∇-Prox: Differentiable Proximal Algorithm Modeling for Large-Scale Optimization

ZEQIANG LAI*, Beijing Institute of Technology, China
KAIXUAN WEI*, Princeton University, USA and McGill University, Canada
YING FU, Beijing Institute of Technology, China
PHILIPP HÄRTEL, Fraunhofer IEE, Germany and Princeton University, USA
FELIX HEIDE, Princeton University, USA

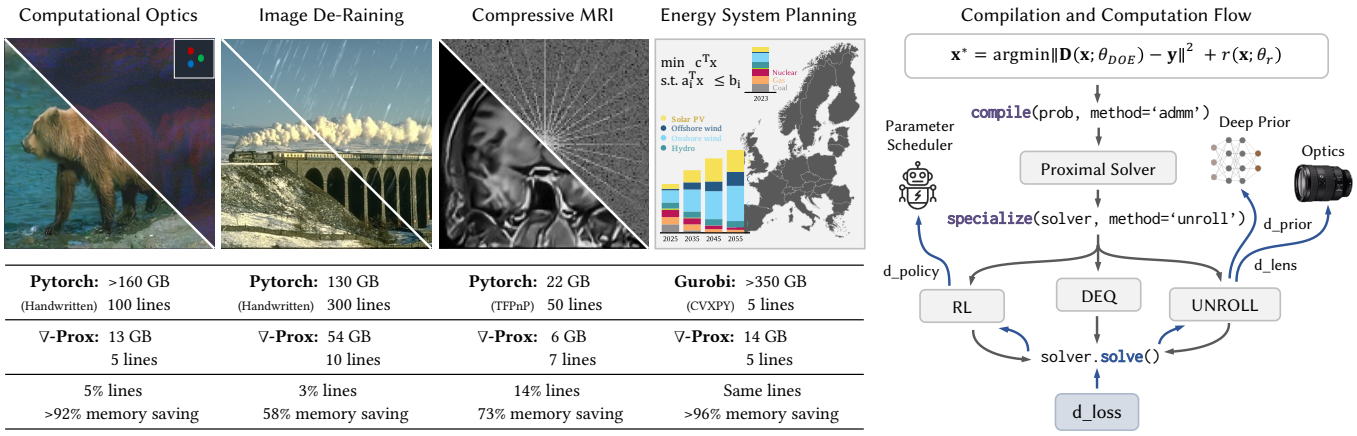


Fig. 1. ∇-Prox is a domain-specific language (DSL) and compiler that transforms optimization problems into differentiable proximal solvers. Departing from handwriting these solvers and differentiating via autograd, ∇-Prox requires only a few lines of code to define a solver that can be specialized to respect a memory or training budget by optimized algorithm unrolling, deep equilibrium learning, and deep reinforcement learning. ∇-Prox allows for rapid prototyping of learning-based bi-level optimization problems for a diverse range of applications. We compare our framework against existing methods with naive implementations. ∇-Prox is significantly more compact in terms of lines of code and compares favorably in memory consumption for diverse tasks.

Tasks across diverse application domains can be posed as large-scale optimization problems, these include graphics, vision, machine learning, imaging, health, scheduling, planning, and energy system forecasting. Independently of the application domain, proximal algorithms have emerged as a formal optimization method that successfully solves a wide array of existing problems, often exploiting problem-specific structures in the optimization. Although model-based formal optimization provides a principled approach to problem modeling with convergence guarantees, at first glance, this seems to be at odds with black-box deep learning methods. A recent line of work shows that, when combined with learning-based ingredients, model-based optimization methods are effective, interpretable, and allow for generalization to a wide spectrum of applications with little or no extra training data. However, experimenting with such hybrid approaches for different tasks by hand requires domain expertise in both proximal optimization and deep learning, which is often error-prone and time-consuming. Moreover, naively unrolling these iterative methods produces lengthy compute graphs, which when differentiated via autograd techniques results in exploding memory consumption, making batch-based training challenging.In this work, we introduce ∇-Prox, a domain-specific modeling language and compiler for large-scale optimization problems using differentiable proximal algorithms. ∇-Prox allows users to specify optimization objective functions of unknowns concisely at a high level, and intelligently compiles the problem into compute and memory-efficient differentiable solvers. One of the core features of ∇-Prox is its full differentiability, which supports hybrid model- and learning-based solvers integrating proximal optimization with neural network pipelines. Example applications of this methodology include learning-based priors and/or sample-dependent inner-loop optimization schedulers, learned with deep equilibrium learning or deep reinforcement learning. With a few lines of code, we show ∇-Prox can generate performant solvers for a range of image optimization problems, including end-to-end computational optics, image deraining, and compressive magnetic resonance imaging. We also demonstrate ∇-Prox can be used in a completely orthogonal application domain of energy system planning, an essential task in the energy crisis and the clean energy transition, where it outperforms state-of-the-art CVXPY and commercial Gurobi solvers.

*indicates equal contribution.

Authors' addresses: Zeqiang Lai, Beijing Institute of Technology, China, laizeqiang@outlook.com; Kaixuan Wei, Princeton University, USA and McGill University, Canada, kxwei@princeton.edu; ✉ Ying Fu, Beijing Institute of Technology, China, fuying@bit.edu.cn; Philipp Härtel, Fraunhofer IEE, Germany and Princeton University, USA, philipp.haertel@iee.fraunhofer.de; Felix Heide, Princeton University, USA, fheide@cs.princeton.edu.

0730-0301/2023/8-ART105 $15.00
https://doi.org/10.1145/3592144

CCS Concepts: • **Computing methodologies** → *Computational photography*; *Regularization*; • **Theory of computation** → *Continuous optimization*; *Reinforcement learning*; • **Mathematics of computing** → *Solvers*; *Automatic differentiation.*

Additional Key Words and Phrases: Computational photography, image reconstruction, differentiable optimization

ACM Reference Format:
Zeqiang Lai, Kaixuan Wei, Ying Fu, Philipp Härtel, and Felix Heide. 2023. ∇-Prox: Differentiable Proximal Algorithm Modeling for Large-Scale Optimization. *ACM Trans. Graph.* 42, 4, Article 105 (August 2023), 19 pages. https://doi.org/10.1145/3592144

1 INTRODUCTION

A broad array of problems in the sciences, engineering, economics, health, and fabrication can be cast as formal optimization problems [Wright et al. 1999]. Remarkably, proximal optimization algorithms [Parikh and Boyd 2014] have emerged across all of these diverse application domains as optimization methods as a direct result of being flexible and handling large-scale problems. In contrast to conventional interior-point or active set methods that solve minimization problems of modest size, proximal algorithms scale to constrained large-scale problems [Boyd et al. 2011] with millions of variables. Defining a class of algorithms, proximal methods sit at a higher level of abstraction than typical optimization methods that directly evaluate first and second-order derivatives for a given problem. Instead, proximal algorithms interact with proximal functions, which themselves can be minimization problems. As such, task-specific proximal algorithms are capable of exploiting application-specific problem structures in domains such as imaging optimization problems [Heide et al. 2016], control and scheduling problems [Parikh and Boyd 2014]. However, although this proximal algorithm modeling provides a principled approach with rigorous convergence guarantee [Komodakis and Pesquet 2015; Nishihara et al. 2015; Wang et al. 2019], existing model-based formal optimization methods appear to be at odds with black-box deep learning methods, which also solve high-dimensional data fitting problems, and, albeit missing convergence guarantees, have been broadly deployed for inverse problems and prediction tasks across almost all application domains [Almagro Armenteros et al. 2019; Goodfellow et al. 2016; Lai et al. 2022; LeCun et al. 2015; Ongie et al. 2020; Ozturk et al. 2020; Silver et al. 2017; Zhao et al. 2019].

Indeed, learning-based methods, especially deep neural networks represent a different paradigm that extracts task-specific priors from data, in lieu of formalized domain knowledge exploited in model-based optimization methods. As a direct result, black-box deep learning methods can learn to fit high-dimensional and complex data with expressiveness far beyond existing analytical models but have trouble generalizing to unseen domains where the data distribution does not resemble that of the training set. In contrast, model-based optimization methods are less expressive but often generalize more effectively.

Recently, these benefits and drawbacks have spurred strong interest in developing hybrid methods that embrace the best of both worlds. Researchers have combined model-based optimization methods with learning-based components, allowing for effective, interpretable, and generalizable methods with little or no extra training data [Diamond et al. 2017; Kamilov et al. 2023; Monga et al. 2021; Shlezinger et al. 2020; Wei et al. 2022a; Zhang et al. 2021]. However, implementing and prototyping such hybrid methods for different tasks by hand requires *domain expertise on both proximal optimization and deep learning*. While modern deep-learning modeling languages allow for fast prototyping, implementing proximal algorithms is often error-prone and time-consuming. This inherent difficulty stems from the ambiguities when formulating a certain (possibly ill-posed) problem as an optimization function and compiling this optimization function to a proximal solver. Finding an effective optimization method thus typically requires exploring a large space of problem formulations, translations, and solvers. This is even more challenging when considering differentiable solver designs that have to account for loop termination and many distinct ways to construct a hybrid pipeline taking other differentiable algorithm components, such as a downstream neural network, into account. As a result, existing methods mostly rely on unrolling these iterative optimization methods to produce lengthy compute graphs that can be differentiated via autograd techniques. This results in exploding memory consumption and making batch-based training challenging without costly GPU hardware with hundreds of GBs of memory available (see Figure 1).

In this work, we address these challenges by introducing ∇*-Prox*, a domain-specific modeling language and compiler for large-scale optimization problems with differentiable proximal algorithms. Although not limited to a specific application domain, we focus our work on large-scale image optimization problems. As such, we build on the proximal image optimization framework *ProxImaL* [Heide et al. 2016]. ∇-Prox allows users to specify optimization objective functions of unknowns concisely at a high level using an intuitive syntax that follows the mathematical objectives and then compiles the problem into efficient differentiable solvers exploiting problem-specific structures automatically. It inherits most assets from ProxImaL, *i.e.*, the compiler pipeline consisting of problem rewriting, splitting, and scaling. But, unlike ProxImaL, ∇-Prox supports full differentiability, which goes beyond conventional auto-differentiation mechanisms [Paszke et al. 2017, 2019]. Specifically, ∇-Prox supports equilibrium learning and reinforcement learning based bi-level optimization schemes that do not require unrolling algorithms into lengthy compute graphs. As such, the proposed DSL supports hybrid model-based and learning-based solvers that merge proximal optimization with deep neural network pipelines in a structured manner. The computational graphs and gradients computed by ∇-Prox seamlessly integrate with modern deep learning modeling frameworks such as PyTorch.

With these differentiable solvers in hand, ∇-Prox enables users to construct hybrid solvers with learning-based task-oriented priors and/or sample-dependent inner-loop schedulers. Based on memory and compute resources available for training, users can train hybrid solvers with distinct learning strategies including algorithm unrolling [Diamond et al. 2017; Monga et al. 2021], deep equilibrium learning [Bai et al. 2019; Gilton et al. 2021] and deep reinforcement learning [François-Lavet et al. 2018; Wei et al. 2020]. As such, the

proposed method allows for rapid experimentation with a variety of learned solvers and training approaches without the pain of manually implementing these methods and training schemes.

We validate the utility of ∇-Prox on a diverse set of imaging applications, including end-to-end computational optics that jointly optimizes diffractive optical element and image reconstruction (joint deconvolution and denoising) algorithm, image deraining, and compressive magnetic resonance imaging. In these applications, differentiable components are learning-based priors and/or sample-dependent inner-loop schedulers as well as the whole proximal optimization pipeline including the parameters themselves. We find that, in many cases, a few lines of ∇-Prox code can generate highly efficient differentiable solvers that achieve state-of-the-art results when trained in an end-to-end fashion. We also show ∇-Prox can be effective in a completely orthogonal application domain of energy system analysis and planning, an essential task in energy security and the transition to climate neutrality.

The main contributions in this work are as follows:

- We introduce ∇-Prox, a domain-specific modeling language and compiler tailored for large-scale optimization using differentiable proximal algorithms. The compiler takes a user problem description and solver choice as inputs, and then automatically compiles them into an efficient differentiable solver that can be combined with other differentiable algorithm components.
- We devise ∇-Prox in a fully differentiable manner, which supports hybrid model-based and learning-based solvers blending proximal optimization with neural network pipelines. To tackle memory and compute in bi-level optimization, ∇-Prox employs reinforcement and deep equilibrium learning beyond unrolling.
- We validate that ∇-Prox is easy to use yet highly performant, achieving state-of-the-art results on a variety of large-scale optimization problems on visual data. We also demonstrate that ∇-Prox allows applying differentiable proximal algorithms to orthogonal application domains.

All code, examples, and documentation are publicly available at https://github.com/princeton-computational-imaging/Delta-Prox.

2 RELATED WORK

Proximal Optimization. Tracing back to the 17th century, to find optima of functions under certain conditions, Fermat and Lagrange found closed-form formulae, while Newton and Gauss proposed iterative methods to find an optimum as a sequence of smaller optimization steps, each moving closer to the optimum. Since Moreau's proximation theorem [Moreau 1965] in the middle of the last century, a large body of work has approached large-scale optimization problems using operator-splitting methods and proximal algorithms [Boyd et al. 2011; Parikh and Boyd 2014]. This includes successful proximal gradient descent (*a.k.a.* forward–backward splitting [Bruck Jr 1975]) and its accelerated versions such as fast iterative shrinkage/thresholding algorithm (FISTA) [Beck and Teboulle 2009], proximal point method [Rockafellar 1976], alternating direction method of multipliers (ADMM) [Gabay and Mercier 1976], half-quadratic splitting (HQS) [Geman and Yang 1995], and primal-dual hybrid gradient, *e.g.*, the Chambolle-Pock algorithm [Chambolle and Pock 2011]. These proximal optimization methods were initially developed to tackle the non-smoothness of convex optimization objectives that are hard to be coped with by classic methods (like Newton's method). Recent works have shown the applicability and superiority of proximal algorithms to non-convex optimization problems and even established local and global convergence guarantees under certain conditions [Attouch et al. 2013; Li and Pong 2015; Li et al. 2017; Mollenhoff et al. 2015; Nishihara et al. 2015; Wang et al. 2019].

Domain-specific Languages for Optimization, Graphics, and Vision. To facilitate the fast development and prototyping of optimization solvers tailored for specific problems, a wide array of domain-specific modeling languages (DSL) for optimization exists. CVX [Grant and Boyd 2014] (and its siblings CVXPY [Diamond and Boyd 2016] and Convex.jl [Udell et al. 2014]) are popular examples that have been applied across disciplines. While these DSLs are expressive, they are restricted to convex optimization, and they struggle to scale to large-scale (image) optimization problems with millions of variables involved. A line of work investigates scaling these optimization DSLs to larger problems, especially prevalent in computer graphics and vision, which includes methods that recognize and exploit fast proximal operators and linear transforms [Becker et al. 2011; Diamond and Boyd 2015] as well as efficient matrix-free methods, symbolic differentiation, scheduling transforms for large-scale nonlinear least squares optimization [DeVito et al. 2017; Mara et al. 2021]. Most closely related to our work is ProxImaL [Heide et al. 2016], an image optimization DSL in this same line of research. ProxImaL allows formulating and compiling non-convex objectives with even non-analytical cost functions (such as image priors characterized by black-box denoisers). In contrast to ProxImaL, ∇-Prox incorporates full differentiability, including efficient training of the bi-level objective beyond autograd approaches, and, with a few lines of code, enables generations of hybrid solvers bridging proximal optimization and differentiable algorithmic ingredients such as neural network pipelines.

Optimization aside, DSLs have been vastly successful in Computer Graphics, with examples of OpenGL [Segal and Akeley 1999] and CUDA delivering rendering operations for broad real-world adoptions. Simulation DSLs such as Ebb [Bernstein et al. 2016] and Simit [Kjolstad et al. 2016] allow users to express and abstract linear algebra operations over heterogeneous data structures of complex geometry. Taichi [Hu et al. 2019a,b] decouples data structure from computation by Cartesian indexing and develops a mini-language to compose data structure hierarchies. Dr.JIT [Jakob et al. 2022] dynamically compiles differential simulations and automatically tracks the data dependency to remove redundant computation by introducing checkpointing [Chen et al. 2016] and Path Replay Backpropagation [Vicini et al. 2021]. Image processing DSLs such as Darkroom [Hegarty et al. 2014], Halide and its extensions [Adams et al. 2019; Li et al. 2018; Mullapudi et al. 2016; Ragan-Kelley et al. 2013] decouple algorithms from schedules for high-performance image processing, and can automatically generate efficient implementations for diverse computing engines such as x86, ARM, GPUs. The proposed ∇-Prox sits at a higher-level abstraction atop those

DSLs, and, while currently being implemented on top of PyTorch, could also potentially be embedded into them to take full advantage while providing flexibility for differentiable proximal algorithm modeling.

Differentiable Optimization and Learned Solvers. ∇-Prox shares its design philosophy with existing differentiable optimization methods which implement optimization algorithms with differentiable programming techniques. Several lines of research investigate differentiable optimization methods by 1) integrating solvers of optimization problems, *e.g.*, quadratic programs [Amos and Kolter 2017], convex cone programs [Agrawal et al. 2019b], general convex problems [Agrawal et al. 2019a], total variation minimization [Yeh et al. 2022] or nonlinear least squares [Pineda et al. 2022], into a deep network as individual layers to explicitly encode hard constraints or dependencies; 2) unroll/unfold a certain truncated iterative solver into a network-like architecture [Diamond et al. 2017; Dong et al. 2018; Gregor and LeCun 2010; Hershey et al. 2014; Monga et al. 2021; Sun et al. 2016; Zhang and Ghanem 2018]; 3) learn fixed point iterations via deep equilibrium learning with implicit differentiation [Bai et al. 2019; Gilton et al. 2021; Liu et al. 2022]; 4) train a neural policy network to automatically tune the internal parameters within an iterative proximal algorithm using reinforcement learning [Ichnowski et al. 2021; Wei et al. 2022a, 2020], and 5) meta-train a model-free learnable optimizer (parameterized by a recurrent neural network) on deep learning tasks by backpropagating through the optimization procedure [Andrychowicz et al. 2016; Chen et al. 2021; Li and Malik 2016; Metz et al. 2022; Wichrowska et al. 2017].

All approaches except the last category are model-based, which means they are, in part, formal optimization instead of black-box architectures. Implementing and testing the above hybrid model-based and learning-based solvers from different categories by hand is time-consuming and error-prone. ∇-Prox provides a shortcut to addressing this challenge — all these model-based algorithms are built-in components and can be implemented by just a few lines of code. This allows users to experiment with the algorithm for a given task in a rapid prototyping fashion, allowing for task-driven optimization method design.

3 FORMULATING PROXIMAL OPTIMIZATION

In this section, we formalize the optimization problems that the proposed domain-specific language and compiler operate on. We consider continuous optimization problems that can be solved by proximal algorithms. To this end, we use a high-level abstraction that allows converting problems into mathematical and programming representations intuitively. Specifically, we formulate problems as general optimization problems with a sum of penalties and a list of constraints. While summed-penalty representations are commonly found in the literature, we extend them with optimizable parameters. With the mathematical representation of these problems in hand, we show how we can cast them into programs with our domain-specific modeling language.

3.1 Canonical Form

We formulate a canonical optimization problem to which we convert all problems considered in this work. We aim to find unknown variables $\mathbf{x} \in \mathbb{R}^n$ that minimize an objective formulated as a sum of penalties f_i on linear transforms $\mathbf{K}_i\mathbf{x}$ with possibly additional constraints c_j

$$\begin{aligned} \underset{\mathbf{x}}{\operatorname{argmin}} \ & \sum_{i=1}^{I} f_i\left(\mathbf{K}_i\left(\mathbf{x};\, \theta_i^K\right);\, \theta_i^f\right), \\ & s.t.\ c_j\left(\mathbf{x};\, \theta_j^c\right) = 0,\ \forall j = 1, ..., J, \end{aligned} \tag{1}$$

where the linear operator $\mathbf{K}_i \in \mathbb{R}^{m_i \times n}$ projects the unknowns to the inputs of the penalty functions $f_i : \mathbb{R}^{m_i} \to \mathbb{R}$, and the constraints $c_j : \mathbb{R}^n \to \mathbb{R}$ can be imposed by either equality or inequality[1]. Here, the terms f_i, $\mathbf{K}_i$ and c_j are optionally parameterized by a set of latent parameters, *i.e.*, θ_i^f, θ_i^K, θ_j^c respectively[2], assuming their differentiability as in the case of a neural network or differentiable physical forward model. Once compiled as a differentiable solver, this canonical form enables us to implement backpropagation to optimize both latent parameters as well as differentiable internal solver parameters using data-driven end-to-end learning.

Example Problem. The canonical form (1) covers a wide array of continuous optimization problems across domains. We focus in this work on image optimization problems and, next, we describe example problems to illustrate the design rationale and core features behind ∇-Prox. In image optimization problems, one typically aims at recovering an underlying unknown image $\mathbf{x} \in \mathbb{R}^n$ from noisy and/or incomplete measurements $\mathbf{y} \in \mathbb{R}^m$. Derived as Bayesian maximum-a-posteriori point estimates [Chambolle and Pock 2016; Xu et al. 2020], the penalty functions typically include a data fidelity term measuring consistency between the reconstructed image and measured data, and a bank of regularizers that enforce prior knowledge of the unknowns in the Bayesian sense.

As a specific example, we consider end-to-end computational optics [Sitzmann et al. 2018; Tseng et al. 2021] that jointly optimizes a diffractive optical element (DOE) and an image reconstruction (joint deconvolution and denoising) algorithm, where the observation $\mathbf{y}$ is obtained by convolving a clear image x by the point spread function (PSF) of the DOE as following

$$\mathbf{y} = \mathbf{D}\left(\mathbf{x};\, \theta_{DOE}\right) + \epsilon,$$

where $\mathbf{D}(\cdot;\, \theta_{DOE})$ indicates a shift-invariant convolution process with an optical kernel, *i.e.* PSF, derived from a DOE image formation model parameterized by θ_{DOE}, and ϵ is measurement noise, *e.g.*, Poissionian-Gaussian noise. As discussed before, to reconstruct target image $\mathbf{x}$ from noise-contaminated measurements $\mathbf{y}$, we minimize the sum of a data-fidelity f and regularizer term r as

$$\min_{x \in \mathbb{R}^n} f\left(\mathbf{D}\left(\mathbf{x};\, \theta_{DOE}\right)\right) + r\left(\mathbf{x};\, \theta_r\right). \tag{2}$$

There are many choices for f and r. For example, one might define f as a sum-of-squares error, a Huber loss, or a Poisson negative

[1]Without loss of generality, the inequality constraint $c_j(x) \leq 0$ could be transformed into an equality constraint by either introducing additional non-negative slack variables (for affine functions) or converting it into $\hat{c}_j(x) = 0$ where $\hat{c}_j(x) \coloneqq \max\{0, c_j(x)\}^2$, see [Giesen and Laue 2016] for general non-linear functions.
[2]We omit θ if θ is a null set ϕ, which implies the operator itself does not hinge on any extra latent parameters.

log-likelihood penalty, depending on the measurement noise distribution. Crafting novel regularizers r plays a key role in the development of optimization-based methods for image reconstruction [Gu et al. 2017; Mairal et al. 2007; Osher et al. 2005; Venkatakrishnan et al. 2013; Xu et al. 2020; Zoran and Weiss 2011]. The regularizer r may be a sparsity-promoting penalty such as total variation [Chambolle and Pock 2016]; data range constraints such as non-negativity; or an implicit image prior characterized by an image denoising algorithm [Tian et al. 2020; Wei et al. 2022b; Zhang et al. 2017a] such as non-local means [Buades et al. 2005] or a convolutional neural network (CNN) [Zhang et al. 2017a]. In practice, researchers often employ a mixture of several regularizers, which is what we assume in our example problem

$$r(\mathbf{x};\theta_r) = \lambda g(\mathbf{x};\theta_r) + I_{[0,\infty)}(\mathbf{x}), \tag{3}$$

where $\lambda \geq 0, g(\cdot;\theta_r)$ denotes an image prior implicitly modeled in an off-the-shelf denoiser (*a.k.a.* Plug-and-Play prior [Venkatakrishnan et al. 2013]). In this example, we use a CNN-based denoiser with U-Net architecture [Ronneberger et al. 2015] parameterized by network weights θ_r. $I_{[0,\infty)}(\mathbf{x})$ is an indicator function whose value is 0 if $\mathbf{x}$ lies at interval $[0,\infty)$ and ∞ otherwise. This compound penalty function encodes both analytical explicit non-negativity priors as well as implicit priors that rely on the properties of image denoisers [Xu et al. 2020]. Problem (4) lists the full optimization problem, where $r(\mathbf{x})$ is defined in (3). The reformulated problem (5) represents one possible choice to transform (4) into the canonical form (1).

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \ \|\mathbf{D}(\mathbf{x};\,\theta_{DOE}) - \mathbf{y}\|_2^2 + r(\mathbf{x};\theta_r) \tag{4}$$

$$\begin{aligned} f_1(\mathbf{v}) &= \|\mathbf{v}-\mathbf{y}\|_2^2, & \mathbf{K}_1 &= \mathbf{D}(\cdot;\,\theta_{DOE}) \\ f_2(\mathbf{v}) &= \lambda g(\mathbf{v};\theta_r), & \mathbf{K}_2 &= \mathbf{I} \\ f_3(\mathbf{v}) &= I_{[0,\infty)}(\mathbf{v}), & \mathbf{K}_3 &= \mathbf{I} \end{aligned} \tag{5}$$

While multiple ways exist to cast the problem into canonical form, the specific choice affects the solver performance. This translation is typically done by hand using expert knowledge. Instead of handcrafted translations, the proposed method provides automatic strategies to find an optimal reformulation.

Once the optimization problem in the canonical form is formulated, we compile it into a specific differentiable solver based on the choice of the proximal algorithm. ∇-Prox supports a variety of typical proximal algorithms, *e.g.*, ADMM, half-quadratic splitting, and Pock-Chambolle algorithm. These algorithms employ a class of abstract operators, namely *proximal operators*, as their base operations instead of the computation of gradients and Hessians in classic optimization algorithms. The proximal operators in our formulation are also differentiable. The evaluation of the proximal operator of a weighted penalty function μf itself involves solving a small convex/non-convex optimization problem as

$$\operatorname{prox}_{\mu f}(\mathbf{v};\,\theta^f) = \underset{\mathbf{x}}{\operatorname{argmin}} \left(f(\mathbf{x};\,\theta^f) + \frac{1}{2\mu}\|\mathbf{x}-\mathbf{v}\|_2^2 \right). \tag{6}$$

These subproblems can be solved with standard methods, but they often admit closed-form solutions or can be solved very quickly with simple specialized methods [Parikh and Boyd 2014]. Note also that the formulation of the proximal operator (6) is mathematically equivalent to the regularized denoising, suggesting one might replace the proximal operator $\operatorname{prox}_{\sigma^2 f}$ by any denoiser $\mathcal{H}_\sigma(\cdot;\theta^f)$ with noise level σ. This yields a framework coined Plug-and-Play (PnP) prior [Venkatakrishnan et al. 2013], where the corresponding penalty function $f(\mathbf{x};\theta^f)$ is implicitly defined by the denoiser itself with parameters θ^f, *i.e.*, the denoiser network weights if the denoiser is a neural network.

3.2 Programming Interface

To describe large-scale optimization problems, ∇-Prox extends the DSL from ProxImaL [Heide et al. 2016] — to specify the optimization objective functions conditioned on parameters (of the objective, unknowns, or the solver itself) that can be optimized with end-to-end training. To make this document self-contained, we introduce the entire DSL here briefly and we explicitly point out differences from ProxImaL.

In ∇-Prox, we start by defining a `Variable` that represents the unknown of interest $\mathbf{x}$, and a `Placeholder` that represents the input measurements $\mathbf{y}$. Then, we define each term of an optimization objective by composing linear operators `LinOp` and proxable functions `ProxFn` with possible latent parameters `Params`. The linear operators `LinOp` themselves can be arbitrarily combined with each other to form a linear expression tree where the leaf nodes are required to be a `Variable`, which is a `LinOp` as well. Each `ProxFn` accepts a linear expression and optional keyword arguments depending on its specific signature. Once all the `ProxFn`s are defined, we mix them in a single optimization objective and wrap it with `Problem` to construct the corresponding optimization problem.

With the language syntax described above, the Problem (4) can be written as

```
        x = Variable()
        y = Placeholder(input)
data_term = sum_squares( conv(x, psf_DOE) - y )
prior_term = deep_prior(x, unet)
objective = data_term + prior_term + nonneg(x)
        p = Problem( objective )
```

where different colors are used to distinguish primitives especially `ProxFn`, `LinOp` and `Params`. In the following, we describe the core primitives and their functionality in the ∇-Prox language.

Tensor Variables. `Variable` in ∇-Prox can refer to multidimensional arrays or single-dimensional vectors, though there is no need to specify the shape beforehand. This tensor definition contrasts ProxImaL, where a problem is specifically bound to a given input shape, and users have to create new problems for inputs with different shapes. ∇-Prox instead provides a built-in mechanism to automatically infer and propagate the shape into each `LinOp` and `ProxFn` for necessary initializations before problem-solving. This new feature not only eases user burden but also allows for optimization over different input shapes in multi-task training.

Linear Operator. `LinOp` $\mathbf{K}$ maps data from a multidimensional space $\mathbb{R}^{n_1\times\cdots\times n_k}$ to another multidimensional space $\mathbb{R}^{m_1\times\cdots\times m_l}$. All linear operators in ∇-Prox (*e.g.*, `grad`, `conv`, `subsample`) are differentiable and allow to evaluate the `forward` operation $\mathbf{x} \rightarrow \mathbf{K}\mathbf{x}$ and `adjoint` operation $\mathbf{x} \rightarrow \mathbf{K}^T\mathbf{x}$. ∇-Prox supports the composition of the arbitrary number of existing linear operators and automatically

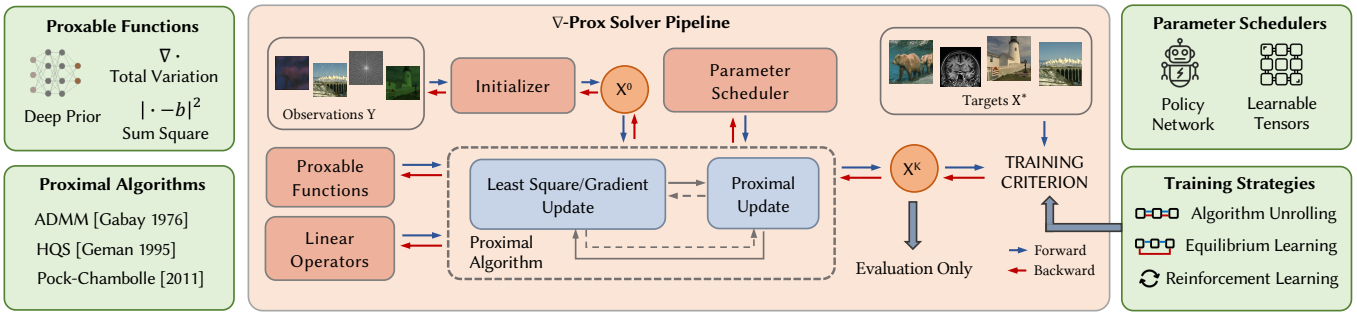


Fig. 2. Overview of the general forward and backward (gradient) computation of the compiled solver (problem-solving stage). Given the observation Y, the solver utilizes an initializer to obtain a guess X^0 for the optimization variable X. The proxable functions and the linear operators interact with the proximal algorithm with a possible additional algorithm parameter scheduler to obtain the final prediction X^N. The gradient can backpropagate through the solver to the learning-based components (green parts) by either unrolling the iterations or implicit differentiation.

converts them to a directed acyclic graph (DAG) for evaluating its linear function and adjoint, following [Diamond and Boyd 2015].

Proxable Function. `ProxFn` represents the penalty function in our canonical problem form that possesses a corresponding proximal operator, which we call the proxable function. Each proxable penalty function in ∇-Prox (*e.g.*, `sum_squares`, `norm1`, `deep_prior`[3]) inherits the base class `ProxFn` and implements the `fn.eval()` and `fn.prox()` for evaluating the function and its proximal operator, though the `fn.eval()` can be omitted if the function cannot be explicitly evaluated, as with the `deep_prior`. The proximal operator of a weighted function μf is defined in (6). All proximal operators in ∇-Prox are differentiable and effectively act as building blocks to construct the proximal differentiable solver.

Placeholder. `Placeholder` is a new primitive that is not present in ProxImaL. As the name implies, it is a placeholder for the actual data it represents. Placeholders can be reused when a problem has to be solved multiple times with different input measurements, a common pattern encountered in traversing an entire dataset for training or evaluation. Using the `Placeholder` `y`, users can freely modify the underlying data of `y` via the assignment, *i.e.*, `y.value = input`, after the problem creation, thereby avoiding problem redefinition.

Latent Parameters. `Params` serves as a primitive to enable differentiable optimization which is not supported by ProxImaL. It encodes learnable parameters associated with differentiable operators in ∇-Prox (such as `LinOp` and `ProxFn`), allowing us to adapt the behavior of the operators to the problem, given a higher-level dataset that the entire solver is optimized over. `Params` can either be explicitly defined by users (for example, manually constructing a denoising neural network `unet` as a deep prior, or parameterizing optical aberrations `psf_DOE` with DOE parameters via an image formation model) or be implicitly defined within an operator itself.

4 COMPILING DIFFERENTIABLE PROXIMAL SOLVERS

With a problem in the canonical form (1), we describe in this section how we compile it into an efficient differentiable solver with the designated proximal algorithm that ∇-Prox supports, including ADMM [Boyd et al. 2011], Pock-Chambolle [Chambolle and Pock 2011], Half Quadratic Splitting [Geman and Yang 1995], Proximal Gradient Descent [Bruck Jr 1975], and their preconditioned and linearized versions [Benning et al. 2015; Liu et al. 2019].

Once the optimization objective is specified and encompassed in the `Problem` class, it is compiled either just-in-time on a function call or explicitly via a `compile` primitive to construct a solver object that enables reusable batch processing.

```
  p = Problem( objective )
out = p.solve(method='admm')
  s = compile(p, method='admm')
out = s.solve({y: input})
```

To solve the problem, ∇-Prox first internally compiles the problem into the designated solver through a series of compilation stages (Section 4.1) and then runs the resulting solver to obtain the solution. A key difference between ∇-Prox and ProxImaL [Heide et al. 2016] is that the `solve` routine is fully differentiable so that users can seamlessly acquire the gradient of learnable components, *e.g.*, a parameterized PSF `psf_DOE` and learning-based prior `unet`, with respect to *e.g.*, a scalar cost function $\mathcal{L}$, through a single call of `L.backward()`. This enables new capabilities, *e.g.*, the specialization of existing solvers into hybrid learning-based ones via a bi-level optimization [Blondel et al. 2021] utilizing the gradients, which greatly increases the solving capabilities.

∇-Prox supports a variety of approaches for differentiating through the solver, including not only vanilla backpropagation-through-time (BPTT)/algorithm unrolling (Section 4.2) but also options for computing gradients via implicit differentiation/deep equilibrium learning (Section 4.4) or crossing the non-differentiable bottleneck with deep reinforcement learning (Section 4.5). The gradient calculations for frequently used routines are also optimized to exploit the problem structure (Section 4.3).

∇-Prox also takes care of the solver pipeline that embeds solver iterations. For example, our compiler provides a fallback option for estimating the initial guess of the solution and internal solver parameters in an automated fashion if they are not provided by the user. The termination of the optimization loop can also be determined manually or automatically with built-in termination criteria.

[3]See Supplementary Material for a complete list of linear operators and proxable functions implemented in ∇-Prox.

ALGORITHM 1: Generalized ADMM for Problem (1)

Input: Parameters $\rho, \lambda_j, \forall j \in \Phi$, number of iterations T

Initialize $\mathbf{x}$, $\mathbf{v}_j = \mathbf{x}$, $u_j = 0 \ \forall j \in \Psi$;

for $t = 1, 2, ..., T$ **do**

$$\mathbf{x}^{(t+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{i\in\Omega} f_i(\mathbf{x}) + \sum_{j\in\Psi} \frac{\rho}{2} \left\| \mathbf{K}_j\mathbf{x} - (\mathbf{v}_j^t - \mathbf{u}_j^t) \right\|^2;$$

$$\mathbf{v}_j^{(t+1)} = \operatorname{prox}_{f_j,\lambda_j} \left(\mathbf{K}_j\mathbf{x}^{(t+1)} + \mathbf{u}_j^{(t+1)} \right) \quad \forall j \in \Psi;$$

$$\mathbf{u}_j^{(t+1)} = \mathbf{u}_j^{(t)} + \left(\mathbf{K}_j\mathbf{x}^{(t+1)} - \mathbf{v}_j^{(t+1)} \right) \quad \forall j \in \Psi;$$

∇-Prox supports not only conventional termination criteria based on relative and absolute residual but also *learning-based termination* that relies on deep reinforcement learning [Wei et al. 2022a], thanks to the differentiability of the compiled solver.

4.1 Solver Compilation Pipeline

Compiling differentiable solvers in ∇-Prox typically involves a series of stages for (a) transforming a given optimization problem into an equivalent simplified one and (b) generating efficient differentiable routines for evaluating linear operators, proxable functions, and optimization updates of proximal algorithms. These include problem transformation, problem partition, preconditioning, and solver generation. The problem transformation stage intelligently translates the original problem into an equivalent form that can be solved more efficiently. The problem partition step splits the penalty functions $\sum_i f_i(\mathbf{x})$ of the problem into two parts $g(\mathbf{x}) = \sum_{f_i\in\Omega} f_i(\mathbf{x})$, $h(\mathbf{z}) = \sum_{f_i\in\Psi} f_i(\mathbf{z})$[4] considering the selected proximal algorithm. The preconditioning scales the problem to accelerate the algorithm convergence, and the final solver generation stage handles the generation of the code of the optimization loop. Note that these concepts for solver construction are directly inherited from ProxImaL, see Supplementary Material for details.

4.2 Differentiating Compiled Solvers

One of the key contributions of ∇-Prox is the differentiability of the entire solver. As shown in Figure 2, our compiler generates solvers that can backpropagate gradients through solvers to trainable parameters of learning-based components, *e.g.*, learnable linear operators [Tseng et al. 2021], proximal regularizers [Zhang et al. 2021], and internal algorithm parameter schedulers [Wei et al. 2020].

In the following, we take Alternative Direction Method of Multiplier (ADMM)[5] as an example for illustrating the basic forward and backward (gradient) computation through the solver (*a.k.a.* algorithm unrolling). The pseudo-code for ADMM is given in Algorithm 1, which, at a high level, repeatedly iterates over sequential updates on the primal variable $\mathbf{x}$, $\mathbf{v}_j$, and dual variables/multipliers $\mathbf{u}_j$. The problem partition of ADMM leaves only the `sum_squares` penalty/proxable functions for the Ω group and all other functions for the Ψ group, thus the $\mathbf{x}$ update reduces to a least-square problem, which can be solved in close-formed or via iterative linear solvers,

[4] Ω and Ψ are a partition of the set of functions $\{f_1, ..., f_I\}$ from Problem (1)

[5] Other proximal algorithms are detailed in the Supplementary Material.

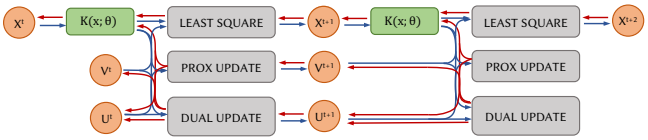


Fig. 3. Differentiation through proximal solvers can be implemented with algorithm unrolling by applying chain rule of derivatives, illustrated for two iterations here, but it often requires manual case-by-case implementation for each algorithm. With ∇-Prox, the unrolling of the proximal solver is implemented automatically for users who can just provide problem descriptions and selected algorithms. Blue arrows indicate forward computational flow and red arrows illustrate backward flow.

e.g., conjugate gradient method. All $\mathbf{v}$ updates call the proximal operators of the given penalty functions, and $\mathbf{u}$ updates are performed with a single-step gradient ascent.

Differentiating the Optimization Loop. Figure 3 provides the computational graph of the unrolled ADMM algorithm 1 truncated to two iterations. It can be observed that all the optimization variables $\mathbf{x}^{t+1}, \mathbf{v}^{t+1}, \mathbf{u}^{t+1}$ at the next step depend on variables at the current step $\mathbf{x}^t, \mathbf{v}^t, \mathbf{u}^t$ through the least square, the proximal, as well as the dual update. Thus, to differentiate through the optimization loop, we have to make every update on subproblems differentiable. ∇-Prox provides a library of differentiable linear operators and proximal functions that can be used for describing an optimization problem. Custom operators and functions can also be easily implemented with minimal code changes. Least squares subproblems appear in many proximal algorithms, including HQS, Pock-Chambolle, and ADMM. In many cases, they cannot be easily solved with exact methods that would require inverting a large-scale matrix. As such, iterative methods are often adopted, which only require the forward and adjoint evaluation of linear operators. However, differentiating these iterative routines can be non-trivial and might be less efficient with auto-diff [Baydin et al. 2018] as we usually need a large number of iterations to ensure the precision of solving results. We later show that the gradient calculation for a linear solver can be optimized with implicit differentiation on both sides of a linear system.

Differentiating Linear Operators. Many linear operators can be represented by matrix-free routines for evaluating their `forward` operations $\mathbf{x} \rightarrow \mathbf{K}\mathbf{x}$ and `adjoint` operations $\mathbf{x} \rightarrow \mathbf{K}^T\mathbf{x}$. For example, the convolution linear operator `conv` can be implemented as element-wise multiplication between inputs and convolution kernel in the frequency domain, instead of costly general matrix multiplication. This allows for reverse-mode auto-diff [Baydin et al. 2018] to efficiently differentiate through these routines. As ∇-Prox supports automatic transformation from composite linear operators to a directed acyclic graph, the differentiation through them can be achieved by evaluating their linear forward function and performing the reversed auto-diff.

Differentiating Proximal Functions. Proximal algorithms utilize the proximal operator of the penalty function to perform updates on primal variables, *e.g.*, v in ADMM. This mandates differentiability of the routines for evaluating the proximal operators to ensure the differentiability of the entire solver. It should be noted that the

proximal operator itself is an optimization problem, as shown in (6). At first glance, differentiating the proximal operator might be as difficult as differentiating the solver itself. Fortunately, most frequently used proximal operators can be expressed in closed form or represented with conventional neural networks, and ∇-Prox subsequently provides auto-diff-compatible implementations of a series of differentiable routines for common proximal operators/functions, *e.g.*, `sum_squares`, `norm1`, `deep_prior`.

4.3 Optimizing Gradient Calculations

Unrolling optimization as a computational graph allows for auto-differentiation [Paszke et al. 2019] for gradient computation. Despite the convenience and versatility, this approach results in clock time and memory complexity that scales with the number of variables (*e.g.*, millions for image optimization problems) and the number of algorithm iterations for forward-mode and backward-mode auto-diff, respectively. This makes *vanilla auto-diff impractical for the large-scale solvers* we focus on in this work. Next, we demonstrate how gradient calculations are optimized in ∇-Prox to facilitate large-scale gradient evaluations in the proposed proximal solvers.

Matrix-free Differentiable Linear Solver. Solving a linear system $\mathbf{Kx} = \mathbf{b}$ is frequently encountered in proximal algorithms. ∇-Prox supports both direct and iterative methods for solving this problem. For example, consider Problem (4), if we choose the problem partition $\Omega = \{f_1\}$, $\Psi = \{f_2, f_3\}$, then we can solve the least square subproblem of ADMM exactly by solving a linear system (the hyperparameters ρ and λ are omitted for brevity).

$$\begin{aligned}\mathbf{x}^{t+1} &= \operatorname*{argmin}_{\mathbf{x}} \left\| \begin{bmatrix} \mathbf{D} \\ \mathbf{I} \\ \mathbf{I} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{y} \\ \mathbf{v}_2^t - \mathbf{u}_2^t \\ \mathbf{v}_3^t - \mathbf{u}_3^t \end{bmatrix} \right\|_2^2 \\ &= \left(\mathbf{D}^T\mathbf{D} + 2\mathbf{I}\right)^{-1} \left(\mathbf{D}^T\mathbf{y} + \sum_j^2 (\mathbf{v}_j^k - \mathbf{u}_j^k)\right),\end{aligned} \tag{7}$$

where $\mathbf{K} = [\mathbf{D}; \mathbf{I}; \mathbf{I}]$ and $\mathbf{b} = [\mathbf{y}; \mathbf{v}_2^t - \mathbf{u}_2^t; \mathbf{v}_3^t - \mathbf{u}_3^t]$ The solution above relies on being mappable to a fast implementation for computing matrix inverse $(\mathbf{D}^T\mathbf{D} + 2\mathbf{I})^{-1}$, which can generally be accessible if the Gram matrices of all the linear operators ($\mathbf{D}$ and $\mathbf{I}$ in this case) are diagonal or diagonal in the frequency domain[6]. If a fast implementation is unavailable, the default choice resorts to iterative methods like the conjugate gradient method.

Auto-diff can be used to efficiently differentiate fast direct linear solvers but is often intractable for iterative linear solvers. In ∇-Prox, we provide an optimized routine to compute the analytic derivatives of linear (iterative) solver outputs with respect to the parameters of linear operators θ and $\mathbf{b}$. Specifically, we differentiate both sides of $\mathbf{Kx} = \mathbf{b}$ to obtain the derivatives $\frac{\partial \mathbf{x}}{\partial \mathbf{b}}$ and $\frac{\partial \mathbf{x}}{\partial \theta}$ as

$$\partial\mathbf{K}\,\mathbf{x} + \mathbf{K}\,\partial\mathbf{x} = \partial\mathbf{b}$$

$$\partial\mathbf{x} = \mathbf{K}^{-1}(-\partial\mathbf{K}\,\mathbf{x} + \partial\mathbf{b}),$$

from which the gradient $\frac{\partial \mathbf{x}}{\partial \mathbf{b}} = \mathbf{K}^{-1}$ can be easily derived. Typically, we are more interested in the gradient of $\mathbf{b}$ with respect to a scalar loss function $\mathcal{L}$, which can be obtained with the chain rule of differential calculus.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \left(\frac{\partial \mathbf{x}}{\partial \mathbf{b}}\right)^T \frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{K}^{-T} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} \tag{8}$$

Since all the linear operators in our system are matrix-free, we cannot directly evaluate (8) for gradient computing. Instead, we transform (8) into

$$\mathbf{K}^T \frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}} \tag{9}$$

where the right-hand-side is the Jacobian of $\mathcal{L}$ with respect to $\mathbf{x}$ that can be efficiently evaluated with auto-diff systems. The calculation of gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ has thus been converted to solving a linear system, requiring significantly less memory.

Similarly, the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ with respect to the parameters θ of the linear operator $\mathbf{K}$ can be derived as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left(\frac{\partial \mathbf{x}}{\partial \theta}\right)^T \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \quad \text{s.t.} \quad \frac{\partial \mathbf{x}}{\partial \theta} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta}\mathbf{x}. \tag{10}$$

Again, $\frac{\partial \mathbf{K}}{\partial \theta}$ cannot be evaluated directly as we consider matrix-free linear operators. To circumvent this obstacle, we use the fact that $\frac{\partial \mathbf{K}}{\partial \theta}\mathbf{x} = \frac{\partial \mathbf{b}}{\partial \theta}$ to transform (10) into

$$\mathbf{K}\frac{\partial \mathbf{x}}{\partial \theta} = -\frac{\partial \mathbf{b}}{\partial \theta}, \tag{11}$$

where $\frac{\partial \mathbf{b}}{\partial \theta}$ can be computed by backpropagating the forward computation $\mathbf{Kx} = \mathbf{b}$. As such, the calculation of gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ and $\frac{\partial \mathcal{L}}{\partial \theta}$ is converted to solving linear systems during backpropagation without requiring storing intermediate states, thereby significantly reducing memory consumption meanwhile saving computation time.

Eliminating Calculations. Linear system solver aside, ∇-Prox automatically detects and removes duplicate or unnecessary forward and gradient computations to further increase efficiency. This is achieved by traversing a computational graph that encompasses all linear operators and proxable functions of a given problem, and iteratively performing constant folding, proxable function fusion, and linear operator absorption on the graph. Please refer to the Supplementary Material for details.

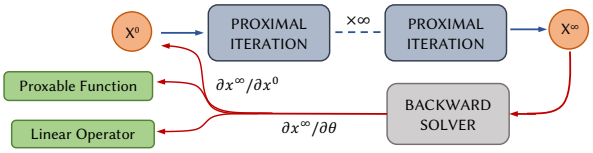
4.4 Implicit Differentiation for Proximal Algorithms

Vanilla algorithm unrolling is suitable for optimization problems that are solved with a fixed and small number of iterations. However, as the number of iterations increases, the computational and memory cost of algorithm unrolling quickly becomes impractical since we need to store all the intermediate variables along the forward iteration trajectory for the backward automatic differentiation.

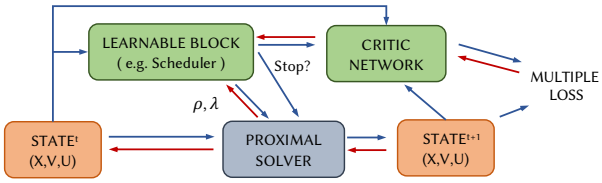
To address this, ∇-Prox supports deep equilibrium learning (DEQ) [Bai et al. 2019; Gilton et al. 2021] that incorporates implicit differentiation to effectively backpropagate the gradient through a proximal optimization solver with, *conceptually, an infinite number of iterations*, as shown in Figure 4a. The adaptation from the original solver into its DEQ version can be easily achieved with a single line of code in ∇-Prox using a `specialize` primitive.

```
s2 = specialize(s, method='deq')
```

[6] ∇-Prox also implements several special fast solutions of the linear system even when the Gram matrices of the linear operators are not all diagonal.



(a) ∇-Prox with Deep equilibrium Learning



(b) ∇-Prox with Deep Reinforcement Learning

Fig. 4. (a) ∇-Prox makes it possible to specialize the solver into deep equilibrium solvers whose gradients can be computed directly without the need of reversing the entire forward optimization iterations. The specialized solver requires significantly less memory and is often more favorable for cases where backpropagating the long iteration is prohibitive. (b) Our compiler utilizes deep reinforcement learning to cross the non-differentiable routines, *e.g.*, the stopping signal predicted by an internal parameter scheduler (parameterized by a policy network).

With this specialization, the forward process of the new solver is translated into solving a fixed-point problem[7],

$$\mathbf{X}^{\infty} = f_{\theta}\left(\mathbf{X}^{\infty}; \mathbf{M}\right), \tag{12}$$

where $\mathbf{X}^{\infty}$ is a compound variable absorbing all optimization variables, *e.g.*, $\mathbf{x}, \mathbf{v}, \mathbf{u}$ for ADMM, f_{θ} represents a composite update of optimization variables $\mathbf{x}, \mathbf{v}, \mathbf{u}$, and $\mathbf{M}$ denotes any other auxiliary inputs, *e.g.*, hyperparameters ρ and observation $\mathbf{y}$.

This fixed-point problem can be solved via vanilla fixed-point iteration or Anderson acceleration [Walker and Ni 2011]. The gradient of the parameters θ with respect to some scalar loss $\mathcal{L}$ can be derived using the implicit differentiation theorem without the need of going backward the entire forward trajectory. Specifically, the backward process first applies the chain rule as,

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left(\frac{\partial \mathbf{X}^{\infty}}{\partial \theta}\right)^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{X}^{\infty}}. \tag{13}$$

The second term $\frac{\partial \mathcal{L}}{\partial \mathbf{X}^{\infty}}$ is the gradient of the scalar loss with respect to the equilibrium point, which can usually be analytically computed or tracked by auto-diff. The first term $\frac{\partial \mathbf{X}^{\infty}}{\partial \theta}$ is the gradient of the equilibrium point with respect to the parameters, which can be computationally expensive to derive with auto-differentiation. To compute it efficiently, we differentiate both sides of (12) with respect to θ to obtain

$$\frac{\partial \mathbf{X}^{\infty}}{\partial \theta} = \left(\mathbf{I} - \frac{\partial f_{\theta}(\mathbf{X}^{\infty}; \mathbf{M})}{\partial \mathbf{X}^{\infty}}\right)^{-1} \frac{\partial f_{\theta}\left(\mathbf{X}^{\infty}; \mathbf{M}\right)}{\partial \theta}. \tag{14}$$

By plugging (14) into (13), we derive an equation for computing $\frac{\partial \mathcal{L}}{\partial \theta}$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \underbrace{\left(\frac{\partial f_{\theta}\left(\mathbf{X}^{\infty}; \mathbf{M}\right)}{\partial \theta}\right)^{T}}_{\text{Single-step Gradient}} \underbrace{\left(\mathbf{I} - \frac{\partial f_{\theta}(\mathbf{X}^{\infty}; \mathbf{M})}{\partial \mathbf{X}^{\infty}}\right)^{-T} \frac{\partial \mathcal{L}}{\partial \mathbf{X}^{\infty}}}_{\text{Inverse Jacobian-vector}}. \tag{15}$$

The inverse Jacobian-vector product can be approximated by another fixed-point problem as the step 2 in Algorithm 2, while the single-step gradient can be computed via auto-diff.

To wrap up, DEQ (Algorithm 2) takes a fixed-point iteration view of the infinite loop of proximal algorithms and utilizes the implicit differentiation theorem to allow for analytic gradient backpropagation through an infinite number of iterations, avoiding the prohibitive memory incurred by vanilla algorithm unrolling.

ALGORITHM 2: Backward Gradient Calculation with DEQ.

Input: Composed iterative mapping f_{θ}, Equilibrium point $\mathbf{X}^{\infty}$, Auxiliary input $\mathbf{M}$, Loss function $\mathcal{L}$.

1 Evaluate $\frac{\partial \mathcal{L}}{\partial \mathbf{X}^{\infty}}$ through auto-differentiation.

2 Derive an approximate fixed-point β^{∞} of the equation

$$\beta = \left(\frac{\partial f_{\theta}(\mathbf{X}^{\infty}; \mathbf{M})}{\partial \mathbf{X}^{\infty}}\right)^{T} \beta + \frac{\partial \mathcal{L}}{\partial \mathbf{X}^{\infty}}.$$

3 Return $\frac{\partial \mathcal{L}}{\partial \theta} = \left(\frac{\partial f_{\theta}(\mathbf{X}^{\infty}; \mathbf{M})}{\partial \theta}\right)^{T} \beta^{\infty}$

4.5 Reinforcement Learning for Proximal Algorithms

The selection of algorithm parameters (*e.g.*, the penalty strength ρ of the multipliers and balancing parameters λ), as well as the termination criterion, are important aspects that significantly affect the performance of proximal algorithms. However, in practice, the optimal parameters for different problems in different proximal algorithms often require tedious case-by-case tuning. To address this issue, learning-based internal parameter schedulers [Wei et al. 2020] have been proposed to automatically estimate the best parameters and terminal time based on the observation of intermediate states of optimization iterations. In practice, training these schedulers is non-trivial because the subsequence of the optimal parameter sequence might not be optimal, which means the greedy search can hardly be effectively used. However, to obtain better schedulers with naïve backpropagation, one would need to backpropagate through the entire optimization loop, resulting in exploding memory consumption. Moreover, learning the sample-dependent termination criterion entails backpropagation through non-differentiable terminal time, which cannot be handled in algorithm unrolling and/or deep equilibrium learning.

To tackle this challenge, ∇-Prox allows for reinforcement-learning-based training strategies, which can be called with two lines of code.

```
s2 = specialize(s, method='rl', policy='resnet')
s2 = s2.train(dataset, {'val': val_dataset}, ...)
```

Specifically, users first `specialize` the original solver by specifying the method as `rl` and choosing the policy/scheduler network, *e.g.*, a vanilla `resnet` [He et al. 2016]. Then, the training of the policy

[7]Note that DEQ has been proposed for infinite-depth networks, an area orthogonal to optimization. In the Supplementary Material, we detail how to transform the proximal optimization routines into a fixed-point iteration.

network can be easily done by calling the `train` method of the specialized solver.

One of the advantages of deep reinforcement learning over existing algorithm unrolling is that we do not need to evaluate the entire optimization trajectory to train the learned components of the solver. Similar to deep equilibrium learning, this reduces the memory requirement for backpropagating the gradients. Another advantage of deep reinforcement learning is the feasibility to bypass the non-differentiable parts, *e.g.*, the termination signal of the optimization loop, to train the required learning-based components.

To learn proximal algorithms in a deep reinforcement approach, we formulate the input and output of learnable parts as a series of *state* and *action* and treat the optimization loop as an *environment*. Then, we can decouple the evaluation of the entire trajectory and the update of trainable parameters into two separate stages. The former stage collects the training data of *states* and can be performed without any gradient calculation. The second stage performs the parameter update with different deep reinforcement learning algorithms, *e.g.*, advantage actor-critic [Mnih et al. 2016] and deterministic policy gradient [Silver et al. 2014]. We note that one of the important advantages of these methods is that we only need to evaluate the optimization loop for only one or a few iterations, and another network, *e.g.*, the critic network in the actor-critic algorithm, predicts the long-term reward to achieve the end-to-end training. Please refer to the Supplementary Material for more details on how these algorithms apply to our proximal framework.

4.6 Implementation

∇-Prox is embedded in Python and works as an external library. All the routines of solver algorithms, linear operators, and proximal operators are implemented following the PyTorch standard on `nn.Module`, so that ∇-Prox can seamlessly be integrated into PyTorch to embrace the broader learning-based components created by the PyTorch community. We utilize the basic operators of PyTorch to build up the ∇-Prox's operators whenever it is possible. Differentiation through these basic routines can then be handled by PyTorch auto-diff without reinventing the wheels. For other routines that are non-differentiable or slow to be differentiated with vanilla PyTorch, we customize their backward/training processes with gradient optimizations, deep equilibrium learning, and deep reinforcement learning. We implement the backward customization with a backward hook in PyTorch so that the internal differentiation is transparent to users.

5 APPLICATIONS AND ANALYSIS

In this section, we evaluate ∇-Prox applied ot diverse large-scale optimization problems in imaging and energy system optimization. We validate that, by relying on bi-level optimization using the proposed differentiable solvers, we can compile hybrid solvers that achieve state-of-the-art quality with a few lines of code for each application. Each application benefits from different aspects of differentiability, including model parameters optimization, solver hyperparameter setting, or learning effective priors in an end-to-end fashion. We discuss each of these benefits for the respective application. We also discuss the effect of different proximal algorithms, priors, and parameter set strategies.

5.1 End-to-End Computational Optics — Learning Model Parameters and Priors

Conventional imaging systems employ compound refractive lens systems that are typically hand-engineered for image quality in isolation of the downstream camera task. Departing from this design paradigm, a growing body of work in computational imaging [Haim et al. 2018; Horstmeyer et al. 2017; Peng et al. 2019; Shi et al. 2022; Sitzmann et al. 2018; Sun et al. 2020a] has explored the design of specialized lens system with diffractive optical elements (DOEs). These DOEs allow for fine-grained modulation of the phase of incident light via diffraction, at the cost of added chromatic aberration that these methods aim to eliminate post-capture. Employing a learnable reconstruction algorithm along with a differentiable wave optics forward model, task-specific computational cameras methods have been proposed for diverse applications from microscopy [Horstmeyer et al. 2017] to single-shot high-dynamic range imaging [Sun et al. 2020a].

Here, we use ∇-Prox to model reconstruction algorithms with a differentiable proximal solver that we co-design with a single thin lens, parameterized by a diffractive phase plate. We adopt a Fourier optics image formation model assuming paraxial optics (*i.e.* small field of view) and incident plane wave (that implies objects at optical infinity), which renders sensor measurement as a shift-invariant convolution of an image and a point spread function (PSF) parameterized by the DOE. Off-axis aberrations like comatic aberration are neglected. The on-axis PSF can then be derived by simulating a free-space wave propagation of a wavefront (plane wave modulated by wavelength-dependent phase profiles of the DOE) propagated to the sensor plane, see [Goodman and Sutton 1996; Sitzmann et al. 2018] for details.

In this experiment, we consider a lens-to-sensor distance of 15 mm, and an aperture size of 3 mm, implying an f-number of 5. The sensor is modeled with a resolution of 748×748 and a pixel pitch of 4 μm. The wave propagation simulation is discretized in a 2μm grid (1496×1496), which meets the computational sampling requirements of a transfer-function-based Fresnel propagator, see again [Goodman and Sutton 1996]. The DOE is modeled as an unconstrained height map. Given the refractive index of the DOE material, we can compute the phase delay in each pixel with respect to different wavelengths, for which we consider 460, 550, 640 nm wavelengths for color imaging. As such, we simulate a three-channel PSF parameterized by the DOE height map θ_h. Since all operations involved are differentiable, we implement this model as a differentiable linear operator with learnable parameters θ_h that can be jointly optimized with a learnable image prior and parameters of the proximal solver within ∇-Prox. The problem of co-designing the DOE and a proximal algorithm that performs image reconstruction can be expressed in a few lines of code:

```
          x = Variable()
        psf = build_doe_model().psf
data_term = sum_squares(conv(x, psf) - y)
 reg_term = deep_prior(x, 'ffdnet', trainable=True)
objective = data_term + reg_term
```

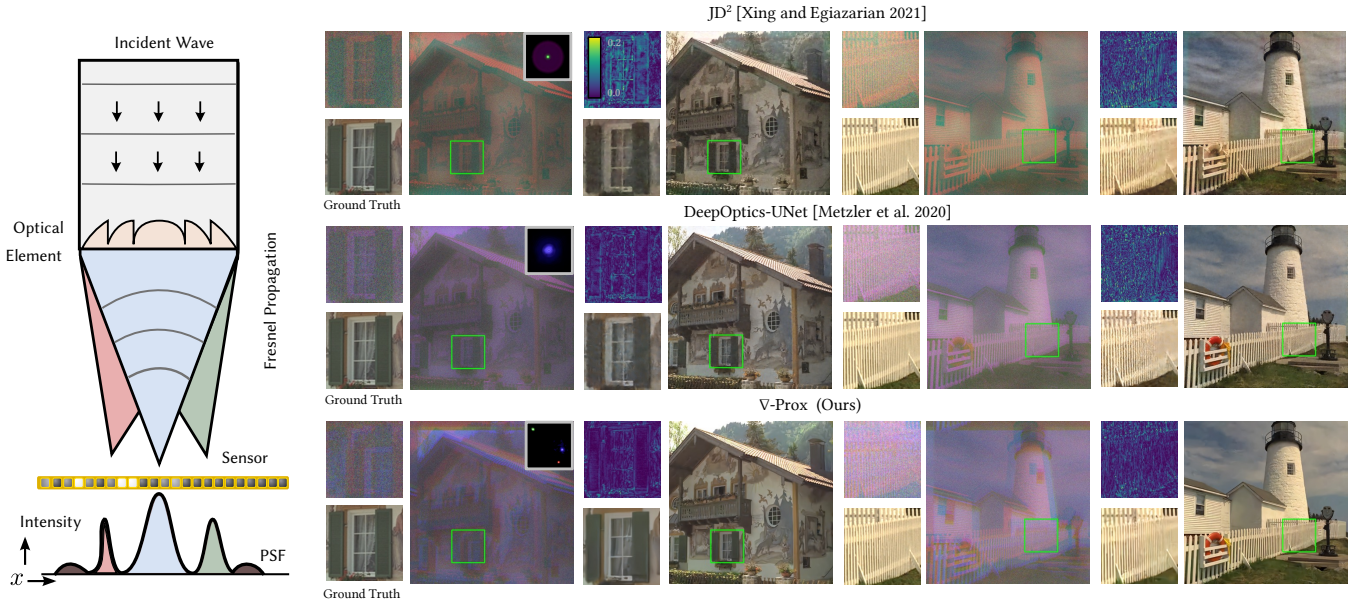


Fig. 5. Visual examples of three state-of-the-art approaches for computational thin-lens imaging. JD^2 directly post-processes imaging measurements from a green-focused Fresnel lens (row 1), while DeepOptics-UNet (row 2) and Our ∇-Prox (row 3) co-design the imager parameters (DOE) and the reconstruction algorithm. The PSFs of imaging systems are displayed in the top right-hand corners of measurement images. It can be seen DeepOptics-UNet designs a PSF with a smaller kernel size compared to the Fresnel baseline, while our ∇-Prox finds an ideal delta-function-like PSF with different spatial shifts for different wavelengths.

```
out = compile(objective, method='admm').solve()
```

Here we use an unrolled ADMM solver, which is the default training strategy in ∇-Prox, with an FFDNet [Zhang et al. 2018] as a deep prior that we train alongside the DOE. We train and validate the method on image datasets CBSD68 [Martin et al. 2001] and Urban100 [Huang et al. 2015]. The quantitative and qualitative findings reported in Table 1 and Figure 5 confirm that ∇-Prox achieves state-of-the-art results and compares favorably not only to recent post-processing methods applied on an unoptimized Fresnel lens profile, including DPIR [Zhang et al. 2021] and JD^2 [Xing and Egiazarian 2021][8], and deep optics methods, such as DeepOptics-UNet [Metzler et al. 2020], that utilize a UNet [Ronneberger et al. 2015] for reconstruction. We note that all baseline methods we compare here have been fine-tuned on the same dataset we use for a fair comparison.

Of note, Figure 5 (rows 2 and 3) highlights that the optimized PSF resulting from the end-to-end optimization problem from above drastically differs from the one optimized with a UNet post-processor. Specifically, the phase pattern we learn focuses three wavelength bands as highly chromatic PSFs for each channel, that is, the red, green, and blue PSFs only focus the specific channel while spreading out energy for other wavelengths over the entire sensor. By spatially separating the corresponding phase patterns, our co-designed network is able to find these chromatic PSFs. As such, ∇-Prox allowed us to find a *novel point in the design space* – turning an out-of-focus deconvolution problem into a transverse chromatic alignment problem. Specifically, the model-based proximal optimization solver compiled with ∇-Prox finds a better local minimum with a significantly improved end-to-end loss (2.3 dB higher than DeepOptics-UNet) validating the effectiveness of the differentiable pipelines we compile.

[8]Note that these post-processing methods do not jointly optimize the optical element, instead, they use a *green-focused Fresnel lens* for imaging.

Table 1. Numerical comparison of different methods for single thin-lens computational imaging, where DPIR and JD^2 perform joint deconvolution and denoising on measurements from a green-focused Fresnel lens, DeepOptics-UNet and ∇-Prox jointly optimize the DOE and the reconstruction algorithm. "T" and "M" refer to running time (s) and memory consumption (GB).

Method	CBSD68		Urban100			
	PSNR	SSIM	PSNR	SSIM	T	M
DPIR [2021]	21.01	0.614	18.56	0.602	0.62	3.1
JD^2 [2021]	25.94	0.903	23.78	0.872	0.54	12.7
DeepOptics-UNet [2020]	29.69	0.924	28.37	0.914	0.48	6.1
∇-Prox	**32.01**	**0.942**	**30.83**	**0.944**	0.64	3.2

5.2 Image Deraining — Learning Unknown Forward Models and Initializer

Departing from image restoration problems that possess structured forward models, such as the computational optics problem above, we next tackle a problem where the forward model is unknown and learned alongside the optimization. We consider here image deraining, i.e., the removal of rain streaks from an image, as an example for this class of problems. This problem is severely ill-posed as the assignment to a rain layer and latent background layer

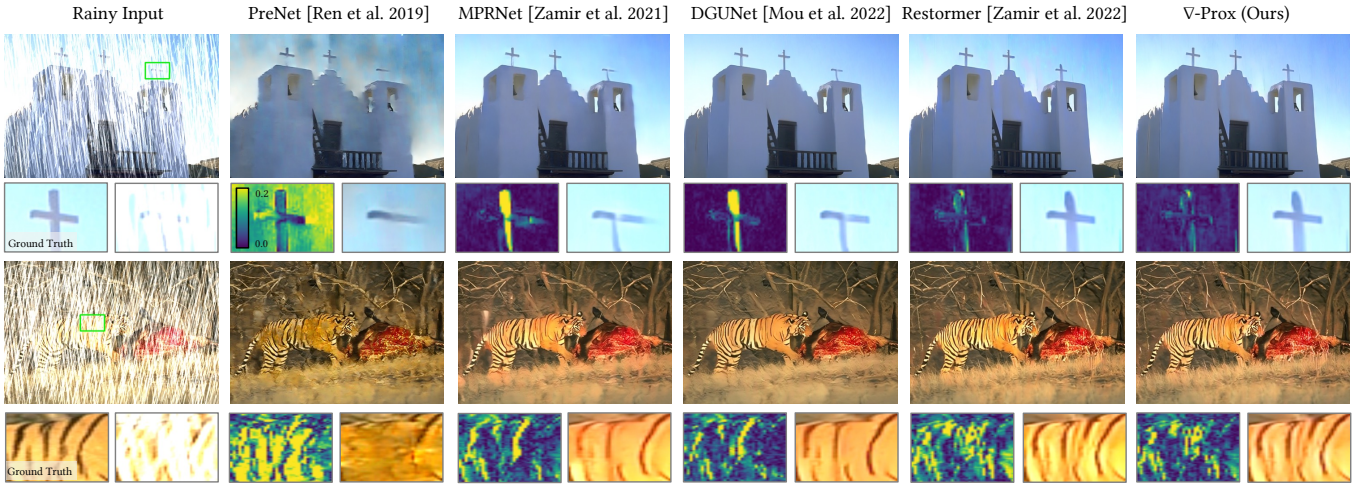


Fig. 6. Qualitative Evaluation for Image Deraining. The outputs from ∇-Prox correspond to "Learn LinOp + Initializer" in Table 2. Combining a transformer-based initializer and a differentiable proximal solver with the learnable forward model, the differentiable solvers that ∇-Prox finds achieve high-quality deraining predictions improving on state-of-the-art methods. The predictions using ∇-Prox accurately recover structure (the cross in row 1) and patterns (the tiger stripes in row 2) that can be easily confused with rain streaks.

is unknown a-priori. As such, researchers typically employ end-to-end black-box neural networks that learn to predict latent clean images from paired datasets. Being able to compile differentiable solvers, ∇-Prox facilitates learnable forward operators by making `LinOp` learnable to encode the raining forward process. Using an unrolled prior [Mou et al. 2022] that considers the prior knowledge interaction between iterations, one can formulate image deraining in ∇-Prox as,

```
          x = Variable()
  data_term = sum_squares( LearnedLinOp(x) - y )
 prior_term = unrolled_prior(x)
     solver = compile(data_term + prior_term, method='admm')
       init = Initializer()
        out = solver.solve(x0=init(b))
```

Here, the unrolled prior and the learned forward operator both have implicit trainable parameters. In ∇-Prox, any parameterized function approximators can be used to model the unknown forward model of the raining process, *e.g.*, convolutional neural networks, and/or transformers. In our example, we use an implementation of the learnable forward operator with two residual blocks to model unknown forward and adjoint models. Please refer to the Supplementary Material for network architecture details.

We report quantitative results in Table 2. ∇-Prox shows that the formal optimization pipeline ("Learn LinOp" in Table 2) can be a robust solver for problems with unknown forward models by learning them. The method achieves competitive performance with state-of-the-art methods out of the box, including very recent approaches such as MPRNet [2021] and DGUNet [2022]. Furthermore, ∇-Prox indicates another interesting direction that integrates an existing (black-box) model as a learnable initializer. By incorporating Restormer [2022] as a learnable initializer, we show that our unrolled optimization pipeline can further post-process the results towards better performance ("Learn LinOp + Initializer" in Table 2 as well as Figure 6). We note that the advantage of our approach over cascading two networks is that we do not need to retrain the entire pipeline as the output of the initializer is the input of the subsequent optimization process. With differentiable optimization powered by ∇-Prox, our approach can be fine-tuned faster and more reliably.

Table 2. Quantitative results (PSNR and SSIM) of image deraining. The best and second-best scores are **highlighted** and underlined.

Method	Rain100H		Test1200	
	PSNR	SSIM	PSNR	SSIM
PreNet [Ren et al. 2019]	26.77	0.858	31.36	0.911
MPRNet [Zamir et al. 2021]	30.41	0.890	32.91	0.916
DGUNet [Mou et al. 2022]	30.66	0.891	33.23	0.920
Restormer [Zamir et al. 2022]	31.46	0.904	33.19	0.926
MHNet [Gao and Dang 2023]	30.34	0.903	**33.42**	0.924
∇-Prox (Learn LinOp)	31.08	0.897	32.95	0.913
∇-Prox (Learn LinOp + Initializer)	**31.62**	**0.905**	33.25	**0.926**

This application highlights the unified interface and flexibility of differentiable proximal solvers in ∇-Prox for a diverse range of problems. The same solver can quickly be transferred from one task to another, and users can rely on the differentiation to learn the unknown part of the optimization process, *e.g.*, switching from a fixed operator to a learnable one if the forward model is unknown beforehand.

5.3 Compressive Magnetic Resonance Imaging — Learning Solver Parameter Scheduler

Magnetic resonance imaging (MRI) [Liang and Lauterbur 2000] is a widely deployed medical imaging modality for diagnosing and disease monitoring. MRI scanners are prohibitively expensive, and their scanning time directly correlates with the accessibility of MRI

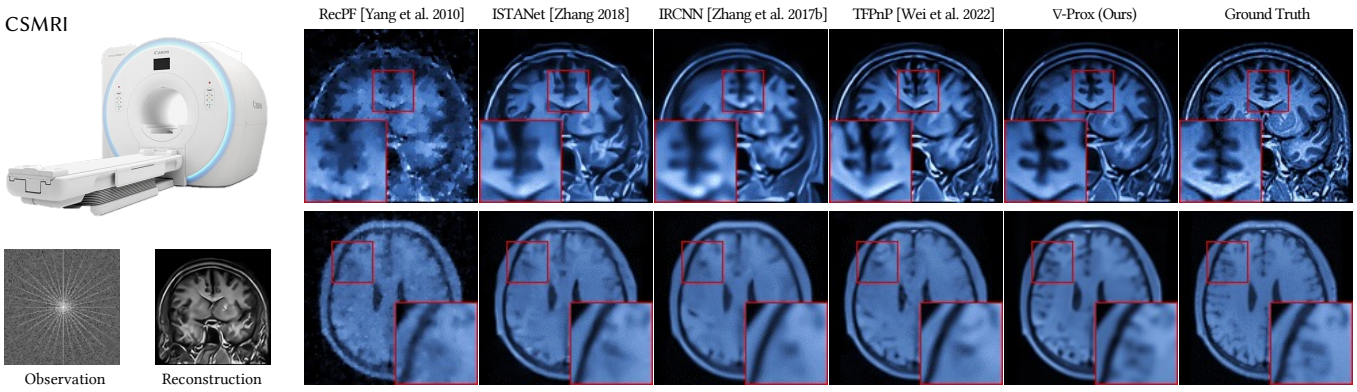


Fig. 7. Qualitative comparisons of CS-MRI reconstruction for human brain scans. The predictions of ∇-Prox correspond to "Learning Scheduler for DRUNet" in Table 3. Using a PnP prior (DRUNet) and an RL-based parameter selection policy, the trained solver output from ∇-Prox achieves unprecedented reconstruction quality compared to state-of-the-art methods. For example, it precisely recovers the cerebral ravine and brain tissues that are inaccurately reconstructed or totally missed in other methods.

scanners to the broader class of patients. Compressed sensing MRI (CS-MRI) accelerates scans by skipping parts of the data-collection process, subsampling K-space, and recovering this data later with computational methods. Recovering high-quality images from undersampled MRI data is an ill-posed inverse imaging problem with a forward model that can be mathematically expressed as

$$\mathbf{y} = \mathcal{F}_p(\mathbf{x}) + \epsilon, \tag{16}$$

where $\mathbf{x} \in \mathbb{C}^N$ is the underlying image, the operator $\mathcal{F}_p : \mathbb{C}^N \to \mathbb{C}^M$, with $M < N$, denotes the partially-sampled Fourier transform, and $\epsilon \sim \mathcal{N}(0, \sigma_n I_M)$ is the additive white Gaussian noise. The data-fidelity term, for the MRI reconstruction, is $\mathcal{D}(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathcal{F}_p(\mathbf{x})\|^2$ whose proximal operator is described in [Eksioglu 2016].

With this forward model in hand, ∇-Prox facilitates using ADMM with learned plug-and-play (PnP) denoising prior via the following lines of code

```
          x = Variable()
objective = csmri(x, b) + deep_prior(x, 'unet')
      out = Problem( objective ).solve(method='admm')
```

As an initial setting, we use a learned denoiser based on a UNet [Ronneberger et al. 2015] as a deep PnP prior, to quickly achieve competitive quantitative and qualitative results. We report our findings in Table 3. To further improve the performance, ∇-Prox implements several denoising prior architectures and allows users to switch to more advanced DRUNet [Zhang et al. 2021] by setting the option of `deep_prior` from `'unet'` to `'drunet'`[9] to obtain an improvement of about 0.3 dB on PSNR.

It is often beneficial to specialize the deep prior to the specific problem at hand. With the variety of differentiable solvers supported in ∇-Prox, users can optimize the deep prior by either unrolling optimization iterations or using deep equilibrium learning. As discussed above, both approaches can be implemented with only a few additional lines of code, specializing the solver and calling the training procedures. Table 3 lists the results achieved by utilizing these two strategies for prior learning in ∇-Prox. We observe consistent improvements over the vanilla PnP (UNet) solver in all tested CS-MRI settings with unrolling while there is one exception for DEQ. However, we note the DEQ solver requires less than a third video memory (*e.g.* 6 GB versus 20 GB in our case), making it particularly amenable to handling large-scale problems.

[9] ∇-Prox supports several built-in denoiser. For custom denoiser, users can pass a denoiser instance that implements a `denoise` function.

Table 3. Quantitative evaluation (PSNR) of CS-MRI reconstruction. The best and second-best scores are **highlighted** and underlined.

Method			Medical7		MICCA	
			4x/n5	4x/n15	4x/n5	8x/n5
RecPF [Yang et al. 2010]			28.67	25.58	33.05	28.35
ISTANet [Zhang and Ghanem 2018]			31.34	28.38	35.46	31.62
IRCNN [Zhang et al. 2017b]			31.36	27.94	35.80	31.66
TFPnP [Wei et al. 2022a]			31.81	28.58	36.17	32.69
∇-Prox	PnP with	UNet	31.42	28.18	35.04	31.66
		DRUNet	31.78	28.43	35.57	32.19
	Learn Prior with	Unroll	31.86	28.66	36.25	32.56
		DEQ	31.32	28.52	35.80	32.28
	Learn Scheduler for	UNet	31.82	28.57	36.21	32.70
		DRUNet	**32.66**	**28.91**	**36.64**	**33.22**

Automatic Parameter Tuning and Termination Time Prediction. Learning an internal algorithm parameter schedule is another essential task in proximal algorithm development. ∇-Prox allows for *automatic parameter tuning* with reinforcement learning [Wei et al. 2022a]. With a few lines of code, our vanilla PnP solver can be equipped with a policy network for optimal parameters and termination time prediction. The training is efficient and can be completed within in a few hours on one modern GPU device. The findings in Table 3, "Learn Scheduler for DRUNet", indicate that the resulting solver can achieve favorable performance just by parameter tuning. The qualitative results in Figure 7 also confirm this finding, highlighting the benefit of parameter and prior optimization using differentiable proximal algorithms.

5.4 Integrated Energy System Planning — Learning Efficient Solvers

Next, we demonstrate ∇-Prox on a seemingly orthogonal problem domain, integrated energy system planning domain — a field that describes the energy system in mathematical models typically formulated as optimization problems. Solving energy planning tasks is essential in the transition to climate neutrality of regional and global energy systems, providing decision support to policymakers by gaining insights into complex interactions and dynamics of increasingly integrated energy systems [Böttger and Härtel 2022; Craig et al. 2022; Frischmuth et al. 2022; Härtel and Korpås 2021].

The planning problems corresponding to large-scale energy systems (easily up to 100 million decision variables) are typically formulated as *continuous linear programming* (LP) [Vanderbei et al. 2020]. Even with tailored decomposition approaches and high-performance linear optimization solvers, such as Gurobi [Gurobi Optimization 2018], the solution to energy system planning problems has approached limits in practice due to increasing complexity and uncertainty in real-world applications. In extremely large instances, the state-of-the-art simplex and interior-point methods (*e.g.*, HiGHS [Huangfu and Hall 2018] and Gurobi Barrier solvers) become intractable in practice due to memory-intensive matrix factorizations. A computationally-tractable alternative is to employ first-order solvers, such as ADMM-based ones (*e.g.*, SCS [O'Donoghue 2021] and OSQP [Stellato et al. 2020]), that use efficient matrix-vector operations at cores. However, the performance of first-order methods, especially the convergence speed, is highly susceptible to its parameter setting, where a bad choice of parameters often leads to an exponentially lengthy optimization trajectory toward convergence. This is particularly pronounced in energy system planning, where the constraint matrices tend to be poorly scaled and ill-conditioned.

Here, we demonstrate that ∇-Prox, albeit initially developed for large-scale visual problems, is also especially suited to cope with the challenges above. It provides differentiable first-order proximal solvers that can be readily applied to large-scale linear programming problems without requiring excessive memory. Solver differentiability naturally provides a tool for automating parameter selection, which is key to our approach. We use ∇-Prox to find a simple yet effective instance-specific parameter tuning strategy motivated by self-supervised learning [Sun et al. 2020b]. For any test LP instance at hand, we first perform a fast test-time parameter adaption that seeks internal parameters specifically tailored to the tackled problem and then run the solver until convergence with the searched parameters. In the parameter adaptation phase, we unroll the proximal solver with a small number of fixed iterations (*e.g.*, 10 iterations) and use the unrolled solver to tackle the target LP problem. We measure the convergence behavior (*i.e.*, the primal and dual residuals and their relative convergence tolerance) and we define a convergence loss (see Supplementary Material for details) that can be backpropagated to algorithm parameters to encourage fast convergence, owing to the differentiable nature of our solver. In practice, we find only 10 gradient updates using Adam optimizer [Kingma and Ba 2014] is sufficient to find an optimal parameter setting, where the extra overhead of this training phase is marginal compared with the solving phase.

Table 4. Problem scales (the sparse constraint matrix size $m \times n$ with the total number of nonzero elements nnz) of integrated energy system analysis and planning test instances (P1/P2/P3). m and n indicate the number of constraints and the dimension of decision variables, respectively.

Problem Scale	P1	P2	P3
m	70,884	683,312	44,785,224
n	35,176	324,150	23,234,424
nnz	167,783	1,422,090	110,066,529

Table 5. Quantitative comparison of different LP solvers on three energy system analysis and planning instances (P1/P2/P3) with different problem scales (small/medium/large). We report memory consumption (M / GB) and running time (T / s) for quantitative comparison. All problem instances are solved in moderate precision (relative tolerance $\epsilon_{\text{rel}} = 10^{-6}$). We set the maximum iterations to 10^6, and the peak memory usage to 32 GB. "OOM" and "OOT" denote out-of-memory and out-of-time errors respectively. Note that memory use and running time are measured on the solvers' compute hardware: NVIDIA A100 GPU for ∇-Prox, and Intel(R) Xeon(R) Gold 6246R CPU @ 3.40GHz (32 physical cores) for others.

Solver	P1		P2		P3	
	M	T	M	T	M	T
Gurobi 10.0 [2018]	2.2	4.4	3.3	153.0	OOM	N/A
HiGHS 1.2.2 [2018]	1.6	30.4	1.7	2800.1	OOM	N/A
CVXPY-SCS [2021]	1.7	2150.0	2.0	OOT	19.6	OOT
CVXPY-OSQP [2020]	1.5	594.3	1.6	OOT	26.8	OOT
∇-Prox (native)	1.3	150.0	1.5	238.3	14.3	17695.2
∇-Prox (params adapted)	1.8	101.8	8.7	192.7	OOM	N/A

Next, we test the energy system planning problem instances[10] with diverse problem scales, whose decision variable dimension ranges from 10^4 to 10^7 (See Table 4 for a summary). Implementing linear programming in ∇-Prox can be achieved via

```
x = Variable()
prob = Problem(c @ x, [A_ub @ x <= b_ub, A_eq @ x == b_eq])
out = prob.solve(method='admm', adapt_params=True)
```

where `c` denotes the cost vector, `A_ub`, `b_ub` and `A_eq`, `b_eq` form the inequality and equality constraints, respectively. The quantitative results of different LP solvers on three representative integrated energy system planning instances are summarized in Table 5. It can be seen that the state-of-the-art solvers Gurobi (Barrier) and HiGHS perform well on small (P1) and medium (P2) scale problems but struggle to tackle the large-scale problem (P3) which runs out-of-memory. Off-the-shelf first-order solvers (SCS and OSQP), on the other hand, are stuck in tailing-off scenarios with slow converge. Without bells and whistles tailored for LP (*e.g.*, the advanced Presolve techniques [Achterberg et al. 2020; Andersen and Andersen 1995] employed by Gurobi), ∇-Prox provides the only solver that successfully tackles all problem instances within reasonable runtime and memory constraints. Its convergence can be further improved by fast parameter adaptation exploiting its differentiable structure,

[10]These instances are created by a modeling framework similar to established single-period energy system models, *e.g.*, SCOPE SD [Härtel and Ghosh 2022] or PyPSA-Eur-Sec [Neumann et al. 2022]. Please see Supplementary Material for a detailed problem description of the underlying modeling framework.

Table 6. Quantitative comparison (PSNR) of different parameter setting strategies. With only two lines of extra code in ∇-Prox, we show stronger performance for a number of image processing tasks. We abbreviate the single image super-resolution and hyperspectral compressive sensing [Fu et al. 2022] as SISR and HCS respectively. See Supplementary Material for the detailed setting of each task.

Strategy	Deblur	Demosaic	Inpainting	SISR	HCS
Fixed	33.43	35.97	36.78	34.89	34.48
Log Descent	33.43	37.22	36.78	34.89	34.48
RL (∇-Prox)	**39.54**	**40.04**	**39.56**	**35.25**	**34.86**

which suggests a promising direction toward learning-based solvers for time-honored linear programming problems. It is also worth noting that state-of-the-art LP solvers (Gurobi and HiGHS) algorithmically better map to CPU-model computing rather than the GPU model. This is an application domain where GPU-based implementations are often inferior to CPU-based implementations. To the best of our knowledge, our work is the first demonstration of GPU-accelerated proximal algorithms outperforming state-of-the-art LP solvers in large-scale LP problems.

5.5 Rapid Differentiable Solver Prototyping Analysis

In the following, we perform additional analysis of the rapid prototyping capabilities of ∇-Prox with respect to solver parameter setting strategies, choice of proximal algorithm, and choice of regularizers. We validate that these knobs, which can be changed with a few lines of code in ∇-Prox, play an essential role in rapid algorithm development.

Effect of Solver Parameter Setting Strategies. The proximal algorithms in ∇-Proxcan be sensitive to their hyperparameters. This traditionally requires a large bunch of trial-and-error for finding the optimal ones. With the differentiable solvers in ∇-Prox, this process can be automated using the learned automatic scheduler, as previously discussed in Section 4.5. Specifically, we here compare the automatic parameter tuning with fixed manual tuned parameter settings [Heide et al. 2014], *i.e.*, a single value is used across iterations for each parameter, and log-descent parameter setting strategy [Zhang et al. 2021] where parameters are gradually decayed in log space from an upper bound to a lower bound. We find the optimal value and bounds for fixed and log-descent parameter setting strategies via brute-force search, and train the automatic parameter schedulers with a small calibration dataset that contains only 200 images in hours. All the experiments for different tasks are conducted with a single deep denoising prior [Lai et al. 2022; Zhang et al. 2018, 2022] and the ADMM algorithm. The experimental results are reported in Table 6. We see that the performance of our automatic strategy consistently outperforms the fixed parameter setting and log-descent strategies on a variety of tasks, with only *two lines of extra code for each task*, illustrating the advantage of ∇-Prox for rapid prototyping.

Effect of Different Proximal Algorithm Choices. The abstraction of proximal algorithms and automatic parameter tuning of the proposed DSL facilitates employing different proximal algorithms. In practice, an algorithm can be changed via the change of a single line of code, *e.g.*, switch `solver='admm'` to `solver='hqs'`. We experiment with four different proximal algorithms, *i.e.*, ADMM [Boyd et al. 2011], linearized ADMM [Boyd et al. 2011], HQS [Geman and Yang 1995], and Pock Chambolle [Chambolle and Pock 2011], on three image reconstruction tasks, deblurring, demosaicking, and CSMRI. The experimental results are shown in Table 7. We observe that linearized ADMM generally works best in conventional deblurring and demosaicing, but worse than vanilla ADMM in CSMRI. Overall, the key finding is that different algorithms perform differently for different applications, and ∇-Prox facilitates fast experimentation with them.

Table 7. Quantitative comparison (PSNR) of different proximal algorithms on color image demosaicking and deblurring. The parameters of all algorithms are *fully automatically set* via ∇-Prox using RL-based schedulers.

Tasks	Demosaic	Deblur	CSMRI
ADMM [Boyd et al. 2011]	40.72	36.56	28.58
HQS [Geman and Yang 1995]	40.77	39.61	27.91
Linearized ADMM [Boyd et al. 2011]	41.34	40.71	28.50
Pock Chambolle [Chambolle and Pock 2011]	35.11	40.18	25.48

Table 8. Quantitative comparison (PSNR) of rapid prototyping with different regularizers on color image deblurring. We use ADMM with an RL-based automatic parameter scheduler for solving the problem.

TV	Non-Negativity	Deep Prior	PSNR
✓	✗	✗	33.79
✓	✓	✗	34.48
✗	✗	✓	36.12
✓	✓	✓	36.80

Effect of Regularizers. The most successful existing methods [Wei et al. 2022a; Zhang et al. 2021] in learned image optimization focus on designing a single regularizer and applying it to different tasks. This approach is largely due to the fact that designing different regularizers is laborious, especially due to extensive manual parameter tuning, and implementing algorithms with different regularizers is complex and error-prone. Enhanced with automatic parameter tuning, we validate that ∇-Prox facilitates testing the effect of different regularizers with the change of a line of code, see Table 8. With a margin of 3 dB, we show that rapidly changing and adding regularizers can drastically improve image quality in conventional image reconstruction problems.

5.6 Discussion and Limitations

∇-Prox versus ProxImaL. ∇-Prox takes a step towards bridging the gap between model-based proximal optimization and learning-based deep-learning algorithms. Built on top of ProxImaL [Heide et al. 2016], ∇-Prox is equipped with numerous algorithmic and engineering improvements that lift its modeling capabilities to support a wide spectrum of features and applications beyond the reach

of ProxImaL. Specifically, ∇-Prox decouples the proximal optimization modeling into three-layer abstractions: problem formulation, solver choices, and training strategies. Components from different layers can be freely chosen and combined, rendering it flexible to experiment with various algorithms in lieu of conventional coupled implementations. The design space is substantially larger than ProxImaL as a result of the new layer of abstraction (training) enabled by full differentiability, thus opening up new solvers and applications (*e.g.* forward optical model optimization, model-free image processing, and non-visual tasks).

Limitations. The RL strategy supported in ∇-Prox is tailored mainly towards determining sample-specific optimal termination time and other optimization-specific parameters. In the future, RL-based solver optimization could be used to optimize entire algorithmic blocks (such as update rules) in the solver. The compilation could be further improved by taking the distributed nature of variable-splitting-style proximal algorithms into account [Boyd et al. 2011]. In this work, we focus on the algorithmic designs for reducing memory consumption incurred by optimization loop differentiation, while compiler techniques, such as checkpointing, are not fully explored in ∇-Prox. Finally, the high degree of encapsulation in ∇-Prox programming can make solver debugging challenging. To facilitate solver development, several debugging utilities had to be built into ∇-Prox.

6 CONCLUSION

We introduce ∇-Prox as a domain-specific modeling language and compiler for differentiable proximal algorithms that solve large-scale optimization problems. ∇-Prox allows users to specify optimization objective functions of unknowns concisely at a high level and compiles the problem into compute- and memory-efficient differentiable solvers. To make proximal algorithms efficiently differentiable, ∇-Prox looks beyond classic auto-differentiation mechanisms and relies on equilibrium learning and reinforcement learning-based bi-level optimization schemes that do not require unrolling algorithms into lengthy compute graphs. As such, the proposed DSL supports hybrid model-based and learning-based solvers that merge proximal optimization with deep neural network pipelines — seamlessly integrating with modern deep learning frameworks such as PyTorch. We assess ∇-Prox on visual computing problems and large-scale planning problems with findings that demonstrate the DSL allows for rapid experimentation with a variety of learned solvers and training approaches without the pain of manually implementing these methods and training schemes. With a few lines of code, we find that ∇-Prox can generate solvers for diverse problems from computational optics to integrated energy system planning of Europe's energy grid towards climate neutrality – each with state-of-the-art performance, task-optimized as a result of the differentiable solvers that allow for parameter and model optimization using stochastic gradient bi-level optimization. Exciting areas for future research include modeling the forward model, or the proximal algorithm itself, as a learnable program and the symbiosis of the proposed differentiable algorithms with differentiable rendering forward models – challenging research areas we hope this work can accelerate with its rapid prototyping capabilities.

ACKNOWLEDGMENTS

We thank Ethan Tseng and Ilya Chugunov for discussion on this work. Felix Heide was supported by an NSF CAREER Award (2047359), a Packard Foundation Fellowship, a Sloan Research Fellowship, a Sony Young Faculty Award, a Project X Innovation Award, and an Amazon Science Research Award. Philipp Härtel was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) in the 7. Energieforschungsprogramm (03EI1027). Ying Fu was supported by the National Natural Science Foundation of China under Grants No.62171038, No.62088101, and No.62006023.

REFERENCES

Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. 2020. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing* 32, 2 (2020), 473–506.

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, et al. 2019. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. 2019a. Differentiable convex optimization layers. *Advances in neural information processing systems* 32 (2019).

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. 2019b. Differentiating through a cone program. *arXiv preprint arXiv:1904.09043* (2019).

José Juan Almagro Armenteros, Konstantinos D Tsirigos, Casper Kaae Sønderby, Thomas Nordahl Petersen, Ole Winther, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. 2019. SignalP 5.0 improves signal peptide predictions using deep neural networks. *Nature biotechnology* 37, 4 (2019), 420–423.

Brandon Amos and J Zico Kolter. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*. PMLR, 136–145.

Erling D Andersen and Knud D Andersen. 1995. Presolving in linear programming. *Mathematical Programming* 71, 2 (1995), 221–245.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems* 29 (2016).

Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. 2013. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods. *Mathematical Programming* 137, 1 (2013), 91–129.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. *Advances in Neural Information Processing Systems* 32 (2019).

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2018. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research* 18 (2018), 1–43.

Amir Beck and Marc Teboulle. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences* 2, 1 (2009), 183–202.

Stephen R Becker, Emmanuel J Candès, and Michael C Grant. 2011. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation* 3, 3 (2011), 165–218.

Martin Benning, Florian Knoll, Carola-Bibiane Schönlieb, and Tuomo Valkonen. 2015. Preconditioned ADMM with nonlinear operator constraint. In *IFIP Conference on System Modeling and Optimization*. Springer, 117–126.

Gilbert Louis Bernstein, Chinmayee Shah, Crystal Lemire, Zachary Devito, Matthew Fisher, Philip Levis, and Pat Hanrahan. 2016. Ebb: A DSL for physical simulation on CPUs and GPUs. *ACM Transactions on Graphics (TOG)* 35, 2 (2016), 1–12.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. 2021. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183* (2021).

Diana Böttger and Philipp Härtel. 2022. On wholesale electricity prices and market values in a carbon-neutral energy system. *Energy Economics* 106 (2022), 105709. https://doi.org/10.1016/j.eneco.2021.105709

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

Ronald E Bruck Jr. 1975. An iterative solution of a variational inequality for certain monotone operators in Hilbert space. *Bull. Amer. Math. Soc.* 81, 5 (1975), 890–892.

Antoni Buades, Bartomeu Coll, and J-M Morel. 2005. A non-local algorithm for image denoising. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 2. Ieee, 60–65.

Antonin Chambolle and Thomas Pock. 2011. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision* 40, 1 (2011), 120–145.

Antonin Chambolle and Thomas Pock. 2016. An introduction to continuous optimization for imaging. *Acta Numerica* 25 (2016), 161–319.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. 2021. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828* (2021).

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

Michael T. Craig, Jan Wohland, Laurens P. Stoop, Alexander Kies, Bryn Pickering, Hannah C. Bloomfield, Jethro Browell, Matteo de Felice, Chris J. Dent, Adrien Deroubaix, Felix Frischmuth, Paula L.M. Gonzalez, Aleksander Grochowicz, Katharina Gruber, Philipp Härtel, Martin Kittel, Leander Kotzur, Inga Labuhn, Julie K. Lundquist, Noah Pflugradt, Karin van der Wiel, Marianne Zeyringer, and David J. Brayshaw. 2022. Overcoming the disconnect between energy system and climate modeling. *Joule* 6, 7 (2022), 1405–1417. https://doi.org/10.1016/j.joule.2022.05.010

Zachary DeVito, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Niessner. 2017. Opt: A domain specific language for non-linear least squares optimization in graphics and imaging. *ACM Transactions on Graphics (TOG)* 36, 5 (2017), 1–27.

Steven Diamond and Stephen Boyd. 2015. Convex optimization with abstract linear operators. In *Proceedings of the IEEE International Conference on Computer Vision*. 675–683.

Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research* 17, 1 (2016), 2909–2913.

Steven Diamond, Vincent Sitzmann, Felix Heide, and Gordon Wetzstein. 2017. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041* (2017).

Weisheng Dong, Peiyao Wang, Wotao Yin, Guangming Shi, Fangfang Wu, and Xiaotong Lu. 2018. Denoising prior driven deep neural network for image restoration. *IEEE transactions on pattern analysis and machine intelligence* 41, 10 (2018), 2305–2318.

Ender M Eksioglu. 2016. Decoupled algorithm for MRI reconstruction using nonlocal block matching model: BM3D-MRI. *Journal of Mathematical Imaging and Vision* 56, 3 (2016), 430–440.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning* 11, 3-4 (2018), 219–354.

Felix Frischmuth, Richard Schmitz, and Philipp Härtel. 2022. IMAGINE – Market-based multi-period planning of European hydrogen and natural gas infrastructure. *2022 18th International Conference on the European Energy Market (EEM)* (2022), 1–11. https://doi.org/10.1109/EEM54602.2022.9921154

Ying Fu, Tao Zhang, Lizhi Wang, and Hua Huang. 2022. Coded hyperspectral image reconstruction using deep external and internal learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 7 (2022), 3404–3420.

Daniel Gabay and Bertrand Mercier. 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications* 2, 1 (1976), 17–40.

Hu Gao and Depeng Dang. 2023. Mixed Hierarchy Network for Image Restoration. *arXiv preprint arXiv:2302.09554* (2023).

Donald Geman and Chengda Yang. 1995. Nonlinear image recovery with half-quadratic regularization. *IEEE transactions on Image Processing* 4, 7 (1995), 932–946.

Joachim Giesen and Sören Laue. 2016. Distributed convex optimization with many convex constraints. *arXiv preprint arXiv:1610.02967* (2016).

Davis Gilton, Gregory Ongie, and Rebecca Willett. 2021. Deep equilibrium architectures for inverse problems in imaging. *IEEE Transactions on Computational Imaging* 7 (2021), 1123–1133.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Joseph W Goodman and P Sutton. 1996. Introduction to Fourier optics. *Quantum and Semiclassical Optics-Journal of the European Optical Society Part B* 8, 5 (1996), 1095.

Michael Grant and Stephen Boyd. 2014. CVX: Matlab Software for Disciplined Convex Programming, version 2.1. http://cvxr.com/cvx.

Karol Gregor and Yann LeCun. 2010. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*. 399–406.

Shuhang Gu, Qi Xie, Deyu Meng, Wangmeng Zuo, Xiangchu Feng, and Lei Zhang. 2017. Weighted nuclear norm minimization and its applications to low level vision. *International journal of computer vision* 121, 2 (2017), 183–208.

LLC Gurobi Optimization. 2018. Gurobi optimizer reference manual.

Harel Haim, Shay Elmalem, Raja Giryes, Alex Bronstein, and Emanuel Marom. 2018. Depth Estimation From a Single Image Using Deep Learned Phase Coded Mask. *IEEE Transactions on Computational Imaging* 4 (2018), 298–310.

Philipp Härtel and Debraj Ghosh. 2022. Modelling Heat Pump Systems in Low-Carbon Energy Systems With Significant Cross-Sectoral Integration. *IEEE Transactions on Power Systems* 37, 4 (2022), 3259–3273. https://doi.org/10.1109/TPWRS.2020.3023474

Philipp Härtel and Magnus Korpås. 2021. Demystifying market clearing and price setting effects in low-carbon energy systems. *Energy Economics* 93 (2021), 105051. https://doi.org/10.1016/j.eneco.2020.105051

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. 2014. Darkroom: compiling high-level image processing code into hardware pipelines. *ACM Trans. Graph.* 33, 4 (2014), 144–1.

Felix Heide, Steven Diamond, Matthias Nießner, Jonathan Ragan-Kelley, Wolfgang Heidrich, and Gordon Wetzstein. 2016. Proximal: Efficient image optimization using proximal algorithms. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–15.

Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pająk, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, et al. 2014. Flexisp: A flexible camera image processing framework. *ACM Transactions on Graphics (ToG)* 33, 6 (2014), 1–13.

John R Hershey, Jonathan Le Roux, and Felix Weninger. 2014. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574* (2014).

Roarke Horstmeyer, Richard Y. Chen, Barbara Kappes, and Benjamin Judkewitz. 2017. Convolutional neural networks that teach microscopes how to image. *ArXiv* abs/1709.07223 (2017).

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. 2019a. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935* (2019).

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019b. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. 2015. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5197–5206.

Qi Huangfu and JA Julian Hall. 2018. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (2018), 119–142.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. 2021. Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 21043–21055.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022. DR. JIT: a just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–19.

Ulugbek S Kamilov, Charles B Bouman, Gregery T Buzzard, and Brendt Wohlberg. 2023. Plug-and-Play Methods for Integrating Physical and Learned Models in Computational Imaging. *IEEE Signal Process. Mag.* 40, 1 (January 2023), 85–97.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Fredrik Kjolstad, Shoaib Kamil, Jonathan Ragan-Kelley, David IW Levin, Shinjiro Sueda, Desai Chen, Etienne Vouga, Danny M Kaufman, Gurtej Kanwar, Wojciech Matusik, et al. 2016. Simit: A language for physical simulation. *ACM Transactions on Graphics (TOG)* 35, 2 (2016), 1–21.

Nikos Komodakis and Jean-Christophe Pesquet. 2015. Playing with duality: An overview of recent primal? dual approaches for solving large-scale optimization problems. *IEEE Signal Processing Magazine* 32, 6 (2015), 31–54.

Zeqiang Lai, Kaixuan Wei, and Ying Fu. 2022. Deep Plug-and-Play Prior for Hyperspectral Image Restoration. *Neurocomputing* (2022).

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

Guoyin Li and Ting Kei Pong. 2015. Global convergence of splitting methods for nonconvex composite optimization. *SIAM Journal on Optimization* 25, 4 (2015), 2434–2460.

Ke Li and Jitendra Malik. 2016. Learning to optimize. *arXiv preprint arXiv:1606.01885* (2016).

Qunwei Li, Yi Zhou, Yingbin Liang, and Pramod K Varshney. 2017. Convergence analysis of proximal gradient with momentum for nonconvex optimization. In *International Conference on Machine Learning*. PMLR, 2111–2119.

Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. 2018. Differentiable programming for image processing and deep learning in Halide. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Zhi-Pei Liang and Paul C Lauterbur. 2000. *Principles of magnetic resonance imaging*. SPIE Optical Engineering Press Bellingham.

Jiaming Liu, Xiaojian Xu, Weijie Gan, Shirin Shoushtari, and Ulugbek S Kamilov. 2022. Online Deep Equilibrium Learning for Regularization by Denoising. *arXiv preprint arXiv:2205.13051* (2022).

Qinghua Liu, Xinyue Shen, and Yuantao Gu. 2019. Linearized ADMM for nonconvex nonsmooth optimization with convergence analysis. *IEEE Access* 7 (2019), 76131–76144.

Julien Mairal, Michael Elad, and Guillermo Sapiro. 2007. Sparse representation for color image restoration. *IEEE Transactions on image processing* 17, 1 (2007), 53–69.

Michael Mara, Felix Heide, Michael Zollhöfer, Matthias Nießner, and Pat Hanrahan. 2021. Thallo–scheduling for high-performance large-scale non-linear least-squares solvers. *ACM Transactions on Graphics (TOG)* 40, 5 (2021), 1–14.

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 2. IEEE, 416–423.

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. 2022. VeLO: Training Versatile Learned Optimizers by Scaling Up. *arXiv preprint arXiv:2211.09760* (2022).

Christopher A Metzler, Hayato Ikoma, Yifan Peng, and Gordon Wetzstein. 2020. Deep optics for single-shot high-dynamic-range imaging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1375–1385.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

Thomas Mollenhoff, Evgeny Strekalovskiy, Michael Moeller, and Daniel Cremers. 2015. The primal-dual hybrid gradient method for semiconvex splittings. *SIAM Journal on Imaging Sciences* 8, 2 (2015), 827–857.

Vishal Monga, Yuelong Li, and Yonina C Eldar. 2021. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine* 38, 2 (2021), 18–44.

Jean-Jacques Moreau. 1965. Proximité et dualité dans un espace hilbertien. *Bulletin de la Société mathématique de France* 93 (1965), 273–299.

Chong Mou, Qian Wang, and Jian Zhang. 2022. Deep Generalized Unfolding Networks for Image Restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 17399–17410.

Ravi Teja Mullapudi, Andrew Adams, Dillon Sharlet, Jonathan Ragan-Kelley, and Kayvon Fatahalian. 2016. Automatically scheduling halide image processing pipelines. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Fabian Neumann, Elisabeth Zeyen, Marta Victoria, and Tom Brown. 2022. Benefits of a Hydrogen Network in Europe. *SSRN Electronic Journal* (2022). https://doi.org/10.2139/ssrn.4173442

Robert Nishihara, Laurent Lessard, Ben Recht, Andrew Packard, and Michael Jordan. 2015. A general analysis of the convergence of ADMM. In *International Conference on Machine Learning*. PMLR, 343–352.

Brendan O'Donoghue. 2021. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization* 31, 3 (2021), 1999–2023.

Gregory Ongie, Ajil Jalal, Christopher A Metzler, Richard G Baraniuk, Alexandros G Dimakis, and Rebecca Willett. 2020. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory* 1, 1 (2020), 39–56.

Stanley Osher, Martin Burger, Donald Goldfarb, Jinjun Xu, and Wotao Yin. 2005. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation* 4, 2 (2005), 460–489.

Tulin Ozturk, Muhammed Talo, Eylul Azra Yildirim, Ulas Baran Baloglu, Ozal Yildirim, and U Rajendra Acharya. 2020. Automated detection of COVID-19 cases using deep neural networks with X-ray images. *Computers in biology and medicine* 121 (2020), 103792.

Neal Parikh and Stephen Boyd. 2014. Proximal algorithms. *Foundations and Trends in optimization* 1, 3 (2014), 127–239.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

Yifan Peng, Qilin Sun, Xiong Dun, Gordon Wetzstein, Wolfgang Heidrich, and Felix Heide. 2019. Learned large field-of-view imaging with thin-plate optics. *ACM Trans. Graph.* 38, 6 (2019), 219–1.

Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, et al. 2022. Theseus: A library for differentiable nonlinear optimization. *arXiv preprint arXiv:2207.09442* (2022).

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

Dongwei Ren, Wangmeng Zuo, Qinghua Hu, Pengfei Zhu, and Deyu Meng. 2019. Progressive image deraining networks: A better and simpler baseline. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3937–3946.

R Tyrrell Rockafellar. 1976. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of operations research* 1, 2 (1976), 97–116.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Mark Segal and Kurt Akeley. 1999. The OpenGL graphics system: A specification (version 1.1).

Zheng Shi, Yuval Bahat, Seung-Hwan Baek, Qiang Fu, Hadi Amata, Xiao Li, Praneeth Chakravarthula, Wolfgang Heidrich, and Felix Heide. 2022. Seeing through obstructions with diffractive cloaking. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

Nir Shlezinger, Jay Whang, Yonina C Eldar, and Alexandros G Dimakis. 2020. Model-based deep learning. *arXiv preprint arXiv:2012.08405* (2020).

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. PMLR, 387–395.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

Vincent Sitzmann, Steven Diamond, Yifan Peng, Xiong Dun, Stephen Boyd, Wolfgang Heidrich, Felix Heide, and Gordon Wetzstein. 2018. End-to-end optimization of optics and image processing for achromatic extended depth of field and super-resolution imaging. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. 2020. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12, 4 (2020), 637–672.

Jian Sun, Huibin Li, Zongben Xu, et al. 2016. Deep ADMM-Net for compressive sensing MRI. *Advances in neural information processing systems* 29 (2016).

Qilin Sun, Ethan Tseng, Qiang Fu, Wolfgang Heidrich, and Felix Heide. 2020a. Learning Rank-1 Diffractive Optics for Single-shot High Dynamic Range Imaging. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. 2020b. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*. PMLR, 9229–9248.

Chunwei Tian, Lunke Fei, Wenxian Zheng, Yong Xu, Wangmeng Zuo, and Chia-Wen Lin. 2020. Deep learning on image denoising: An overview. *Neural Networks* 131 (2020), 251–275.

Ethan Tseng, Ali Mosleh, Fahim Mannan, Karl St-Arnaud, Avinash Sharma, Yifan Peng, Alexander Braun, Derek Nowrouzezahrai, Jean-Francois Lalonde, and Felix Heide. 2021. Differentiable compound optics and processing pipeline optimization for end-to-end camera design. *ACM Transactions on Graphics (TOG)* 40, 2 (2021), 1–19.

Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. 2014. Convex optimization in Julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*. IEEE, 18–28.

Robert J Vanderbei et al. 2020. *Linear programming*. Springer.

Singanallur V Venkatakrishnan, Charles A Bouman, and Brendt Wohlberg. 2013. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, 945–948.

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path replay backpropagation: differentiating light paths using constant memory and linear time. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14.

Homer F Walker and Peng Ni. 2011. Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.* 49, 4 (2011), 1715–1735.

Yu Wang, Wotao Yin, and Jinshan Zeng. 2019. Global convergence of ADMM in nonconvex nonsmooth optimization. *Journal of Scientific Computing* 78, 1 (2019), 29–63.

Kaixuan Wei, Angelica Aviles-Rivero, Jingwei Liang, Ying Fu, Hua Huang, and Carola-Bibiane Schönlieb. 2022a. TFPNP: Tuning-free plug-and-play proximal algorithms with applications to inverse imaging problems. *Journal of Machine Learning Research* 23, 16 (2022), 1–48.

Kaixuan Wei, Angelica Aviles-Rivero, Jingwei Liang, Ying Fu, Carola-Bibiane Schönlieb, and Hua Huang. 2020. Tuning-free plug-and-play proximal algorithm for inverse imaging problems. In *International Conference on Machine Learning*. PMLR, 10158–10169.

Kaixuan Wei, Ying Fu, Yinqiang Zheng, and Jiaolong Yang. 2022b. Physics-based noise modeling for extreme low-light photography. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 11 (2022), 8520–8537.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. 2017. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*. PMLR, 3751–3760.

Stephen Wright, Jorge Nocedal, et al. 1999. Numerical optimization. *Springer Science* 35, 67-68 (1999), 7.

Wenzhu Xing and Karen Egiazarian. 2021. End-to-end learning for joint image demosaicing, denoising and super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3507–3516.

Xiaojian Xu, Yu Sun, Jiaming Liu, Brendt Wohlberg, and Ulugbek S Kamilov. 2020. Provable convergence of plug-and-play priors with MMSE denoisers. *IEEE Signal Processing Letters* 27 (2020), 1280–1284.

Junfeng Yang, Yin Zhang, and Wotao Yin. 2010. A fast alternating direction method for TVL1-L2 signal reconstruction from partial Fourier data. *IEEE Journal of Selected Topics in Signal Processing* 4, 2 (2010), 288–297.

Raymond A Yeh, Yuan-Ting Hu, Zhongzheng Ren, and Alexander G Schwing. 2022. Total Variation Optimization Layers for Computer Vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 711–721.

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. 2022. Restormer: Efficient transformer for high-resolution image restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5728–5739.

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. 2021. Multi-stage progressive image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 14821–14831.

Jian Zhang and Bernard Ghanem. 2018. ISTA-Net: Interpretable optimization-inspired deep network for image compressive sensing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1828–1837.

Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. 2021. Plug-and-play image restoration with deep denoiser prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017a. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing* 26, 7 (2017), 3142–3155.

Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. 2017b. Learning deep CNN denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3929–3938.

Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. FFDNet: Toward a fast and flexible solution for CNN-based image denoising. *IEEE Transactions on Image Processing* 27, 9 (2018), 4608–4622.

Tao Zhang, Ying Fu, and Jun Zhang. 2022. Guided Hyperspectral Image Denoising with Realistic Data. *International Journal of Computer Vision* 130, 11 (2022), 2885–2901.

Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems* 30, 11 (2019), 3212–3232.

Daniel Zoran and Yair Weiss. 2011. From learning models of natural image patches to whole image restoration. In *2011 international conference on computer vision*. IEEE, 479–486.