Delft University of Technology

MASTER'S THESIS

COSSE PROGRAMME

Topology Optimization and Physics-Informed Neural Networks for Metamaterial Optics Design

Dylan Everingham
(5034027)

August 25th, 2022

Contents

1 Introduction

The development of optical metamaterials in recent years has enabled the design of novel optical devices with exciting properties and applications ranging across many fields, including in scientific instrumentation for space missions. This in turn has led to demand for computational methods which can produce efficient device designs.

Traditional optical devices admit a closed-form solution for this inverse design problem. However, in the presence of strong multiple scattering, which is often the case when considering optical metamaterials, the inverse problem becomes ill-posed. As a result, many optimization and machine learning techniques have been applied towards discovering good solutions.

In this MSc thesis project, several of the most promising of these techniques are applied to a specific problem, the discovery of silicon metamaterial lens designs for the CoPILOT high-altitude balloon project. Ultimately, a software tool capable of producing effective and admissible designs is produced and demonstrated.

First, an overview of the CoPILOT design problem is presented. Next, relevant background material topics, including properties of metamaterials and computational methods for simulating them, are covered in some detail. After this, methods used to solve optical design problems in past literature are described and contrasted. Then, a comprehensive explanation of the method developed and used for this project, including important design considerations, is given. The best solutions found using this lens optimization method are shown and compared. Finally, fruitful areas of future work on this topic are listed.

2 Overview of the Design Problem

This section introduces the design problem at hand: the search for metamaterial lens shapes for the CoPILOT high-altitude balloon mission. First, basic information about the mission itself and its goals is given. Next, the manufacturing constraints and design cases, which were guiding factors in the development of solution methods, are described. Finally, the research questions to be addressed in these experiments are introduced.

2.1 The CoPILIOT Project

CoPILOT is a planned mission by the French space agency CNES with the goal of improving existing maps of low density cold gas in the Milky Way and local group galaxies, otherwise known as the interstellar medium (ISM). Far from being inert, the ISM forms a dynamic system interlinked with the processes of star formation, evolution, and death. Observing the distribution and heating of the ISM is therefore instrumental in understanding these processes as well as the long-term structure of galaxies.

To accomplish this mapping, CoPILOT will observe the fine structure line of ionized carbon (C+) which occurs at $157.679\mu m$. This spectral line corresponds to what is known as a forbidden process, which can only occur frequently enough to be observed under conditions present in low-density, cold gas clouds, therefore uniquely characterizing the ISM. The intensity of emissions by this process also depends on the density and temperature of the gas, making it possible to learn something about the local conditions of the ISM based off of these observations. These characteristics, combined with the abundance of carbon in the universe, make it an ideal tracker for ISM conditions in star-forming regions.

Because the Earth's atmosphere is significantly opaque in the far-infrared, observations of the C+ fine structure line are not possible from ground-based observatories. As a result, airborne or space-based instruments are required. Several such experiments have already produced C+ emission observations in the past, including the Far InfraRed Absolute Spectrophotometer (FIRAS) aboard the COsmic Background Explorer (COBE) satellite[1], the HIFI and PACS instruments on the Herschel space observatory[2], FIFI-LS and GREAT on the SOFIA airborne observatory[3], and the Balloon-borne Infrared Carbon Explorer (BICE)[4]. While these missions, COBE in particular, have provided state-of-the-art C+ emission mappings, imperfections in instrument capabilities have led to properties of the data which could be improved in future mappings, including low angular resolution, low sensitivity to non-bright regions, and a lack of wide-field mapping.

CoPILOT aims to produce a C+ map of the sky which improves on previous maps in these areas. Specifically, the combination of a wide field of view, high-speed mapping capabilities and high sensitivity as compared to FIRAS will lead to the production of a high quality map suitable for use towards several science objectives. These include validation of ISM models via observation of distributions of ISM attributes including temperature, density, metallicity, and radiation field, estimation of galactic and inter-galactic star formation rate, and identification of regions of interest for future very high resolution observations.

2.2 Lens Design

CoPILOT is a follow-up to the 2017 PILOT balloon-borne experiment, and will reuse and modify some of its pre-existing instrumentation. One potential modification is the replacement of traditional refractive optics with lenses made from a non-resonant terahertz metamaterial. An explanation of metamaterials as well as methods for modelling their optical behavior can be found in section 3, background material.

Such lenses have several advantageous properties, most notably being both lighter and smaller in the focal direction. In high-altitude and space-based instruments, low-weight and physically compact components are highly sought after because smaller and lighter systems reduce cost and enable more additional features to be carried in the payload.

This research aims to evaluate the possibility of using metamaterial lenses on CoPILOT by seeking metamaterial lens shapes which can match or outperform the effectiveness of the existing optics while adhering to mission and manufacturing design constraints. Furthermore, it seeks to produce a software tool which can produce on-demand designs for similar applications.

2.2.1 Physical Description

The existing conventional refractive lens, proposed to be replaced, consists of a array of silicon lenslets, as seen in Figure 1.

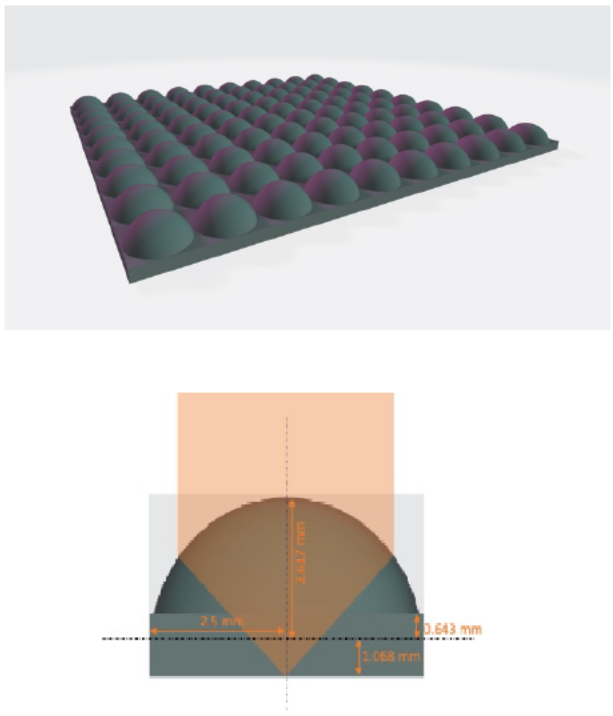

Figure 1: Existing lens design from PILOT experiment.

The proposed metamaterial lens, or metalens, would be composed of an array of flat metamaterial lenslets. Each lenslet is divided into a grid of square regions called ’pixels’, each of which has a uniform, discrete height. The surface the lens is only etched with detail to a certain depth, below which is a uniform substrate layer. This arrangement can be seen in Figure 2.

The dimensions of each pixel and the possible pixel heights are determined by manufacturing constraints. The total thickness of each lens, the dimensions of the lenslet array, and the focal length are determined instead by CoPILOT design constraints.

2.2.2 Manufacturing Process and Constraints

The entirety of the device, both the surface etching as well as the underlying substrate layer, will consist of high-resistivity, float-zone silicon, which is a type of highly pure silicon obtained via a process called vertical zone melting. This

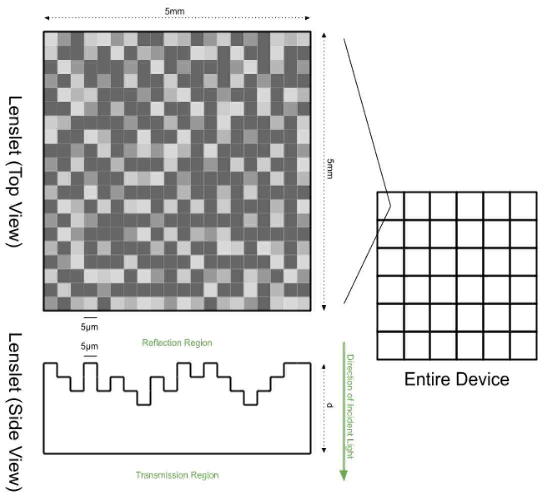

Figure 2: Basic proposed design of a metamaterial lens for CoPILOT. This design shows 20×20 pixels per lenslet, which in reality can be increased up to 250×250. Lenslet thickness d and dimensions of lenslet array depend on the use case, as discussed below.

silicon is one of the most transparent dielectric materials in the THz domain, making it well suited for construction of non-resonant THz metamaterials[5]. It has a relative permittivity of $\epsilon = 12.04$, and displays several favorable optical properties including very little dispersion over the 0.5-4.5 THz band. Techniques for layered etching of silicon, as is required to manufacture the CoPILOT device, are also well established[6].

Effective lens designs found during this research can be fabricated as actual devices by the Kavli Nanolab Delft[7], a nanofabrication research facility at TU Delft which uses the Van Leeuwenhoek Laboratory (VLL) cleanroom facilities. This will enable experimental verification of discovered designs, as well as the eventual production of flight lenses.

The following constraints are imposed by the manufacturing process:

- The minimum size of etched features is $20\mu m$, with a tolerance of $\pm 2\mu m$.
- The size of each lenslet is set at $5mm \times 5mm$. This, combined with the first constraint, puts a limit on the number of pixels per lenslet at 250×250.
- The surface etching can have a maximum of 5 layers, each of thickness $50\mu m$. This leads to 6 different possibilities for the discrete height of each pixel.
- "Tunnels" are not allowed in the surface. This means that for any pixel, there may be no layer which is filled with material above a layer which is not.

2.2.3 Design Constraints and Cases

CoPILOT calls for two separate lens designs, which we call the two design cases. The **near field case** calls for wave focusing in the device's near field, meaning a focal length of less than one wavelength. The **far field case** calls for the opposite, namely a focal distance longer than one wavelength.

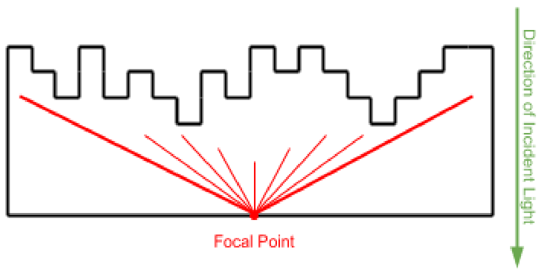

Figure 3: Near field design case. The focal point of each lenslet is the center of its transmission side surface, and all lenslets are identical.

In the near field design, depicted in Figure 3, the entire optical device consists of an array of 10×10 lenslets. Each lenslet should focus at the transmission side of each lenslet, or equivalently, each should have a focal length of 0. In this design, the total thickness of the device, including both the surface etching and substrate layer, should be $1.2mm$. However, this thickness, as discussed in section 7, is one of several design parameters which may be optimized. The design should be optimized for an incident plane wave with wavelength $\lambda = 120\mu m$ or $2.5THz$.

The far field design, shown in Figure 4, demands instead a single lenslet which focuses on a point in the far field.

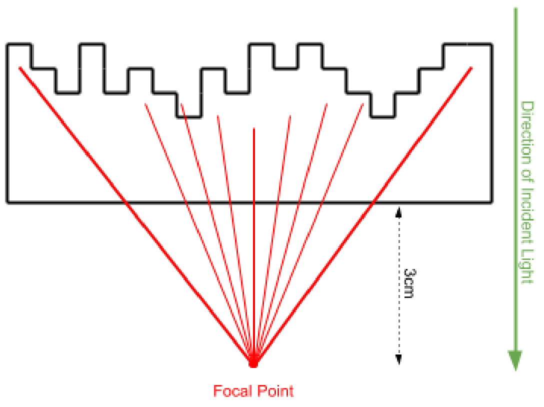

Figure 4: Far field design case. All lenslets focus on a single faraway point, meaning that now they must be not identical.

Here, the entire lens consists only of one lenslet, with a focal length of $3cm$.

The total thickness of the device should be $500\mu m$, but this thickness, again, may be optionally optimized. This design should be optimized for an incident plane wave with wavelength $\lambda = 158\mu m$ or $1.9THz$.

2.3 Research Questions

The primary research questions at hand in this project are:

What technique is the most effective and computationally efficient for determining good CoPILOT lens designs? What does a good software implementation of such a method look like?

The high-level goal of the project is to construct effective lens designs and ultimately demonstrate functioning metalenses via fabrication. Of course, this is impossible without an optimization method which is both able to find such good designs and computationally efficient enough to handle many geometric parameters. And a method itself is not particularly useful without an effective software implementation.

In section 3, several methods used for similar metamaterial optics optimizations applications in the literature are discussed and examined for applicability to the CoPILOT problem. Answering this research question is a matter of implementing these methods, adapting them to the CoPILOT problem, and comparing them in terms of produced solution quality and computational efficiency.

In addition, for any working optimization method, there will be a number of model hyperparameters which can affect the performance of the method. For typical machine learning methods, these include values such as the learning rate, initial guesses, and maximum number of training iterations. The choice of these parameters has measurable effects on solution quality, so finding good values is another necessary step towards an effective optimization implementation.

Other research questions suitable for future work on this topic are detailed in section 7.

3 Background Material

In this section, some fields of background material necessary to construct solutions to the CoPILOT lens design problem are introduced and discussed. Because the lenses are to be made of an all-dielectric metamaterial, the first topic addressed concerns what exactly a metamaterial is. Next comes the formulation and discussion of rigorous coupled-wave analysis, or RCWA, a numerical solver method which is used to simulate wave scattering as part of the design process. Finally there is an introduction to algorithmic differentiation, or AD, a mathematical programming technique which is essential to the physics-based machine learning and optimization methods used in the lens design methods.

3.1 Terahertz Metamaterials

There is no universally accepted definition of the term **metamaterial**, but it refers in general to engineered composite materials which may exhibit physical properties, most notably electromagnetic response, which are not observed in nature or in their constituent materials. Usually, these properties arise due primarily to some sub-wavelength-scale lattice structure of the material rather than directly from its chemical properties.

Metamaterials may possibly exhibit unusual properties such as having a negative index of refraction for particular wavelengths[8]. They might also be "left-handed" - that is, inside such a material, electromagnetic waves obey a "left-hand rule" and can convey energy in the direction opposite to their phase velocity[9]. This has led to great interest in recent years in their use for developing novel optical devices, such as ultra-thin flat lenses and collimators[10][11][12] and hybrid systems for correcting chromatic and spherical aberrations in traditional refractive lenses[13]. Dielectric metamaterial lenses, or **metalenses**, have been developed in recent years with favorable properties such as full phase coverage[14][15], polarization insensitivity[16][17], high numerical aperture[17], and dynamically controllable focal length and intensity[18]. Applications for such devices exist in many fields and include sub-diffraction limit super lenses, cloaking devices, medical imaging, and flat space optics.

In general, the periodicity of the sub-wavelength-scale structure of a metamaterial is too small for incident electromagnetic waves to scatter between adjacent elements, and so the wave itself is not resonant. However, when the metamaterial contains a metallic element, an incident wave may induce an oscillating current which itself is resonant and emits some response. Such metamaterials are called **resonant**. Other materials, lacking conductive components and resonant currents, are called **nonresonant**, derive their properties simply from how their structure scatters incident power, and can have arbitrarily small periodicity. Metamaterials can broadly be divided into these two classes.

The CoPILOT lens design problem concerns an etched silicon surface with features smaller than the wavelength of the THz radiation it affects. This is an example of a nonresonant terahertz metamaterial - a metamaterial which exhibits desirable properties in the terahertz range, usually defined as 0.1 to 10 THz. The lack of naturally occurring materials and practical technologies which interact with waves in this frequency band is called the **terahertz gap** and has led to great interest in techniques for the development of devices which function in this range.

The term **dielectric metasurface** can be used to describe the CoPILOT device. This refers to a surface of sub-wavelength dielectric structures whose optical properties are governed by a large number of geometrical parameters. This project deals with the question of how to effectively and efficiently choose such parameters.

In general, the **forward problem**, that is, finding how light is scattered by a particular metasurface, is well understood. In simple cases it admits analytical solutions, and is otherwise solved commonly via a number of numerical solver methods for Maxwell's Equations. However, finding effective metasurface parameters requires solving the **inverse problem** - that of finding an appropriate lens topology given a desired scattering pattern. This reverse problem becomes intrinsically ill-posed in the presence of multiple light scattering, which is the case when dealing with metamaterials[19]. As a result, normal numerical solver methods for the reverse problem are computationally intractable and the investigation of more powerful computational techniques from optimization and machine learning is necessary.

Several papers discussed in the section on previous results [19][20][21][22][23][24] have already proposed and implemented such techniques for similar, but not identical, applications. In order to understand how these techniques work and decide which to apply, first methods to solve the forward problem must be understood.

3.2 Computational Electromagnetics

A key component of the most common techniques for metasurface design is a method for predicting a device's electromagnetic response to some incident wave, that is, a **forward solver**. Normally this amounts to solving Maxwell's equations numerically for approximations of the electric and magnetic field values in the space around the device. In order to do so, some understanding of computational electromagnetics is required.

In this section Maxwell's equations are briefly reintroduced and used as a motivation for rigorous coupled-wave analysis, a semi-analytical fast solver method which is integral to the successful optimization of lenses for CoPILOT.

3.2.1 Maxwell's Equations

In the following table, **Maxwell's equations**, a set of partial differential equations which constitute a model of classical electromagnetics and underlie all solver methods used to analyze and design metasurfaces, are summarized. They are considered here in differential form.

Name	Statement	Explanation
Gauss's Law	$\nabla \cdot \mathbf{E} = \rho/\epsilon_0$	Electric flux through a closed surface is proportional to the charge enclosed by that surface.
Gauss's Law for Magnetism	$\nabla \cdot \mathbf{H} = 0$	Magnetic monopoles do not exist.
Faraday's Law of Induction	$\nabla \times \mathbf{E} = -\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}$	A time-varying magnetic field always accompanies a spatially varying electric field.
Ampere's Law (with Maxwell's Addition)	$\nabla \times \mathbf{H} = \mathbf{J} + \epsilon_0 \frac{\mathrm{d}\mathbf{E}}{\mathrm{d}t}$	The magnetic field induced around a closed loop is related to the electric current through that loop.

Table 1: Statement of Maxwell's Equations in a vacuum, differential form. Here, $\mathbf{E}$ is the electric field intensity (V/m), $\mathbf{H}$ the magnetic field intensity (A/m), ρ a charge density (C/m^3), $\mathbf{J}$ a current density (A/m^2), ϵ_0 the permittivity of free space (F/m), μ_0 the permeability of free space (H/m), and $\nabla\cdot, \nabla\times$ the divergence and curl differential operators.

In the remainder of this section, only propagating $\mathbf{E}$ and $\mathbf{H}$ fields with no charge or current sources are considered, so $\rho = 0$ and $\mathbf{J} = 0$.

A number of numerical simulation methods for finding approximate solutions to these equations exist, which hold for general incident waves and device

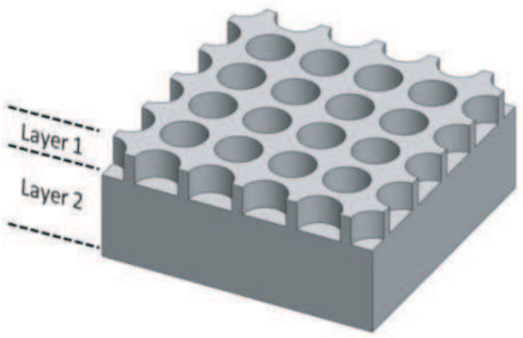

Figure 5: A layered, periodic dielectric structure[26].

shapes. Among these, those most widely adopted include the finite element method (FEM), the finite difference time domain method (FDTD), and the finite difference frequency domain method (FDFD). Other methods may rely on assumptions about the geometry of a scattering device but can gain increased performance as a trade-off.

3.2.2 Formulation of RCWA

Rigorous coupled-wave analysis, or RCWA, is a semi-analytical method used to solve Maxwell's equations for scattering from layered, periodic dielectric structures, a class of objects which include some metasurfaces. It is considered the dominant method in the analysis of such structures, owing to its high computational efficiency[25].

The following mathematical formulation of RCWA is adapted from the Computational Electrodynamics course notes by Dr. Raymond C. Rumpf[26], who also originated key components of the formulation[25].

Consider a 3d dielectric structure such as the one depicted in Figure 5.

The structure is uniform in the z (longitudinal) direction, and periodic in the x and y (transverse) directions. A single spatial period of the structure is called a **unit cell**. As a "semi-analytical" method, RCWA seeks to solve for the response analytically in the longitudinal direction, while solving numerically in the transverse directions via discretization in Fourier space.

To describe the solution inside one of the device layers, some material parameters must be introduced, namely some space-dependent **relative permittivity** and **permeability** $\epsilon_r(x, y)$ and $\mu_r(x, y)$. Permittivity represents the degree to which a dielectric material is polarized in response to an applied electric field, while permeability measures magnetization of the material in response to an applied magnetic field. "Relative" here denotes that the ratio of these values to the permittivity and permeability of free space is used. For float-zone silicon at room temperature, a relative permittivity of 11.9 and relative permeability of 1.0 is observed[27]. Note that these parameters do not depend on z because each of the layers is regular in the longitudinal direction.

In order to derive this method, Maxwell's equations must first be framed in their time-harmonic form. Because the first two equations in Table 1 are not time-dependent, they remain unchanged, and Faraday's Law and Ampere's Law become:

$$\boldsymbol{\nabla} \times \mathbf{E} = k_0 \mu_r(x,y)\tilde{\mathbf{H}} \text{ and } \boldsymbol{\nabla} \times \tilde{\mathbf{H}} = k_0 \epsilon_r(x,y)\mathbf{E},$$

$$\text{where } k_0 = \omega\sqrt{\mu_0 \epsilon_0}, \text{ and } \tilde{\mathbf{H}} = -i\sqrt{\mu_0/\epsilon_0}\mathbf{H}$$

Here, k_0 is the wavenumber corresponding to each harmonic of frequency ω, and $\tilde{\mathbf{H}}$ is the normalized magnetic field intensity. This normalization is helpful because it eliminates any sign inconsistency introduced with the complex values and equalizes the amplitudes of $\mathbf{E}$ and $\mathbf{H}$.

A complex Fourier series expansion of the material properties ϵ_r and μ_r in the transverse directions can now be taken, leaving the z direction alone:

$$\epsilon_r(x,y) = \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} a_{m,n} e^{i(m\mathbf{T_1}+n\mathbf{T_2})\mathbf{r}}$$

$$a_{m,n} = \frac{1}{\Lambda_x \Lambda_y} \int_{-\Lambda_x/2}^{\Lambda_x/2} \int_{-\Lambda_y/2}^{\Lambda_y/2} \epsilon_r(x,y) e^{-i(m\mathbf{T_1}+n\mathbf{T_2})\mathbf{r}} \, dy \, dx$$

$$\mu_r(x,y) = \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} b_{m,n} e^{i(m\mathbf{T_1}+n\mathbf{T_2})\mathbf{r}}$$

$$b_{m,n} = \frac{1}{\Lambda_x \Lambda_y} \int_{-\Lambda_x/2}^{\Lambda_x/2} \int_{-\Lambda_y/2}^{\Lambda_y/2} \mu_r(x,y) e^{-i(m\mathbf{T_1}+n\mathbf{T_2})\mathbf{r}} \, dy \, dx$$

$\mathbf{T_1} = \frac{2\pi}{\Lambda_x}\hat{x}$ and $\mathbf{T_2} = \frac{2\pi}{\Lambda_y}\hat{y}$ are called **reciprocal lattice vectors**, and provide a simpler way to write the expansion terms. M and N are the number of spatial harmonics in the expansion in the x and y directions, respectively. They should both be odd and rounded down when divided by 2 so that the expansion is centered on $(m,n) = (0,0)$. Λ_x and Λ_y are the lengths of the spatial interval of the expansion in x and y, or equivalently the transverse dimensions of the unit cell. Following this expansion, all information about the permittivity and permeability distributions is contained in the Fourier coefficients $a_{m,n}$ and $b_{m,n}$.

The fields can then be expanded as

$$\mathbf{E}(x,y,z) = \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} \mathbf{S}(m,n;z) e^{-i(k_x(m,n)x + k_y(m,n)y)}$$

$$\tilde{\mathbf{H}}(x,y,z) = \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} \mathbf{U}(m,n;z) e^{-i(k_x(m,n)x + k_y(m,n)y)}$$

$$\text{where } \mathbf{k}_{xy}(m,n) = \boldsymbol{\beta} - m\mathbf{T_1} - n\mathbf{T_2}, \text{ and}$$

$$k_x(m,n) = \beta_x - mT_{1,x} - nT_{2,x}, \; k_y(m,n) = \beta_y - mT_{1,y} - nT_{2,y}$$

This expansion is a representation of the $\mathbf{E}$ and $\mathbf{H}$ fields as a finite sum of spatially infinite 2d plane waves, or **spatial harmonics**. m and n parameterize the angle and period of each plane wave, and $\mathbf{S}$ and $\mathbf{U}$ are their (z-dependent) complex amplitudes. I adopt the convention that e^{-ikz} represents a wave travelling in the $+z$ direction.

$\mathbf{k}_{xy}$ is the **wave vector** which characterizes each plane wave in our expansion, pointing the direction normal to the wave and with magnitude equal to its wavenumber. Solutions to Maxwell's equations must obey **Blotch's theorem**, which states that waves in a periodic dielectric structure take the form of a plane wave modulated by a periodic function. $\boldsymbol{\beta}$ is the wave vector of this aperiodic plane wave component, which can be thought of equivalently as the wave number of an incident wave to the device. As an example, assuming free space outside of our layer, for an incident plane wave of frequency ω at incident polar angle θ and azimuthal angle ϕ, there is simply

$$\boldsymbol{\beta} = \omega\epsilon_0\mu_0 \begin{bmatrix} \sin(\theta)\cos(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\theta) \end{bmatrix}$$

The goal is to find the complex amplitudes $\mathbf{S}$ and $\mathbf{U}$, which can then be converted back into field values in cartesian space. To do so, the curl operators in our time-harmonic Maxwell equations can be expanded and then the Fourier expansions for $\epsilon_r, \rho_r, \mathbf{E}$ and $\tilde{\mathbf{H}}$ can be substituted in. Consider for Ampere's Law:

Equation Curl Expanded

$$\boldsymbol{\nabla}\times\mathbf{E} = k_0\mu_r(x,y)\tilde{\mathbf{H}} \implies \begin{cases} \frac{\mathrm{d}E_z}{\mathrm{d}y} - \frac{\mathrm{d}E_y}{\mathrm{d}z} = k_0\mu_r(x,y)\tilde{H}_x \\ \frac{\mathrm{d}E_x}{\mathrm{d}z} - \frac{\mathrm{d}E_z}{\mathrm{d}x} = k_0\mu_r(x,y)\tilde{H}_y \\ \frac{\mathrm{d}E_z}{\mathrm{d}x} - \frac{\mathrm{d}E_x}{\mathrm{d}y} = k_0\mu_r(x,y)\tilde{H}_z \end{cases} \implies$$

Semi-Analytical Fourier Space

$$\begin{cases} -i\tilde{k}_y(m,n)S_z(m,n;\tilde{z}) - \dfrac{\mathrm{d}S_y(m,n;\tilde{z})}{\mathrm{d}\tilde{z}} = \displaystyle\sum_{q=-M/2}^{M/2}\sum_{r=-N/2}^{N/2} b_{m-q,n-r}U_x(q,r;\tilde{z}) \\ \dfrac{\mathrm{d}S_x(m,n;\tilde{z})}{\mathrm{d}\tilde{z}} + i\tilde{k}_x(m,n)S_z(m,n;\tilde{z}) = \displaystyle\sum_{q=-M/2}^{M/2}\sum_{r=-N/2}^{N/2} b_{m-q,n-r}U_y(q,r;\tilde{z}) \\ -i\tilde{k}_x(m,n)S_y(m,n;\tilde{z}) + i\tilde{k}_y(m,n)S_x(m,n;\tilde{z}) = \displaystyle\sum_{q=-M/2}^{M/2}\sum_{r=-N/2}^{N/2} b_{m-q,n-r}U_z(q,r;\tilde{z}) \end{cases}$$

Here the normalized wave vector components $\tilde{k}_x = k_x/k_0, \tilde{k}_y = k_y/k_0$ and $\tilde{k}_z = k_z/k_0$ are used, as well as the normalized longitudinal coordinate $\tilde{z} = k_0 z$. These equations hold true for all choices of $m \in [-M/2, M/2]$ and $n \in [-N/2, N/2]$, and so the combined set of all equations for each choice can be written as one matrix equation. To do so, some matrices and column vectors are introduced:

$$\mathbf{s}_x = \begin{bmatrix} S_x(1,1), & S_x(1,2), & \cdots & S_x(M,N) \end{bmatrix}^T$$

and equivalently for $\mathbf{s}_y, \mathbf{s}_z, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z$,

$$\tilde{\mathbf{K}}_x = \begin{bmatrix} \tilde{k}_x(1,1) & & & 0 \\ & \tilde{k}_x(1,2) & & \\ & & \ddots & \\ 0 & & & \tilde{k}_x(M,N) \end{bmatrix}, \text{ and equivalently for } \tilde{\mathbf{K}}_y, \tilde{\mathbf{K}}_z.$$

In addition, there are some useful definitions for the material properties:

$$[\![\epsilon_r]\!] = [\epsilon_{r(m,n)}], \text{ for } \epsilon_{r(m,n)} = \sum_{q=-M/2}^{M/2} \sum_{r=-N/2}^{N/2} a_{m-q,n-r} \text{ and}$$

$$[\![\mu_r]\!] = [\mu_{r(m,n)}], \text{ for } \mu_{r(m,n)} = \sum_{q=-M/2}^{M/2} \sum_{r=-N/2}^{N/2} b_{m-q,n-r}$$

These $[\epsilon_r]$ and $[\mu_r]$ are symmetric convolution matrices which contain all information about the permittivity and permeability distributions in the layer. Ampere's Law in semi-analytical Fourier space is then

$$\begin{cases} -i\tilde{\mathbf{K}}_y \mathbf{s}_z - \dfrac{\mathrm{d}}{\mathrm{d}\tilde{z}}\mathbf{s}_y = [\mu_r]\mathbf{u}_x \\ \dfrac{\mathrm{d}}{\mathrm{d}\tilde{z}}\mathbf{s}_x + i\tilde{\mathbf{K}}_x \mathbf{s}_z = [\mu_r]\mathbf{u}_y \\ \tilde{\mathbf{K}}_x \mathbf{s}_y - \tilde{\mathbf{K}}_y \mathbf{s}_x = [\mu_r]\mathbf{u}_z \end{cases}$$

The longitudinal term $\mathbf{u}_z$ can be eliminated by solving for it in the third equation. After expanding the result and simplifying, there is the block matrix form

$$\frac{\mathrm{d}}{\mathrm{d}\tilde{z}} \begin{bmatrix} \mathbf{s}_x \\ \mathbf{s}_y \end{bmatrix} = \mathbf{P} \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix}, \ (1)$$

$$\text{where } \mathbf{P} = \begin{bmatrix} \tilde{\mathbf{K}}_x[\epsilon_r]^{-1}\tilde{\mathbf{K}}_y & [\mu_r] - \tilde{\mathbf{K}}_x[\epsilon_r]^{-1}\tilde{\mathbf{K}}_x \\ \tilde{\mathbf{K}}_y[\epsilon_r]^{-1}\tilde{\mathbf{K}}_y - [\mu_r] & -\tilde{\mathbf{K}}_y[\epsilon_r]^{-1}\tilde{\mathbf{K}}_x \end{bmatrix}$$

Applying the same steps for Faraday's Law, there is

$$\frac{\mathrm{d}}{\mathrm{d}\tilde{z}} \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} = \mathbf{Q} \begin{bmatrix} \mathbf{s}_x \\ \mathbf{s}_y \end{bmatrix}, \ (2)$$

$$\text{where } \mathbf{Q} = \begin{bmatrix} \tilde{\mathbf{K}}_x[\mu_r]^{-1}\tilde{\mathbf{K}}_y & [\epsilon_r] - \tilde{\mathbf{K}}_x[\mu_r]^{-1}\tilde{\mathbf{K}}_x \\ \tilde{\mathbf{K}}_y[\mu_r]^{-1}\tilde{\mathbf{K}}_y - [\epsilon_r] & -\tilde{\mathbf{K}}_y[\mu_r]^{-1}\tilde{\mathbf{K}}_x \end{bmatrix}$$

Finally, taking the derivative of (2) with respect to $\tilde{z}$ and substituting the result into (1), the **matrix wave equation** for $\mathbf{s}_x$ and $\mathbf{s}_y$ is obtained:

$$\frac{\mathrm{d}^2}{\mathrm{d}\tilde{z}^2} \begin{bmatrix} \mathbf{s}_x \\ \mathbf{s}_y \end{bmatrix} - \mathbf{\Omega}^2 \begin{bmatrix} \mathbf{s}_x \\ \mathbf{s}_y \end{bmatrix} = 0, \text{where } \mathbf{\Omega}^2 = \mathbf{PQ}$$

(*Standard PQ Form*)

This is simply a large number of ODEs which each admit an analytical solution. The solutions are of the form

$$\begin{bmatrix} \mathbf{s}_x(\tilde{z}) \\ \mathbf{s}_y(\tilde{z}) \end{bmatrix} = e^{-\mathbf{\Omega}\tilde{z}}\mathbf{s}^+(0) + e^{\mathbf{\Omega}\tilde{z}}\mathbf{s}^-(0)$$

where $\mathbf{s}^+(0)$ and $\mathbf{s}^-(0)$ are initial values of the problem for forward and backwards propagating waves, respectively.

Now all that is required are the matrices $e^{-\mathbf{\Omega}\tilde{z}}$ and $e^{\mathbf{\Omega}\tilde{z}}$, which can be found using the property

$$f(\mathbf{A}) = \mathbf{W}f(\boldsymbol{\lambda})\mathbf{W}^{-1}, \text{ where}$$

$\mathbf{A}$ is an arbitrary full rank matrix, f an arbitrary matrix function, $\mathbf{W}$ the matrix of A's eigenvectors, and $\boldsymbol{\lambda}$ the matrix of corresponding eigenvalues. Applying this to $\mathbf{\Omega}^2$ with each $f(A) = e^{A\tilde{z}}$ and $f(A) = e^{-A\tilde{(z)}}$ gives

$$e^{-\mathbf{\Omega}\tilde{z}} = \mathbf{W}e^{-\boldsymbol{\lambda}\tilde{z}}\mathbf{W}^{-1} \text{ and } e^{\mathbf{\Omega}\tilde{z}} = \mathbf{W}e^{\boldsymbol{\lambda}\tilde{z}}, \text{ where}$$

$$e^{\boldsymbol{\lambda}\tilde{z}} = \begin{bmatrix} e^{\sqrt{\lambda_1^2}\tilde{z}} & & 0 \\ & \ddots & \\ 0 & & e^{\sqrt{\lambda_{N_\lambda^2}}\tilde{z}} \end{bmatrix}$$

Here, $\mathbf{W}$ is the matrix of $\mathbf{\Omega}^2$'s eigenvectors, and $\lambda_1 ... \lambda_{N_\lambda}$ are its corresponding eigenvalues. This gives a final solution for $\mathbf{s}$

$$\begin{bmatrix} \mathbf{s}_x(\tilde{z}) \\ \mathbf{s}_y(\tilde{z}) \end{bmatrix} = \mathbf{W}e^{-\boldsymbol{\lambda}\tilde{z}}\mathbf{c}^+ + \mathbf{W}e^{\boldsymbol{\lambda}\tilde{z}}\mathbf{c}^-, \text{ where}$$

$$\mathbf{c}^+ = \mathbf{W}^{-1}\mathbf{s}^+(0) \text{ and } \mathbf{c}^- = \mathbf{W}^{-1}\mathbf{s}^-(0)$$

Writing a similar expression for $\mathbf{u}$ and combining in order to write the final solution for both s and u inside a single layer, there is

$$\boldsymbol{\psi}(\tilde{z}) = \begin{bmatrix} \mathbf{s}_x(\tilde{z}) \\ \mathbf{s}_y(\tilde{z}) \\ \mathbf{u}_x(\tilde{z}) \\ \mathbf{u}_y(\tilde{z}) \end{bmatrix} = \begin{bmatrix} \mathbf{W} & \mathbf{W} \\ -\mathbf{V} & \mathbf{V} \end{bmatrix} \begin{bmatrix} e^{-\boldsymbol{\lambda}\tilde{z}} & 0 \\ 0 & e^{\boldsymbol{\lambda}\tilde{z}} \end{bmatrix} \begin{bmatrix} \mathbf{c}^+ \\ \mathbf{c}^- \end{bmatrix}, \text{ where}$$

$$\mathbf{V} = \mathbf{Q}\mathbf{W}\boldsymbol{\lambda}^{-1}$$

Therefore, finding the complex amplitudes $\mathbf{s_x}$, $\mathbf{s_y}$ and $\mathbf{u_x}$, $\mathbf{u_y}$, and thus the solution to Maxwell's equations for wave scattering inside a single dielectric layer, amounts to solving the eigenvalue problem for $\mathbf{\Omega}^2$ to obtain $\mathbf{W}$ and $\boldsymbol{\lambda}$. There are no strong assumptions about $\mathbf{\Omega}^2$, meaning that this is in general a complex-valued degenerate eigenproblem. The implications of this for solving the inverse problem are significant and discussed in section 4.3.

However, if the layer is homogeneous, meaning that the material is isotropic and ϵ_r and μ_r are constant values which no longer depend on x and y, some simplifying assumptions can be made in order to obtain the solution

$$\mathbf{W} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I} \end{bmatrix}, \ \boldsymbol{\lambda} = \begin{bmatrix} i\tilde{\mathbf{K}}_z & 0 \\ 0 & i\tilde{\mathbf{K}}_z \end{bmatrix}, \ \tilde{\mathbf{K}}_z = (\sqrt{\mu_r^*\epsilon_r^*\mathbf{I} - \tilde{\mathbf{K}}_x^2 - \tilde{\mathbf{K}}_y^2})^*, \ \mathbf{V} = \mathbf{Q}\boldsymbol{\lambda}^{-1}$$

This means that for homogeneous layers, the eigenvalue problem does not actually need to be solved.

Now consider the problem for a device consisting of multiple non-homogeneous periodic layers. At each layer, $[\![\epsilon_r]\!]$ and $[\![\mu_r]\!]$ are different, and therefore also $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{\Omega}^2$, $\mathbf{W}$, $\mathbf{V}$, $\boldsymbol{\lambda}$, $\mathbf{c}^+$ and $\mathbf{c}^-$. However, boundary conditions dictate that $\tilde{\mathbf{K}}_x$ and $\tilde{\mathbf{K}}_y$ remain the same between layers. This means that, for some layer i of thickness L_i, sandwiched by two homogeneous layers 1 and 2 of zero thickness each with constant ϵ_0 and μ_0, there are boundary conditions

$$\boldsymbol{\psi}_1 = \boldsymbol{\psi}_i(0) \Rightarrow \begin{bmatrix}\mathbf{W}_0 & \mathbf{W}_0\\ -\mathbf{V}_0 & \mathbf{V}_0\end{bmatrix}\begin{bmatrix}\mathbf{c}_0^+\\ \mathbf{c}_0^-\end{bmatrix} = \begin{bmatrix}\mathbf{W}_i & \mathbf{W}_i\\ -\mathbf{V}_i & \mathbf{V}_i\end{bmatrix}\begin{bmatrix}\mathbf{c}_i^+\\ \mathbf{c}_i^-\end{bmatrix}, \text{ and}$$

$$\boldsymbol{\psi}_i(k_0L_i) = \boldsymbol{\psi}_2 \Rightarrow \begin{bmatrix}\mathbf{W}_i & \mathbf{W}_i\\ -\mathbf{V}_i & \mathbf{V}_i\end{bmatrix}\begin{bmatrix}e^{-\boldsymbol{\lambda}_i k_0 L_i} & 0\\ e^{\boldsymbol{\lambda}_i k_0 L_i} & 0\end{bmatrix}\begin{bmatrix}\mathbf{c}_i^+\\ \mathbf{c}_i^-\end{bmatrix} = \begin{bmatrix}\mathbf{W}_2 & \mathbf{W}_2\\ -\mathbf{V}_0 & \mathbf{V}_0\end{bmatrix}\begin{bmatrix}\mathbf{c}_0^+\\ \mathbf{c}_0^-\end{bmatrix}$$

Here, $\mathbf{W}_0$, $\mathbf{V}_0$, $\mathbf{c}_0^+$ and $\mathbf{c}_0^-$ are calculated using the solution for homogeneous layers. For any such layer i a **scattering matrix** $\mathbf{S}^{(i)}$ can be defined such that

$$\begin{bmatrix}\mathbf{c}_1^-\\ \mathbf{c}_2^+\end{bmatrix} = \mathbf{S}^{(i)}\begin{bmatrix}\mathbf{c}_1^+\\ \mathbf{c}_2^-\end{bmatrix}, \mathbf{S}^{(i)} = \begin{bmatrix}\mathbf{S}_{11}^{(i)} & \mathbf{S}_{12}^{(i)}\\ \mathbf{S}_{21}^{(i)} & \mathbf{S}_{22}^{(i)}\end{bmatrix}$$

$$\mathbf{S}_{11}^{(i)} = (\mathbf{A}_i - \mathbf{X}_i\mathbf{B_i}\mathbf{A}_i^{-1}\mathbf{X}_i\mathbf{B}_i)^{-1}(\mathbf{X}_i\mathbf{B}_i\mathbf{A}_i^{-1}\mathbf{X}_i\mathbf{A}_i - \mathbf{B}_i)$$

$$\mathbf{S}_{12}^{(i)} = (\mathbf{A}_i - \mathbf{X}_i\mathbf{B}_i\mathbf{A}_i^{-1}\mathbf{X}_i\mathbf{B}_i)^{-1}\mathbf{X}_i(\mathbf{A}_i - \mathbf{B}_i\mathbf{A}_i^{-1}\mathbf{B}_i)$$

$$\mathbf{S}_{21}^{(i)} = \mathbf{S}_{12}^{(i)},\ \mathbf{S}_{22}^{(i)} = \mathbf{S}_{11}^{(i)}$$

$$\mathbf{A}_i = \mathbf{W}_i^{-1}\mathbf{W}_0 + \mathbf{V}_i^{-1}\mathbf{V}_0,\ \mathbf{B}_i = \mathbf{W}_1^{-1}\mathbf{W}_0 - \mathbf{V}_i^{-1}\mathbf{V}_0,\ \mathbf{X}_i = e^{-\boldsymbol{\lambda}_i k_0 L_i}$$

Here, c_1 and c_2 correspond to the reflection and transmission sides of the layer, respectively. An entire device can be modelled as a series of such layers, separated by homogeneous gap layers of zero thickness with constant ϵ_0 and μ_0. Such zero thickness gaps have no effect on the physical validity of the model and simplify the calculation of each scattering matrix by ensuring that each depends on only the physical parameters of its corresponding layer and not those of adjacent ones. Thus if the device contains multiple identical layers in different positions, the corresponding scattering matrix only needs to be constructed once.

This convention for the scattering matrix ensures that it is symmetric and thus highly efficient to calculate, as well as consistent with convention[25]. In addition, it allows the matrices of individual layers to be easily combined into a **global scattering matrix** as

$$\mathbf{S}^{(global)} = \mathbf{S}^{(ref)} \otimes \mathbf{S}^{(device)} \otimes \mathbf{S}^{(trn)},$$
$$\mathbf{S}^{(device)} = \mathbf{S}^{(1)} \otimes \mathbf{S}^{(2)} \otimes \cdots \otimes \mathbf{S}^{(N_L)} \text{ for } N_L \text{ layers}$$

Here, $\otimes$ is the **Redheffer star product**[28] defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{bmatrix} \otimes \begin{bmatrix}\mathbf{B}_{11} & \mathbf{B}_{12}\\ \mathbf{B}_{21} & \mathbf{B}_{22}\end{bmatrix}$$

$$= \begin{bmatrix}\mathbf{B}_{11}(\mathbf{I} - \mathbf{A}_{12}\mathbf{B}_{21})^{-1}\mathbf{A}_{11} & \mathbf{B}_{12} + \mathbf{B}_{11}(\mathbf{I} - \mathbf{A}_{12}\mathbf{B}_{21})^{-1}\mathbf{A}_{12}\mathbf{B}_{22}\\ \mathbf{A}_{21} + \mathbf{A}_{22}(\mathbf{I} - \mathbf{B}_{21}\mathbf{A}_{12}^{-1}\mathbf{B}_{21}\mathbf{A}_{11} & \mathbf{A}_{22}(\mathbf{I} - \mathbf{B}_{21}\mathbf{A}_{12})^{-1}\mathbf{B}_{22}\end{bmatrix}$$

The scattering matrices $\mathbf{S}^{(ref)}$ and $\mathbf{S}^{(trn)}$ represent the device's reflection (i.e. $-z$) and transmission ($+z$) regions, respectively, and are calculated using some constant physical parameters in those regions $\epsilon_{r,ref}, \epsilon_{r,trn}, \mu_{r,ref}$ and $\mu_{r,trn}$ as

$$\mathbf{S}_{11}^{(ref)} = -\mathbf{A}_{ref}^{-1}\mathbf{B}_{ref},\ \mathbf{S}_{12}^{(ref)} = 2\mathbf{A}_{ref}^{-1},$$

$$\mathbf{S}_{21}^{(ref)} = \frac{1}{2}(\mathbf{A}_{ref} - \mathbf{B}_{ref}\mathbf{A}_{ref}^{-1}\mathbf{B}_{ref}),\ \mathbf{S}_{22}^{(ref)} = \mathbf{B}_{ref}\mathbf{A}_{ref}^{-1},$$

$$\mathbf{A}_{ref} = \mathbf{W}_0^{-1}\mathbf{W}_{ref} + \mathbf{V}_0^{-1}\mathbf{V}_{ref},\ \mathbf{B}_{ref} = \mathbf{W}_0^{-1}\mathbf{W}_{ref} - \mathbf{V}_0^{-1}\mathbf{V}_{ref},$$

$$\text{with equivalent expressions for } \mathbf{A}_{trn}, \mathbf{B}_{trn}, \text{ and } \mathbf{S}^{(trn)}$$

Knowing $\mathbf{S}^{(global)}$, an approximation of the transmitted and reflected fields for some incident wave and and arbitrary periodic, layered device can finally be obtained. For some normalized incident polarization vector $\mathbf{P} = [p_x, p_y, p_x]^T, \|\mathbf{P}\| = 1$, and vector $\boldsymbol{\delta}_{0,pq}$ of length $M \times N$, which has 0s in all positions except for a 1 in the p, q^{th} position indicating the mode of the incident wave (normally the center position), there is

$$\mathbf{c}_{inc} = \mathbf{W}_{inc}^{-1}\begin{bmatrix} p_x\boldsymbol{\delta}_{0,pq} \\ p_y\boldsymbol{\delta}_{0,pq} \end{bmatrix}, \text{ and}$$

$$\mathbf{r}_T = \begin{bmatrix} \mathbf{r}_x \\ \mathbf{r}_y \end{bmatrix} = \mathbf{W}_{ref}\mathbf{S}_{11}\mathbf{c}_{inc},\ \mathbf{t}_T = \begin{bmatrix} \mathbf{t}_x \\ \mathbf{t}_y \end{bmatrix} = \mathbf{W}_{ref}\mathbf{S}_{21}\mathbf{c}_{inc}$$

These are the transverse components of the complex wave amplitudes in the reflected and transmitted regions. The longitudinal components are

$$\mathbf{r}_z = -\tilde{\mathbf{K}}_{z,ref}^{-1}(\tilde{\mathbf{K}}_x\mathbf{r}_x + \tilde{\mathbf{K}}_y\mathbf{r}_y),\ \mathbf{t}_z = -\tilde{\mathbf{K}}_{z,trm}^{-1}(\tilde{\mathbf{K}}_x\mathbf{t}_x + \tilde{\mathbf{K}}_y\mathbf{t}_y), \text{ with}$$

$$\tilde{\mathbf{K}}_{z,ref} = -\left(\sqrt{\mu_{r,ref}^*\epsilon_{r,ref}^*\mathbf{I} - \tilde{\mathbf{K}}_x^2 - \tilde{\mathbf{K}}_y^2}\right)^*, \text{ and equivalently for } \tilde{\mathbf{K}}_{z,trn}$$

To get the final fields in both regions, these plane wave amplitudes can be transformed back to cartesian space, and then propagated any distance z away from the transmission or reflection surface of the device. Several methods for calculating this propagation exist, such as the band-limited angular spectrum method[29]. An implementation of this propagation method is provided in the code of Colburn and Majumdar[23].

3.3 Algorithmic Differentiation

Differentiability of a process with respect to its inputs often brings many advantages and enables many methods for modelling it. Having access to the derivatives of a forward solver enables direct optimization algorithms such as gradient descent for the inverse problem. Access to derivatives is also required when characterization of the quality of a solution depends on its derivatives, a property of physics-informed machine learning methods.

However, for an arbitrary function implemented as computer program, such as a numerical solver, the derivative is not, in general, available. There are several techniques which attempt to address this.

In **numerical differentiation**, the method of finite differences is used to compute an approximation to the derivative using several evaluated function values. While simple to implement, this technique can encounter problems with floating-point round-off errors due to the discretization introduced. In addition, its performance does not scale well to higher-order derivatives or derivatives with respect to many inputs. These problems make it ill-suited to an optimization problem in which the target function is highly nonlinear and has many inputs, such as scattering from a metasurface.

In **symbolic differentiation**, one attempts to represent the function as a single mathematical object or expression, which can be used to analytically compute the exact value of a derivative. However, in practice, this is a complicated task which requires the function to be rewritten in a number of fundamental ways - for example, normally numbers in the program must be converted to arbitrary-precision representation, so that the precision limits of standard floating point types do not impact the exactness of the computation. This is highly inefficient for programs with many inputs and operations, which includes most numerical solvers for physical problems.

A third option, **algorithmic differentiation** (also called automatic differentiation, computational differentiation, autodiff, or simply **AD**), attempts to solve all of these complications. AD takes advantage of the fact that any sequential computer program, no matter its complexity, is essentially a series of elementary operations. As long as the derivative of each elementary operation is known, the chain rule can be applied to evaluate the derivative of such a composed function.

AD contrasts with symbolic differentiation in that it relies only on the chain rule and knowledge about derivatives of elementary operations. Symbolic differentiation additionally relies on analytical results about the derivatives of more complicated operations, up to and including an entire program, which is the source of the method's difficulties.

AD has two operating modes, each of which computes the same values but with derivatives of the composing operations computed in a different order. In **forward accumulation**, derivatives of the first intermediate value (i.e. that after the first operation has been applied) with respect to the inputs are evaluated first, and the products of each operation's derivative are accumulated as we progress through the operations towards the output. For a function $y = f^n(f^{n-1}(...(f^1(x))$ with intermediate values $w_0 = x$, $w_i = f^{i-1}(w_{i-1})$, $w_n = y$, this computes

$$\frac{\mathrm{d}w_i}{\mathrm{d}x} = \frac{\mathrm{d}w_i}{\mathrm{d}w_{i-1}}\frac{\mathrm{d}w_{i-1}}{\mathrm{d}x} \text{ for } i = 1, 2...n$$

The alternative is **reverse accumulation**, which instead starts from the output and progresses backwards towards the input, computing

$$\frac{\mathrm{d}y}{\mathrm{d}w_i} = \frac{\mathrm{d}y}{\mathrm{d}w_{i+1}}\frac{\mathrm{d}w_{i+1}}{\mathrm{d}w_i} \text{ for } i = n-1, n-2...1$$

Each of these modes has performance advantages in certain applications. For computing the gradient of a function with a large number of inputs relative to the number of outputs, such as a loss function for a neural network, the reverse mode is far more computationally efficient[23].

3.3.1 Correctness of AD

Standard autodiff frameworks following this framework can be formally shown to be correct, in the sense of correctly computing the derivative at almost all points in the domain for some function. This applies to a large class of possibly non-differentiable functions, given that the function is differentiable almost everywhere and that some assumptions hold on the set of elementary operations. Specifically, each of the elementary operations utilized by the program must be **piecewise analytic under analytic partition** (or **PAP**)[30].

A PAP function $f : X \to Y$ is a piecewise function made up of partitions $\{A^i, f^i\}_{i\in\{1,2...\}}$ such that all f^i are analytic functions over their domains $X^i = \{x \in A^i | f^i(x)\}$, and A^i are analytic partitions[30].

Recall that an analytic function is infinitely differentiable and equal to its Taylor expansion. An analytic partition is any set $A \in \mathbb{R}^n$ for which some analytic functions $g_j^+ : X_j^+ \to \mathbb{R}$ and $g_k^- : X_k^- \to \mathbb{R}$ on open domains $X_j^+, X_k^- \subseteq \mathbb{R}^n$ for $j \in J$, $k \in K$, $J, K \in \mathbb{Z}_{>0}$ exist such that $A = \{x \in X_j^+ | (\forall j \in J) g_j^+(x) > 0\} \cup \{x \in X_k^- | (\forall k \in K) g_k^-(x) \leq 0\}$. In other words, an analytic partition can be broken up into a number of regions on each of which some analytic function exists such that for all points in that region, the analytic function is strictly positive or less than or equal to zero.

Most of the elementary operations used in common implementations of AD, such as PyTorch and TensorFlow, hold to this condition, and so these autodiff systems can be shown to be correct for all programs written using these operations. An example is the *relu* function $\phi(x) = max(0, x)$, a common activation function for neural networks. This function is not differentiable at 0, but the TensorFlow implementation returns 0 for its derivative evaluated at 0 anyway. This choice makes the TensorFlow implementation of *relu* in fact a PAP function and leads to AD correctness for programs using this operation.

3.3.2 Implementation of AD

There are several approaches to implementing programs such as to utilize algorithmic differentiation, a programming paradigm known as **differentiable programming**. A relatively straightforward approach is to overload basic arithmetic operators so that they return information about their derivatives in addition to their results. This can be accomplished efficiently in many languages - for example, C++.

Another approach formulation of the program as a computational graph. The nodes of such a graph represent operations, and its edges represent data dependency between them. This is the strategy employed by TensorFlow, a widely-used, flexible, open-source software framework for machine learning[31]. First released in 2015, TensorFlow provides a high-level scripting interface which can be used to compose such program graphs from elementary operations such as matrix multiplications and convolutions, and to execute these programs on powerful heterogeneous computer architectures including GPUs. This has enabled many applications of differential programming and machine learning across sectors.

Implementations of differentiable versions of common solver methods from computational electromagnets using AD have already been shown, including FDTD[32], FDFD[33], and RCWA[23].

4 Past Methods and Results from Literature

In this section, a number of previous papers which address the use of machine learning and optimization techniques for the design of metamaterial optical devices and metasurfaces are presented. The methods they employ are discussed in depth and categorized into three groups based on their general approach.

4.1 Scientific Machine Learning

The first class of solutions belongs to what can be described broadly as **scientific machine learning**. This refers simply to the application of standard methods in machine learning, such as modelling processes via deep neural networks, to scientific data sets. Applied to an inverse scattering problem, this approach leads to a neural network which replicates the results of a numerical solver method, efficiently predicting electromagnetic response for a device given the parameters of a metasurface. Such a network can be called a **forward network**. This enables, in turn, the training of an **inverse network** which can predict metasurface parameters given some desired response.

4.1.1 Jiang et al.

Jiang et al.[21] applies this idea in order to design metalenses for phase manipulation. Their goal is the training of a deep neural network, or DNN, which can predict metasurface geometric parameters of a device that produces some desired phase spectra. Six geometric parameters are sought. These are the transverse dimensions of three small TiO_2 structures called nanofins of fixed height, which are tiled across a substrate surface. The arrangement can be seen in Figure 6.

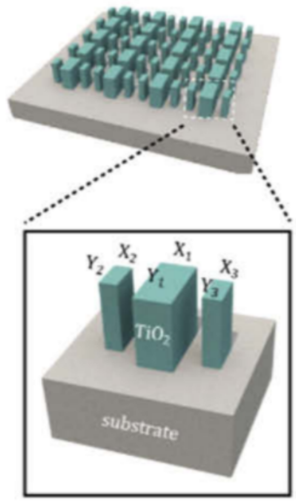

Figure 6: Metasurface design sought by Jiang et al. composed of repeating nanofin structures[21].

First, an FDTD solver is used to generate phase spectra, sampled at 81 frequency points over the $380nm$-$780nm$ band, for a sample of 7680 parameter combinations. The phase coefficients can be calculated from the scattering matrices used by FDTD, which resemble those from RCWA. Care is taken to cluster more samples around discontinuities in the phase, so that these discontinuities may be properly learned. A DNN with six hidden layers composed of approximately 600 neurons each is implemented using TensorFlow[31] and trained on the majority of the generated data, using a mean absolute error loss function which compares predicted spectra to generated spectra. Network hyperparameters such as batch size and learning rate are varied and compared in terms of network performance, with the best out of those tried selected, but no sophisticated hyperparameter search method is employed. This results in a forward network which models the FTFD solver with an error of <0.2 on 93% of the test data.

Next, the inverse network is initialized. The structure is the same as the forward network except that the loss function now compares predicted spectra to some desired spectra, and that the input layer of geometric parameters is now treated as a hidden layer and the parameters as trainable weights. The rest of the network weights are reused from the forward network and fixed.

In the reverse network optimization process, some arbitrary initial guesses for geometric parameters are first chosen. The network output is calculated, resulting in a loss, which can then be used to update the parameter guess via backpropagation. This is repeated until the loss is small enough to meet a stopping condition, and then the result is checked for correctness with a FDTD simulation. This optimization process is repeated to find other ideal parameters for any additional desired phase spectra. The authors also extend their process to predict and design for other phase properties, including group delay and group delay dispersion, in order to demonstrate the generalizable power of their DNN approach.

4.1.2 An et al.

An et al.[20] attempt a similar goal with a similar approach. The authors consider arrays of small dialectric structures of varying shape with 3-5 geometric parameters and possibly a variable permittivity. They call these building blocks **meta-atoms**, and arrange them on top of a substrate material of lower refractive index, as seen below.

As in Jiang et al.[21], a large amount of simulation-generated amplitude and phase spectra training data is created - over 50000 combinations of geometric parameters, and 31 frequency samples over the 30-60THz (5-10μm wavelength) band. Again, the problem of how to ensure that phase discontinuities are properly captured by the DNN model arises. Finding that these discontinuities occur in the amplitude and phase but that the real and imaginary parts of the complex transmission coefficient, from which the amplitude and phase can be calculated, are continuous, the authors decide to construct two independent networks which predict these parts. The outputs of these two DNNs can then be combined to predict amplitude and phase according to the formulas

$$\text{amplitude} = \sqrt{\text{Im}(T)^2 + \text{Re}(T)^2}, \text{ and}$$

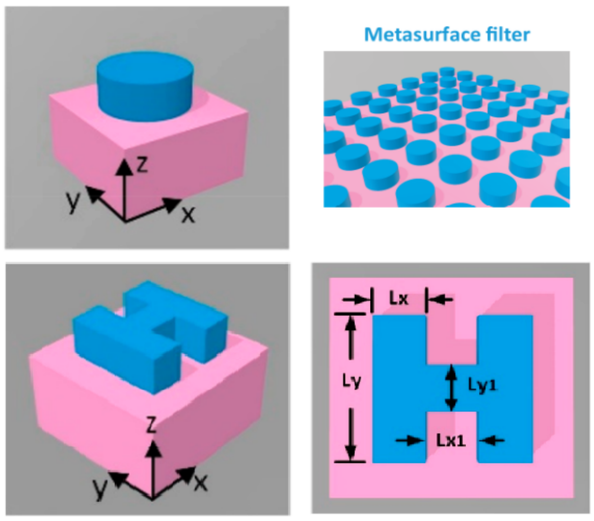

Figure 7: Metasurface design sought by An al. composed of arrays of dielectric "meta-atoms". Separate DNNs are employed for parameter selection of each meta-atom structure, such as the cylindrical and H-shaped ones seen here[20].

$$\text{phase} = \tan^{-1}\frac{\text{Im}(T)}{\text{Re}(T)}$$

Here, the transmission coefficient T is the ratio of the amplitude of the incident wave to that of the transmitted wave. T can be calculated directly from the results of any full-wave simulation such as FDTD and therefore can be used in the loss function for these DNNs. Each network is again implemented with TensorFlow[31] has three hidden layers consisting of 500, 500, and 200 neurons each. For a simple cylindrical meta-atom design, the resulting trained networks achieve an average mean squared error on the test set of 0.00035 for the real part and 0.00023 for the imaginary part of T. The authors go on to repeat this process, training networks to predict transmission coefficient components for over 30000 meta-atom structures, including the H-shaped one shown in Figure 7.

These authors take a slightly different approach to the design of the inverse network, which they call a "meta-filter design network". For any one meta-atom structure, the meta-filter design network is a separate DNN with four hidden layers of widths 500, 500, 500, and 50 neurons each which takes as input a desired phase spectra and produces design parameters as output. The resulting parameters are fed to the forward network, which produces a predicted spectra which can be compared to the target to produce a loss. This is then backpropagated in order to update the many weights and biases of the design network, while the forward network parameters remain unchanged. In this way, the trained forward network is used as part of the loss function, effectively a more computationally efficient stand-in for a standard numerical solver.

Because the meta-filter design network is a proper DNN of its own here, rather than Jiang's one layer added to the forward network, it can be trained once on a data set consisting of many desired phase spectra and afterwards be used to efficiently predict meta-atom parameters for many other desired spectra. This is in contrast to Jiang's method, wherein the inverse network must be re-optimized for each desired spectra.

4.2 Physics-Informed Neural Networks

While the methods discussed above may fall under the category of scientific machine learning, they do not actually take advantage of any physical knowledge known about the system. The closest they come to this is using numerical solvers based on it to generate training data. A more clever approach might be to embed information about the physical laws, that is, Maxwell's equations or some other set of PDEs, into the learning process itself. This can be achieved by incorporating the PDEs that govern the data set into the network's loss function. The network is then called a **physics-informed neural network**, or a **PINN**[34].

Say that, for a function $u(\mathbf{x})$ with $\mathbf{x} = (x, y)$ in a domain $\Omega \subseteq \mathbb{R}^2$, there is a PDE of the form

$$f\left(\mathbf{x}; \frac{\mathrm{d}u}{\mathrm{d}x}, \frac{\mathrm{d}u}{\mathrm{d}y}, \frac{\mathrm{d}^2u}{\mathrm{d}x^2}, \frac{\partial^2 u}{\partial x \partial y}, \frac{\mathrm{d}^2u}{\mathrm{d}y^2}; \lambda\right) = 0, \text{ for } \mathbf{x} \in \Omega,$$

with some mixed boundary conditions

$$u(\mathbf{x}) = g_D(\mathbf{x}) \text{ on } \Gamma_D \subset \partial\Omega \text{ and } \frac{\mathrm{d}u(\mathbf{x})}{\mathrm{d}\mathbf{x}} = \mathbf{g}_R(u, x, y) \text{ on } \Gamma_R \subset \partial\Omega$$

Here λ is an unknown parameter in the PDE system, which could represent, for example, some metasurface parameters. The goal of the PINN is to recover λ.

The PINN then consists of a DNN with an output $\hat{u}(x, y; \theta)$ which approximates u, where θ is a vector of all DNN weights and biases to be trained. The goal is then to train the DNN and optimize θ and λ so as to minimize the error between u and $\hat{u}$. This is achieved with a loss function of the form

$$\mathcal{L}(\theta, \lambda) = w_f \mathcal{L}_f(\theta, \lambda; \mathcal{T}_f) + w_i \mathcal{L}_i(\theta, \lambda; \mathcal{T}_i) + w_b \mathcal{L}_b(\theta, \lambda; \mathcal{T}_b), \text{ where}$$

$$\mathcal{L}_f(\theta, \lambda; \mathcal{T}_f) = \tfrac{1}{|\mathcal{T}_f|} \textstyle\sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \tfrac{\mathrm{d}\hat{u}}{\mathrm{d}x}, \tfrac{\mathrm{d}\hat{u}}{\mathrm{d}y}, \tfrac{\mathrm{d}^2\hat{u}}{\mathrm{d}x^2}, \tfrac{\partial^2\hat{u}}{\partial x \partial y}, \tfrac{\mathrm{d}^2\hat{u}}{\mathrm{d}y^2}; \lambda\right) \right\|_2^2,$$

$$\mathcal{L}_i(\theta, \lambda; \mathcal{T}_i) = \tfrac{1}{|\mathcal{T}_i|} \textstyle\sum_{\mathbf{x} \in \mathcal{T}_i} \|\hat{u}(\mathbf{x}) - u(\mathbf{x})\|_2^2,$$

$$\mathcal{L}_b(\theta, \lambda; \mathcal{T}_{bD}, \mathcal{T}_{bR)} = \tfrac{1}{|\mathcal{T}_{bD}|} \textstyle\sum_{\mathbf{x} \in \mathcal{T}_{bD}} \|(\hat{u}(\mathbf{x}) - g_D(\mathbf{x}))\|_2^2$$

$$+ \tfrac{1}{|\mathcal{T}_{bR}|} \textstyle\sum_{\mathbf{x} \in \mathcal{T}_{bR}} \left\| \left(\tfrac{\mathrm{d}\hat{u}(\mathbf{x})}{\mathrm{d}\mathbf{x}} - g_R(u, \mathbf{x}) \right) \right\|_2^2$$

Here, $\mathcal{L}_f$, $\mathcal{L}_i$ and $\mathcal{L}_b$ are loss terms representing adherence to the PDE, to a desired solution $u(\mathbf{x})$, and to the boundary conditions, respectively, and w_f, w_i and w_b are their weights. $\mathcal{T}_f$ and $\mathcal{T}_i$ are some set of points sampled from Ω at which these conditions are checked, which may be on a grid or chosen randomly. Similarly, $\mathcal{T}_{bD}$ are sampled from Γ_D and $\mathcal{T}_{bR}$ from Γ_R. If the PINN is implemented with an AD system such as TensorFlow, then the derivatives $\frac{\mathrm{d}\hat{u}(\mathbf{x})}{\mathrm{d}\mathbf{x}}$ are easily available at each training step.

The inclusion of the $\mathcal{L}_i$ term makes this an **inverse** problem which finds λ, but requires some known solution $u(\mathbf{x})$. In other words, given some known solution to the PDE, this network can find the PDE parameter λ such that the PDE admits such a solution. A remarkable fact is that with the minimal change of omitting $\mathcal{L}_i$, the PINN functions in a forward direction instead, finding an admissible solution $\hat{u}(\mathbf{x})$ given some λ. Adding this term is of insignificant computational cost, meaning that this PINN framework can solve forward and inverse problems on equal footing[19].

To train the PINN, normal forward evaluation and backpropagation is iterated until the loss is smaller than some σ. Afterwards, the obtained λ and $\hat{u}(\mathbf{x})$ can be verified using a numerical simulation method. The entire architecture is summarized in Figure 8.

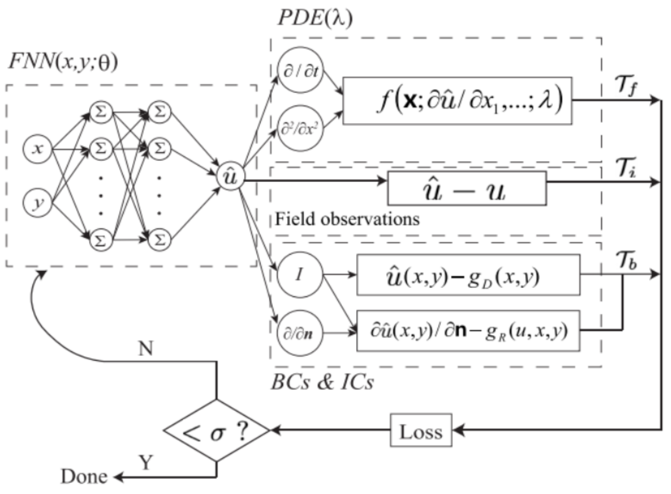

Figure 8: Framework of a general PINN, for the inverse problem to obtain PDE parameter λ. The left FNN (feed-forward neural network) is a model of the PDE solution, and the right side shows the terms of the loss function[19].

4.2.1 Chen et al.

Chen et al.[19] applies this general framework to the problem of homogenizing finite-size metamaterials. That is, they seek to replace a lattice of cylindrical dielectric meta-atoms each having some constant permittivity ϵ with a single dielectric cylinder having a permittivity profile $\epsilon(x, y)$. Both the lattice of meta-atoms and the single scatterer should produce the same scattering pattern. For this problem, the relevant PDE is the Helmholtz equation for weakly inhomogeneous 2d media under transverse magnetic (TM) polarization excitation, for the unknown electric field z component $\mathbf{E}_z$:

$$\nabla^2 \mathbf{E}_z(x, y) + \epsilon_r(x, y) k_0^2 \mathbf{E}_z = 0$$

Here k_0 is the wavenumber of the incident wave in free space and $\epsilon_r(x, y)$ is the sought permittivity profile. In terms of the general PINN framework,

$u(x, y)$ is generated from a single numerical simulation of the original meta-atom lattice, and the sought parameter λ is ϵ_r. The situation and results can be seen in Figure 9.

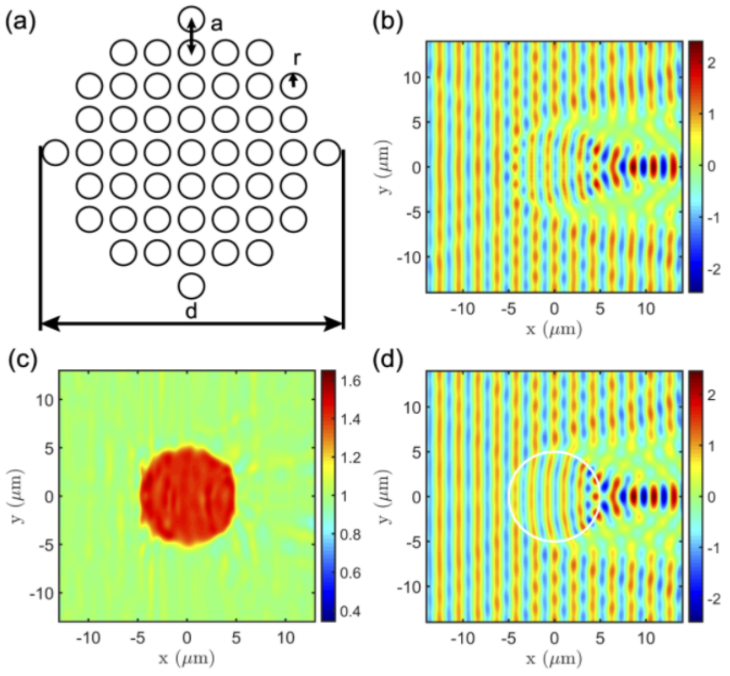

Figure 9: Use of a PINN for retrieval of an effective permittivity profile $\epsilon(x, y)$. a) Arrangement of original meta-atoms whose scattering pattern is desired, with $a = 500nm, r = 125nm, d = 10\mu m, N = 317, \epsilon = 3$. b) Simulated scattering pattern $u(x, y)$ of the meta-atom lattice with incident plane wave wavelength $\lambda_0 = 2.1\mu m$. c) Permittivity profile $\epsilon_r(x, y)$ for a single cylinder, predicted by the PINN. d) Scattering pattern for predicted permittivity profile, with 2.82% error compared to the desired pattern in b)[19].

Chen et al. use DeepXDE[35], a Python deep learning library, to implement their PINN with 4 layers of 64 neurons each. An FEM simulation, sampling both $\mathbf{E}_z$ and ϵ_r at a 150x150 square lattice spatial resolution, is used both to generate the desired scattering pattern as well as verify the result. The hyperbolic tangent activation function is used, and the training rate is set to 10^{-3}. 150000 iterations of training were run, after which a loss value of 10^{-2} was reached. The retrieved permittivity profile $\epsilon_r(x, y)$ produced a scattering pattern with only 2.82% error in the L^2 norm when compared to the desired pattern.

The authors go on to show that their PINN implementation is valid for solving the homogenization problem for other arrangements of meta-atoms, such as a Vogel spiral. By using the framework to recover effective permittivity for coated cylinders and arrangements of multiple cylinders, they show that their method can be extended to deal with multiple materials and objects. Finally, they apply their method to the problem of invisible cloaking design, that is, finding the permittivity of a cloaking material which, when applied to a nanocylinder, cancels its scattering for a given incident plane wave.

4.3 Topology Optimization

The use of classical optimization methods for metasurface design has been criticized due to the high performance cost of running many forward simulations[21][22][19]. This criticism has been leveled in particular at the use of adjoint methods, a class of general methods for numerically computing gradients which can be used for optimization in metamaterial design. This has motivated the application of machine learning techniques to the inverse design problem, as discussed in the previous two sections.

However, papers in recent years have demonstrated the recovery of many metasurface parameters at similar levels of accuracy and efficiency to the deep learning solutions through the use of more traditional optimization methods, enabled through the use of more powerful forward solvers such as RCWA and efficient software implementation with strong low-level optimization in C, parallelism, and AD using TensorFlow[24][23]. These techniques are able to determine unique metalens shapes from an extensive design space consisting of a high number of geometric degrees of freedom, and can be said to belong to the field of **topology optimization**.

4.3.1 Colburn and Majumdar

As discussed in detail in section 3.2.2, despite offering a very high standard of performance for solving the multiple scattering problem for periodic, layered devices, RCWA has issues with respect to differentiability. Specifically, at its core, one must find solutions to the matrix wave equation

$$\frac{\mathrm{d}^2}{\mathrm{d}\tilde{z}^2}\begin{bmatrix}\mathbf{s}_x\\ \mathbf{s}_y\end{bmatrix} - \mathbf{\Omega}^2\begin{bmatrix}\mathbf{s}_x\\ \mathbf{s}_y\end{bmatrix} = 0$$

which amounts to solving the eigenproblem for the matrix $\mathbf{\Omega}^2$. In order for the entire method to be differentiable and for an AD implementation to be possible, access to the derivatives of these eigenvectors and eigenvalues is required. For certain scatterer geometries, $\mathbf{\Omega}^2$ is Hermitian and has no repeated eigenvalues. However, this does not hold for general scatterers, leading to a complex-valued, degenerate eigenproblem wherein the eigenvector gradients are undefined[36].

Without differentiability, standard optimization techniques for nonlinear functions are not applicable to RCWA. However, it is possible to make some small modifications to the method in order to make an AD implementation of RCWA possible.

Colburn and Majumdar[23] formulate and apply these corrections, resulting in a TensorFlow implementation which can be directly optimized for metasurface parameters. Two innovations are combined. First, they consider an eigenequation of the form $\mathbf{\Omega}^2\mathbf{W} = \mathbf{W\Lambda}$, where the columns of $\mathbf{W}$ are the eigenvectors of $\mathbf{\Omega}^2$ and $\mathbf{\Lambda}$ is the diagonal matrix of its eignvalues, and some real scalar function $J = f(\mathbf{W}, \mathbf{\Lambda})$ which depends on the eigendecomposition. Using Wirtinger derivatives, operators from complex analysis which enable a differential calculus for complex functions completely analogous to ordinary calculus for real functions, the authors derive an expression for the sensitivity of J to changes in $\mathbf{\Omega}^2$. This is written as

$$\frac{\mathrm{d}J}{\mathrm{d}\mathbf{\Omega}^2} = \mathbf{W}^{-H}\left(\frac{\mathrm{d}J}{\mathrm{d}\mathbf{\Lambda}^*} + \mathbf{F}^* \circ \left(\mathbf{W}^H \frac{\mathrm{d}J}{\mathrm{d}\mathbf{W}^*}\right)\right)\mathbf{W}^H$$

Here, $\mathbf{F}$ is defined as $\mathbf{F}_{ij} = 1/(\lambda_i - \lambda_j)$ if $i \neq j$ and $\mathbf{F}_{ij} = 0$ otherwise, with $\lambda_0, \lambda_1 ... \lambda_n$ the eigenvalues of $\mathbf{\Omega}^2$. $\circ$ is the Hadamard product defined on two matrices A and B of the same dimensions as $(A \circ B)_{ij} = (A)^*_{ij}(B)_{ij}$.

This effectively removes any differentiability problems caused by the presence of complex eigenvalues. However, if any eigenvalues are repeated, i.e. are degenerate, $\mathbf{F}_{ij}$ remains undefined. to avoid this problem, the authors redefine $\mathbf{F}$ as

$$\mathbf{F}_{ij} = \frac{\lambda_i - \lambda_j}{(\lambda_i - \lambda_j)^2 + \varepsilon}$$

where ε is a small real number. The introduction of ε also introduces some small error in $\mathbf{F}_{ij}$ but ensures that the reverse sensitivity of J is defined. The result is a slightly modified RCWA which can be implemented correctly with AD, using TensorFlow. The authors present such an implementation, and validate its output against another commonly used implementation of RCWA originating from the electrical engineering department of Stanford University, S4 [37], in order to certify that the introduced error is insignificant.

Having this implementation, Colburn and Majumdar now cleverly imagine the reverse scattering problem of optimizing inverse RCWA for some metasurface parameters a_i as equivalent to optimizing weights of a neural network via backpropagation. The scheme is illustrated in Figure 10.

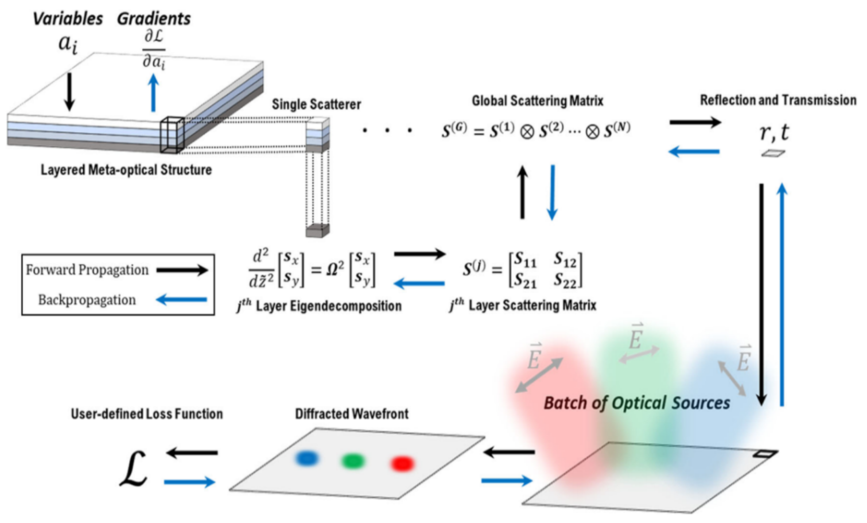
Variables
Gradients
a_i
$\frac{\partial \mathcal{L}}{\partial a_i}$
Single Scatterer
Global Scattering Matrix
$S^{(G)} = S^{(1)} \otimes S^{(2)} \cdots \otimes S^{(N)}$
Reflection and Transmission
r, t
Layered Meta-optical Structure
Forward Propagation
Backpropagation
$\frac{d^2}{d\tilde{z}^2}\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \Omega^2 \begin{bmatrix} s_x \\ s_y \end{bmatrix}$
j^{th} Layer Eigendecomposition
$S^{(j)} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix}$
j^{th} Layer Scattering Matrix
$\vec{E}$
$\vec{E}$
$\vec{E}$
Batch of Optical Sources
User-defined Loss Function
Diffracted Wavefront
$\mathcal{L}$

Figure 10: Colburn and Majumdar's scheme for metamaterial design with RCWA and algorithmic differentiation. The figure follows all notation used in the section on RCWA[23].

The authors treat the metamaterial parameters a_i as trainable weights of a neural network. After initial guesses for the a_i are provided, RCWA is used to calculate the $\mathbf{E}$ field in the reflection and transmission regions, following the black arrows. This result is passed to some scalar loss function $\mathcal{L}$, which defines an optimization criteria - for example, maximizing intensity on a focal plane. Because this has all been implemented using TensorFlow, the gradients $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}a_i}$ are readily calculated using the reverse mode of AD, following the blue arrows. Then, updates can be applied to the a_i in order to produce new best guesses. This process is iterated until the loss is sufficiently small, at which point some optimized a_i have been obtained. All code is available at `https://github.com/scolburn54/rcwa_tf`.

The approach turns out to be both flexible and powerful. This optimization scheme is shown to find effective parameters for a broad range of periodic, layered metasurface structures, including arrays of independently sized nanopost resonators and multilayer gratings. In addition, a number of optimization objectives and loss functions are demonstrated, such as minimizing reflection and focusing red, blue and green light at different points. For a relatively simple case of optimizing reflectivity for an array of nanocylinders on a substrate, the scheme is able to find a design maximizing reflectivity at 99.8% of the incident light.

Additionally, the method is shown to work for aperiodic structures. As an example, a polarization- multiplexed metalens is designed to direct vertically and horizontally polarized incident light to separate focal spots. The applicability of RCWA here to this device and other similar metasurfaces is based on the local phase approximation, meaning that scatterers only slowly vary with position.

Great care is taken by the authors to ensure high performance. In addition to the inherent speed of RCWA, the scheme benefits from the efficiency and parallelizability of backpropagation. Implementation in TensorFlow allows for easy usage of dedicated parallel architecture including GPUs[31], which is demonstrated to increase performance by 14x to 24x for certain problems.

Compared specifically to adjoint methods, a speedup is demonstrated in terms of time per optimization iteration. Iterations of an adjoint method normally cost twice the forward simulation time because they require two simulations, the forward and adjoint simulations, to compute a gradient. The authors demonstrate that their scheme, which performs one forward simulation and a backpropagation update per iteration, results in a 1.4x speedup over the adjoint standard of two forward simulations for multiple problems. This result and the GPU speedup are shown in Figure 14.

4.3.2 Lin et al.

The paper of Lin et al.[24] adds additional support to the efficiency of topology optimization schemes for periodic, layered metasurface design. The authors use RCWA as their primary forward solver method for each unit cell of the periodic device, with FDFD used to perform a full-device simulation after results are obtained for verification. Unlike Colburn and Majumdar, however, Lin et al. do not leverage AD or backpropagation for performance gains, instead using the adjoint state method and relying just on efficient implementations of the forward solvers in C and massive parallelization using the message-passing interface (MPI) library.

This choice turns out to be effective. The authors claim that their straightforward topology optimization approach allows them to handle thousands of

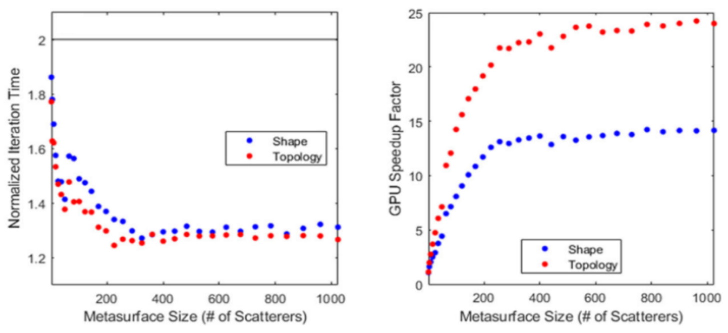

Figure 11: Performance results for the Colburn-Majumdar metaoptics optimization scheme.Both simple optimization of scatterer shape and full topology optimization are considered. Left: optimization iteration time, normalized to the time of a single forward simulation, are compared to the adjoint method (black horizontal line). Right: speedup factor from GPU parallelization vs. problem complexity[23].

degrees of freedom per unit cell, far exceeding the number of parameters optimized by the machine learning approaches. This is demonstrated through the recovery of multi-layer aperiodic topologies for monochromatic and multi-wavelength focusing metalenses, in both 2d and 3d. One example is given in Figure 12.

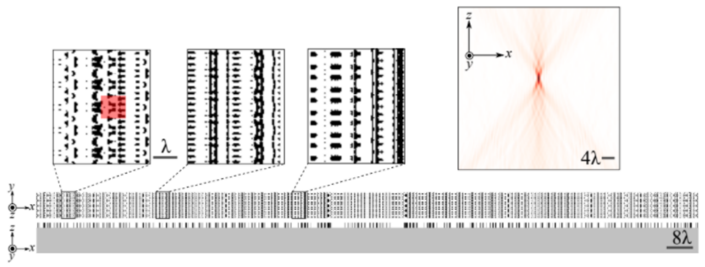

Figure 12: Monochrmatic 3d cylcindrical metalens topography achieved through adjoint method optimization by Lin et al. Red section shows a single λxλ unit cell. Plot on the right shows simulation of the final focusing behavior[24].

This lens topology contains 4x10^4 degrees of freedom in total, and was computed in parallel on 25 CPUs. The final FDTF validation simulation gives a transmission efficiency of 75%.

Optimization of arbitrary aperiodic topologies using RCWA is achieved though approximating the aperiodic structures as a set of periodic scattering problems, one for each unit cell. In other words, to solve for the near field for each unit cell, a separate problem is solved in which the single unit cell is tiled across an entire device. Each of the problems may be solved entirely in parallel. Afterwards, the final near field is approximated via the periodic near field solutions.

The authors conclude by positing the expectation that methods such as

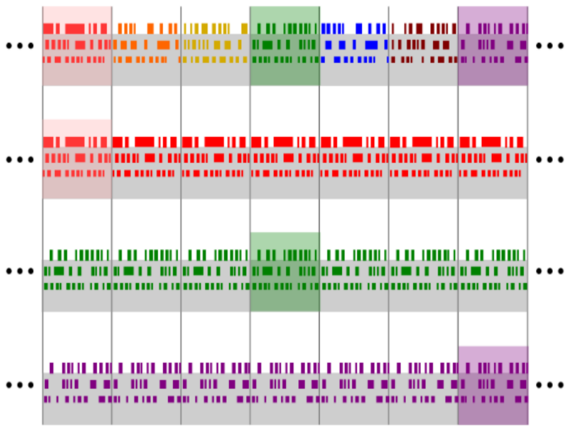

Figure 13: Lin et al. approximate the near field scattering pattern of an arbitrary aperiodic layered metasurface, shown at top, by solving a separate periodic scattering problem for each unit cell. These solutions, shown in the lower three rows, are combined to approximate the near field on the entire surface[24].

theirs will become indispensable for tackling design problems with large design spaces and multiple layers.

4.4 Relationship Between Methods

Based on this literature survey, it is possible to make some reasonable preliminary comparisons between these methods. The timescale of a Master's thesis project simply does not allow for all of them to be implemented and experimentally compared. Thus evaluating their individual advantages and disadvantages is instrumental to arriving at a good solution method for the CoPILOT design problem.

4.4.1 Physical Accuracy

The methods from scientific machine learning do not strictly enforce the physical correctness of their solutions, because they use only general neural network frameworks not specially adapted for an application in optics. Despite this, they can derive physical accuracy from the use physically accurate training data. This relies on an accurate, pre-existing numerical forward solver which can be used to generate such data. Given that the solver is accurate and that the training data covers a representative sample of the design space, then there is a reasonable expectation that the trained neural network will produce physically accurate solutions. This can always be verified via additional numerical simulations of the results.

However, when modelling highly nonlinear and possibly discontinuous functions such as those which describe multiple scattering in metamaterials, the dependency of a neural network's accuracy on training data coverage of the design space can be detrimentally strong. In order to maintain a reasonable model accuracy, either a prohibitively large corpus of training data must be produced or the design space must be made smaller by reducing the number of optimized parameters. Both Jiang[21] and An[20] achieve model accuracy via the latter approach, allowing for only 3 to 6 geometrical metasurface parameters.

Similarly, these methods cannot strictly enforce manufacturing and design constraints. If the training data only contains examples of solutions which are admissible in terms of these constraints, then the network will not have access to any information about the distinction between admissible and non-admissible solutions, and the admissibility of results is not guaranteed. For example, Jiang et al.[21] find that sometimes the network will return negative parameter values which have no physical meaning.

This can potentially be solved via regularization terms in the training loss function which disincentivize non-admissible and non-physical solutions. This strategy carries with it the risk that adding such terms may make the loss function non-convex. If this happens, the training of the network may no longer converge, leading to unstable behavior and a failure of the method to find good solutions. Either way, as long as non-physical and non-admissible results remain technically possible, they must be weeded out via validation.

The PINN approach also relies on such terms in the loss function, but brings several advantages due to the fact that they are codified in terms of the boundary conditions of the relevant physical problem. Because of this, the local convexity and existence of minima of the loss function depends only on the existence of optimal solutions to the physical problem. In the case of metasurface design, this means that the existence of good designs leads to convergence of the PINN[21]. While the existence of good solutions can't be presupposed, without them, the optimization problem would fail for even a perfect method. PINNs, however, still face the same difficulties as scientific machine learning methods with respect to enforcing design and manufacturing constraints - if included in as loss terms, these constraints may cause loss of loss function convexity and create convergence issues for the training algorithm.

The physical accuracy of solutions produced by the topology optimization approach depends only on the accuracy of the numerical forward solver used. As long as the physical predictions of the solver at each optimization step are accurate and good solutions to the design problem exist, then gradient descent is guaranteed to arrive at least at a local minimum. This class of methods still faces the same problem as the previous two concerning problematic regularization terms for forcing admissible solutions, and strategies to tackling this problem are discussed in depth in section 5. However, given that a fast and accurate forward solver exists for the design problem, which is now the case in most applications, this seems like the strongest promise of physical accuracy given by any of the three discussed approaches.

4.4.2 Computational Cost

Prior to the introduction of machine learning based approaches to metamaterial optics design, more traditional optimization strategies were employed. One such idea is the construction of a large "library" of possibly metasurface designs, each of which is modelled with a forward solver to determine a scattering response. Afterwards, some classical search method, such as a straightforward grid search is used to traverse the library in order to locate desirable metasurface shapes. This idea is criticized as being inefficient and time-consuming, both due to the time cost of running many numerical simulations as well as the common over-reliance on trial and error or empirical reasoning in the search method which is not sound when applied to the highly nonlinear problem of multiple scattering in metamaterials[21][20]. These approaches can be used as a baseline to compare against those considered here in terms of performance.

Evaluating a neural network in the forward direction is typically an inexpensive operation compared to running a numerical solver. Thus, once the network is trained, scientific machine learning methods outclass all others in terms of performance. But the true performance cost is derived from the training, and depends on the size of the body of training data. Because a numerical solver must be used to generate the training data, more training data means more use of the solver. Although this is a one-time cost after which the network can be trained multiple times on the same data, the cost of training also scales with the size of the training corpus.

Because the physical accuracy of the model relies on heavy coverage of the design space by the training data, when using these methods there is therefore an unavoidable and potentially unfavorable correlation between physical accuracy and cost. Lacking some sort of advanced heuristic approach to adjusting the density of training data across the design space in advantageous ways, it is likely that increasing the amount of training data will only have diminishing returns on solution quality while continuing to impact performance. At some point, time constraints will impose a cap on the amount of training data which can be used and with it a cap on the accuracy of the model. Equivalently, this restricts the number of parameters which can be optimized for by putting a cap on the size of a design space for which the model can remain reasonably accurate while maintaining a reasonable execution time.

The magic of the PINN framework is that, by penalizing non-physical solutions, it restricts the space of admissible solutions to a manageable size. Compared to normal machine learning methods applied to the same problem, a PINN loss function acts as a regularization agent and amplifies the information content of used data, allowing the PINN to generalize well even when using much less training data. Only one piece of training data is generated and used, the desired scattering pattern u, compared to the prior approaches in which thousands or hundreds of thousands of simulations are required to generate a massive training corpus. In this small-data regime, most other general machine learning techniques fail to provide any sort of convergence guarantees[34].

In fact, because the inverse PINN requires no data on the parameters λ which it predicts, it belongs to the class of methods called **unsupervised learning**, to which reinforcement learning also belongs[19].

These features of PINNs enable them to overcome the training data limitations of the previous methods and allows for the prediction of greater numbers of metasurface parameters. This is clearly seen in the results of Chen et al.[19], in which the permittivity profile ϵ_r is predicted at every point on a 150×150 square spatial lattice across the domain - 22500 parameters, if considered independently.

A downside to the PINN framework is that a single trained PINN does not function as a general solution to the inverse problem for multiple design cases. This means that when changing the design goal by, for example, adjusting the focal length, the PINN network needs to be retrained. In Chen at al.[19] the PINN must be retained to predict metasurface parameters for each desired scattering pattern. By contrast, networks trained according to methods from scientific machine learning are able to predict geometrical parameters for numerous devices of differing goals without needing to be retrained. In An et al.[20], the authors are able to predict metasurface parameters for a great number desired phase profiles with repeated evaluations of the same meta-filter design network, without retraining needed in between each.

If designs for many diverse devices are routinely required and the network is forward evaluated many times, this is highly computationally efficient. This is rightly cited as a performance boon. However, if the number of devices needed to be designed is small, this provides virtually no upside.

How can the topology optimization approach be brought into the performance discussion? Algorithmically, there are some overlapping areas for which strong comparisons can be made between all three discussed methods. In all three, an iterative optimization is performed for some parameters. In the case of the scientific machine learning methods, it is the backpropagation process to find network weights and biases. For a PINN, the geometrical metasurface parameters are included alongside these. For topology optimization, it is the normal gradient descent to find metasurface parameters. As discussed in section 5, this comparison has interesting implications for the interoperability of the methods.

But how can these processes be compared? What is the cost of each iteration, and how many iterations are required? A PINN attempts to optimize the greatest number of parameters - both the network parameters as well as metasurface parameters. Yet, due to the size of the training data required, the training iterations of the scientific machine learning methods are likely to be the most expensive of the three. In addition, due to the efficient use of the boundary conditions to restrict the design space, it is possible but not certain that a PINN will outclass these methods in terms of the number of training iterations required to reach a suitably accurate model.

When it comes to the topology optimization approach, the performance comparison is less straightforward. It could be theorized that the cost of one iteration of gradient descent may be cheaper than one PINN training iteration because the number of parameters being optimized, and therefore the number of gradients needed to be calculated, is smaller in the case of topology optimization. However, the question of how many such iterations is required for convergence has no clear answer without analytical knowledge about the shape of the loss function, the lack of which motivates the entire search for an optimization method. Ultimately, the answers to these questions are likely to be application specific, with the optimally performant solution possibly only being able to be determined via experimental comparison, exploration of efficient implementations, and optimization of model hyperparameters.

4.4.3 Applicability to CoPILOT Design Problem

All three of the discussed method classes are sufficiently general in order to be applicable to the CoPILOT lens design problem, but the advantages and disadvantages of each with regards to physical accuracy and computational performance must be considered. Physical accuracy is of course a primary concern, without which a method is not viable. And because the problem requires the optimization of a large number of geometrical lens parameters, computational efficiency is perhaps even more important - without it, a given method may simply be unable to handle the scale of the problem.

The primary upside of scientific machine learning compared to the others is the speed of model evaluation for many designs. However, for CoPILOT, only two design cases exist, meaning that this advantage is wasted. In consideration of the other problems these methods face with accuracy and performance in comparison to the others, this makes these methods less suited to the design problem at hand.

PINNs present a very flexible and powerful framework better suited to single device designs. However, in terms of implementaion complexity, they carry some problems compared to topology optimization. The core of the PINN is the framing of a loss function in terms of boundary conditions to a PDE which is descriptive of the physical problem. But in the CoPILOT case, these conditions take the form of reflection and transmission coefficients on the material boundaries between device layers, as seen in RCWA. This means that the formulation of the boundary conditions, and therefore the loss function, changes as the metasurface shape is optimized. There is also the question of how to take advantage of RCWA in this context. A PINN seeks to replace a numerical forward solver with a trained network, but this is done primarily because a differentiable version of a conventional forward solver is not presumed to exist. In the CoPILOT case, such solvers actually do exist, in the form of RCWA implementations using AD or the adjoint state method. What is then unclear how this can be optimally used in conjunction with a PINN - perhaps part of the network can be replaced with known operations from RCWA.

Good solutions to these problems will likely only arise after long periods of development and experimentation. For this reason, a PINN approach is less viable for projects under more limited time constraints.

As seen in the next section, the final implementation arrived upon relies mostly on the topology optimization approach, while also adopting and experimenting with ideas from PINNs. Topology optimization has been shown in the literature to be generalizable to many applications in metaoptics, flexible enough to take advantage of diverse techniques including AD, and powerful enough to handle many metasurface parameters, making it the most well suited to the CoPILOT problem.

5 Lens Optimization Implementation

In this section, a detailed description of a working optimization method for producing quality, admissible solutions to the CoPILOT metamaterial lens design problem is given. This method is based primarily on the topology optimization approach described in the previous section, but certain features and algorithmic improvements take inspiration from the PINN and scientific machine learning approaches. The method was implemented over the course of a few months as part of this thesis project and is available for use as an easily configurable python library. All source code and Jupyter notebook examples can be found in the public Github repository `https://github.com/deveringham/metalens_optimization`.

First, a pseudocode overview of the method is given. After this, important choices made in its design are explained and motivated. Next, important tunable model hyperparameters are listed and described. The interactions between these parameters are explored, and the grid search method used to find their optimal values is introduced. Finally appears a brief exploration of implementation ideas which did not make the final cut, either because they did not work for this application or were outperformed by other ideas. This entire section is written keeping in mind the future student or researcher working on a similar problem, in the hope that a detailed description of the work that went into this project may help them with theirs.

5.1 Complete Algorithm Overview

A psuedocode sketch of the complete metasurface lens optimization algorithm is given below.

```
def optimize_device():

    # Get the initial guess.
    H = init_metasurface(initial_height = h_i)

    # Define optimizer.
    opt = optimizer.Adam(H, learning_rate = L)

    # Initialize the sigmoid coefficient.
    S = 1.0

    # Optimize.
    for i in range(N):

        # Do forward pass, calculating loss.
        loss = loss_function(H, S, r_f, w_1, w_2)

        # Calculate gradients using AD.
        loss.backward()

        # Apply gradients to pixel heights.
        opt.step()

        # Increase the sigmoid coefficient.
        S += (S_f / N)

    # Round off to a final, admissible, solution.
    H = torch.round(H)

    # Return the final metasurface representation.
    return H
```

The parameters L, N, S_f, h_i, r_f, w_1, and w_2 are **model hyperparameters** which affect the behavior of the algorithm and the obtained results but not its basic structure or physical accuracy. A description of each is given in section 5.3. The function `init_metasurface` generates the initial guess for the metasurface, and is described there along with h_i. The choice of loss function, denoted here simply as `loss_function`, is described in the section on implementation considerations.

An optimizer object is instantiated in the psuedocode, which represents an interface to an implementation of some standard optimization method. Specifically, the **Adam** algorithm is used, which is a computationally efficient and memory-light stochastic optimization method named after its core innovation of adaptive memory estimation [38]. Adam has a convergence bound comparable to other common methods, can outperform such methods for a wide class of objective functions, and even scales well to high-dimensional problems. These features, especially the low memory usage, make it well-suited for use on the CoPILOT problem.

The pseudocode also relies on an interface to an algorithmic differentiation framework on lines 19 and 22. It is similar to that which would be used in order to implement the code using TensorFlow[31] or PyTorch[39]. In practice, two implementations were produced, each taking advantage of one of these frameworks, and their performance was compared. The results of this comparison are discussed in section 5.2.3.

The original version of the *rcwa_tf* solver, utilizing TensorFlow, is the only part of this algorithm which has not been created as part of this project. It has been heavily modified for use on the CoPILOT problem, and its port to PyTorch, described in section 5.2.3, is new here. In addition, all parts described in the following sections, including the formulation of the metasurface, the differentiable thresholding strategy, the formulation of the loss function, the memory optimization strategies, and the hyperparameter tuning strategies, are original work.

5.2 Implementation Considerations

Consider the topology optimization approach as applied to the CoPILOT lens design problem. A differentiable RCWA solver, implemented with TensorFlow so as to provide access to its gradients via AD, is given by Colburn and Majumdar in the form of their library *rcwa_tf*[23]. The most naive method, then, is using a straightforward gradient descent to optimize the geometrical lens parameters, namely the heights of each pixel.

In such a method, at each optimization step, the current best device is represented as a 2-dimensional array of pixel heights. A forward simulation of the current device produces an image of the electric field intensity on the focal plane. Then this image is fed to a loss function, which outputs a loss value which characterizes the quality of the solution. A backwards pass of AD provides the gradients of this loss with respect to the pixel heights. These gradients can then be applied to current heights to produce a next best guess. This process is iterated until the loss converges. What implementation considerations must be taken into account in order to adapt such a method to the problem at hand?

5.2.1 Forcing Admissible Solutions

The first issue is that this naive method does nothing to ensure that the solutions it produces adhere to the manufacturing constraints. The most important of these is the requirement that the pixel heights take on only one of 6 possible discrete values. Because the shape of the surface is formulated in terms of pixel heights rather than layers, this requirement also enforces of the "no tunnels" rule. Solutions which adhere to this requirement can be called **admissible solutions**.

In the naive approach, heights at each step must be represented as floating point values in order for their free optimization via gradient descent to make sense. By contrast, heights in an admissible solution can be represented as integers. While methods for discrete optimization of integers exist, these are not in general compatible with gradient descent, typical machine learning techniques, or AD frameworks including TensorFlow and PyTorch. A more flexible idea is to instead keep the heights as floats during the optimization process, but at some point to convert or force them towards integers.

The most obvious idea is to conduct unconstrained float optimization and to simply round off those values to the nearest integer after convergence. However, while the converged float solution may produce a good result, there is no guarantee that the nearest integer solution will do the same. In most cases, this final round-off will produce a significant spike in loss at the final iteration, as seen in the example learning curve given in Figure 14.

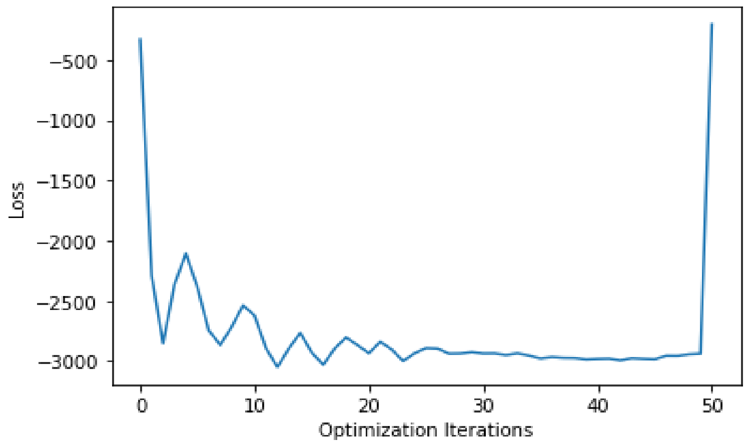

Figure 14: Learning curve for free optimization of metasurface relative permittivity, with admissibility constraints applied once at the end. Note the spike in loss at the last iteration, when these constraints are applied.

The idea of freely optimizing float heights also faces a problem with the use of RCWA. RCWA relies on a description of the device being simulated as consisting of a discrete number of layers with fixed thicknesses. If the height of each pixel can change continuously at each optimization step, then at each step the number device layers and their thicknesses can also change - adding some complication to the process of calculating the gradients. The cost of each iteration is no longer guaranteed to be stable. In addition, allowing each pixel to have a unique height leads to the possibility of needing up to $n^2 + 1$ layers in order to describe the device. Because the computational complexity of a forward solve in RCWA depends linearly on the number of layers, this situation quickly becomes computationally intractable.

To address this issue, the float heights can be mapped to another parameter of the metasurface which can be freely optimized without additional computational cost. Specifically, the relative permittivity ϵ of each pixel at each layer is used. A continuous function `height_to_stacked`, which maps heights to permittivity values at each layer, can be constructed such that when the height takes on an integer value, an admissible solution for that pixel is output. For 3 layers, such a function must fulfil

$$\texttt{height\_to\_stacked}(h) = \begin{cases} [1,1,1] \text{ for } h = 0, \\ [1,1,\epsilon] \text{ for } h = 1, \\ [1,\epsilon,\epsilon] \text{ for } h = 2, \\ [\epsilon,\epsilon,\epsilon] \text{ for } h = 3. \end{cases}$$

Here, 1 is the relative permittivity of free space, and $\epsilon > 1$ is the relative permittivity of the device material. As long as this function is implemented such that it is continuous for all $h \in [0, 3]$, heights can then be freely optimized without concern as to intractable computational complexity of simulating the device.

However, solutions must still be restricted to being admissible. This can be achieved by implementing `height_to_stacked` using **thresholding**, that is, by rounding off or nudging towards admissible ones at each optimization step. A successful thresholding strategy must meet two criteria:

- It must not overly restrict the ability of the optimization algorithm. If, early in the optimization process, the solution is repeatedly thresholded to a solution which has previously been reached, then the process has gotten stuck. This is unfavorable because it likely prevents a better admissible solution from being reached.
- It must be differentiable. In order for AD to successfully return gradients at each step, every operation executed during the forward pass must be differentiable. This includes not just the mathematics of the solver but also the conversion from heights to permittivities and any thresholding.

These requirements immediately rule out any round-off operations, as they are not differentiable. A differentiable alternative is the **sigmoid function**, given by

$$Sig(x, S, a, b) = \frac{a}{1 + e^{-S(x-b)}}$$

where a gives the maximum height of the sigmoid curve, b gives its horizontal translation, and S governs the horizontal scaling of the curve or, equivalently, its slope in the transition region. In the rest of this description, S is simply called the **sigmoid coefficient**. The sigmoid function is plotted in Figure 15 for a few values of S.

Sigmoid functions are commonly used in neural networks as activation functions and to introduce nonlinearity into a model. In this application, a sigmoid function can be used to differentiably "nudge" a solution towards an admissible one, with the magnitude of the solution restriction being related to the magnitude of S.

Several properties of the sigmoid function make it suitable for this role. Firstly, the sigmoid function maintains the side of the transition region on which each of its inputs occur. By that, it is meant that

$$\text{for } x_{out} = Sig(x_{in}, S, a, b),$$

$$x_{out} = \begin{cases} > \frac{a}{2}, \text{ if } x_{in} > b, \text{ and} \\ < \frac{a}{2}, \text{ if } x_{in} < b. \end{cases} .$$

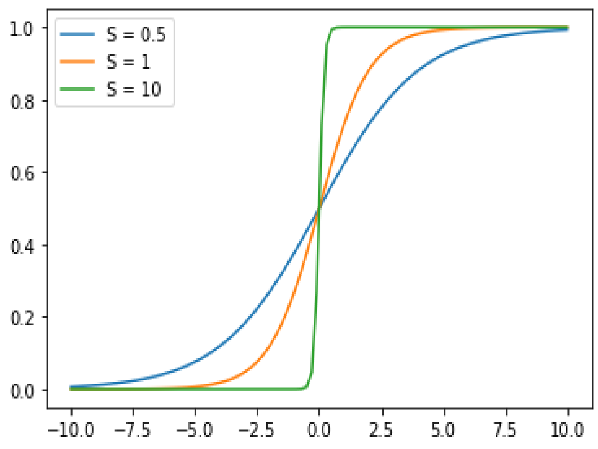

Figure 15: Sigmoid function plotted for $a = 1$, $b = 0$, and several values of S.

That is, for points x_{in} which lie to the left of the transition at $x = b$, the output x_{out} is less than half of the sigmoid curve height a, and the opposite holds true for points on the positive side of the transition. This property is useful because it means that for an input float height and an offset b which is halfway between two admissible integer values, the output height returned by the sigmoid is always closest to the same admissible value which the input already was - in other words, the closest admissible value for some input is invariant under the sigmoid operation.

Another favorable property of the sigmoid function is that this thresholding behavior becomes more pronounced, continuously, as S increases. As seen in the figure above, $Sig(x, 10, 1, 0) = 0$ if $x < 0$ and $= 1$ if $x > 0$, unless x is very close to 0. When S is large, almost all input values are forced all the way to an admissible value, in this case, 0 or 1. If the output is then rounded off to a final admissible solution, the difference will be very small.

A very good solution strategy for the CoPILOT design problem, then, is to apply differential thresholding operation in the form of a sigmoid function at each optimization step, while increasing S after each step. This allows for free optimization of the heights early in the optimization process, and gradual forcing towards an admissible solution as the process progresses. This results in a learning curve like shown in Figure 16, where the quality of the solution, as characterized by a small loss value, increases early on, then eventually converges to a lower quality but admissible solution as the sigmoid coefficient is annealed.

To realize this idea in practice, a differentiable form of the function `height_to_stacked` is implemented using several sigmoid functions. For 3 layers, it takes the form

$$\texttt{diff\_height\_to\_stacked}(h, S) =$$

$$[\, 1 + Sig(h, S, \epsilon - 1, \frac{1}{2}),\ 1 + Sig(h, S, \epsilon - 1, \frac{3}{2}),\ 1 + Sig(h, S, \epsilon - 1, \frac{5}{2})\,]$$

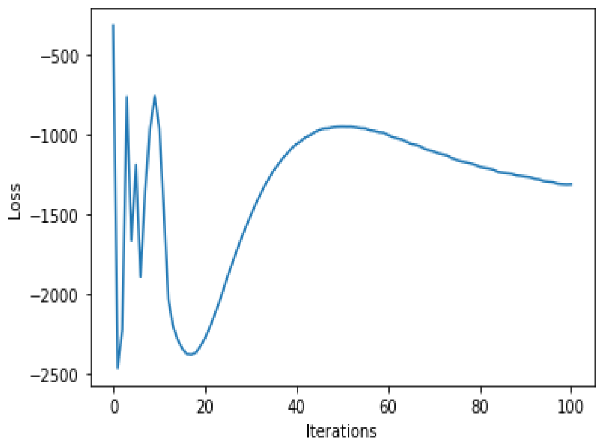

Figure 16: Learning curve for optimization with differential thresholding and sigmoid coefficient annealing. Note the smoother final convergence behavior vs. Fig. 15, but with the tradeoff of loss increase in the middle of the curve due to steadily increasing the admissibility forcing.

$$= \begin{cases} [1, 1, 1] \text{ for } h = 0, \\ [1, 1, \epsilon] \text{ for } h = 1, \\ [1, \epsilon, \epsilon] \text{ for } h = 2, \\ [\epsilon, \epsilon, \epsilon] \text{ for } h = 3. \end{cases}$$

The choice of b in each term horizontally translates the sigmoid such that the transition region is halfway between two admissible values. This function can be simply extended with additional terms for devices with more layers.

5.2.2 Choice of Loss Function

Of course, in order to perform an optimization iteration, a loss function must be defined. This function should take a height representation of a CoPILOT metasurface and return a float value representing the quality of that solution. A higher quality solution should correspond to a smaller return value, and in order to find an optimal solution, the loss should be minimized. At each optimization iteration, the gradients of this loss value with respect to the pixel heights are calculated during the backwards step of AD, and these gradients can be applied to update the heights. The choice of a loss function therefore affects the optimization algorithm greatly is and is the second important implementation consideration.

CoPILOT's near field and far field design cases, while requiring different focal distances, share the goal of maximizing the focusing of light at the center of the focal plane. Assume access to a differentiable function $solve(\mathbf{M})$, which uses RCWA to solve the scattering for a metasurface described by the given 3D tensor of relative permittivities $\mathbf{M}$ and returns a matrix representing electric field intensity at some physical sampling rate on the focal plane. Such functionality is provided by the $rcwa\_tf$ solver[23]. Then, a loss function which encodes the proper optimization goal simply converts a given matrix of pixel heights $\mathbf{H}$ to a permittivity representation $\mathbf{M}$ using `diff_height_to_stacked`, passes the result to *solve*, and returns the negative sum of all focal plane intensity values within some small distance r of the center.

```
1   def loss_function(H, S, r):
2
3       # Convert height representation of metasurface
4       # to stacked representation suitable for solver,
5       # using differentiable thresholding.
6       M = diff_height_to_stacked(H, S)
7
8       # Solve for scattering from metasurface.
9       focal_plane = solve(M)
10
11      # Sum electric field intensities around
12      # center of focal plane.
13      i = focal_plane.shape(0) / 2
14      return -1 * sum(focal_plane[i-r:i+r, i-r:i+r])
```

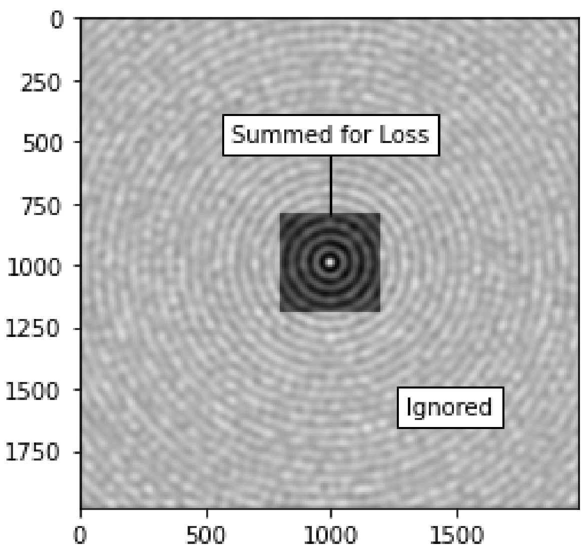

Figure 17: Diagram of loss function for $r_f = 200$. The units are in terms of the physical sampling rate used by the solver. An example scattering pattern from an optimized lens, shown on the $5mm \times 5mm$ region of the focal plane immediately behind the device, is used as the background. A large r_f is chosen for visual clarity. In practice, a much smaller value around 10 is chosen.

This loss function incentivizes intensity at the focal spot. The matrix slicing on line 14 selects for only the square region of width and height r in the center of the focal plane, as seen in Figure 17.

A slightly more sophisticated loss function also disincentivizes intensity elsewhere:

```
1   def loss_function(H, S, r, w_1, w_2):
2
3       # Convert height representation of metasurface
```

```
4       # to stacked representation suitable for solver,
5       # using differentiable thresholding.
6       M = diff_height_to_stacked(H, S)
7
8       # Solve for scattering from metasurface.
9       focal_plane = solve(M)
10
11      # Sum electric field intensities around
12      # center of focal plane.
13      i = focal_plane.shape(0) / 2
14      loss_1 = -1 * sum(focal_plane[i-r:i+r, i-r:i+r])
15
16      # Sum electric field intensities everywhere
17      # except around center of focal plane.
18      loss_2 = w_2 * (sum(focal_plane[:,0:i-r])
19                     + sum(focal_plane[:, i+r:-1])
20                     + sum(focal_plane[0:i-r,i-r:i+r])
21                     + sum(focal_plane[i+r:-1,i-r:i+r]))
22
23      # Final loss is weighted sum of these
24      # two terms.
25      return w_1 * loss_1 + w_2 * loss_2
```

The slicing on lines 18-21 selects the region of the focal plane everywhere except the square used in the first term, like shown in Figure 18.

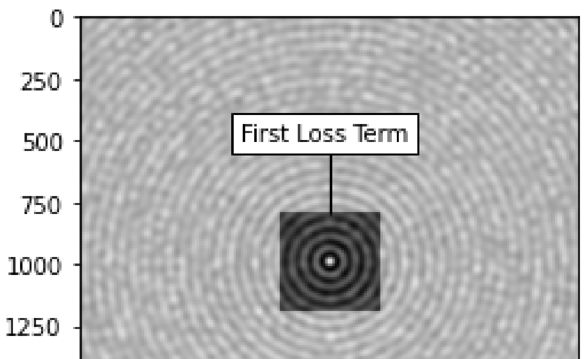

Figure 18: Diagram of improved loss function for $r_f = 200$.

The focal spot radius r and the loss term weights w_1, w_2 are now method hyperparameters. When changed, they affect the optimization algorithm, but not the physical accuracy of the method, and may lead to changes in convergence and solution quality. There are a number of other such hyperparameters, which are discussed in section 5.3.

5.2.3 Meeting Memory Constraints

A final important consideration has to do with memory performance of the method. The machine available for this project, a server owned by TU Delft's Optics group, possesses an NVIDIA A6000 GPU with 48GB of GPU memory. Execution of the optimization code on the GPU carries large performance benefits in terms of execution time, as seen in Figure 19. In order to optimize the CoPILOT devices in a reasonable amount of time, then, it is necessary that the optimization code is able to operate under this 48GB GPU memory constraint.

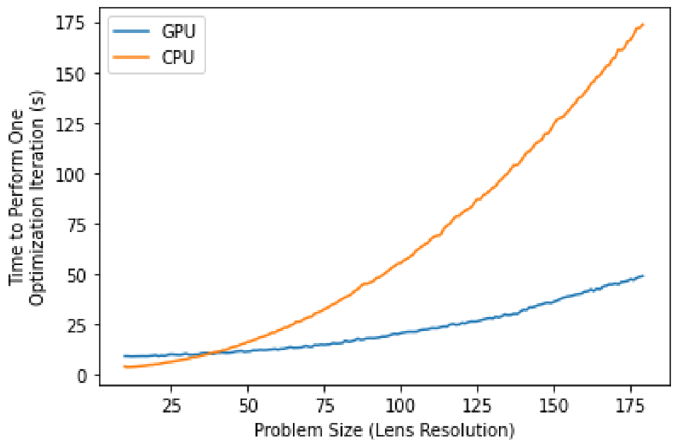

Figure 19: Performance comparison of the CoPILOT metalens optimization algorithm running on GPU vs CPU. For small problems, the overhead of initializing the GPU dominates, but for larger problems, the speedup due to GPU parallelization makes it by far the best option.

The most memory intensive part of the optimization algorithm is the computation of the gradients of the pixel heights with respect to the loss during the backwards mode of AD. The memory consumed during this computation depends on some parameters of the device being optimized, including the number of pixels in each transverse dimension and the number of layers. It also depends on some parameters of the RCWA solver, including the number of spatial harmonics used in the Fourier decomposition and the physical sampling rate desired of the focal plane scattering pattern. Therefore, the ability of the algorithm to optimize devices with many pixels and layers while maintaining physical accuracy of the simulations at each step depends strongly on the efficiency of its memory utilization.

The device's pixel count is particularly relevant. An optical device only functions as an optical metamaterial if it includes features smaller than the wavelength of light with which it interacts, and the effect is most pronounced for materials with features less than $\frac{1}{3}$ of the wavelength. Because the CoPILOT lenslets have a fixed size of $5mm/times5mm$, if the metasurface etching is made to cover the entire surface of each lenslet, then the size of each pixel in each transverse dimension is the device size in that dimension divided by the number of pixels in that dimension. Therefore, whether a CoPILOT lenslet displays metamaterial behavior is dependent on the number of pixels, or the **resolution**, of that device.

Importantly, a large increase in the quality of solutions is seen when passing this limit, as shown in Figure 20. Therefore, it is imperative for the discovery of effective lens designs that the algorithm is memory efficient enough to simulate and optimize large resolution devices.

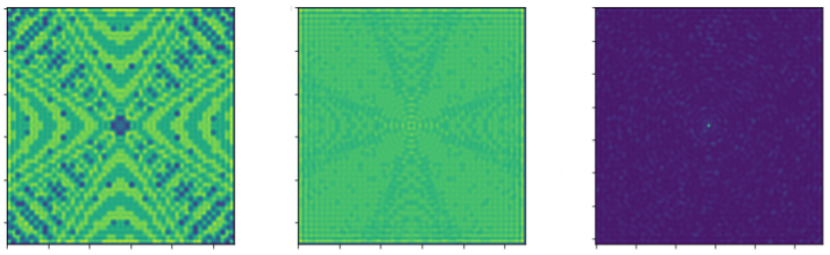

Figure 20: Comparison of converged solution quality for CoPILOT devices of different resolutions. The nearfield design case is considered, with hperparameters $N = 50, S_f = 10.0, L = 0.8$, and $h_i = 0$. These plots show the electric field intensity produced on the focal plane by an optimized design. The first is produced by an optimized device with 50×50 pixels, and at this resolution the scattering pattern typically reflects the blocky metasurface shape. The second come from a device of resolution 100×100 pixels, and this behaviour is somewhat reduced. The final device, at 150×150 pixels, surpasses the $\lambda/3$ feature size limit at which metamaterial behavior emerges, and so is able to produce a much more favorable scattering pattern.

For the nearfield design case, the source wavelength is $120\mu m$, which gives a minimum desired lenslet resolution of 125×125 pixels. The farfield case is slightly more lenient with a source wavelength of $158\mu m$ and minimum desired resolution of 95×95. In each case, the maximum resolution is 250×250, limited by manufacturing constraints on the minimum etchable feature size.

The first complete implementation of the optimization algorithm was unable to meet these requirements, being only able to simulate a 75×75 device with 7×7 spatial harmonics before using up all available GPU memory. Therefore, some clever approaches to reducing memory usage by AD were required.

Firstly, the memory efficiency of the algorithm was compared when using different AD frameworks. The $rcwa\_tf$ solver, originally written to take advantage of the TensorFlow for AD, was rewritten using the alternative framework PyTorch[39]. It was found that, even without any high-level algorithmic changes, PyTorch was more memory efficient on the GPU for this problem.

It is not surprising that TensorFlow and PyTorch perform differently. Their efficiencies, both in terms of execution time and in terms of memory utilization, are independent due to differences in the underlying AD implementations and mathematical implementations of various operations. Thus, which framework performs better is expected to be highly application-dependent.

For both frameworks, the GPU memory utilization of the code can be monitored using the NVIDIA System Management Interface utility ($nvidia - smi$). By breaking the solver into many small steps and measuring memory usage after each during the AD backwards step, the most memory-heavy operations can be identified. For the *rcwa_tf* TensorFlow implementation of RCWA, one operation alone used nearly 50 percent of all memory used by the program. This was a **matrix exponential** operation, defined as

$$e^{\mathbf{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{A}^k$$

The matrix exponential plays a key role in the solution of differential equations, and thus has been used in many wide-ranging applications and been the subject of a large amount of literature. In RCWA, it is necessary for constructing the scattering matrix after solving the matrix wave equation for the complex spatial harmonic amplitudes within the device. For this calculation, TensorFlow uses a form of the scaling and squaring method using Padé approximations, which derives from the 2005 SIAM article of Higham[40]. PyTorch's implementation is instead based on a version of the scaling and squaring method which uses a Taylor expansion of the exponential rather than Padé approximants, and originates from the 2019 paper of Bader et. al.[41]. Bader claims that this improvement reduces the number of required matrix calculations while maintaining comparable error estimates.

Because fewer matrix calculations in the forward solve corresponds to fewer results needed to be stored in order to calculate gradients, using the Taylor expansion method for the matrix exponential leads to improved memory utilization when calculating gradients. This one change was found to be responsible for the majority of the difference in memory consumption between the TensorFlow and PyTorch implementations of the metalens optimization algorithm. The remainder of the difference is likely due to differences in the implementations of other heavy operations required by RCWA, such as the eigenvalue decomposition and Fourier transforms.

Next, a technique called **gradient recompute checkpointing** is used to optimize AD for reduced memory consumption. Recompute checkpointing is commonly used to reduce memory usage during the training of deep neural network in order to enable increased model sizes. However, because recompute checkpointing takes advantage of the computational graph underlying AD used for DNN training, it is also applicable to any algorithm which relies on AD, including the metalens topology optimization algorithm. Both TensorFlow and PyTorch provide implementations of recompute checkpointing.

For an algorithm consisting of n sequential, differentiable operations, the computational graph used by AD for computing the gradients of algorithm's outputs with respect to its inputs is depicted in Figure 21.

Here, A_0 is the algorithm's input, B_n is its output, and B_0 is its computed gradients $\frac{\mathrm{d}B_n}{\mathrm{d}A_0}$. In order to calculate B_n during the forward pass, all forward operation nodes f must be evaluated. Then to calculate the backwards nodes b and the final gradients B_0, the outputs of the f nodes are required. Therefore, they must be stored during the forward pass, and can only be released when the backwards step has progressed far enough such they are no longer needed. This storing of intermediate results is the source of memory usage during AD.

For a simple feed-forward DNN, the f nodes correspond to the network layers. If each layer is the same size, then memory consumption grows linearly with the number of layers n, or equivalently can be said to be of order $\mathcal{O}(n)$. The number of calculations performed and the execution time is also of order

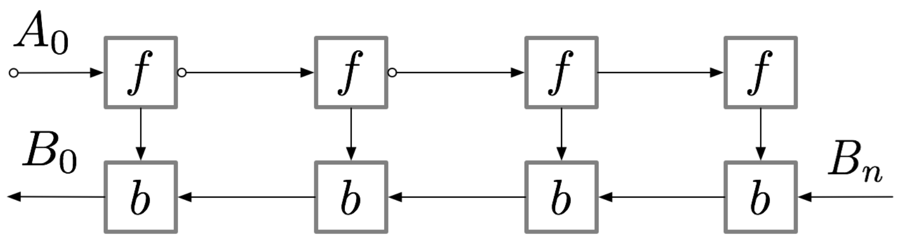

Figure 21: Diagram of computational graph for computing gradients of an arbitrary function. The function represented here is composed of 4 differentiable steps or operations. Each is represented by a forward node f and a backwards node b.

$\mathcal{O}(n)$. For the RCWA solver, each f represents an intermediate result reached after some calculation. Each result may be of a different size and require more memory to store, and therefore are not all of equal weight. This was clearly seen with the particular heaviness of the matrix exponential step.

By choosing to recompute intermediate results instead of storing them, tradeoffs can be made between computational efficiency and memory usage. Consider first the **memory-poor backpropagation** strategy, in which all nodes are recomputed during the gradient calculation. Using this method, memory consumption is capped at a constant $\mathcal{O}(1)$, determined by the single heaviest operation. However, the computational efficiency increases to $\mathcal{O}(n^2)$, which is intractable for most applications, including lens optimization.

A suitable middle ground between the normal backpropagation strategy and the memory-poor version exists the the form of checkpointing. Here, specific nodes are selected as **checkpoints**, and only the intermediate results of these f nodes are stored during the forwards pass. As the backwards pass progresses, nodes are recomputed and results stored until a checkpointed f node is reached, at which point all nodes so far recomputed can be cleared from memory. Each node is recomputed at most once, and the maximum memory usage is limited by the distance between checkpoints.

For the feed-forward DNN with uniformly sized layers, it is optimal to checkpoint every $\sqrt{n}^{th}$ node. Then, the number of nodes between each checkpoint, and therefore also the memory usage, is on order $\mathcal{O}(\sqrt{n})$. This is a marked improvement compared to the original $\mathcal{O}(n)$. In this simple case, the optimal checkpoints can be chosen automatically. In general, the checkpoints must be selected manually, and optimal checkpointing depends on the relative weights of each computational step. If the total amount of work done in one run of the program is m, then optimally the checkpoints should divide the program into subroutines which each perform work on the order of $\mathcal{O}(\sqrt{m})$.

For the lens optimization algorithm, finding good checkpoints is a matter of trial and error. Several ideas were tried, including checkpointing alternating steps in the RCWA algorithm and subdividing heavy calculations like the matrix exponential. The total achieved memory usage reduction of the PyTorch version of the algorithm with gradient recompute checkpointing was significant enough to allow for optimization of sufficiently high-resolution devices in both the nearfield and farfield cases, as shown in Figure 22.

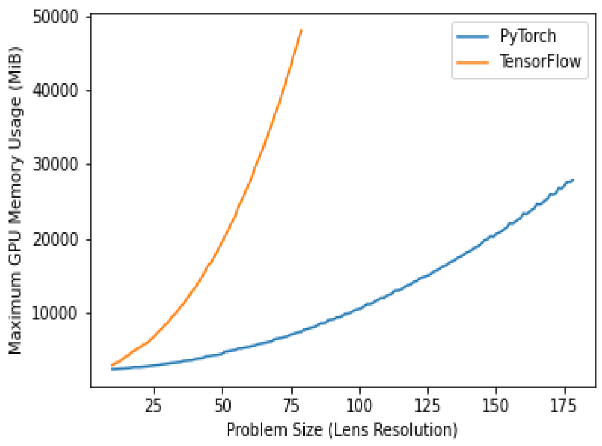

Figure 22: Final memory usage comparison of the AD frameworks, for the CoPILOT metalens optimization algorithm. Gradient recompute checkpointing was used for both frameworks. For devices with a resolution higher than about 75, the TensorFlow representation runs out of memory on the NVIDIA A6000 GPU.

5.3 Tuneable Hyperparameters

The output of the complete lens optimization algorithm also depends on a few **model hyperparameters** which affect the optimization process but not the physical validity of the results. Because the loss function used in this optimization model is highly nonlinear, there are many local minima, which correspond to different solutions of varying quality. The choice of hyperparameters may determine which of these solutions is reached, and therefore finding the best solutions requires careful tuning of their values. The most important model hyperparameters and their interactions are described below.

- **Learning Rate** L
  The learning rate is a parameter of the Adam algorithm for stochastic gradient descent which is at the core of the lens optimization algorithm. It is a coefficient multiplied with the calculated gradients which determines the size of the step which is taken at each step.
  A larger learning rate means that pixel heights are adjusted more at each step and a greater distance in the design space is covered. As a result, it opens up the possibility of finding minima which are global over a larger area of the design space. This can mean, however, that good local minima are skipped over. In addition, if the learning rate is very great, it can lead to chaotic behavior and failure to converge.
  A small learning rate means that pixel heights are only adjusted slightly at each step. This means that the algorithm will certainly not skip over any local minima it encounters. However, these local minima may actually be poor choices compared to some others which might be discovered with a larger learning rate. Also, a very small learning rate means the algorithm may take many steps to converge. The best learning rate is a middle value which balances these tradeoffs. Ideally, The learning rate is large enough

to allow the algorithm to skip over undesirable local minima, while being small enough to avoid chaotic, nonconvergent behavior.

- **Maximum Iterations** N
  N is the maximum number of optimization steps that the algorithm takes before rounding off the pixel heights to admissible values and returning the final solution. The effect of N on the method's behavior is closely related to both the learning rate and the sigmoid coefficient.
  If N is very great, the algorithm takes more steps and explores more of the design space. This may allow for convergence even with a smaller learning rate. It also allows for forcing solutions to be admissible at a greater rate while still maintaining freedom of optimization early in the algorithm, as discussed in relation to S below. In general, increasing N should produce higher quality solutions.
  However, greater N means more iterations and therefore a greater computational cost and execution time. An ideal N is large enough to allow for exploration of a sufficiently large area of the design space and sufficiently smooth forcing of admissibility while not making the execution time of the algorithm unacceptably long.

- **Sigmoid Coefficient** S
  The sigmoid coefficient, as described in section 5.2.1, is a parameter which describes how much a solution is forced to adhere to manufacturing and design constraints at each optimization step. The central idea of the CoPILOT lens optimization approach is to start with a small S and gradually increase it as optimization proceeds. This lets the method first freely discovery good, non-admissible solutions, and then converge to the admissible solutions which is closest in the design space.
  A final value of the sigmoid coefficient, S_f, is chosen, and then the coefficient is updated at each step using the formula

  $$S \mathrel{+}= S_f/N$$

  The initial values of S is set to 0.1. Then, S_f essentially represents the amount that solutions are forced towards admissibility at the final optimization step, before all pixel heights are actually rounded off to the nearest admissible integer values.
  If S_f is very large, then this rounding-off operation is likely to not produce much change in the solution, which is good for the stability and convergence of the algorithm. On the other hand, a large S_f means that the sigmoid coefficient is increased by a greater amount at each step and solutions are more rapidly forced towards admissible ones. This interferes more in the free optimization part of the algorithm, and may prevent discovery of better solutions.
  If S_f is small, then the forcing of admissibility is slow and smooth, meaning that the free optimization is less constrained and perhaps more likely to discovery better solutions. But this also means that the final rounding-off operation may induce greater error, leading to unpredictable and inconsistent results.
  Ideally, some intermediate S_f is chosen which allows good solutions to be discovered while minimizing error in the final rounding operation. The

best value is likely related to the maximum number of iterations N. A greater N allows greater S_f, because the rate of admissibility forcing scales inversely with N.

- **Initial Height h_i**
  The results of the optimization are also sensitive to the initial guess used for the pixel heights. The simplest guess is to start all pixels at one uniform height, which can be expressed as an integer h_i.
  If $h_i = 0$, this corresponds to the initial device having 0 pixel height everywhere - in other words, it begins as a flat surface and pixels of material are "stacked" on top as optimization proceeds. If instead $h_i = (\#Layers - 1)$, then the initial device is flat at the maximum pixel height, and space is instead "etched" into that surface as optimization progresses.
  Another option is random initialization. Here, the initial pixel heights are chosen independently and randomly. With this option enabled, the optimization algorithm can be run many times, with the same hyperparameters enabled, and different results will be reached.
  Without having detailed knowledge of the shape of the loss function, it is not possible to predict which initial guess will enable the algorithm to reach the best possible solution. It is possible, also, that the best initial guess is different even for different selections of the other parameters L, N and S. Therefore, this is a parameter for which several values should be tried for each choice of other hyperparameters.

- **Focal Spot Radius r_f**
  The focal spot radius determines the size of the region in the center of the focal plane in which greater electric field intensity is incentivized by the loss function. The ideal solution has a minimal focal spot radius, with all light focused precisely at the center of the focal plane. However, the smaller that r_f is, the more sensitive the loss is to small changes in the metasurface. This means that for very small r_f, the optimization process may be more chaotic and have issues with convergence, unless the learning rate is very small or N is very large. In addition, it is possible that an "ideal" solution, in the sense of focusing maximal light within a minimal radius, does not exist. In the worst case, the loss function written with r_f may not even be convex. In other words, using a minimal r_f may make the algorithm more unstable for no upside.
  Therefore, a non-minimal r_f should be chosen in order to trade the possibility of finding perfect solutions in favor of algorithmic stability.

- **Loss Weights w_1 and w_2**
  As discussed in section 5.2.2, when multiple terms are used, they should be combined in a weighted sum. The weights used for each term are then hyperparameters which affect the shape of the loss function. Here, w_1 corresponds to the first loss term, which incentivizes light focusing at the center of the focal plane, and w_2 corresponds to the second, which disincentivizes light scattering elsewhere. If $w_1 = 0$ or $w_2 = 0$, then the corresponding loss term is simply left out.
  In theory, including the second loss term disincentivizes solutions which focus a lot of light at the center of the focal plane, but also have significant secondary focal spots elsewhere. For CoPILOT, such solutions are not

desireable because the secondary focal spots may result in light scattering into other instruments and impacting their performance in unpredictable ways. However, as with r_f, the choice of values for w_1 and w_2 affect the shape of the loss function, and so some choices may lead to algorithmic instability and issues with convergence.
Early testing showed no significant improvement in solution quality when including the second loss term. It is possible that it is simply a physics-based feature of the loss function that solutions which centrally focus light do not also coherently focus light in secondary focal spots. In other words, the type of solutions which the second loss term seeks to disincentivize may not actually be physically possible anyway. In order to avoid any potential complications from including the second term, w_2 was then set to 0 and w_1 to 1 for all future runs.

5.3.1 Hyperparameter Grid Search

The simplest and most common method for tuning hyperparameter values in machine learning applications is a grid search. This involves defining a search space of possible hyperparameter values, and selecting a "grid" of values within the search space which will be tried. Then each position on the grid is one possible selection of hyperparameter values. Then the model is trained and evaluated using the values from each grid point, and the results are compared using some evaluation metric in order to determine the most effective hyperparameter choices.
A script for the grid search was implemented which runs the CoPILOT lens optimization algorithm many times for different hyperparameter searches. The results for each run are written to individual log files, which can later be read in order to compare the quality of solutions. Each run is defined by a choice of L, N, S_f and h_i, leading to a 4-dimensional grid.

Increasing the resolution of the grid, that is, increasing the number of hyperparameter values to try, is likely to increase the quality of the best solution found. However, it also increases the cost of the entire grid search. In addition, choice of N in particular impacts the cost of each individual optimization run, meaning that choice of N on the grid must be capped at some reasonable value. In the end, a suitable grid was selected based on early tests, and the grid search was run for several days for each of the CoPILOT design cases. The specific grid used and the results of the search are detailed in the results section.

5.4 Evaluation and Comparison of Solutions

Analysis of the results of the hyperparameter grid search relies on a way to compare the solutions returned by each optimization run. Therefore some evaluation metric is required in order to quantify the quality of solutions.

In typical machine learning and optimization applications, evaluation metrics are distinct and different from the loss function used during training. This is usually because the desired evaluation metric is not able to be easily used as a loss function for some reason. It may be non-differentiable, discontinuous, non-convex, too complicated to be implemented in the context of the training loop. For example, in an image classification problem, cross-entropy loss may be used for training but model accuracy used for an evaluation metric.

The CoPILOT problem, however, specifies a design goal which can be simply and differentiably implemented as a loss function. This is the loss function as already defined: the sum of electric field intensities around the center of the focal plane. Therefore, this function can be used both as an evaluation metric and the loss function minimized during optimization.

Therefore, to compare solutions generated during the hyperparameter grid search, the final, admissible solution can simply be passed once more to the loss function and the computed loss used as an evaluation score. Because loss is minimized, during optimization, a smaller evaluation score corresponds to a higher quality solution.

A secondary, optional, evaluation metric can be formulated in terms of the diffraction limit of the produced devices. The **diffraction limit** of an optical system is a principal, theoretical limit on its angular resolution based on the source wavelength and the device's numerical aperture. It is formulated as

$$d = \frac{\lambda}{2 \cdot NA} = \frac{\lambda}{2n \cdot \sin(\theta)}$$

Here λ is the source wavelength, n is the index of refraction of the device material, and θ is half angle subtended by the optical objective lens. That is, if the total transverse width of the device is l and its total longitudinal thickness is t, then $\theta = \tan^{-1}(l/2t)$. The quantity $2n \cdot \sin(\theta)$ is called the device's **numerical aperture** or NA.

For an imaging device, d gives the minimum resolvable distance of observable features. For a scattering device such as a CoPILOT lens, this corresponds to the minimum half-width of the central gaussian in the focal plane transverse intensity profile. For any particular CoPILOT device with some such half-width w, it must hold that $d < w$. Furthermore, the closer that w is to d, the closer that the device is to functioning as a theoretically ideal "diffraction-limited" device. Thus, it is suitable to use the ratio $|\frac{w}{d}|$ as another evaluation metric, where a larger score indicates a better solution.

In section 6, this diffraction limit metric is applied to compare some of the higher-scoring solutions with regards to the loss function metric.

5.5 Ideas which Did Not Work

In the process of producing the working lens optimization algorithm implementation, many ideas were tried which were ultimately left on the cutting room floor. Some were not practical to implement, others led to poor results, and others still seemed conceptually interesting while not panning out in practice. In this section, a few of these ideas are described in order to give some insight into the implementation process for future students.

- **Optimization Directly via Relative Permittivity**
  Before fully developing the idea of converting between representations of the metasurface using the function `diff_height_to_stacked`, the idea was to directly optimize the relative permittivity of the device at each pixel rather than the height. This is because *rcwa_tf* required as input what was called by convention a **stacked representation** of the metasurface, a 3D tensor which specifies the relative permittivity of the device at each pixel and layer. This is as opposed to the **height representation**, a 2D tensor

which specifies the metasurface height at each pixel. Because initially it was not clear how to differentiably convert between these two representations, an implementation was produced which directly optimized a stacked representation.
This approach did produce solutions, but had several problems. Firstly, the number of metasurface parameters was vastly increased versus the height optimization approach, because there is one parameter per pixel and per layer rather than only one per pixel. This leads to decreased performance, both in terms of execution time and memory usage.
Secondly, it leads to issues enforcing the CoPILOT manufacturing constraints. A height representation is defined in such a way that the "no tunnels" constraint is disallowed, but the stacked representation allows it. That is, a metasurface specified by a stacked representation may have areas of no material below areas filled with material, something which is not allowed. Therefore this constraint must be enforced by disincentivizing tunnels in the loss function. It was found that this, too, was difficult to implement in a differentiable and computationally efficient way. When such an implementation was produced, it was then found that this made the loss function non-convex, leading to poor or no convergence during optimization.

- **Building Device Layer-By-Layer**
  This problem did not exist, however, as long as the device to be optimized had only one layer. Therefore, an alternative to disincentivizing tunnels via a loss function term was to instead build the device one layer a time through multiple, single-layer optimizations. In the first cut of the algorithm, a single pass was performed, building the layer furthest from the source first and then progressing upwards. At each layer, only pixels which contained material on the previous layer were allowed to be filled. This did work, but led to designs for which upper layers were almost entirely empty. It seemed that the single-pass approach was too drastically reducing the design space of possible metasurfaces.
  Next, a "cocktail shaker" approach was tried, in which this single pass of layer optimizations was repeated several times. After the first pass through the layers, a layer is chosen at random and optimized again, keeping constant the structure of all other layers. Then, all layers above it are optimized once again, in order, applying the "no tunnels" constraint. This is repeated some number N number of times, with a final pass of all layers starting at the bottom. This did succeed in sometimes producing good, densely-featured designs, but the algorithm was on the whole unstable. The quality of solutions depended a lot on the randomness of the layer selections and not very much on N, instead of smoothly converging as N increases. This was found to be undesirably unpredictable, and motivated the development of the final, working algorithm.

6 Lens Optimization Results

In this section, good solutions to the CoPILOT lens design problem, found via the optimization algorithm and hyperparameter grid search described in the previous section, are presented.

6.1 Near Field Case

The best result found for the far field design case is detailed in Figures 23, 24, and 25.

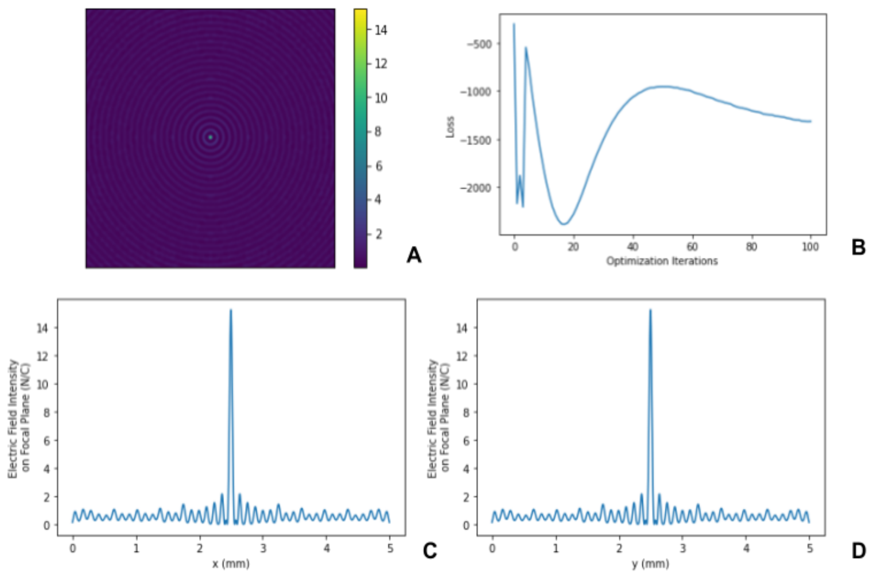

Figure 23: Best result found by the hyperparameter search, for the near field design case. The final scattering pattern on the focal plane, in terms of electric field intensity, is given (A) as well as cross sections of the scattering pattern in the transverse directions (C,D). Also given is the learning curve for this optimization (B).

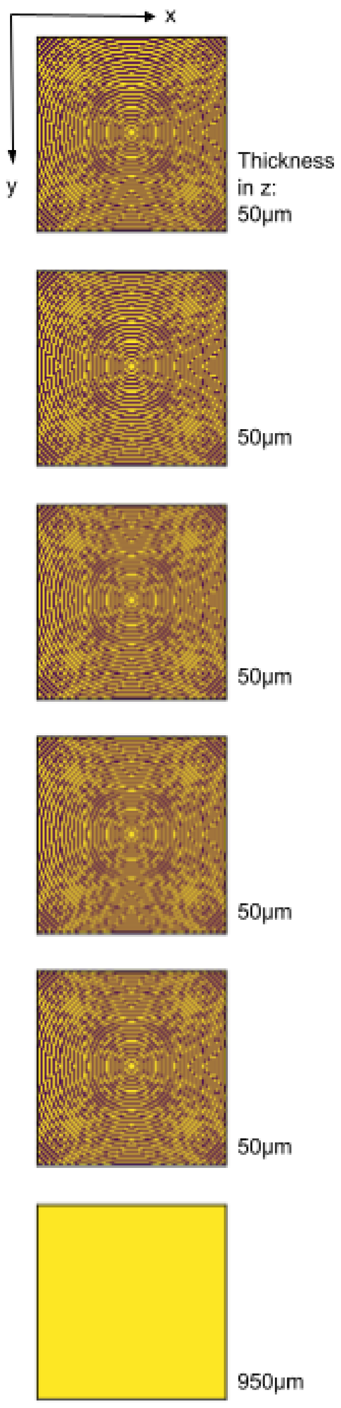

Figure 24: Best discovered metasurface shape for the near field case. Each plot shows one layer of the device, with transverse dimensions $5mm$ x $5mm$ and their thicknesses listed. The device resolution is 180×180 pixels. The topmost plot represents that layer which is closest to the light source, and the bottom-most is the uniform substrate layer. Yellow pixels mark locations filled with silicon, while purple pixels mark locations void of material.

This solution was obtained for learning rate $L = 0.8$, maximum iterations $N = 100$, maximum sigmoid coefficient $S_f = 10.0$, and initial pixel height $h_i = 3$.

There are several remarkable features of the discovered lens shape - firstly, its symmetry. Symmetry along the diagonal axis from the top left to lower right corners of the design is expected based on the transverse electric (TE) polarization used for the source. Full symmetry along both the x and y axes is not required, but appears to have been incentivized in this case.

Nearly all pixels have either the maximum or minimum possible height. This, combined with the concentric circles in the design, causes the lens to resemble a **fresnel lens**. This makes sense, considering that the main benefits of fresnel lenses are that they can produce short focal lengths while remaining compact in design compared to conventional refractive lenses. Apparently, the error imposed by realizing such a device under the discretizing manufacturing constraints is not so great as to negate its benefits.

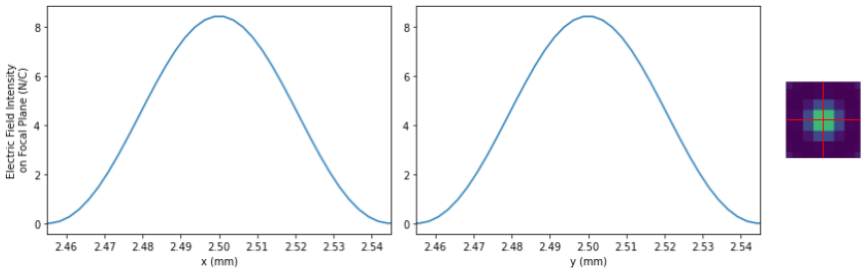

Figure 25: Diffraction limit evaluation metric as applied to the best discovered solution in the near field design case. For the near field case, the diffraction limit can be calculated as $d_{near} = \lambda/2 \cdot NA \approx 171.43\mu m$. As seen in the plot, the half-width of the focal spot gaussian w_{near}, in both x and y, is $\approx 650\mu m$, giving a final evaluation score of $d_{near}/w_{near} \approx 0.26$. While this means that the device is certainly not acting at the diffraction limit, it is also a nontrivial value, and is comparable or better to the value obtained for other far field solutions.

6.2 Far Field Case

The best result found for the far field design case is detailed in Figures 26, 27, and 28.

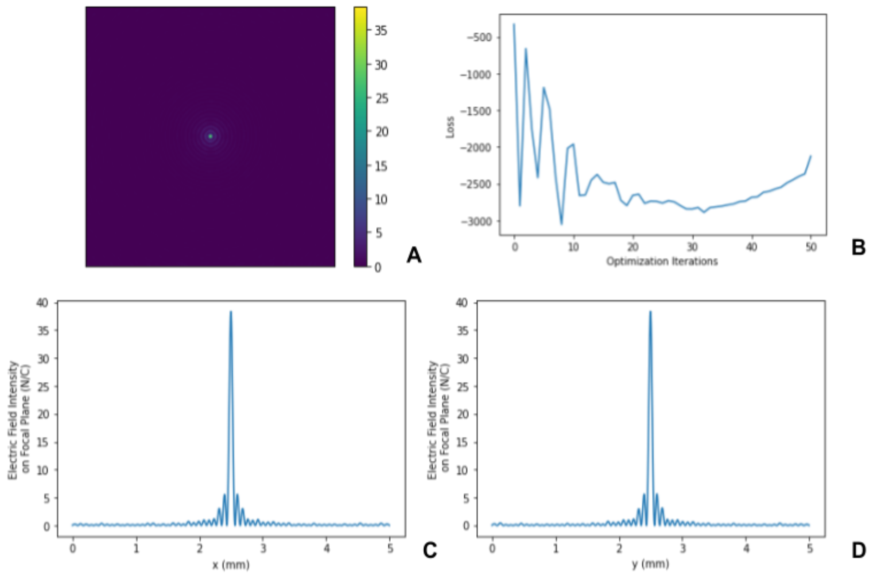

Figure 26: Best result found by the hyperparameter search, for the far field design case. The final scattering pattern on the focal plane, in terms of electric field intensity, is given (A) as well as cross sections of the scattering pattern in the transverse directions (C,D). Also given is the learning curve for this optimization (B).

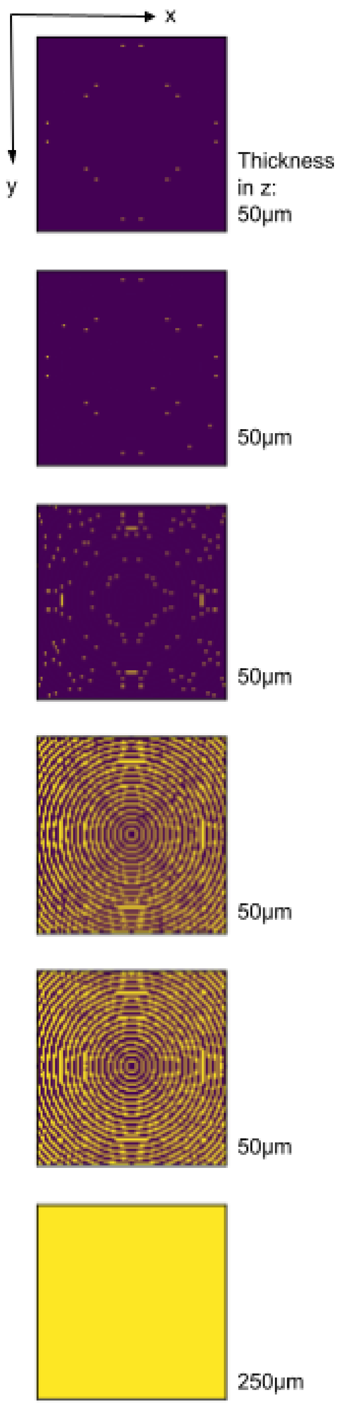

Figure 27: Best metasurface shape for the far field case. Each plot shows one layer of the device, with transverse dimensions $5mm$ x $5mm$ and their thicknesses listed. The device resolution is 180×180 pixels. The topmost plot represents that layer which is closest to the light source, and the bottom-most is the uniform substrate layer. Yellow pixels mark locations filled with silicon, while purple pixels mark locations void of material.

This solution was obtained for learning rate $L = 0.8$, maximum iterations $N = 50$, maximum sigmoid coefficient $S_f = 10.0$, and initial pixel height $h_i = 0$.

The first two layers strongly resemble the best device from the near field case, consisting of concentric rings of material reminiscent of a fresnel lens. However, the center of the design is an empty circle, as opposed to the filled circle of the near field case. Also, the near field design has more irregularities near the edges of the design as opposed to the far field one.

This design also shows the diagonal symmetry as expected from the source polarization. Symmetry along the x and y axes is also present in all layers except the 4th, where only the diagonal symmetry reigns. This may show that x and y axis symmetry is often, but not always, a beneficial characteristic for these designs.

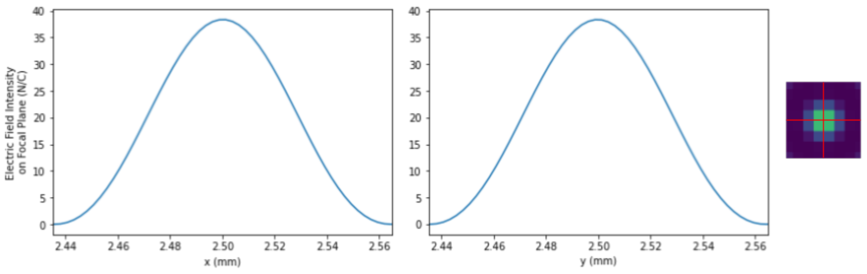

Figure 28: Diffraction limit evaluation metric as applied to the best discovered solution in the far field design case. In this case, the diffraction limit is $d_{far} = \lambda/2 \cdot NA \approx 225.71\mu m$. As seen in the plot, the half-width of the focal spot gaussian w_{far}, in both x and y, is $\approx 450\mu m$, giving a final evaluation score of $d_{far}/w_{far} \approx 0.50$. As in the near field case, the best solution is not operating at the diffraction limit. However, in terms of the diffraction limit metric, this solution is even more optimal than the best discovered in the near field case, and again is comparable or better to other results obtained in the far field case. It is possible that having a nonzero focal length enables better performance in this metric.

All other solutions found during the hyperparameter grid search runs can be found in the project's Github repository at `https://github.com/deveringham/metalens_optimization`. In the */results* directory, each solution is stored in a single, JSON-readable file, with the filename identifying the hyperparameter values used, the device resolution, and the design case.

7 Future Work

While this project has certainly been a success in finding an effective metalens optimization method and constructing good solutions to the CoPILOT design problem, there is always room for algorithmic improvement and future work. In this section, further research questions of interest on the topic of metalens construction for CoPILOT are listed. Also, additional implementation ideas, which could not be realized due to time limitations, are suggested.

7.1 Additional Research Questions

Listed here are a few research questions which were not able to be answered within the limited scope of this project, but which may be of interest to future students who might continue the work.

Can the solution quality be improved by including some optional design parameters? Is the computational feasibility of the optimization algorithm maintained?

As specified by the CoPILOT design constraints, there are a number of extra lens design parameters which are left to be optionally determined as part of the optimization process. It is possible that considering one or more of these, by including them in the optimization algorithm, loss function, or evaluation metrics, may help improve solution quality. The extra design parameters include:

- **Lenslet Thickness**
  The thickness of each lenslet, labeled as d in Figure 2, may be specified before optimization or left as one additional optimizable parameter. If left to be optimized, this may be treated as a single additional free geometric parameter, which is not required to be discretized like the pixel heights. Because solution methods must already handle a large number of geometric parameters in the form of the pixel heights, it is unlikely that adding one more will have a significant effect on performance. However, like handling most of these extra design parameters, it adds some development cost with no clear promise of improving solutions.

- **Single-Sided vs Double-Sided Device**
  It is possible for two separate topologies to be etched on the transmission and reflection sides of the device. Doing so would double the number of geometrical metasurface parameters, leading to significantly increased cost of optimization. However, this also substantially increases the design space of possibly lenses, which may lead to the discovery of more effective designs.

- **Single or Multiple Materials**
  The use of multiple materials, perhaps one per layer, with different indices of refraction, is a possibility. This introduces a single extra free optimization parameter per layer, corresponding to the index of refraction of each material. This has a similar additional computational cost as freely optimizing device thickness, and likely a similar potential payoff.

- **Anti-reflective Coating**
  It is a common practice in optics design to incorporate thin-film anti-reflective coatings on the surface of devices in order to eliminate stray light. This can be treated as a single extra uniform layer on the reflection side of each pixel, with its thickness optionally as another free parameter. This may be both tricky to implement and computationally heavy, as it would involve doubling the number of layers in the device in order to allow for coating on each pixel regardless of its height. However, it allows for evaluation of whether optimization of the metasurface shape alone can offer reflection mitigation comparable to using an anti-reflective coating.

Determining which of these optional design parameters are most beneficial to include in the device optimization method would be both an interesting exploration of more exotic lens design possibilities and potentially a serious benefit to the quality of produced lenses.

How sensitive are lens designs to manufacturing inaccuracies and to the angles of incidence of source waves, and can these sensitivities be minimized?

It is desirable for lenses to remain effective under source illumination which is not exactly normal to the device surface. In addition, they should ideally also remain effective under slight changes in the width of pixels which might arise due to nonzero manufacturing tolerances. Therefore, the final research question concerns the robustness of device performance to these perturbations.

A first step would be to characterize sensitivity to perturbations in source angle and pixel width. This can be done by starting with a optimized solution, then applying changes to these values and quantifying the change in device performance.

A second step would be to incorporate these perturbable parameters as part of the optimization method in order to minimize solution sensitivity. This could take the form of additional terms in the loss function. Alternatively, at each optimization step, simulations of several devices, each representing a randomly perturbed form of the current device, could be simulated. Then the loss resulting from each of these designs could be summed for a final loss. Because evaluating many loss function terms or designs at each optimization step would likely be computationally expensive, the best strategy might be to use the previously performed characterization of sensitivities in order to make informed decisions about which source angle directions and pixel edges are the most sensitive and should therefore be included in the optimization.

7.2 Additional Implementation Ideas

Finally, there are a few alterations to the successful lens optimization algorithm which might be considered by a future student.

- **Random Initialization**
  As described in section 5.3, random initialization of the metasurface was implemented as a possibility for initial guesses. However, this was not extensively tested, or included as part of the hyperparameter search. Thus, this remains as an open topic of investigation. During the hyperparameter search, it was found that solution quality was very sensitive to initial guess. Therefore, it stands to reason that better solutions may be found for some initial guess which could be generated randomly. However, For any random initial guess, the quality of solution that the algorithm reaches also depends on the selection of other hyperaparameters N, L and S_f. Therefore, a proper exploration of the benefits of random initialization probably requires many runs for each hyperparameter selection, and would take a long time. If resources are available, this is easy to set up and run over an extended period.

- **Exploiting Symmetry**
  The symmetry of the discovered metasurface designs was discussed in section 6. Because some symmetry is expected based on the polarization of the light source and additional symmetry may be a characteristic of good solutions, it is possible that restricting the design space to include symmetrical designs would reduce the search space, increasing the efficiency of the algorithm, while not significantly hurting the quality of produced solutions. If symmetry across both the x and y axes is assumed, then a $2.5mm \times 2.5mm$ device can be optimized and then simply reflected across x and y to produce the final device. Theoretically, this could lead to a huge speedup, perhaps allowing the device resolution of N to be increased

beyond what is currently possible.
However, this is a bit tricky in practice. If the entire $5mm \times 5mm$ device is simulated via RCWA at each step, then no computations are saved, even if the device is constrainted to be symmetrical. If, however, only one $2.5mm \times 2.5mm$ device is simulated, performance is increased but the loss function must then be modified to incentivize focus at a corner of the focal plane rather than at the center. This seems possible. However, initial tests showed that the algorithm had trouble producing designs which could focus well at the corner. In addition, it is probable that in most cases, each of the 4 corners of the device do not function independently - that is, one corner device alone will function differently when it is placed next to the other three. This means that optimizing one corner of the device and then reflecting it to produce a final design may not actually result in a convergent algorithm or good solutions.
However, this idea still holds a lot of promise. Maybe the RCWA solver can be improved to take advantage of symmetry and improve performance. Or maybe by including slightly more than one quarter of the device in each simulation, the ability to focus at the corner can be improved. Some algorithmic improvement seems necessary in order to facilitate optimization of maximal resolution, 250×250 pixel devices, and something along these lines seems promising.

- **Nonlinear Sigmoid Coefficient Annealing**
  In the current algorithm, the sigmoid coefficient is increased linearly with the iteration number. It is possible, however, that this is not optimal, and that increasing the coefficient according to some other function may allow better solutions to be reached.
  The simplest possibility is to first allow the sigmoid coefficient to remain at its initial value for some number of iterations before beginning its linear increase. This allows a longer period of free optimization before the solution is forced to be admissible, which may result in discovery of better solutions, similar to the effect of increasing N without actually increasing the execution time of the algorithm.
  Other possibilities are numerous, and require some testing. The coefficient can be increased exponentially, quadratically, or according to any arbitrary nonlinear function. An especially interesting option is using a periodic function, such as a sin or cos curve, and perhaps composed with another, increasing, function. This may allow for repeated refinement of a discretized solution.

8 Acknowledgements

To all who have read this far, thank you. Special thanks to my supervisors Aurèle Adam and Matthias Möller, COSSE programme coordinator Kees Vuik, and friends Ben Levin, Leonie Schulze, Mirko Kemna, and Genya Crossman, all of whom have lent incredible support throughout this project.

Bibliography

[1] *About the COBE Far Infrared Absolute Spectrophotometer (FIRAS).* https://lambda.gsfc.nasa.gov/product/cobe/about_firas.html.

[2] *About the Herschel Space Observatory Instruments.* https://www.herschel.caltech.edu/page/instruments.

[3] *Sofia Observatory Instruments.* https://www.sofia.usra.edu/instruments/sofia-instruments-suite.

[4] Takao Nakagawa et al. "Far-Infrared [C scpii/scp] Line Survey Observations of the Galactic Plane". In: *The Astrophysical Journal Supplement Series* 115.2 (Apr. 1998), pp. 259–269. DOI: 10.1086/313082. URL: https://doi.org/10.1086%2F313082.

[5] Jianming Dai et al. "Terahertz time-domain spectroscopy characterization of the far-infrared absorption and index of refraction of high-resistivity, float-zone silicon". In: *Journal of The Optical Society of America B-optical Physics - J OPT SOC AM B-OPT PHYSICS* 21 (July 2004). DOI: 10.1364/JOSAB.21.001379.

[6] Kimmo Solehmainen et al. "Development of multi-step processing in silicon-on-insulator for optical waveguide applications". In: *Journal of Optics A: Pure and Applied Optics J. Opt. A: Pure Appl. Opt* 8 (July 2006), pp. 455–460. DOI: 10.1088/1464-4258/8/7/S22.

[7] TU Delft. *Kavli Nanolab Description.* 2022. URL: https://www.tudelft.nl/tnw/over-faculteit/afdelingen/quantum-nanoscience/kavli-nanolab-delft (visited on 03/15/2022).

[8] D. R. Smith et al. "Composite Medium with Simultaneously Negative Permeability and Permittivity". In: *Phys. Rev. Lett.* 84 (18 May 2000), pp. 4184–4187. DOI: 10.1103/PhysRevLett.84.4184. URL: https://link.aps.org/doi/10.1103/PhysRevLett.84.4184.

[9] D. R. Smith et al. "Left-Handed Metamaterials". In: *Photonic Crystals and Light Localization in the 21st Century.* Ed. by Costas M. Soukoulis. Dordrecht: Springer Netherlands, 2001, pp. 351–371. ISBN: 978-94-010-0738-2. DOI: 10.1007/978-94-010-0738-2_25. URL: https://doi.org/10.1007/978-94-010-0738-2_25.

[10] Xue Jiang et al. "All-dielectric metalens for terahertz wave imaging". In: *Opt. Express* 26.11 (May 2018), pp. 14132–14142. DOI: 10.1364/OE.26.014132. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-26-11-14132.

[11] Takehito Suzuki, Kota Endo, and Satoshi Kondoh. "Terahertz metasurface ultra-thin collimator for power enhancement". In: *Opt. Express* 28.15 (July 2020), pp. 22165–22178. DOI: 10.1364/OE.392814. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-28-15-22165.

[12] Takehito Suzuki et al. "Metalens mounted on a resonant tunneling diode for collimated and directed terahertz waves". In: *Opt. Express* 29.12 (June 2021), pp. 18988–19000. DOI: 10.1364/OE.427135. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-29-12-18988.

[13] Rajath Sawant et al. "Aberration-corrected large-scale hybrid metalenses". In: *Optica* 8.11 (Nov. 2021), pp. 1405–1411. DOI: 10.1364/OPTICA.434040. URL: http://opg.optica.org/optica/abstract.cfm?URI=optica-8-11-1405.

[14] Delin Jia et al. "Transmissive terahertz metalens with full phase control based on a dielectric metasurface". In: *Opt. Lett.* 42.21 (Nov. 2017), pp. 4494–4497. DOI: 10.1364/OL.42.004494. URL: http://opg.optica.org/ol/abstract.cfm?URI=ol-42-21-4494.

[15] Chun-Chieh Chang et al. "Demonstration of a highly efficient terahertz flat lens employing tri-layer metasurfaces". In: *Opt. Lett.* 42.9 (May 2017), pp. 1867–1870. DOI: 10.1364/OL.42.001867. URL: http://opg.optica.org/ol/abstract.cfm?URI=ol-42-9-1867.

[16] Yufei Gao et al. "Polarization Independent Achromatic Meta-Lens Designed for the Terahertz Domain". In: *Frontiers in Physics* 8 (2020). ISSN: 2296-424X. DOI: 10.3389/fphy.2020.606693. URL: https://www.frontiersin.org/article/10.3389/fphy.2020.606693.

[17] Hao Chen et al. "Sub-wavelength tight-focusing of terahertz waves by polarization-independent high-numerical-aperture dielectric metalens". In: *Opt. Express* 26.23 (Nov. 2018), pp. 29817–29825. DOI: 10.1364/OE.26.029817. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-26-23-29817.

[18] Pei Ding et al. "Graphene aperture-based metalens for dynamic focusing of terahertz waves". In: *Opt. Express* 26.21 (Oct. 2018), pp. 28038–28050. DOI: 10.1364/OE.26.028038. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-26-21-28038.

[19] Yuyao Chen et al. "Physics-informed neural networks for inverse problems in nano-optics and metamaterials". In: *Opt. Express* 28.8 (Apr. 2020), pp. 11618–11633. DOI: 10.1364/OE.384875. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-28-8-11618.

[20] Sensong An et al. "A Deep Learning Approach for Objective-Driven All-Dielectric Metasurface Design". In: *ACS Photonics* 6.12 (2019), pp. 3196–3207. DOI: 10.1021/acsphotonics.9b00966. eprint: https://doi.org/10.1021/acsphotonics.9b00966. URL: https://doi.org/10.1021/acsphotonics.9b00966.

[21] Li Jiang et al. "Neural network enabled metasurface design for phase manipulation". In: *Opt. Express* 29.2 (Jan. 2021), pp. 2521–2528. DOI: 10.1364/OE.413079. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-29-2-2521.

[22] Fardin Ghorbani et al. "Deep neural network-based automatic metasurface design with a wide frequency range". In: *Scientific Reports* 11 (Mar. 2021). DOI: 10.1038/s41598-021-86588-2.

[23] Shane Colburn and Arka Majumdar. "Inverse design and flexible parameterization of meta-optics using algorithmic differentiation". In: *Communications Physics* 4 (Mar. 2021). DOI: 10.1038/s42005-021-00568-6.

[24] Zin Lin et al. "Topology optimization of freeform large-area metasurfaces". In: *Opt. Express* 27.11 (May 2019), pp. 15765–15775. DOI: 10.1364/OE.27.015765. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-27-11-15765.

[25] Raymond Rumpf. "Improved formulation of scattering matrices for semi-analytical methods that is consistent with convention". In: *Progress In Electromagnetics Research B* 35 (Aug. 2011), pp. 241–261. DOI: 10.2528/PIERB11083107.

[26] *EMPossible Computational Electromagnetics Course Notes.* https://empossible.net/academics/emp5337/.

[27] H. H. Li. "Refractive index of silicon and germanium and its wavelength and temperature derivatives". In: *Journal of Physical and Chemical Reference Data* 9.3 (1980), pp. 561–658. DOI: 10.1063/1.555624. eprint: https://doi.org/10.1063/1.555624. URL: https://doi.org/10.1063/1.555624.

[28] F. Mistiri and Alan P. Wang. "The Star-product and its Algebraic Properties". In: *Journal of the Franklin Institute* 321.1 (1986), pp. 21–38. ISSN: 0016-0032. DOI: https://doi.org/10.1016/0016-0032(86)90053-0. URL: https://www.sciencedirect.com/science/article/pii/0016003286900530.

[29] Kyoji Matsushima and Tomoyoshi Shimobaba. "Band-Limited Angular Spectrum Method for Numerical Simulation of Free-Space Propagation in Far and Near Fields". In: *Opt. Express* 17.22 (Oct. 2009), pp. 19662–19673. DOI: 10.1364/OE.17.019662. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-17-22-19662.

[30] Wonyeol Lee et al. "On Correctness of Automatic Differentiation for Non-Differentiable Functions". In: *NeurIPS 2020 - 34th Conference on Neural Information Processing Systems.* Vancouver / Virtual, Canada, Dec. 2020. URL: https://hal.inria.fr/hal-03081582.

[31] Martın Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).* Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[32] Tyler W. Hughes et al. "Forward-Mode Differentiation of Maxwell's Equations". In: *ACS Photonics* 6.11 (Oct. 2019), pp. 3010–3016. ISSN: 2330-4022. DOI: 10.1021/acsphotonics.9b01238. URL: http://dx.doi.org/10.1021/acsphotonics.9b01238.

[33] Logan Su et al. "Nanophotonic inverse design with SPINS: Software architecture and practical considerations". In: *Applied Physics Reviews* 7.1 (2020), p. 011407. DOI: 10.1063/1.5131263. eprint: https://doi.org/10.1063/1.5131263. URL: https://doi.org/10.1063/1.5131263.

[34] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999118307125`.

[35] Lu Lu et al. “DeepXDE: A deep learning library for solving differential equations”. In: *SIAM Review* 63.1 (2021), pp. 208–228. DOI: `10.1137/19M1274067`.

[36] Matthias Seeger et al. “Auto-Differentiating Linear Algebra”. In: (Oct. 2017).

[37] Victor Liu and Shanhui Fan. “S^4 : A free electromagnetic solver for layered periodic structures”. In: *Computer Physics Communications* 183.10 (Oct. 2012), pp. 2233–2244. DOI: `10.1016/j.cpc.2012.04.026`.

[38] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: `10.48550/ARXIV.1412.6980`. URL: `https://arxiv.org/abs/1412.6980`.

[39] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems.* Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

[40] Nicholas J. Higham. “The Scaling and Squaring Method for the Matrix Exponential Revisited”. In: *SIAM J. Matrix Anal. Appl.* 26.4 (Apr. 2005), pp. 1179–1193. ISSN: 0895-4798. DOI: `10.1137/04061101X`. URL: `https://doi.org/10.1137/04061101X`.

[41] Philipp Bader, Sergio Blanes, and Fernando Casas. “Computing the Matrix Exponential with an Optimized Taylor Polynomial Approximation”. In: *Mathematics* 7.12 (2019). ISSN: 2227-7390. DOI: `10.3390/math7121174`. URL: `https://www.mdpi.com/2227-7390/7/12/1174`.