf{x}^{(n-1)}; \mathbf{y})$
4: $\quad \mathbf{x}^{(n)} \leftarrow \arg\min_{\mathbf{x}} \mathcal{P}(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{z}^{(n)}\|_2^2$
5: $\quad n \leftarrow n + 1$
6: **end for**

---

acceleration rate), while computing gradients via backpropagation does.

III. BACKGROUND

The forward process for a typical computational imaging system describes how information about the image to be reconstructed, $\mathbf{x}$, is encoded into the measurements, $\mathbf{y}$. Specifically,

$$\mathbf{y} = \mathcal{A}(\mathbf{x}) + \mathbf{n}, \tag{1}$$

where $\mathcal{A}$ is the forward model that characterizes the measurement system physics and $\mathbf{n}$ is noise. The forward model is a continuous process, but is often represented by a discrete approximation. The inverse problem (i.e. decoding) is commonly formulated as an optimization problem,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \mathcal{D}(\mathbf{x}; \mathbf{y}) + \mathcal{P}(\mathbf{x}), \tag{2}$$

where $\mathcal{D}(\cdot)$ is a data consistency penalty and $\mathcal{P}(\cdot)$ is a signal prior penalty. When the noise, $\mathbf{n}$, is governed by a known noise model, the data consistency penalty can be written as the negative log-likelihood of the appropriate distribution. Proximal gradient descent (PGD) and half quadratic splitting (HQS) are two choices of algorithm for minimizing the objective in Eq. 2 and can be used to form PbNs (Fig. 1) that alternate between minimizing the data consistency and signal prior penalties.

PGD is efficient in the case when $\mathcal{A}$ is non-linear and/or $\mathcal{P}(\mathbf{x})$ is not smooth in $\mathbf{x}$ (e.g. ℓ_1, total variation). The PGD algorithm (Alg. 1) is composed of alternating gradient and proximal update steps. In Alg. 1, α is the gradient step size, $\nabla_{\mathbf{x}}$ is the gradient operator, and $\mathbf{x}^{(k)}$ and $\mathbf{z}^{(k)}$ are intermediate variables for the kth iteration.

Algorithm 2: Half Quadratic Splitting.

Inputs $\mathbf{x}^{(0)}$-initialization, μ-penalty parameter, N-maximum number of iterations
Output $\mathbf{x}^{(N)}$-final estimate of image
1: $n \leftarrow 1$
2: **for** $n < N$ **do**
3: $\quad \mathbf{z}^{(n)} \leftarrow \arg\min_{\mathbf{z}} \mathcal{D}(\mathbf{z}; \mathbf{y}) + \mu\|\mathbf{z} - \mathbf{x}^{(n-1)}\|_2^2$
4: $\quad \mathbf{x}^{(n)} \leftarrow \arg\min_{\mathbf{x}} \mathcal{P}(\mathbf{x}) + \mu\|\mathbf{x} - \mathbf{z}^{(n)}\|_2^2$
5: $\quad n \leftarrow n + 1$
6: **end for**

HQS is a more efficient algorithm when the forward model, $\mathcal{A}$, is linear and $\mathcal{P}(\mathbf{x})$ is not smooth in $\mathbf{x}$. While similar to PGD in alternating between data consistency and prior updates, HQS instead performs a full model inversion rather than a single gradient step for the data consistency update. This process is described in Alg. 2, with μ representing a penalty parameter that weights the data consistency and signal prior penalties. When the noise has a normal distribution and $\mathbf{A}$ is linear, Line 3 of Alg. 2, can be efficiently solved via the conjugate gradient (CG) method.

The structure of the PbN is determined by unrolling N iterations (Fig. 1) of the optimizer (Eq. 2) to form N layers of a network (e.g. for PGD, Line 3 and 4 of Alg. 1 form a single layer and for HQS, Line 3 and 4 of Alg. 2 form a single layer). The input to the network is the initial guess for the reconstructed image, $\mathbf{x}^{(0)}$, and the output is the resultant, $\mathbf{x}^{(N)}$. Commonly, the learnable parameters are optimized using gradient-based methods (e.g. stochastic gradient descent (SGD) or adaptive moment estimation (Adam) [36]). Machine learning toolboxes' (e.g. PyTorch [37], Tensor Flow [38]) auto-differentiation functionalities are used to compute the gradients. Auto-differentiation creates a computational graph composed of the PbN's operations and stores intermediate variables in memory on the forward pass of the network. On the backward pass, auto-differentiation traces through the graph from the output to the input, computing the Jacobian-vector product for each operation.

IV. METHODS

Inspired by from concepts of reverse recalculation described in Griewank & Walther [18] and Gomez *et al.* [21], our method improves the storage and computational complexity of backpropagation for PbNs. First, we treat the single large graph for auto-differentiation as a series of smaller graphs. Then, we rely on each physics-based layer's invertibility to reform each smaller graph from the network's output in reverse order. By only requiring a single layer to be stored in memory at a time, we save a factor of N in memory. By computing the smaller graphs in reverse order, we save on computation compared to other methods such as forward checkpointing.

Consider a PbN, $\mathcal{F}$, composed of a sequence of layers,

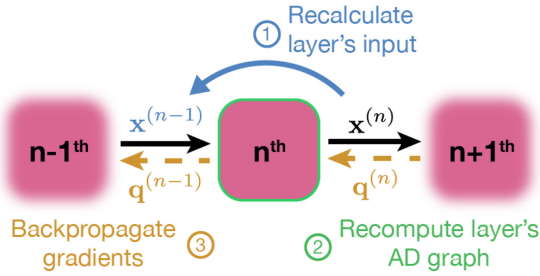$$\mathbf{x}^{(n)} = \mathcal{F}^{(n)}\left(\mathbf{x}^{(n-1)}; \theta^{(n)}\right), \tag{3}$$

Fig. 2. Memory-efficient learning procedure for a single layer: 1) Recalculate the layer's input, $\mathbf{x}^{(n-1)}$, from the output, $\mathbf{x}^{(n)}$, by applying that layer's inverse operations. 2) Recompute the auto-differentiation graph for that single layer. 3) Backpropagate gradients, $\mathbf{q}^{(n)} = \partial\mathcal{L}/\partial\mathbf{x}^{(n)}$, through the layer's auto-differentiation graph.

Algorithm 3: Memory-Efficient Learning for Physics-Based Networks.

Inputs $\mathbf{x}^{(N)}$-output of physics-based network,
  $\mathbf{q}^{(N)}$-gradient of loss with respect to state at output
Output $\{\nabla_{\theta^{(n)}}\mathcal{L}\}_{n=1}^{N}$-gradients with respect to learnable
  parameters at each layer
1: $n \leftarrow N$
2: **for** $n > 0$ **do**
3:     $\mathbf{x}^{(n-1)} \leftarrow \mathcal{F}_{\text{inverse}}^{(n)}(\mathbf{x}^{(n)}; \theta^{(n-1)})$
4:     $\mathbf{v}^{(n)} \leftarrow \mathcal{F}^{(n)}(\mathbf{x}^{(n-1)}; \theta^{(n-1)})$
5:     $\mathbf{q}^{(n-1)} \leftarrow \frac{\partial\mathbf{v}^{(n)}}{\partial\mathbf{x}^{(n-1)}}\mathbf{q}^{(n)}$
6:     $\nabla_{\theta^{(n)}}\mathcal{L} \leftarrow \frac{\partial\mathbf{v}^{(n)}}{\partial\theta^{(n)}}\mathbf{q}^{(n)}$
7:     $n \leftarrow n - 1$
8: **end for**

---

where $\mathbf{x}^{(n-1)}$ and $\mathbf{x}^{(n)}$ are the nth layer input and output, respectively, and $\theta^{(n)}$ are its learnable parameters. When performing reverse-mode differentiation, the method treats a PbN of N layers as N separate smaller graphs, generated on demand, processed and stored one at a time, rather than as a single large graph, thereby saving a factor N in memory. As outlined in Alg. 3 and Fig. 2, we first recalculate the current layer's input, $\mathbf{x}^{(n-1)}$, from its output, $\mathbf{x}^{(n)}$, using $\mathcal{F}_{\text{inverse}}^{(n)}$ (Alg. 3 line 3), and then form one of the smaller graphs by recomputing the output of the layer, $\mathbf{v}^{(n)}$, from the recalculated input (Alg. 3 line 4). To compute gradients, we then rely on auto-differentiation of each layer's smaller graph to compute the gradient of the loss, $\mathcal{L}$, with respect to $\mathbf{x}^{(n)}$ (denoted $\mathbf{q}^{(n)}$) (Alg. 3 line 5) and $\nabla_{\theta^{(n)}}\mathcal{L}$ (Alg. 3 line 6). The procedure is repeated for all N layers in reverse order.

In order to perform the reverse-mode differentiation efficiently, our method must be able to compute each layer's inverse operation, $\mathcal{F}_{\text{inverse}}^{(n)}$. The remainder of this section overviews the procedures to invert gradient and proximal update layers. In addition, special treatment is given to the proximal operation performed in the least-squares update layer to highlight several computational details.

Algorithm 4: Inverse for Gradient Layer.

Inputs $\mathbf{z}$-output of gradient descent layer, L-number of
  iterations
Output $\mathbf{x}^{(L)}$-estimate of gradient descent layer's input
1: $l \leftarrow 0$
2: $\mathbf{x}^{(l)} \leftarrow \mathbf{z}$
3: **for** $l < L$ **do**
4:     $\mathbf{x}^{(l+1)} \leftarrow \mathbf{z} + \alpha\nabla_{\mathbf{x}}\mathcal{D}(\mathbf{x}^{(l)}; \mathbf{y})$
5:     $l \leftarrow l + 1$
6: **end for**

A. Inverse of Gradient Update Layer

A common interpretation of gradient descent is as a forward Euler discretization of the continuous-time ordinary differential process [15] gradient flow. With steps of size α, the nth gradient descent step is,

$$\mathbf{x}^{(n)} \leftarrow \mathbf{x}^{(n-1)} - \alpha\nabla_{\mathbf{x}}\mathcal{D}(\mathbf{x}^{(n-1)}; \mathbf{y}). \quad (4)$$

As a consequence, the inverse of the gradient update layer (Eq. 4) can be viewed as a backward Euler step,

$$\mathbf{x}^{(n-1)} = \mathbf{x}^{(n)} + \alpha\nabla_{\mathbf{x}}\mathcal{D}(\mathbf{x}^{(n-1)}; \mathbf{y}). \quad (5)$$

This is an implicit equation and $\mathbf{x}^{(n)}$ can be solved for iteratively via the backward Euler method using a fixed-point algorithm (Alg. 4) [15]. Convergence is guaranteed if

$$\text{Lip}\left(\alpha\nabla_{\mathbf{x}}\mathcal{D}(\mathbf{x}; \mathbf{y})\right) < 1, \quad (6)$$

where $\text{Lip}(\cdot)$ computes the Lipschitz constant of its argument [39]. In the setting when $\mathcal{D}(\mathbf{x}; \mathbf{y}) = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2$ and the forward model, $\mathbf{A}$, is linear, this can be ensured if $\alpha < \frac{1}{\sigma_{max}(\mathbf{A}^H\mathbf{A})}$, where $\sigma_{max}(\cdot)$ computes the largest singular value of its argument. Finally, as given by Banach Fixed Point Theorem, the fixed point algorithm (Alg. 4) will have an exponential rate of convergence [39].

B. Inverse of Proximal Update Layer

The proximal update is defined as the optimization problem [15],

$$\mathbf{x}^{(n)} \leftarrow \text{prox}_{\mathcal{P}}(\mathbf{x}^{(n-1)}) \quad (7)$$

$$\leftarrow \arg\min_{\mathbf{v}} \mathcal{P}(\mathbf{v}) + \frac{1}{2}\|\mathbf{v} - \mathbf{x}^{(n-1)}\|_2^2. \quad (8)$$

For differentiable $\mathcal{P}(\cdot)$, the solution to Eq. 8 gives,

$$\mathbf{x}^{(n)} = \mathbf{x}^{(n-1)} - \nabla_{\mathbf{x}}\mathcal{P}(\mathbf{x}^{(n)}). \quad (9)$$

In contrast to the gradient update layer, the proximal update layer can be thought of as a backward Euler step of a continuous-time ordinary differential process [15]. This allows its inverse to be expressed as a forward Euler step,

$$\mathbf{x}^{(n-1)} = \mathbf{x}^{(n)} + \nabla_{\mathbf{x}}\mathcal{P}(\mathbf{x}^{(n)}), \quad (10)$$

when the proximal function is bijective (e.g. prox_{ℓ_2}). If the proximal function is not bijective (e.g. prox_{ℓ_1}), the inversion is not straightforward; however, in many cases we can substitute it

with a bijective function with similar behavior. For example, soft thresholding, the proximal operator of ℓ_1 norm, is not bijective, but can be made so by adding a small slope (further discussed in Sec. VIII-A).

C. Inverse of Least-Squares Update Layer

The least-squares update is used in optimizers such as HQS and alternating direction method of multiplers (ADMM) and is more efficient than PGD in the number of unrolled layers required as it performs the complete minimization of the data consistency penalty at each iteration. When examined, this update is technically a proximal operation and thanks to its differentiability it has an exact inverse as outlined in the previous section (Sec. IV-B). We treat it separately because of its importance in many algorithms and because of efficient solution using CG.

This update minimizes the data consistency penalty regularized by the previous estimate, $\mathbf{x}^{(n-1)}$,

$$\mathbf{x}^{(n)} \leftarrow \arg\min_{\mathbf{v}} \|\mathcal{A}(\mathbf{v}) - \mathbf{y}\|_2^2 + \mu\|\mathbf{v} - \mathbf{x}^{(n-1)}\|_2^2, \quad (11)$$

where μ varies the amount of regularization. Giving its name, this optimization can be solved in closed form using least-squares when the forward model, $\mathcal{A}(\cdot)$, is linear,

$$\mathbf{x}^{(n)} \leftarrow \left(\mathbf{A}^H\mathbf{A} + \mu\mathbf{I}\right)^{-1} \left(\mathbf{A}^H\mathbf{y} + \frac{\mu}{2}\mathbf{x}^{(n-1)}\right), \quad (12)$$

where $\mathbf{A}$ denotes the linear forward model. When $\mathbf{A}$ models a linear translation invariant system, it is a circular convolution, the Fourier transform diagonalizes the model, and Eq. 12 can be computed in closed form by dividing by the power spectrum of the system's convolution kernel. However, computational imaging systems often form $\mathbf{A}$ not as an explicit matrix, but as a series of operators. In this case, the inversion can be efficiently computed using the CG method.

The inverse operation of this layer (Eq. 12) can be expressed in closed form as

$$\mathbf{x}^{(n-1)} = \frac{1}{\mu}\left(\left(\mathbf{A}^H\mathbf{A} + \mu\mathbf{I}\right)\mathbf{x}^{(n)} - \mathbf{A}^H\mathbf{y}\right). \quad (13)$$

When using a CG method to perform the forward model inversion (Eq. 12), Eq. 13 is accurate only if CG performs the forward model inversion accurately. This is source of numerical error is further discussed in Sec. V.

V. HYBRID REVERSE RECALCULATION AND CHECKPOINTING

Reverse recalculation of the unstored variables is non-exact, as the operations to calculate the variables are not identical to forward calculation. The result is numerical error between the original forward and reverse calculated variables. As more iterations are unrolled, numerical errors can accumulate.

To mitigate these effects, we can use checkpointing. Some of the intermediate variables can be stored from forward calculation and used in substitution for the recalculated variables, that could incur accumulated numerical errors. Memory permitting, as many checkpoints as possible should be stored to ensure accuracy while performing reverse recalculation. Due to the size of the intermediate variables, large-scale PbNs cannot afford to store all variables required for reverse-mode differentiation, but it is often possible to store a few as checkpoints.

Further, when enough iterations of the reconstruction optimization (Eq. 2) are unrolled, convergence of the intermediate variables can often be observed. When this occurs, inversion of each layer's operations (Alg. 3 line 3) becomes ill-posed. For example, when PGD converges the gradient of the reconstruction loss will vanish (i.e. become zero), thus Alg. 4 will return its input and the inversion will fail.

Checkpointing can again be used to reduce these effects. If convergence behavior is observed, then checkpoints can be stored during later layers to correct inversion error. Economically, checkpoints should be placed closer together for later layers and less frequently for earlier layers (this further discussed in Sec. VIII-A).

VI. RESULTS

We first demonstrate our memory-efficient learning method with a small-scale compressed sensing system as an example, then with two real-world large-scale applications. In the compressed sensing example, we learn the measurement matrix to improve reconstruction performance. In the first of our large-scale applications, we improve the image quality for 3D multi-channel compressed sensing MRI by learning signal priors to regularize the reconstruction. In the second, we improve the temporal resolution of super-resolution microscopy (Fourier Ptychography) by learning the system's experimental design.

A. Learned Measurements for Compressed Sensing

Compressed sensing combines random measurements and regularized optimization to reduce the sampling requirements of a signal below the Nyquist rate [40]. It has seen practical success in many fields (e.g. MRI [41], holography [42], optical imaging [43]). A natural question to ask is, which measurements provide the best signal recovery for a class of signals? Specifically, we recover arbitrary one-sparse signals from linear measurements and learn the linear measurement matrix with a PbN. We learn a set of 7 coded 1D masks; each scalar measurement is the dot product of a mask with the signal. We optimize recovery of the signal in terms of mean square error, for a problem with relatively small dimensions and scale. This small-scale problem lets us rapidly demonstrate the accuracy of our method and compare against other methods. The PbN is constructed by unrolling PGD for the reconstruction loss,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_1. \quad (14)$$

where $\mathbf{A} \in \mathbb{R}^{7 \times 10}$, $\mathbf{x} \in \mathbb{R}^{10}$ is a one-sparse signal, $\mathbf{y} \in \mathbb{R}^7$ is a measurement signal, and λ trades off the data consistency and sparsity prior penalties. The PbN is formed from 800 unrolled iterations of PGD with a step size of 0.05 and $\lambda = 0.06$. For our method, a modified soft thresholding function is used as the proximal operator, where a small slope (on the order of $1e^{-6}$) is added to the zeroed region to make it an invertible function (as discussed in Sec. IV-B). Training was conducted for 20 epochs with 20 training data points, batch size of 4, and learning rate
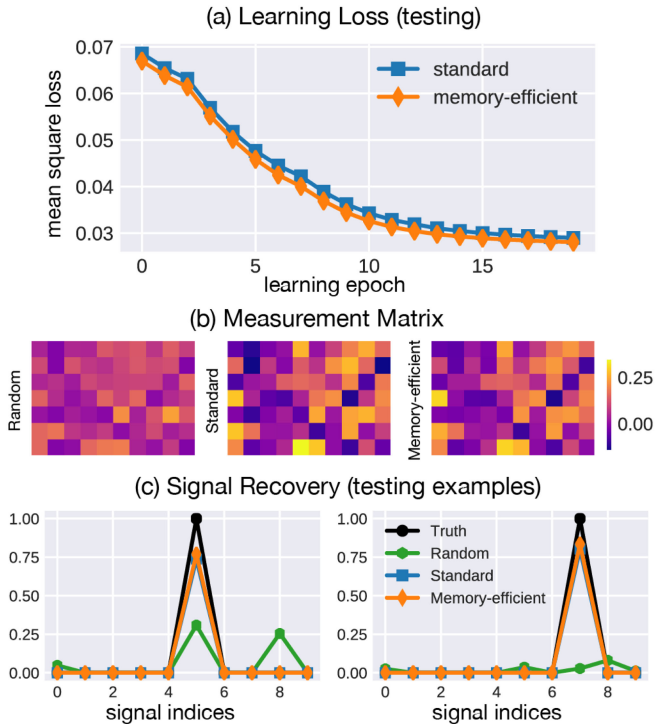
Fig. 3. Learned measurements for 1D compressed sensing: (a) Mean testing loss for learning using standard and memory-efficient learning techniques. (b) Initial (Gaussian randomly distributed) and learned measurement matrices using standard and memory-efficient techniques. (c) Two testing examples of reconstructions with random and learned measurement schemes, demonstrating both improved signal recovery using the learned measurements in comparison to the random measurements and similarity between standard and memory-efficient learning (while requiring 4.1KB, $\sim 800\times$ less memory than standard backpropagation).

of $1e^{-2}$ using ADAM [36]. 50 checkpoints are used to mitigate error due to numerical precision (as discussed in Sec. V).

Fig. 3 shows a comparison between the testing loss for standard and memory-efficient learning techniques, initial random and optimized measurement matrices, and several testing data points for the ground truth with the signal recovered using the learned and initial matrix (random Gaussian variable). As seen in Fig. 3(c), the learned measurement matrices have better signal recovery than the random matrix. The learned measurements and signal recovery using our method and standard learning are similar, however, our method uses $\sim 800\times$ less memory. Where our method uses a modified soft thresholding function, standard learning uses the ordinary version of the function. The quality of the results are comparable (Fig. 3) between the uses of the two functions, suggesting the affect of the modification is negligible; further discussion is included in Sec. VIII-A.

B. Learned Priors for Multi-Channel MRI

As our first large-scale example of real-world applications, we look at MRI, a powerful medical imaging modality that non-invasively captures rich biophysical information without ionizing radiation. Since MRI acquisition time is often directly proportional to the number of acquired measurements, reducing measurements leads to immediate impact on scan time, patient

throughput, and enables capturing fast-changing physiological dynamics. Multi-channel MRI is the standard-of-care in clinical systems and uses multiple receive coils distributed around the body to acquire measurements in parallel. This parallel imaging technique reduces the total number of required acquisition frames for decoding [44]. Further, scan time and noise amplification can be additionally reduced by relying on signal prior knowledge, allowing undersampling of the acquisition frames (i.e. with compressed sensing [41]). Recently, PbNs have been developed to learn the signal priors, achieving state-of-the-art performance for multi-channel accelerated MRI [7], [11]. However, the PbNs are limited in network size and number of unrolled iterations due to the amount of memory required for training. This is an especially prominent problem when moving to high-dimensional problems (e.g. 3D anatomical imaging, temporal dynamics, etc.). Our memory-efficient learning reduces memory footprint at training time, thereby enabling learning for larger problems.

To validate our method, we first show results for the 2D problem in [11], which has small enough memory requirements for the standard backpropagation to fit on our GPUs. The PbN is formed from 4 unrolled iterations of the HQS method and uses a learnable Resnet as the image prior [4], [11]. Specifically, the objective the PbN is minimizes

$$\mathbf{x}^{\star} = \arg\min_{\mathbf{x}} \|\mathbf{PFSx} - \mathbf{y}\|_2^2 + \mu\|\mathbf{x} - \mathcal{R}(\mathbf{x})\|_2^2, \qquad (15)$$

where $\mathbf{S}$ are the multi-channel coil sensitivities, $\mathbf{F}$ denotes Fourier transform, and $\mathbf{P}$ is the undersampling mask used for compressed sensing. The image prior is learned using a network, $\mathcal{R}(\mathbf{x})$, with the invertible residual convolutional neural network (RCNN) [21], [30], [45] architecture composed of a 5-layer CNN where each layer has 64 channels and filters of 3×3. The RCNN's learnable parameters are shared between each PbN layer. To ensure that the invertible RCNN's performance is similar to the standard RCNN, we train with and without required invertibility (supplement Sec.1).

We learn to reconstruct 256×320 slices with measurements from 8 channels and variable density Poisson Disc Fourier undersampling at a rate of $4\times$. Data used for training and testing is from [11], where ground truth brain images are used and data is synthetically generated given the sensitivity and undersampling masks. Training was conducted for 10 epochs with 350 training data points and 10 testing data points, a batch size of 4, and a learning rate of $1e^{-5}$ using ADAM [36]. In Fig. 4, we compare image reconstructions using the priors learned by standard and memory-efficient learning. As shown in Fig. 4(a), testing losses and image reconstruction quality are similar for both methods (Fig. 4(d), (e)), but our method uses $4.82\times$ less memory, while only requiring a $1.09\times$ increase in time.

Finally, we demonstrate our method's ability to learn priors for a 3D volume reconstruction from under-sampled multi-channel measurements - a problem that does not fit within typical GPU memory limits. Specifically, we reconstruct volumes of $50 \times 256 \times 320$ with measurements from 8 channels and variable density Poisson Disc undersampling at a rate of $4\times$. Data used is from [11] and is augmented to create more training
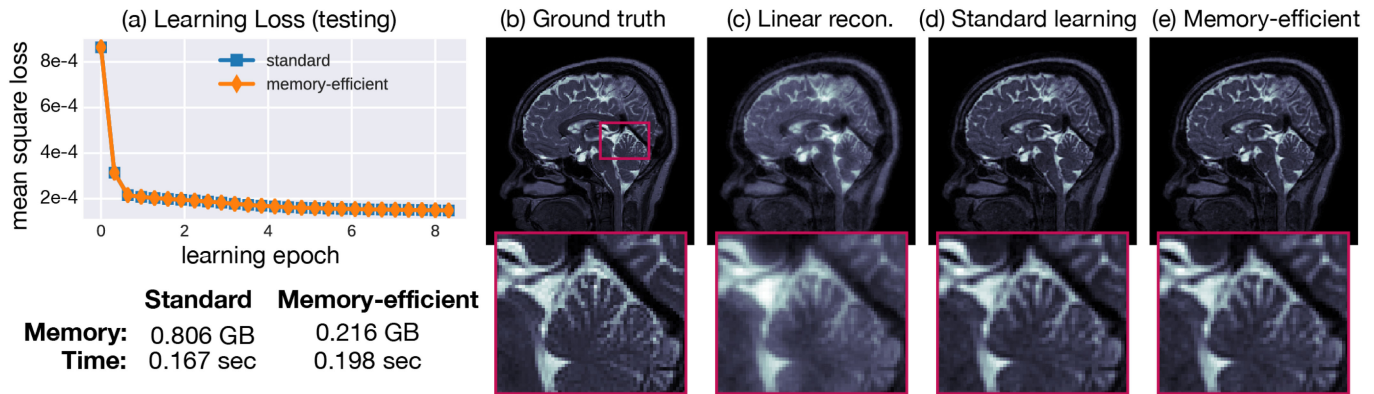
Fig. 4. Learned priors for multi-channel 2D under-sampled MRI: (a) Mean testing loss is similar for both standard backpropagation and memory-efficient learning. (b) Ground truth reconstruction using fully sampled measurements, (c) linear parallel imaging reconstruction (no prior), (d) PbN reconstruction learned using standard backpropagation and (e) PbN reconstruction learned using memory-efficient learning ($3.7\times$ reduced memory requirement, $1.2\times$ increase in compute time). Insets highlight fidelity of high-resolution features and noise reduction in both of the learned designs, as compared to the CG reconstruction. Reported memory and time required is for a single learning update with batch size one.
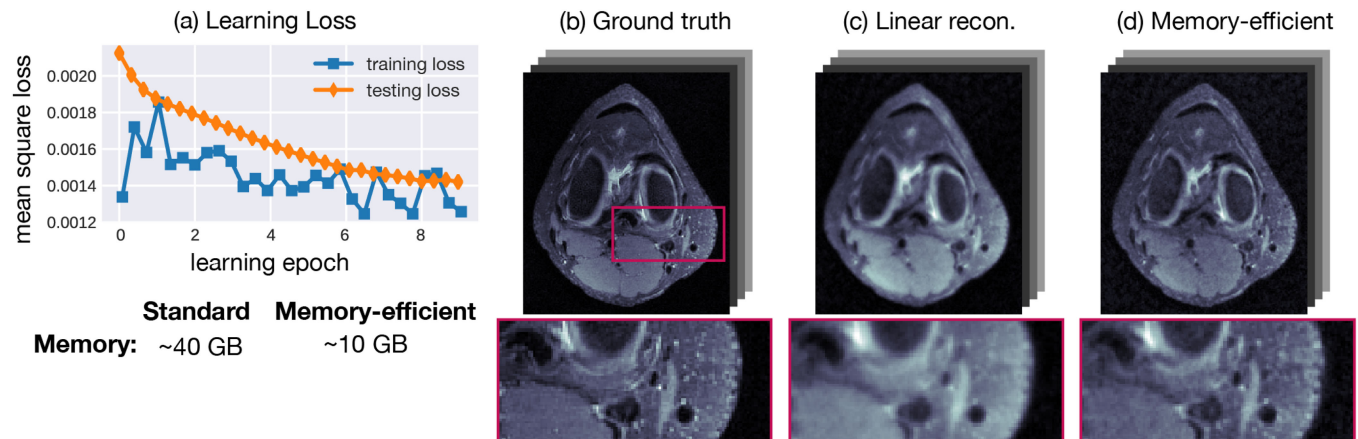

Fig. 5. Learned priors for multi-channel under-sampled 3D MRI: (a) Mean training and testing loss for learning with our proposed memory-efficient technique. (b) One slice of ground truth 3D reconstruction using fully sampled measurements, (c) linear parallel imaging reconstruction (no prior), (d) PbN reconstruction using memory-efficient learning with ~ 10 GB of memory. Standard learning is not shown because it requires more memory than would fit on our GPU.

examples by cropping down larger volumes to $50 \times 256 \times 320$. We use a similar PbN architecture as before for the reconstruction and training parameters, but now with a RCNN with 3D filters ($3 \times 3 \times 3$) and 32 channels. This model would ordinarily require ~ 40 GB of memory using standard backpropagation (~ 10 GB per unrolled iteration), but only requires ~ 10 GB of memory using our method. In Fig. 5, we show results of the learning loss and a single slice of the reconstructed volumes from the ground truth (fully sampled), conjugate gradient (no learning or signal prior), and after learning priors with our memory-efficient learning scheme.

To further motivate the need for large-scale models, we perform an analysis of performance (PSNR) versus number of unrolled iterations (Fig. 6). While the return diminishes as the number of unrolled iterations increases, performance continues to increase past 20 iterations. In this setting, when even a single-layer and checkpoint require approximately 12 GB of memory, our method will be more computationally efficient than other memory-friendly methods such as forward checkpointing (further discussed in Sec. VII).
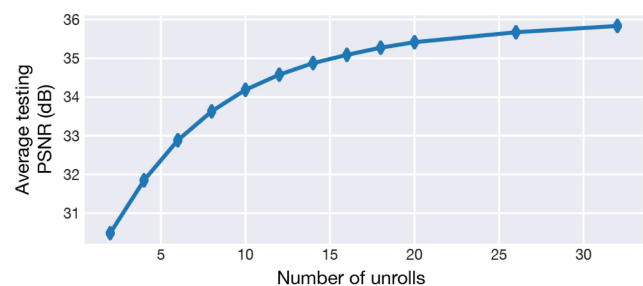
Fig. 6. Multi-channel MRI performance versus number of unrolls: Average testing PSNR for varying number of unrolled iterations. In this example we learn independent parameters for each layer.

C. Learned Experimental Design for Fourier Ptychographic Microscopy

Bright-field microscopy is a standard method for imaging biological samples *in vitro*. As with most microscopes, one must trade-off field-of-view (FoV) and resolution. Fourier Ptychographic Microscopy (FPM) [46] is a super-resolution (SR)

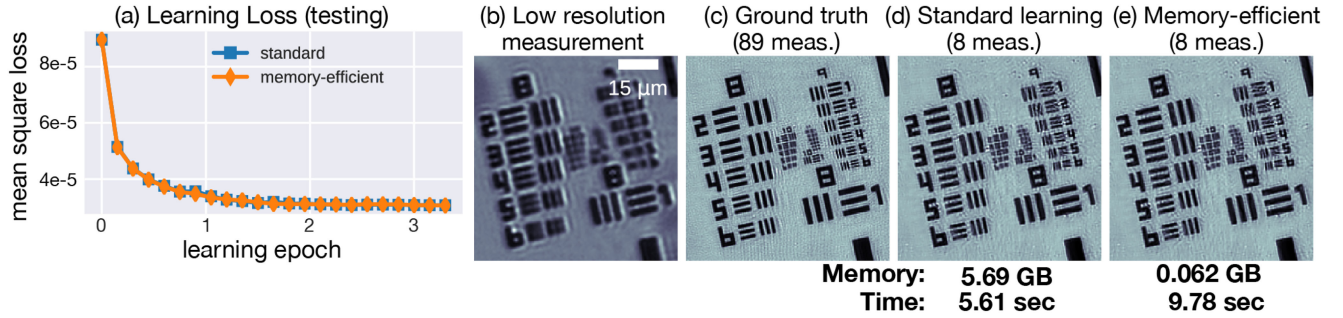Fig. 7. Learned illumination design for Fourier Ptychographic Microscopy (FPM): (a) Mean testing loss is similar for both standard backpropagation and memory-efficient learning. (b) Example low-resolution measurement, (c) ground truth reconstruction using all 89 LED measurements to perform $3.1\times$ super resolution, (d) reconstruction from only 8 measurements learned using standard backpropagation and (e) memory-efficient learning ($92\times$ reduced memory requirement, $1.7\times$ increase in compute time). Reported memory and time required is for a single learning update with batch size one.

method that computationally reconstructs gigapixel-scale images with both large FoV and high resolution from a series of low-resolution images acquired with different illumination settings. The illumination patterns can be conveniently created by a programmable LED array source [47]. The system's dependence on many measurements limits its ability to image live fast-moving biology. Multiplexing schemes for reducing the number of measurements have been proposed [48].

Recently, state-of-the-art performance was achieved by forming a PbN and learning its experimental design (the LED array patterns) [8], [9]. However, the PbN was limited in scale due to GPU memory constraints; terabyte-scale memory would be required for learning patterns with all of the LEDs. Here, we show that our proposed memory-efficient learning framework reduces the necessary memory to only a few gigabytes, thereby enabling full-scale learning on a consumer-grade GPU, which in turn allows us to achieve higher factors of super resolution.

The PbN for learning FPM LED source patterns is formed from the following phase retrieval optimization:

$$\mathbf{x}^{\star} = \arg\min_{\mathbf{x}} \sum_{k=1}^{K} \left\| \mathbf{y}_{\mathrm{m}_k} - \sum_{l=1}^{L} c_{kl} |\mathbf{A}_l \mathbf{x}|^2 \right\|_2^2, \qquad (16)$$

where $\mathbf{y}_{\mathrm{m}_k}$ is the kth multi-LED measurement, $\mathbf{A}_l = \mathbf{F}^H \mathbf{P}_l \mathbf{F}$ is the forward model for the lth LED [9], [46], $\mathbf{P}_l$ is the microscope's pupil function for the lth LED, $\mathbf{F}$ denotes 2D Fourier transform, and c_{kl} is the learnable brightness for the lth LED in the kth measurement. A PbN is formed from N unrolled iterations of gradient descent. We then minimize the loss between the output of the PbN and the ground truth to learn LED brightnesses over the dataset.

We again start by validating our method's accuracy on a small-scale problem that fits in GPU memory using standard learning. We reproduce results in [9], learning illumination patterns for eight measurements, which gives $3.1\times$ resolution improvement and $10\times$ faster data capture. We set $L = 4$, the number of fixed point iterations to invert gradient layers, and checkpoints every 10 unrolled iterations. The testing loss between our method and standard learning are similar (Fig. 7(a)), and the SR reconstructions with learned designs using standard (Fig. 7(d)) and memory-efficient (Fig. 7(e)) methods are both

similar to the 'ground truth' reconstruction using 89 measurements (Fig. 7(c)). Our memory-efficient learning approach, however, reduces memory required from 5.69 GB to 0.062 GB, with compute time increasing by less than a factor of $2\times$. Hence, our method produces comparable quality results as the standard learning, but with significantly reduced (more than $91\times$) memory requirements.

Next, we use our memory-efficient learning scheme to solve a larger-scale problem than was previously possible. For FPM, that means using all 293 LEDs to achieve a higher factor of super resolution ($4.2\times$). 200 iterations are unrolled to create the PbN, we set $L = 4$, and checkpoints every 13 unrolled iterations. For this problem, standard backpropagation would require ~ 500 GB of memory, while our method only requires ~ 3 GB (using 15 checkpoints). In Fig 8, we demonstrate our learned design's ability to reduce the number of measurements required from 293 to 16, demonstrating $20\times$ faster data capture with comparable image quality to ground truth.

In this example, the number of unrolled iterations is determined by the conditioning of the problem as aspects of the reconstruction are not learned (as they are in Sec. VI-B). We perform an analysis of performance (PSNR) versus number of unrolled iterations (Fig. 9) and show that for this problem at least 100 iterations must be unrolled before we see diminishing returns in performance. This analysis is in the context of this level of super resolution and the number of measurements learned. As the degree of super resolution grows or the number of measurements is decreased, the problem will become worst conditioned and more unrolled iterations will be required.

VII. MEMORY-COMPUTATION ANALYSIS

In Sec. VI we demonstrated several example uses of our method, each representing a single point in the storage-computation trade-off space. Here, we provide analysis to determine when our method provides advantage over standard backpropagation and forward checkpointing. We visualize the complete storage-computation space and calculate when each method is best (the fastest method that accommodates the memory required). Specifically, we visualize the relationship between storage-computation for varying input sizes (referred
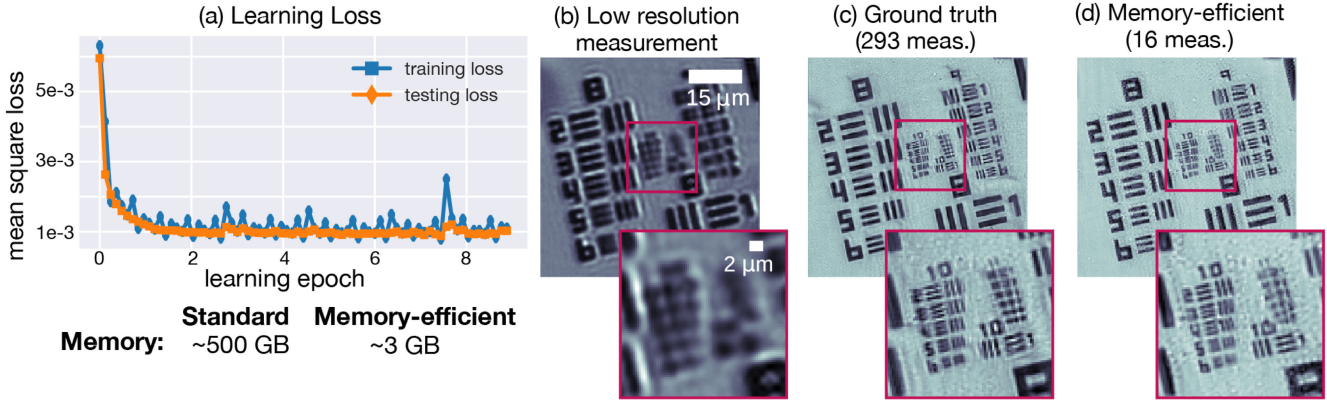
Fig. 8. Large-scale learned illumination design for FPM: (a) Training and testing loss for memory-efficient learned design. (b) Example low-resolution measurement, (c) ground truth reconstruction using all 293 LED measurements to perform $4.2\times$ super resolution, (d) reconstruction from only 16 measurements learned using memory-efficient learning with ~ 3 GB memory. Standard learning is not shown because it would require ~ 500 GB of memory, which is not available on our GPU. Insets highlight high-resolution features.
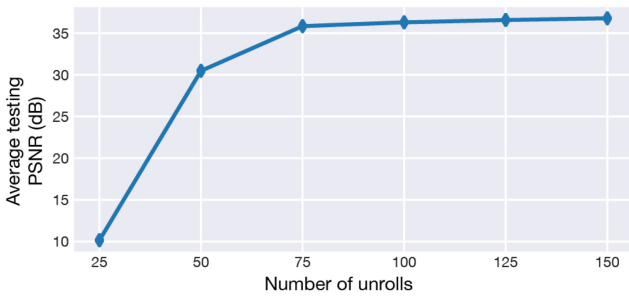

Fig. 9. Fourier Ptychographic Microscopy performance versus number of unrolls: Average testing PSNR for varying number of unrolled iterations.

to as checkpoint sizes), varying physics-based layer sizes, varying numbers of unrolls, and varying amounts of computation required.

First, we calculate the memory and computation time requirements for all three methods. For standard backpropagation the memory required is $N \times A + B$, where N is the number of layers, A is the memory for a single physics-based layer, and B is the memory for a single data input. The time required is $N \times (T_{fwd} + T_{bck})$, where T_{fwd} and T_{bck} are the computation times for running a layer forward and backwards, respectively. Forward checkpointing stores as many checkpoints as memory affords. Once there are more layers than available memory for checkpoints, layers are recomputed from the previous closest checkpoint. Because backpropagation is performed in the reverse order of the forward pass, it is computationally expensive to recompute layers when the number of layers far exceeds the number of checkpoints. When checkpoints are stored every K iterations the computational time required is $N \times (T_{fwd} + T_{bck}) + \frac{N(K+1)}{2} \times T_{fwd}$. Finally, our method only requires $A + B$ in memory and $N \times (2T_{fwd} + T_{bck} + T_{inv})$, where T_{inv} is the time to invert each layer's operations.

Both forward checkpointing and our method are memory-friendly, but the computation time they require varies drastically based on how many checkpoints can be stored and the computational cost of performing a layer's forward versus its inverse operations. Once forward checkpointing cannot store a single
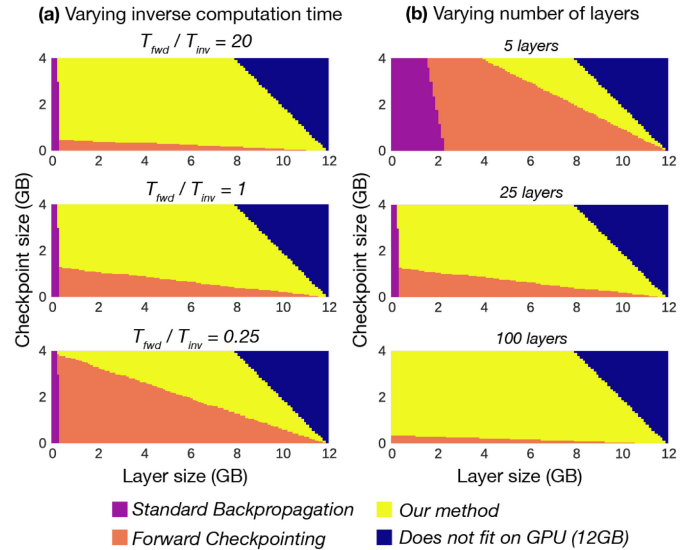

Fig. 10. Fastest method for varying layer and checkpoint size: (a) Fastest method for three scenarios where the ratio between forward and inverse layer computation varies. Our method performs better with larger ratios because it calls each layer's inverse more and forward checkpointing calls each layer's forward more. (b) Fastest method for three scenarios with increasing number of unrolled iterations. Our method performs better as more iterations are unrolled. Sector color corresponds to the fastest method: purple, orange, yellow, and blue representing standard backpropagation, forward checkpointing, our method, and problems that do not fit on a single GPU (12 GB), respectively.

checkpoint per layer, its computation time grows quadratically, while our method remains linear. Further, forward checkpointing relies on the evaluation of layers' forward operations, while our method relies on the evaluation of layers' inverse operations.

Fig. 10 illustrates when each method is fastest for varying size physics-based layers and checkpoint sizes (determining how many checkpoints can fit in memory). This analysis is done using a GPU with 12 GB of memory. Fig. 10(a) shows the fastest choice of method for three different cases: when inverse layer computation is faster, the same speed, and slower than forward layer computation. Our method (yellow) spans a larger area of the trade-off space than forward checkpointing (orange) and

standard backpropagation (purple) when inverse layer evaluation is faster. This is the case when evaluating proximal and least-squares update layers, often implemented using iterative methods such as conjugate gradient, but whose inverse can be expressed in closed form. For this experiment the number of unrolls is held constant with a value of 25. Fig. 10(b) shows which method is fastest for varying size problems as the number of unrolled iterations varies. As more iterations are unrolled, forward checkpointing can store relatively fewer checkpoints and thus slows relative to our method. For this experiment $T_{fwd} = T_{inv}$.

The work of Putzky and Welling [17] ensures invertibility by passing two intermediate variables into each layer requiring double a checkpoint's memory size in storage. Our method only requires a single variable to be stored to ensure invertibility, thereby enabling larger physics-based networks. The computation time required in [17] to perform inversion for each layer is equal to the forward evaluation time, $T_{fwd}/T_{inv} = 1$. For our method, the amount of computation required for a layer's inversion varies depending on its architecture. For gradient layers, the typical amount of computation is $4\times$ slower than its forward evaluation, $T_{fwd}/T_{inv} < 1$. For proximal layers, iterative in nature, the inverse can often be expressed in closed form, i.e. $T_{fwd}/T_{inv} > 1$.

VIII. REMARKS

A. Discussion

Our proposed memory-efficient learning opens the door to using unrolled physics-based networks for learning the design of large-scale computational imaging systems that are not otherwise possible due to GPU memory constraints, without a significant increase in training time. Within this work we detail how physics-based networks composed of gradient and proximal update layers can be reversible to allow for memory-efficient gradient computation. While we have demonstrated our procedure for PbNs formed from PGD and HQS methods, the update layers we describe form the fundamental building blocks of many larger PbNs (e.g. unrolling the updates of alternating minimization).

For our method, the physics-based network must be invertible. To achieve this at the layer level, sufficient conditions for invertibility must be met. For gradient update layers, this comes in the form of a Lipschitz constant constraint (Eq. 6). At the network level, the convergent behavior of physics-based networks and reconstruction optimization (Eq. 2) makes accurate reverse recalculation ill-posed and can cause numerical error accumulation (as outlined in Sec. V). This is not an issue for many PbNs as they truncate the number of unrolled iterations prior to the optimizer's convergence as an additional form of regularization (i.e. early stopping) or to save computation. In the case when convergent behavior is observed, checkpoints should be used. A possible option is to measure the difference between successive intermediate variables on the forward pass of the network. If that quantity falls below a threshold, then the optimization is approaching convergence and checkpoints should be placed to mitigate the accumulation of error on the reverse pass.

In some situations the relationship between storage and computational complexity can be traded off with accuracy. For gradient descent layers, the fixed-point method outlined in Alg. 4 is used to invert and, if not run to convergence, the inversion will be less accurate. When the Lipschitz constant of the gradient operator is large, more iterations (a larger value of L) will be required to accurately invert the layer. Unfortunately, the ideal Lipschitz constant for a gradient descent layer is larger. Practically, we find that only a few (e.g. 4 to 8) iterations are required. For proximal layers, the inversion is accurate up to numerical precision, but requires the iterative forward process of the layer to be computed accurately for our method to be accurate.

In Sec. VI-A a modified soft thresholding function is used in place of the proximal operator for the ℓ_1. While results in Fig. 3 suggest the effect of this change is negligible, the performance of the reconstruction could be reduced to allow for the invertibility of the operation (Sec. IV-B) and use of our method. Depending on the slope added the soft thresholding function, the performance and invertibility are traded off. When the slope is very small (on the order of machine epsilon), the performance of the reconstruction will behave similar to the ordinary function, however, it will be less invertible due to floating point quantization. When the slope is larger, the reconstruction performance could be reduced because the operator does not well model the original proximal function, but will be more linear, thus be less affected by quantization and more invertible.

Acceleration layers to improve convergence of image reconstruction are commonly used in variants of PGD (termed FISTA [49]) and can be incorporated into our framework. Typically, such layers linearly combine the output of the current and previous layers. The acceleration layer cannot be inverted from only the current layer's output, however, with the storage of additional information (this layer's output and the previous layer's output) it is possible to invert an acceleration layer by computing the inverse of a 2×2 matrix.

Work by Mardani *et al.* (2019) [13] demonstrates that the number of unrolled iterations can be reduced without significant losses in performance. In Sec. VII, we showed that even for large layer and checkpoint sizes and a moderate number of unrolls (10-100), memory-efficient learning is required and our method is computationally faster than forward checkpointing. Further, it is not always appropriate to learn a signal prior. In Sec. VI-A and Sec. VI-C only the experimental design is learned and the image reconstruction is fixed. In such settings, methods in [13] do not apply.

A limitation of our method is when each smaller auto-differentiation graph (discussed in Sec. IV) is still too large to fit in memory. In this situation more context-specific solutions (e.g. coil compression for multi-channel MRI, using a smaller field-of-view) or more efficient implementation of the system's fundamental operations is required.

B. Conclusion

Memory-efficient learning with physics-based networks is a practical tool for large-scale computational imaging problems.

Using the concept of reversibility, we implemented reverse-mode differentiation with favorable storage and computational complexities. We demonstrated our method on several representative large-scale applications: 3D multi-channel compressed sensing MRI and super-resolution optical microscopy, and expect many other computational imaging systems to fall within our framework.

With the ever-continuing upward trend in the growth of computational imaging system's size and dimensionality, the need for memory-efficient learning techniques will continue to be in demand. Now, examples of even larger-scale systems exist: extreme MRI [50] and XD-GRASP [51] in the area of medical imaging, 3D fluorescence microscopy [52], [53] and optical diffraction tomography [54] in the area of biological imaging, and hyper-spectral imaging in the area of remote sensing.
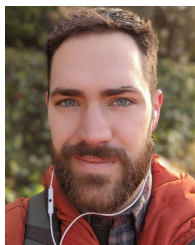
ACKNOWLEDGMENT

REFERENCES

[1] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 399–406.

[2] J. Sun *et al.*, "Deep ADMM-Net for compressive sensing MRI," in *Proc., Adv. Neural Inf. Process. Syst.*, 2016, pp. 10–18.

[3] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1256–1272, Jun. 2016.

[4] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3929–3938.

[5] J. Zhang and B. Ghanem, "ISTA-net: Interpretable optimization-inspired deep network for image compressive sensing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1828–1837.

[6] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein, "Unrolled optimization with deep priors," 2017, *arXiv:1705.08041*.

[7] K. Hammernik *et al.*, "Learning a variational network for reconstruction of accelerated MRI data," *Magn. Reson. Med.*, vol. 79, no. 6, pp. 3055–3071, Nov. 2017.

[8] M. Kellman, E. Bostan, N. Repina, and L. Waller, "Physics-based learned design: Optimized coded-illumination for quantitative phase imaging," *IEEE Trans. Comput. Imag.*, vol. 5, no. 3, pp. 344–353, Sep. 2019.

[9] M. Kellman, E. Bostan, M. Chen, and L. Waller, "Data-driven design for Fourier ptychographic microscopy," in *Proc. Int. Conf. Comput. Photography*, 2019, pp. 1–8.

[10] V. Sitzmann *et al.*, "End-to-end optimization of optics and image processing for achromatic extended depth of field and super-resolution imaging," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–13, Jul. 2018.

[11] H. K. Aggarwal, M. P. Mani, and M. Jacob, "Modl: Model-based deep learning architecture for inverse problems," *IEEE Trans. Med. Imag.*, vol. 38, no. 2, pp. 394–405, Feb. 2018.

[12] P. Putzky and M. Welling, "Recurrent inference machines for solving inverse problems," in *Proc. Int. Conf. Learn. Representations*, 2019.

[13] M. Mardani *et al.*, "Neural proximal gradient descent for compressive imaging,," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9573–9583.

[14] G. Ongie, A. Jalal, C. A. M. R. G. Baraniuk, A. G. Dimakis, and R. Willett, "Deep learning techniques for inverse problems in imaging," *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 39–56, May 2020.

[15] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.

[16] D. Geman and C. Yang, "Nonlinear image recovery with half-quadratic regularization," *IEEE Trans. Image Process.*, vol. 4, no. 7, pp. 932–946, Jul. 1995.

[17] P. Putzky and M. Welling, "Invert to learn to invert," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 444–454.

[18] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. Philadelphia, PA, USA: SIAM, 2008.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Computer-Assisted Intervention*, 2015, pp. 234–241.

[20] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2113–2122.

[21] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2214–2224.

[22] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, "Reversible architectures for arbitrarily deep residual neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018.

[23] L. Ardizzone *et al.*, "Analyzing inverse problems with invertible neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.

[24] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear independent components estimation," in *Proc. Int. Conf. Learn. Representations*, 2015.

[25] L. Dinh, J. Sohl Dickstein, and S. Bengio, "Density estimation using real NVP," 2016, *arXiv:1605.08803*.

[26] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 10 215–10 224.

[27] J.-H. Jacobsen, A. Smeulders, and E. Oyallon, "i-RevNet: Deep invertible networks," 2018, *arXiv:1802.07088*.

[28] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6571–6583.

[29] W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form continuous dynamics for scalable reversible generative models," 2018, *arXiv:1810.01367*.

[30] J. Behrmann, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," 2018, *arXiv:1811.00995*.

[31] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," 2017, *arXiv:1703.00443*.

[32] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," 2018, *arXiv:1806.04910*.

[33] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 113–124.

[34] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 690–701.

[35] R. Liao *et al.*, "Reviving and improving recurrent back-propagation," 2018, *arXiv:1803.06396*.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[37] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Proc. Neural Inf. Process. Syst.*, 2017.

[38] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[39] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fundam. Math.*, vol. 3, no. 1, 1922.

[40] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[41] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magn. Reson. Med.*, vol. 58, no. 6, pp. 1182–1195, Dec. 2007.

[42] D. J. Brady, K. Choi, D. L. Marks, R. Horisaki, and S. Lim, "Compressive holography," *Opt. Express*, vol. 17, no. 15, pp. 13 040–13 049, 2009.

[43] M. F. Duarte *et al.*, "Single-pixel imaging via compressive sampling," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 83–91, Mar. 2008.

[44] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger, "SENSE: Sensitivity encoding for fast MRI," *Magn. Reson. Med.*, vol. 42, no. 5, pp. 952–962, Nov. 1999.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[46] G. Zheng, R. Horstmeyer, and C. Yang, "Wide-field, high-resolution Fourier ptychographic microscopy," *Nature Photon.*, vol. 7, no. 9, pp. 739–745, Jul. 2013.

[47] Z. F. Phillips, R. Eckert, and L. Waller, "Quasi-dome: A self-calibrated high-NA LED illuminator for Fourier ptychography," in *Proc. Imag. Appl. Opt.*, Jun. 2017.

[48] L. Tian, Z. Liu, L.-H. Yeh, M. Chen, J. Zhong, and L. Waller, "Computational illumination for high-speed in vitro Fourier ptychographic microscopy," *Optica*, vol. 2, no. 10, pp. 904–908, Oct. 2015.

[49] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[50] F. Ong *et al.*, "Extreme MRI: Large-scale volumetric dynamic imaging from continuous non-gated acquisitions," *Magn. Reson. Med.*, vol. 84, no. 4, pp. 1763–1780, 2020.

[51] L. Feng, L. Axel, H. Chandarana, K. T. Block, D. K. Sodickson, and R. Otazo, "XD-GRASP: Golden-angle radial MRI with reconstruction of extra motion-state dimensions using compressed sensing," *Magn. Reson. Med.*, vol. 75, no. 2, pp. 775–788, 2016.

[52] B.-C. Chen *et al.*, "Lattice light-sheet microscopy: Imaging molecules to embryos at high spatiotemporal resolution," *Science*, vol. 346, no. 6208, pp. 1257998-1–1257998-14, 2014.

[53] F. L. Liu, G. Kuo, N. Antipa, K. Yanny, and L. Waller, "Fourier diffuserscope: Single-shot 3D Fourier light field microscopy with a diffuser," 2020, *arXiv:2006.16343*.

[54] S. Chowdhury *et al.*, "High-resolution 3D refractive index microscopy of multiple-scattering samples from intensity images," *Optica*, vol. 6, no. 9, pp. 1211–1219, 2019.

Jon Tamir (Member, IEEE) received the B.S. degree in electrical and computer engineering (ECE) from The University of Texas at Austin, Austin, TX, USA, in 2011 and the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, Berkeley, CA, USA, in 2018, where he was also a Research Associate in 2019. He is an Assistant Professor with The University of Texas at Austin with joint appointments in ECE and the Dell Medical School. His research interests include computational magnetic resonance imaging (MRI), machine learning for inverse problems, and clinical translation. His work focuses on applying fast imaging and efficient reconstruction techniques to MRI, with the goal of enabling real clinical adoption. He is a core Developer of the Berkeley Advanced Reconstruction Toolbox.

Michael Kellman received the B.S. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2015, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, Berkeley, CA, USA, in 2017 and 2020, respectively. He is a National Science Foundation Graduate Research Fellow. His research is focused on the design of large-scale computational imaging systems for application in optical microscopy and medical imaging.

Emrah Bostan received the M.Sc. and Ph.D. degrees from EPFL, Lausanne, Switzerland, advised by Michael Unser. He is an Assistant Professor with the Informatics Institute, University of Amsterdam (UvA). He was a Postdoctoral Researcher with the Computational Imaging Lab, University of California, Berkeley, sponsored by Prof. Laura Waller. He was also affiliated with the Berkeley Artificial Intelligence Research Laboratory and Berkeley Center for Computational Imaging in Berkeley.

Kevin Zhang is currently working toward the undergraduate degree with the University of California, Berkeley, Berkeley, CA, USA, and is intending to graduate in 2021. He has previously worked as a Research Assistant with the Computational Imaging Lab from 2018 to 2020 as well as a Teaching Assistant for the Electrical Engineering and Computer Science (EECS) Department from 2019 to 2020 with the University of California, Berkeley. Currently, he is a research assistant with the Computer Vision Group as well as a Research Assistant in a group headed by Professor Gireeja Ranade. His current research interests include the application of machine learning to control theory and computational imaging and the self-supervised extraction of visual features from videos. He has several publications in the field of deep learning applied to computational imaging.

Michael Lustig received the B.Sc. degree in electrical engineering from the Technion - Israel Institute of Technology, Haifa, Israel, in 2002, and the M.Sc. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2004 and 2008, respectively. He is an Associate Professor in EECS. He joined the faculty of the EECS Department, UC Berkeley in Spring 2010. His research interests focus on computational imaging methods in medical imaging, particularly magnetic resonance imaging (MRI). He is a jolly good Fellow of the Society of Magnetic Resonance in Medicine.

Eric Markley received the B.S. degree in applied mathematics and biomedical engineering from the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, in 2019. He is a National Science Foundation Graduate Research Fellow in the Bioengineering program with University of California, Berkeley. Currently, he is a student in Prof. Laura Waller's lab working on joint experimental design and image reconstruction problems in optical microscopy.

Laura Waller received the B.S., M.Eng., and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2004, 2005, and 2010, respectively. She is the Ted Van Duzer Associate Professor of Electrical Engineering and Computer Sciences (EECS) with the University of California, Berkeley, a Senior Fellow with the Berkeley Institute of Data Science, and affiliated with the UCB/UCSF Bioengineering Graduate Group. She was a Postdoctoral Researcher and a Lecturer of Physics with Princeton University from 2010 to 2012. She is a Packard Fellow for Science and Engineering, Moore Foundation Data-driven Investigator, Bakar Fellow, OSA Fellow, and Chan-Zuckerberg Biohub Investigator. She has recieved the Carol D. Soc Distinguished Graduate Mentoring Award, Agilent Early Career Professor Award (Finalist), NSF CAREER Award, and the SPIE Early Career Achievement Award.