

# PyMieDiff: A differentiable Mie scattering library

Oscar K. C. Jackson,<sup>1</sup> Simone De Liberato,<sup>1,2,3</sup> Otto L. Muskens,<sup>1,\*</sup> and Peter R. Wiecha<sup>4,†</sup>

<sup>1</sup>Physics and Astronomy, Faculty of Engineering and Physical Sciences,  
University of Southampton, SO17 1BJ Southampton, UK

<sup>2</sup>Istituto di Fotonica e Nanotecnologie – Consiglio Nazionale delle Ricerche (CNR), Piazza Leonardo da Vinci 32, Milano, Italy

<sup>3</sup>Sensorium Technological Laboratories, Nashville, Tennessee 37210, USA

<sup>4</sup>LAAS-CNRS, Université de Toulouse, UPS, Toulouse, France

Light scattering by spherical-shaped particles of sizes comparable to the wavelength is foundational in many areas of science, from chemistry to atmospheric science, photonics and nanotechnology. With the new capabilities offered by machine learning, there is a great interest in end-to-end differentiable frameworks for scattering calculations. Here we introduce PyMieDiff, a fully differentiable, GPU-compatible implementation of Mie scattering for core-shell particles in PyTorch. The library provides native, autograd-compatible spherical Bessel and Hankel functions, vectorized evaluation of Mie coefficients, and APIs for computing efficiencies, angular scattering, and near-fields. All inputs – geometry, material dispersion, wavelengths, and observation angles and positions – are represented as tensors, enabling seamless integration with gradient-based optimisation or physics-informed neural networks. The toolkit can also be combined with “TorchGDM” for end-to-end differentiable multi-particle scattering simulations. PyMieDiff is available under an open source licence at <https://github.com/UoS-Integrated-Nanophotonics-group/MieDiff>.

**Keywords:** Mie theory, core-shell particles, automatic differentiation, gradient optimisation, PyTorch, GPU acceleration

## I. INTRODUCTION

Mie theory provides an analytical solution for light scattering by spherical particles, which underpins many nano-optical applications (e.g., color generation, dielectric metamaterials, nanoantennas, radiative cooling).<sup>1</sup> Multi-layer (core–shell) spheres extend this to complex nanoparticle designs. While the forward scattering problem can be solved by standard Mie formulae,<sup>2</sup> the inverse design of such particles remains challenging: it typically requires many repeated forward simulations and costly optimization. Recent work has therefore turned to machine learning, training neural nets to predict scattering spectra and act as differentiable surrogates.<sup>3–7</sup> An alternative approach, which involves integrating the exact Mie solution directly into gradient-based design workflows has not been fully exploited, as analytic derivatives tend to become very bulky,<sup>8</sup> especially for core-shell spheres.

A variety of open-source packages implement Mie theory for spheres. For example, “MiePython” is a pure-Python (NumPy/Numba) implementation of Mie theory for homogeneous spheres.<sup>9</sup> “PyMieSim” is an open-source Python/C++ toolkit that supports scattering from spheres, infinite cylinders and core–shell geometries.<sup>10</sup> “Scattnlay” is a c++ programm, focused on the calculation of nearfields inside and around multi-layer spherical particles.<sup>11</sup> “pyMieCS” is a vectorized NumPy library specialized for core–shell nanoparticles.<sup>12</sup> These tools offer validated, high-speed forward solvers, but none provide native differentiation capabilities or GPU backends.

Automatic differentiation (AD) is the key numerical technique behind deep learning, and allows to calculate arbitrary derivatives for any numerical calculation with close to analytical precision.<sup>13</sup> The photon-

ics community has seen rapid growth of differentiable simulation tools based on AD. Recent efforts have produced open-source Maxwell solvers with autodifferentiation (e.g., FEM, Fourier modal, FDTD or volume integral approaches) to enable gradient-based inverse design and topology optimization.<sup>14–20</sup> For Mie computations however, so far still external libraries are required, which are not AD-compatible.<sup>20</sup>

Recent works have highlighted the power of integrating physics-based models with deep learning. An important example are deep learning based nanoparticle design approaches which, so far, often rely on data-based tandem networks or surrogate models.<sup>5,6,21,22</sup> These solutions often suffer from a lack of fidelity of the surrogate models<sup>23</sup>, as their interpolation is limited to the dataset they are trained on. These large datasets only cover a limited parameter space, and new problems require new datasets. Replacing such surrogate models by AD capable analytical solvers would suppress this typical point of failure.

We present the PyTorch-based toolkit “PyMieDiff” which implements analytical Mie theory for core–shell spherical particles with full support for automatic differentiation. Our library implements the standard Mie recurrence (spherical Bessel/Hankel forward and backward recurrences, angular functions, vector spherical harmonics<sup>24</sup>) entirely in PyTorch. All quantities – particle geometry (layer thicknesses, materials), wavelengths, and scattering angles – can be treated as differentiable tensors. As a result any Mie-derived observable can be backpropagated to compute gradients with respect to any input parameter. The code is fully vectorized over Mie orders, wavelengths, angles, and positions. It supports batched evaluation of many particles in parallel and runs on GPU. Our simple API exposes a `Particle` class for defining core-shell spheres with possibly dispersive mate-

rials (using the `refractiveindex.info` format<sup>25</sup>). It implements methods to compute extinction, absorption and scattering efficiencies, far-fields and S-matrix elements, as well as near-fields, all with autograd support, as illustrated in figure 1.

Key features of our toolkit include:

- Mie coefficients, efficiencies and cross sections, angular scattering, and near-fields for core-shell particles
- Compatibility with TorchGDM for multi-particle scattering simulations with autodiff support.<sup>20</sup>
- End-to-end auto-differentiability: the entire Mie computation is implemented in PyTorch, allowing gradient-based optimization of particle parameters (sizes, layer thicknesses, refractive indices, etc.) through backpropagation.
- GPU-accelerated vectorization: batches of particles, wavelengths and angles are processed in parallel, potentially yielding orders-of-magnitude speedups when workloads are large (especially relative to serial codes).
- Flexible materials interface: refractive index data (from `refractiveindex.info`) are interpolated through a PyTorch implementation, so material dispersion enters the gradient graph transparently.
- User-friendly API: a high-level object oriented API (`Particle` class) as well as a simple to use functional API make it easy to plug the Mie solver into other pipelines, including physics-informed neural networks or design loops.

By embedding analytic Mie theory in an autodiff framework, our toolkit enables new design approaches in nano-optics. For example, we demonstrate using the analytic forward model in gradient-based inverse design of core-shell particles (including a “tandem” neural network architecture whose forward layer is our Mie solver). Because the Mie calculations are exact, this avoids the need for approximate surrogate training.<sup>6</sup> It also opens the door to differentiable sensitivity analyses and rapid parameter retrieval from data.

Potential applications of our autodiff Mie solver include (but are not limited to):

- Inverse design and optimization: using gradients to fit particle geometries or material choices to a target scattering spectrum or objective (extinction, backscatter, field enhancement, etc.), even in high-dimensional parameter spaces.
- Machine learning integration: embedding the Mie forward model as a layer in neural networks (e.g., tandem or physics-informed networks) so that analytic gradients flow through the scattering physics for training.<sup>26</sup>

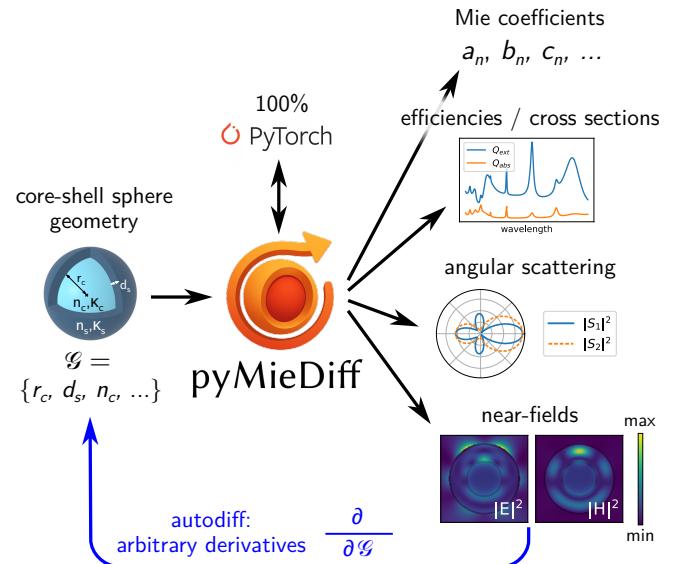


FIG. 1. Overview of PyMieDiff’s features. The core of the code is a differentiable Mie solver, which is used to compute various differentiable observables (efficiencies, angular scattering, near-fields). All calculations are fully implemented in PyTorch, they all support automatic differentiation and run on GPU. The shown results are from a gold-silicon core-shell particle with core radius  $r_c = 100\text{ nm}$  and shell radius  $r_s = 200\text{ nm}$ , placed in vacuum and illuminated by a linearly polarized plane wave.

- Fast batch simulation: efficient GPU-based evaluation of large ensembles of spheres over broad frequency bands for applications like cloud radiative transfer, hyperspectral imaging design, or Monte Carlo light propagation.
- Adjoint sensitivity analysis: computing how scattering observables change with small perturbations in geometry or material (for uncertainty quantification or experimental fitting), leveraging the computed autograd derivatives.

In summary, our PyTorch Mie toolkit provides a fully autodifferentiable, vectorized and GPU-accelerated implementation of core-shell Mie theory. This bridges analytic nanophotonics and modern ML tools, enabling gradient-based design methods that were previously difficult or impossible with black-box Mie solvers. While we demonstrate the toolkit on optical-frequency examples, the implementation is completely general and can be applied at any wavelength (from microwaves to X-rays) where Mie scattering is relevant.

## II. IMPLEMENTATION

Mie theory describes the interaction of light with a spherical particle through field expansion coefficients

$a_n, b_n, c_n$  and  $d_n$ , where  $a_n$  and  $b_n$  are the coefficients for the outgoing (scattered) field, while  $c_n$  and  $d_n$  are the expansion coefficients for the incoming (internal) fields.

In the case of a core-shell particle of core radius  $r_c$  and shell radius  $r_s$ , the Mie coefficients are functions of the scale factors  $x = kr_c$  and  $y = kr_s$  (wavevector  $k = k_0 n_{\text{env}}$ ,  $k_0 = 2\pi/\lambda_0$ ) and the relative refractive indices,  $m_c = n_c/n_{\text{env}}$ , and  $m_s = n_s/n_{\text{env}}$  for the core and shell, respectively. The explicit form of the core-shell Mie scattering coefficients is given in the appendix F. The evaluation of the Mie coefficients requires Riccati-Bessel functions and their derivatives:

$$\psi_n(z) = z j_n(z), \quad \xi_n(z) = z h_n^{(1)}(z), \quad \chi_n(z) = z y_n(z). \quad (1)$$

where  $j_n$  and  $y_n$  are spherical Bessel functions of the first, respectively second kind, and  $h_n^{(1)} = j_n + iy_n$  are spherical Hankel functions of the first kind.

The derivatives can be obtained through the identity:

$$\frac{d}{dz} f_n(z) = f_{n-1}(z) - \frac{n+1}{z} f_n(z), \quad (2)$$

where  $f$  represents any spherical Bessel or Hankel function (see also appendix D).

In PyTorch (as of version 2.9), spherical Bessel functions are not implemented. The key contribution of our work is therefore to provide PyTorch-based, AD-capable, vectorized, fast and stable spherical Bessel routines. We implement two versions of AD enabled spherical Bessel functions: (1) a SciPy wrapper and (2) a native PyTorch implementation based on recurrences. While the SciPy wrapper is based on stable and tested SciPy routines, it is not GPU compatible and comes with additional memory transfer overhead. This limits its efficiency, especially in batched evaluations. The native torch implementation on the other hand is fully GPU capable and entirely vectorized. On single, non-parallelized evaluations the SciPy version is faster, whereas in cases where many wavelengths, several particles, high orders, or multiple angles are to be calculated, the native PyTorch implementation becomes advantageous.

In our PyTorch implementation we compute the spherical Bessel functions  $j_n$  using downward recurrence, while the spherical Neumann functions  $y_n$  are obtained by upward recurrence from analytic low-order seeds. This separation is necessary because forward propagation of the  $j_n$  recurrence rapidly becomes unstable for large  $n$  and for  $z$  with a large imaginary part (e.g., for spheres with strongly absorbing / metallic permittivities).

Using the Riccati-Bessel form  $\psi_n(z) = z j_n(z)$  the three-term recurrence reads:

$$\psi_{n+1}(z) = \frac{2n+1}{z} \psi_n(z) - \psi_{n-1}(z). \quad (3)$$

To obtain  $j_n$ , we run through Eq. (3) as backward recurrence, starting from seeds  $j_n = 0$  and  $j_{n+1} = 10^{-25}$  at a sufficiently large order<sup>24</sup>. The sequence is finally renormalized to the known  $j_0$  ( $j_0(z) = \sin z/z$ ). The Neumann functions  $y_n$  are generated via the same three-term recurrence, applied upward from analytic seeds

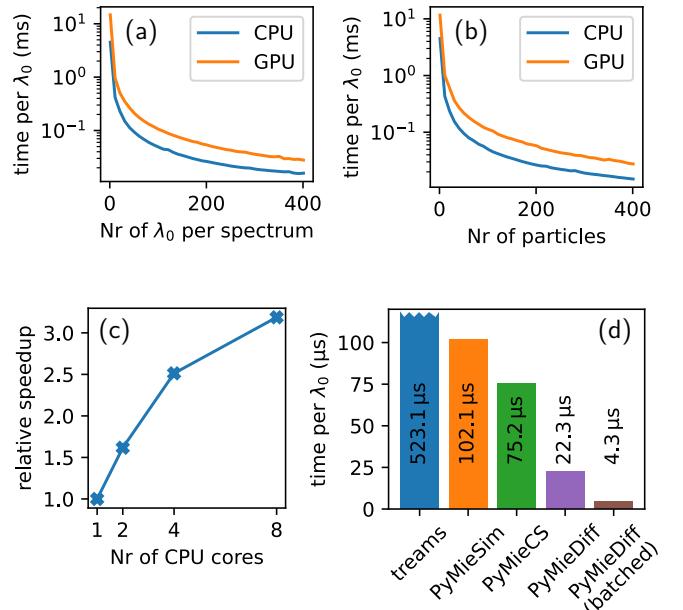


FIG. 2. Benchmarks. (a) timing for increasing numbers of simultaneously evaluated wavelengths. (b) timing for increasing numbers of simultaneously evaluated particles. (c) parallelization speedup on multi-core CPUs. (d) timing comparison to other Mie toolkits (on CPU). The batched example evaluates 256 different particles simultaneously, each at 256 wavelengths. Note that “treams” is not performance-optimized for Mie evaluation since it is not a dedicated Mie toolkit (but T-matrix), in consequence the time-bar for the treams result is not fully shown.

( $y_0(z) = -\cos z/z$ ,  $y_1(z) = -\cos z/z^2 - \sin z/z$ ).<sup>27</sup> Implementing both branches using only differentiable PyTorch tensor operations preserves automatic differentiation, while avoiding overflow/underflow and the loss of accuracy encountered with lossy materials, for example in plasmonic parameter regimes. Spherical Hankel functions and derivatives are generated from the spherical Bessel relations (see Eqs. (2) and appendix D).

With these spherical Bessel and Hankel functions and their derivatives, the Mie coefficients can be readily evaluated (see appendix F). Based on these, scattering efficiencies and angular scattering patterns can be calculated, as detailed in appendix G.

As “PyMieDiff” is entirely implemented in PyTorch, automatic differentiation is fully supported through all calculations.

### III. BENCHMARK

In figure 2 we assess the computational performance of PyMieDiff and compare it against existing Mie scattering toolkits. Owing to its fully vectorized implementation utilizing PyTorch tensors, PyMieDiff exhibits efficient scaling for any independent parameter, like number

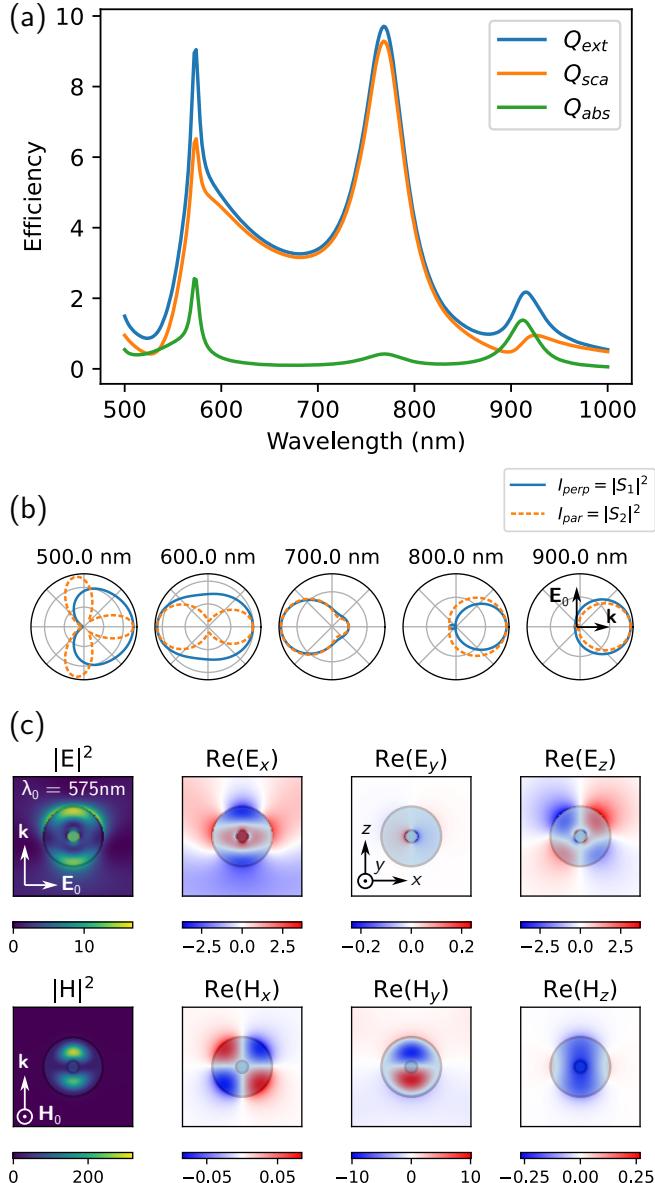


FIG. 3. Mie forward evaluation by the example of scattering from a gold-silicon core-shell particle with core radius  $r_c = 20\text{ nm}$  and shell radius  $r_s = 100\text{ nm}$ , placed in vacuum and illuminated by a linearly polarized plane wave. Tabulated material permittivities are taken from literature.<sup>28,29</sup> (a) Extinction (blue), scattering (orange), and absorption (green) efficiency spectra. (b) Scattering radiation patterns in the scattering plane at selected wavelengths for perpendicularly (blue lines) and parallel (orange dashed lines) polarized light. (c) Near-fields (electric and magnetic in top and bottom row) inside and around the particle, evaluated at the resonance at  $\lambda_0 = 575\text{ nm}$ . The shown maps are  $400 \times 400\text{ nm}^2$ . Colorbars indicate field intensity (leftmost panels) and amplitude (second from left to right panels), relative to the incident field absolute amplitude.

of wavelengths per spectrum (Fig. 2a), incident angles,

```

1 import torch
2 import pymiediff as pmd
3
4 # --- setup
5 r_core = 20.0 # nm
6 r_shell = 100.0 # nm
7 mat_core = pmd.materials.MatDatabase("Au")
8 mat_shell = pmd.materials.MatDatabase("Si")
9 n_env = 1.0
10
11 p = pmd.Particle(
12     r_core=r_core,
13     r_shell=r_shell,
14     mat_core=mat_core,
15     mat_shell=mat_shell,
16     mat_env=n_env,
17 )
18
19 # --- evaluation
20 w10 = torch.linspace(500, 1000, 50) # nm
21 k0 = 2 * torch.pi / w10
22
23 # efficiency spectra
24 cs = p.get_cross_sections(k0)
25
26 # angular scattering (angles in radian)
27 theta = torch.linspace(0.0, 2 * torch.pi, 100)
28 angular = p.get_angular_scattering(k0, theta)
29
30 # nearfields (positions in nm)
31 x, z = torch.meshgrid(
32     torch.linspace(-250, 250, 100),
33     torch.linspace(-250, 250, 100),
34 )
35 y = torch.ones_like(x)
36 r_probe = torch.stack([x, y, z], dim=-1)
37
38 fields = p.get_nearfields(k0=k0, r_probe=r_probe)

```

Listing 1. Particle class usage example for forward Mie calculations.

evaluation positions or the number of particles (Fig. 2b). The evaluation time saturates only, when the memory transfer overhead is of similar magnitude as the actual computation time of one calculation batch.

Note that, while GPU execution is supported, the performance is currently constrained by the recurrences operating internally at double precision, which is required for numerical stability. On consumer grade GPUs (for Fig. 2 an NVIDIA RTX 4090 was used), double precision is significantly less efficient than lower precision types, which explains the superior performance on CPUs. Yet, GPU support can still benefit integration into deep learning schemes, as memory transfer during model evaluation is reduced.

On multi-core CPUs, PyMieDiff demonstrates good parallel efficiency, which seems to approach a plateau after eight cores (Fig. 2c). These parallelization capabilities are automatically used since they are inherent to PyTorch. This scaling behaviour is particularly beneficial for large-scale parameter sweeps, many-particle optimization tasks or machine learning integration with batched training schemes. In a direct comparison with other publicly available toolkits (Fig. 2d), PyMieDiff achieves markedly lower evaluation times per wavelength as soon as batched evaluation is possible, when our toolkit provides more than one order-of-magnitude improvement over the other evaluated tools. Note that

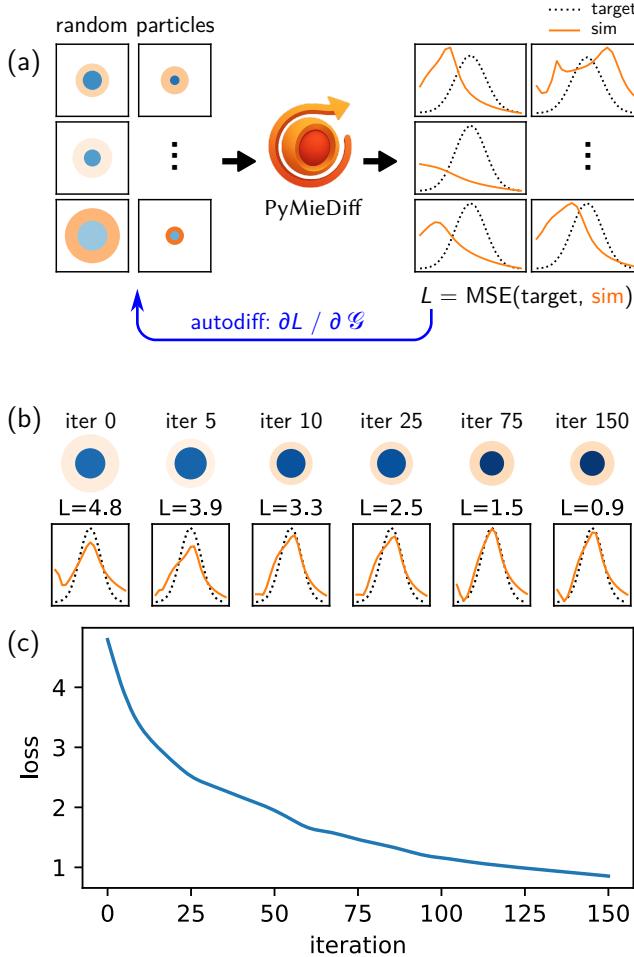


FIG. 4. Optimization example. (a) Sketch of the particle optimization procedure. A large number of random core-shell particles is randomly initialized and evaluated using PyMieDiff. Their optical properties (orange lines, here scattering efficiency spectra) are compared with the design target (dotted black lines, here a Gaussian response). The derivatives of the loss function (here MSE) with respect to the geometry parameters are obtained via PyTorch autograd and used to iteratively update the geometries. (b) Sketches of the best particles of selected iteration during an optimization (top row) together with their spectral response (bottom row). Relative sizes are on the same scale, the shades of the colors indicate the real part of the refractive index (darker means higher). (c) Loss function during the optimization.

while the T-Matrix package “treams”<sup>30</sup> is included for completeness, it is not parallelized and not specifically optimized for single particle Mie calculations.

Overall, these benchmarks establish PyMieDiff as a performant and scalable solution for Mie scattering computations, combining differentiability, GPU compatibility, and multi-core efficiency in a single framework.

```

1 # - gradient of scattering wrt wavelength
2 wl = torch.as_tensor(500.0) # nm
3 wl.requires_grad = True
4 cs = p.get_cross_sections(k0=2 * torch.pi / wl)
5
6 cs["q_sca"].backward()
7 dQdwl = wl.grad

```

Listing 2. Automatic differentiation usage example.

## IV. EXAMPLES

### A. Forward evaluation

As a first example, we demonstrate the forward solver by simulating the extinction, scattering and absorption efficiencies, as well as angular scattering patterns of a gold-silicon core-shell particle. The efficiency spectra are shown in figure 3a, scattering patterns of perpendicular and parallel polarized light are given in Fig. 3b. Electric and magnetic near field intensities and real parts of all field components are shown in Fig. 3c.

The code to configure the particle and calculate the spectra, angular scattering patterns and near-fields is given in listing 1, where tabulated materials are handled by the `MatDatabase` class and the `Particle` class acts as an easy-to-use high-level interface to PyMieDiff.

We compared the results to several other, openly available Mie solvers and obtained results identical to machine precision. This comparison can be found in the examples of the online documentation.

### B. Gradient based optimization

A central challenge in nanophotonics is designing nanostructures with desired optical properties. Design problems are typically ill-posed and need to be solved by optimization. While global optimization is useful in cases with many local minima or for non-continuous / discrete problems, this demonstration of local gradient based optimization provides a straightforward implementation with very fast convergence, more advanced schemes may be implemented when required.

The automatic differentiation capability of PyMieDiff is therefore the key contribution of this work. In the following example we demonstrate how to use autodiff for optimization of a core-shell particle to implement as closely as possible a predefined Gaussian scattering response.

We start by defining a loss function, using the mean square error (MSE) of the current iteration’s particle responses vs. the target response:

$$L = \frac{1}{n} \sum_{i=1}^n (R_{\text{target},i} - R_i)^2 .$$

Here the  $R_i$  serve as general placeholders for observables calculated by Mie theory, and  $n$  is the total number of cal-

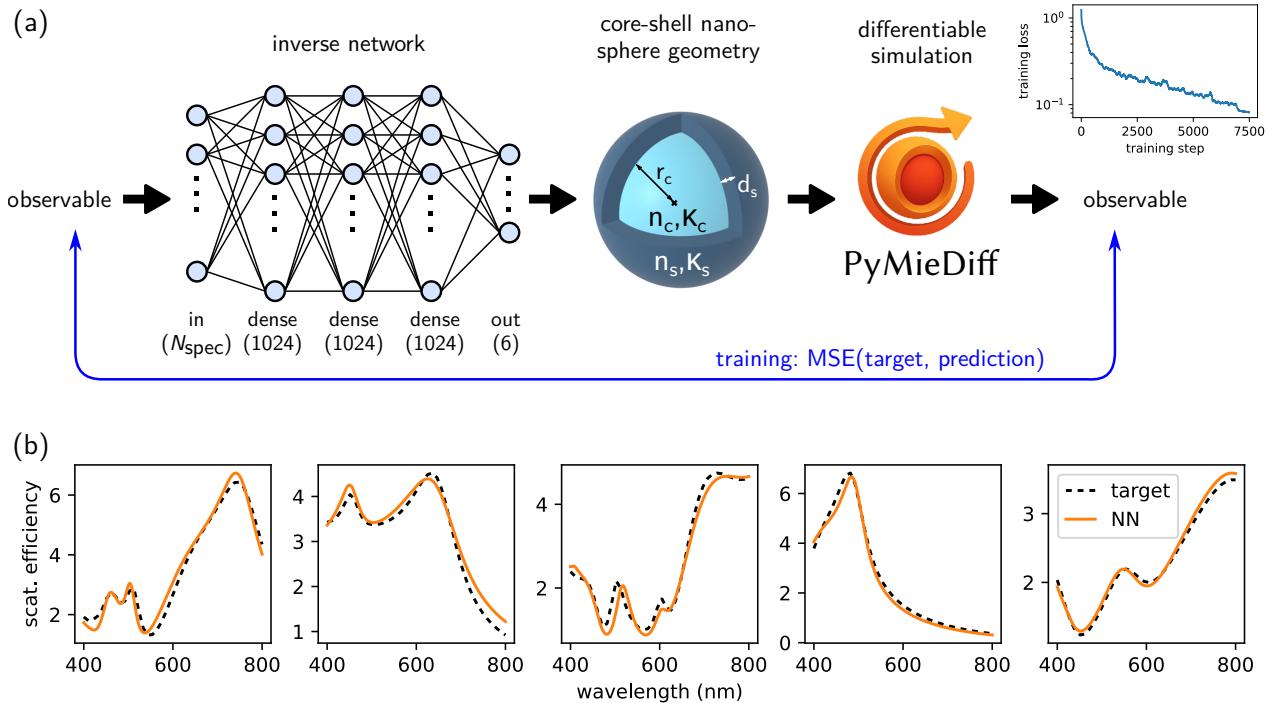


FIG. 5. “Mie-informed” tandem network. (a) Sketch of the training configuration. The design observable (e.g., scattering efficiency, or any other Mie-calculated value) is fed into the “inverse network” (here a simple MLP). The NN result is interpreted as core-shell geometry parameters and fed into PyMieDiff, which calculates analytically the response of the particle. The result of the Mie calculation is compared to the original input data using some loss function (here MSE). The inverse network is trained using backpropagation of the loss through the differentiable Mie solver and then the network itself. The inset shows the training loss for an inverse network trained on scattering efficiency spectra of dielectric particles. (b) Examples of inverse design responses (orange lines) vs. randomly sampled design targets (dashed black lines) from the training dataset .

culations (e.g., multiple wavelengths, angles, particles). The optimization targets are given by  $R_{\text{target},i}$ . Via automatic differentiation, we calculate the partial derivatives of  $L$  with respect to the geometric parameters  $\partial L / \partial \mathcal{G}$ , and update the particles to minimize  $L$ . To avoid getting stuck in local minima, we can perform this optimization on a large number of initial guesses. This optimization procedure is depicted in figure 4a.

As example, we optimize the scattering efficiency  $Q_{\text{sca}}$  of a dielectric core-shell particle, by modifying its core and shell size as well as the material refractive indices. We limit the radii to  $10 - 100$  nm and the complex refractive indexes to dielectric materials with  $n_{\text{real}}$  ranging from 1 to 4.5 and  $n_{\text{imag}}$  from 0.0 to 0.1. To keep the optimization within these restrictions, we optimize normalized values using a sigmoid activation inside the optimization loop.

In this example, the popular Adam optimizer is used.<sup>31</sup> While Adam is ideal for mini-batch based optimization as typically used for neural network training, it performs well also on our optimization task. We demonstrate in the online documentation how to use the L-BFGS optimizer as an alternative.<sup>32</sup>

We perform a batch optimization on 100 random core-shell particles in parallel, which takes roughly 100 ms per

iteration on a typical office CPU. The spectra from the iteratively improved solutions are shown in figure 4b alongside the target spectra. The convergence curve is shown in figure 4c. We found that also higher learning rates are usually stable, allowing convergence within some 20-30 iterations. Yet as optimization is fast, lower, very stable learning rates can be used without major inconvenience.

### C. Core-shell design: Mie informed tandem model

Our toolkit being entirely implemented in Pytorch, it can be directly implemented within any PyTorch written machine learning pipeline. Here we demonstrate how to train a deep learning neural network through a Mie-theory based loss function.

As a simple technical example, we train a design network, capable to predict core-shell geometries that fulfill given target optical properties. This so-called “tandem” model is a commonly used deep learning method for inverse design.<sup>21,22</sup> It works by regularizing the training of the ill-posed inverse problem through a forward predictor, as depicted in figure 5a. Typically, the forward model is a trained neural network that approximates a

physics solver, adding automatic differentiation capabilities. With PyMieDiff as fast, autograd-compatible Mie solver, analytical Mie theory can be used as an error-free forward model through which the loss can be backpropagated to train the inverse network.

We train the Mie-informed tandem network on the same type of particles as in the optimization example, limiting the materials to constant dielectrics and training the model on spectra in the visible light range ( $400 - 800$  nm). A dataset of 20,000 spectra, calculated from randomized particles is used for training the model. The inverse model consists of a fully connected neural network (multilayer perceptron, MLP) with three hidden layers, each containing 1024 neurons, connected by ReLU activation functions. A sigmoid function is used at the output layer to ensure positive predictions within our defined particle limits. Using a basic manually optimized training schedule, with two learning rates (first  $10^{-4}$ , then  $10^{-5}$ ) and incremental increase of the batch size (from 32 up to 256 in four steps), we obtain fast convergence after a few thousand training steps (see inset in Fig. 5a). The training takes a few minutes on an 8-core office CPU.

Once the inverse model is trained, we use scattering spectra from particles that were not in the training set as test-targets and predicting the geometry parameters by the inverse network. A few examples are given in figure 5b, showing good performance of the model. Note that, as the forward model is “perfect” (exact Mie solution), the discrepancy of the solutions with respect to the target spectra comes solely from the inverse network, for which we deliberately use a very simple dense network, chosen to have a short training time of only a few minutes.

#### D. Autodiff multi-scattering: End-to-end metasurface design

As a final example, we demonstrate how PyMieDiff can be combined with the autodiff light scattering simulation toolkit “TorchGDM”,<sup>20</sup> to perform multi-particle scattering simulations. PyMieDiff evaluates the Mie response of one or several core-shell particles, which are then used by torchGDM to generate a structure model and calculate the optical response of an ensemble of many scatterers, where all optical interactions between the particles are taken into account. Since both tools fully support PyTorch automatic differentiation, gradient optimization can be used to iteratively optimize the positions and core/shell size parameters for any target observable. This is illustrated in figure 6a.

We demonstrate this capability on a simple toy problem, not meant to be relevant for an actual application. We calculate the field intensity enhancement at a target focal position, to optimize a diffractive lens made of silicon-gold core-shell particles and optimize for the core and shell radii and the particle positions on the XY plane.

Starting from the same initial conditions ( $10 \times 10$  identical 90 nm/15 nm silicon/gold core/shell spheres on a regular grid), we compare two scenarios: (1) Optimization of the positions of many identical particles, illustrated in figure 6b, and (2) optimization of particle positions and each particle’s core and shell size, illustrated in figure 6c. As expected, we obtain a higher focal intensity if more degrees of freedom are available (individual particle sizes).

The multi-scattering capability may be useful for instance in the design of bottom-up metasurfaces or for fitting optical scattering data from dense, spherical solutions, where homogenization models fail.

## V. LIMITATIONS AND PERSPECTIVES

Here we list current limitations of PyMieDiff and possible future extensions:

- PyMieDiff is currently limited to homogeneous spheres and core-shell particles. In the future we may implement multi-shell spheres as well as 2D Mie theory for infinite cylinders. The latter are more challenging to implement with pure PyTorch autodiff support, as 2D Mie theory requires cylindrical Bessel and Hankel functions, for which it is technically more difficult to implement efficient recurrences.
- The recurrences currently require to be performed at double precision, which significantly slows down the code on consumer grade GPUs. In the future, more stable algorithms may be implemented such as logarithmic derivatives.<sup>33,34</sup>
- Vector spherical harmonics are currently implemented only for order  $l = 1$ , which limits the field calculations to particles with spherical symmetry. We plan to add pure pytorch implementations of general vector spherical harmonics that could be used for field calculations of general T-matrices. A possible route could be to expand on NVIDIA’s recent work about spherical Fourier neural operators.<sup>35</sup>
- The current recurrences can become unstable in cases where many expansion orders are contributing to the response, for instance with very large particles or plasmonic/dielectric interfaces in larger particles. Be aware of these limitations. This could be mitigated in the future by implementing more stable algorithms, such as logarithmic derivatives recurrences.<sup>2</sup>

## VI. CONCLUSIONS

In this work, we present PyMieDiff, an implementation of Mie scattering for core-shell particles fully built

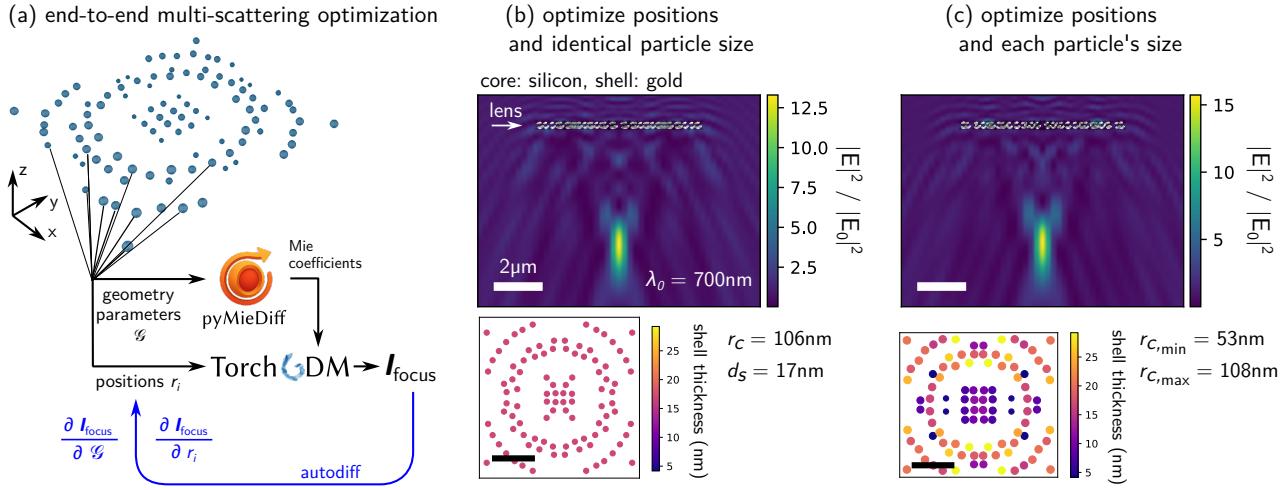


FIG. 6. Combining PyMieDiff with torchGDM<sup>20</sup> for end-to-end multi-scattering autograd calculations. (a) Sketch of the gradient calculation scheme. PyMieDiff is used to calculate differentiable Mie coefficients, which are in turn used to create TorchGDM structure models for multi-particle scattering simulations. Gradients can then be obtained with respect to all input parameters, including particle geometry and material properties (represented by  $\mathcal{G}$ ) and particle positions ( $r_i$ ). (b) Optimization of a diffractive lens, composed of identical particles. Positions as well as the size parameters of the particle are optimized. Top: field intensity enhancement in side view, bottom: top view of the geometry. (c) Same as (b), but now each particle's core and shell sizes are optimized individually. Core sizes are indicated by the circle size, shell size by the color code. All scale bars are  $2 \mu\text{m}$ .

in the automatic differentiation framework PyTorch. It is available as open source software at <https://github.com/UoS-Integrated-Nanophotonics-group/MieDiff>. The package enables gradient-based optimisation and hybrid physics-informed deep learning models. It can be combined with other PyTorch-based toolkits, such as torchGDM, to perform end-to-end differentiable multi-particle scattering simulations. The toolkit was designed with flexibility and performance in mind, offering both, a SciPy-wrapped interface and a native PyTorch implementation with GPU support.

We demonstrate the capabilities by several examples of inverse design problems, including the iterative reconstruction of core-shell particle geometries from a target scattering spectrum, neural network training through analytical Mie calculations, and the gradient-based design of a diffractive lens made of core-shell spheres, for which PyMieDiff is combined with the multi-particle scattering toolkit “TorchGDM”.

At the time of the submission, we became aware of a manuscript developing similar ideas.<sup>7</sup> This underpins the timeliness and importance of the differentiable formulation of algorithms that solve multiple-scattering problems, a key requirement for inverse design of photonic nanostructures.

## ACKNOWLEDGMENTS

OJ acknowledges support by EPSRC through a Ph.D. studentship. OLM acknowledges support by EPSRC

through Grant No. EP/W024683/1. PRW acknowledges financial support by the French Agence Nationale de la Recherche (ANR) under grants ANR-22-CE24-0002 (project NAINOS) and ANR-23-CE09-0011 (project AIM). SDL acknowledges financial support under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call for tender No. 1409 published on 14/09/2022 by the Italian Ministry of University and Research (MUR), funded by the European Union – NextGenerationEU – Project Title MINAS - CUP B53D23028420001 - Grant Assignment Decree No. 1380 adopted on 01/09/2023 by the Italian Ministry of University and Research (MUR).

## DATA AVAILABILITY

The supporting data used in this work are openly available from the University of Southampton repository at <https://doi.org/10.5258/SOTON/D3776>

## APPENDIX

### A. Example code

All examples shown in this work, further examples, as well as the full technical documentation of the package are available online at <https://uos-integrated-nanophotonics-group>.

[github.io/MieDiff/index.html](https://github.io/MieDiff/index.html).

## B. Vectorization conventions

PyMieDiff routines are implemented along the following conventions:

Mie orders: The (Mie) order  $n$  is passed as an integer specifying the maximum order. Then, all PyMieDiff routines (special functions, Mie coefficient routines, etc...) calculate all orders up to the maximum order. An additional, order dimension is added as the first dimension to the shape of the passed argument(s).

Vectorization dimensions: Internal vectorization conventions are the following:

- First dim.: Order / Mie order  $n$  (an additional, first dimension is added upon function calls)
- Second dim.: Number of particles  $N_{\text{particles}}$
- Third dim.: Number of wavenumbers  $k_0$ :  $N_{k0}$
- Fourth dim.: Number of angles or number of positions (angular scattering / fields)

Further dimensions (e.g., for field components) can be added after these four main dimensions and will be broadcasted automatically.

## C. Spherical Bessel recurrences

For comparison and testing, we provide an alternative PyTorch-based scheme to compute the spherical Bessel functions  $j_n$  using a continued fraction calculation.<sup>36</sup> Using the Riccati–Bessel form  $\psi_n(z) = z j_n(z)$  the three-term recurrence reads

$$\psi_{n+1}(z) = \frac{2n+1}{z} \psi_n(z) - \psi_{n-1}(z), \quad (4)$$

and defining the ratio  $A_n(z) = \psi_n(z)/\psi_{n-1}(z)$  gives the continued-fraction relation for stable downward ratios:

$$A_n(z) = \frac{1}{\frac{2n+1}{z} - \frac{1}{\frac{2n+3}{z} - \frac{1}{\ddots}}}. \quad (5)$$

We truncate the continued fraction at a sufficiently large  $n_{\text{max}}$ ,<sup>24</sup> compute the  $A_n$  by backward recurrence, reconstruct  $\psi_n$  (and hence  $j_n = \psi_n/z$ ) up to a multiplicative constant, and fix the global scale using  $j_0(z) = \sin z/z$ . Both implementations for  $j_n$  are similarly stable and can be selected with the argument `which_jn` (“ratios” or “recurrence”).

## D. Spherical Hankel functions and derivatives of spherical Bessel functions

Spherical Hankel functions can be obtained from spherical Bessel functions of first and second kind:

$$h_n^1(z) = j_n(z) - i y_n(z). \quad (6)$$

Derivatives of spherical Bessel functions can be obtained by either of the following recurrences:

$$\begin{aligned} f'_n(z) &= f_{n-1}(z) - \frac{n+1}{z} f_n(z), \\ f'_n(z) &= -f_{n+1}(z) + \frac{n}{z} f_n(z), \end{aligned} \quad (7)$$

where  $f_n(z)$  is any spherical bessel function.

Note: In practice, to avoid negative orders, derivatives for  $n > 0$  are calculated using the first relation in Eqs. (7), while the derivative for  $n = 0$  is obtained from the second relation:

$$f'_0(z) = -f_1(z). \quad (8)$$

## E. Vector spherical harmonics

The vector spherical harmonics  $\mathbf{N}_{lm}^{(k)}$  and  $\mathbf{M}_{lm}^{(k)}$  can be used to describe the near fields of Mie scattering.<sup>37</sup> For spherical symmetry, we need only the  $l = 1$  odd and even order harmonics, which are given by:<sup>2</sup>

$$\mathbf{N}_{o1n}^{(k)} = \begin{bmatrix} \frac{1}{\rho} \sin \varphi n(n+1) \sin \theta \pi_n(\cos \theta) z_n(\rho) \\ \frac{1}{\rho} \sin \varphi \tau_n(\cos \theta) [\rho z_n(\rho)]' \\ \frac{1}{\rho} \cos \varphi \pi_n(\cos \theta) [\rho z_n(\rho)]' \end{bmatrix}, \quad (9)$$

$$\mathbf{N}_{e1n}^{(k)} = \begin{bmatrix} \frac{1}{\rho} \cos \varphi n(n+1) \sin \theta \pi_n(\cos \theta) z_n(\rho) \\ \frac{1}{\rho} \cos \varphi \tau_n(\cos \theta) [\rho z_n(\rho)]' \\ -\frac{1}{\rho} \sin \varphi \pi_n(\cos \theta) [\rho z_n(\rho)]' \end{bmatrix}, \quad (10)$$

$$\mathbf{M}_{o1n}^{(k)} = \begin{bmatrix} 0 \\ \cos \varphi \pi(\cos \theta) z_n(\rho) \\ -\sin \varphi \tau(\cos \theta) z_n(\rho) \end{bmatrix}, \quad (11)$$

$$\mathbf{M}_{e1n}^{(k)} = \begin{bmatrix} 0 \\ -\sin \varphi \pi(\cos \theta) z_n(\rho) \\ -\cos \varphi \tau(\cos \theta) z_n(\rho) \end{bmatrix}. \quad (12)$$

Here,  $m$  has become the Mie order  $n$ ,  $\rho = kr$ ,  $\varphi$  is the azimuthal and  $\theta$  the polar angle. The spherical Bessel functions of order  $n$  are denoted by  $z_n$ . The kind of  $z_n$  is indicated by the superscript  $(k)$  of  $\mathbf{N}$  and  $\mathbf{M}$ :  $^{(1)}$  denotes the first kind,  $j_n(kr)$ ,  $^{(2)}$  the second kind,  $y_n(kr)$  and  $^{(3)}$  denotes spherical Hankel functions  $h_n^{(1)}(kr)$ .

## F. Mie coefficients

The Mie coefficients for a core-shell spherical particle are given by<sup>2</sup>

$$a_n = \frac{\psi_n(y)[\psi'_n(m_2y) - A_n \chi'_n(m_2y)] - m_2 \psi'_n(y)[\psi_n(m_2y) - A_n \chi_n(m_2y)]}{\xi_n(y)[\psi'_n(m_2y) - A_n \chi'_n(m_2y)] - m_2 \xi'_n(y)[\psi_n(m_2y) - A_n \chi_n(m_2y)]} \quad (13)$$

$$b_n = \frac{m_2 \psi_n(y)[\psi'_n(m_2y) - B_n \chi'_n(m_2y)] - \psi'_n(y)[\psi_n(m_2y) - B_n \chi_n(m_2y)]}{m_2 \xi_n(y)[\psi'_n(m_2y) - B_n \chi'_n(m_2y)] - \xi'_n(y)[\psi_n(m_2y) - B_n \chi_n(m_2y)]}, \quad (14)$$

where  $m_1 = n_1/n_{\text{env}}$  and  $m_2 = n_2/n_{\text{env}}$  are the relative refractive indices for the core and shell, respectively and  $x = ka$  and  $y = kb$  are the scale factors with the wavevector in the host medium  $k = k_0 n_{\text{env}}$ . The Riccati-Bessel functions are given as  $\psi_n(z) = z j_n(z)$ ,  $\xi_n(z) = zh_n^{(1)}(z)$  and  $\chi_n(z) = zy_n(z)$ .

$$A_n = \frac{m_2 \psi_n(m_2x) \psi'_n(m_1x) - m_1 \psi'_n(m_2x) \psi_n(m_1x)}{m_2 \chi_n(m_2x) \psi'_n(m_1x) - m_1 \chi'_n(m_2x) \psi_n(m_1x)}, \quad (15)$$

and

$$B_n = \frac{m_2 \psi_n(m_1x) \psi'_n(m_2x) - m_1 \psi_n(m_2x) \psi'_n(m_1x)}{m_2 \psi_n(m_1x) \chi'_n(m_2x) - m_1 \psi'_n(m_1x) \chi_n(m_2x)}. \quad (16)$$

The Mie coefficients for internal fields can be found in literature.<sup>2</sup>

### G. Mie far-field observables

The scattering, absorption and extinction efficiencies can be obtained from the Mie coefficients as,

$$Q_{\text{ext}} = \frac{1}{2\pi r_s} \frac{2\pi}{k^2} \sum_{n=1}^{\infty} (2n+1) \operatorname{Re}\{a_n + b_n\}, \quad (17)$$

$$Q_{\text{sca}} = \frac{1}{2\pi r_s} \frac{2\pi}{k^2} \sum_{n=1}^{\infty} (2n+1) (|a_n|^2 + |b_n|^2), \quad (18)$$

$$Q_{\text{abs}} = Q_{\text{ext}} - Q_{\text{sca}}. \quad (19)$$

The cross sections are given by the efficiencies times the geometric cross section of the particle  $\sigma_{\text{geo}} = \pi r^2$ .

The angular dependent scattering i.e. the scattered irradiance per unit incident irradiance for perpendicularly and parallel polarised illumination light (with respect to the scattering plane), as well as for unpolarised light, are given by,

$$i_{\text{par}} = |S_2|^2, \quad i_{\text{per}} = |S_1|^2, \quad i_{\text{unp}} = \frac{i_{\text{par}} + i_{\text{per}}}{2}. \quad (20)$$

Where,

$$S_1(\theta) = \sum_{n=1}^{n_{\max}} \frac{2n+1}{n(n+1)} (a_n \pi_n(\mu) + b_n \tau_n(\mu)), \quad (21)$$

$$S_2(\theta) = \sum_{n=1}^{n_{\max}} \frac{2n+1}{n(n+1)} (a_n \tau_n(\mu) + b_n \pi_n(\mu)), \quad (22)$$

where  $\mu = \cos \theta$ . The angular functions  $\pi_n(\cos \theta)$  and  $\tau_n(\cos \theta)$  are defined from the associated Legendre polynomials  $P_n^1(\cos \theta)$ .<sup>2</sup> These satisfy the recurrence relations

$$\pi_0(\mu) = 0, \quad (23)$$

$$\pi_1(\mu) = 1, \quad (24)$$

$$\pi_{n+1}(\mu) = \frac{2n+1}{n} \mu \pi_n(\mu) - \frac{n+1}{n} \pi_{n-1}(\mu), \quad (25)$$

$$\tau_n(\mu) = n\mu \pi_n(\mu) - (n+1) \pi_{n-1}(\mu). \quad (26)$$

### H. Near-field observables

The near-fields can be found by matching the continuity conditions at each spherical interface, which yields following formulae:<sup>2</sup>

$$\begin{aligned} \mathbf{E}_i &= \sum_{n=1}^{\infty} E_n \left( \mathbf{M}_{o1n}^{(1)} - i \mathbf{N}_{e1n}^{(1)} \right), \\ \mathbf{H}_i &= \frac{-k}{\omega \mu} \sum_{n=1}^{\infty} E_n \left( \mathbf{M}_{e1n}^{(1)} + i \mathbf{N}_{o1n}^{(1)} \right), \\ \mathbf{E}_1 &= \sum_{n=1}^{\infty} E_n \left( c_n \mathbf{M}_{o1n}^{(1)} - id_n \mathbf{N}_{e1n}^{(1)} \right), \\ \mathbf{H}_1 &= \frac{-k_1}{\omega \mu_1} \sum_{n=1}^{\infty} E_n \left( d_n \mathbf{M}_{e1n}^{(1)} + ic_n \mathbf{N}_{o1n}^{(1)} \right), \\ \mathbf{E}_2 &= \sum_{n=1}^{\infty} E_n \left( f_n \mathbf{M}_{o1n}^{(1)} - ig_n \mathbf{N}_{e1n}^{(1)} + v_n \mathbf{M}_{o1n}^{(2)} - iw_n \mathbf{N}_{e1n}^{(2)} \right), \\ \mathbf{H}_2 &= \frac{-k_2}{\omega \mu_2} \sum_{n=1}^{\infty} E_n \left( g_n \mathbf{M}_{e1n}^{(1)} + if_n \mathbf{N}_{o1n}^{(1)} + w_n \mathbf{M}_{e1n}^{(2)} + iv_n \mathbf{N}_{o1n}^{(2)} \right), \\ \mathbf{E}_3 &= \sum_{n=1}^{\infty} E_n \left( ia_n \mathbf{N}_{e1n}^{(3)} - b_n \mathbf{M}_{o1n}^{(3)} \right), \\ \mathbf{H}_3 &= \frac{k}{\omega \mu} \sum_{n=1}^{\infty} E_n \left( ib_n \mathbf{N}_{o1n}^{(3)} + a_n \mathbf{M}_{e1n}^{(3)} \right). \end{aligned}$$

The subscripts indicate the region and type of field. “ $i$ ”: incident field in the background medium, “ $1$ ”: total field in the core, “ $2$ ”: total field in the shell medium, and “ $3$ ”: scattered field in the background medium. The total fields in the background medium are given by  $\mathbf{E}_{\text{tot}} = \mathbf{E}_i + \mathbf{E}_3$  and  $\mathbf{H}_{\text{tot}} = \mathbf{H}_i + \mathbf{H}_3$ .

## I. Second derivatives of the Spherical Bessel functions

To add automatic differentiation capabilities to the SciPy interface, we add custom PyTorch autodiff classes with manually implemented derivatives. To allow autograd for the first order derivatives of spherical Bessel functions that occur in the Mie coefficients, we need to implement analytic formulae for their second order derivatives. Starting from the recurrence relation Eq. (7) we substitute  $n = n + 1$  into the first equation to get,

$$f'_{n+1}(z) = f_n(z) - \frac{n+2}{z} f_{n+1}(z) \quad (27)$$

Taking the derivative of the second equation,

$$\frac{d^2}{dz^2} f_n(z) = -\frac{d}{dz} f_{n+1}(z) + n \frac{d}{dz} \left( \frac{f_n(z)}{z} \right), \quad (28)$$

$$f''_n(z) = -f'_{n+1}(z) + n \left( \frac{f'_n(z)z - f_n(z)}{z^2} \right). \quad (29)$$

Rearrange this to,

$$z^2 f''_n(z) = -z^2 f'_{n+1}(z) + nz f'_n(z) - n f_n(z), \quad (30)$$

and then substitute the modified first equation and the second equation to get,

$$z^2 f''_n(z) = -z^2 \left( f_n(z) - \frac{n+2}{z} f_{n+1}(z) \right) + \\ nz \left( -f_{n+1}(z) + \frac{n}{z} f_n(z) \right) - n f_n(z).$$

Rearrange this to get the equation for  $f''_n(z)$ ,

$$z^2 f''_n(z) = f_n(z) (-z^2 + n^2 - n) + f_{n+1}(z) (z(n+2) - nz), \quad (31)$$

$$f''_n(z) = \frac{1}{z^2} [(n^2 - n - z^2) f_n(z) + 2z f_{n+1}(z)]. \quad (32)$$

\* e-mail : [O.Muskens@soton.ac.uk](mailto:O.Muskens@soton.ac.uk)

† e-mail : [pwiecha@laas.fr](mailto:pwiecha@laas.fr)

- <sup>1</sup> Mie, G. Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. *Annalen der Physik* 330, 377–445 (1908).
- <sup>2</sup> Bohren, C. F. & Huffman, D. R. *Absorption and Scattering of Light by Small Particles* (Wiley, 1998).
- <sup>3</sup> Kuhn, L., Repän, T. & Rockstuhl, C. Inverse design of core-shell particles with discrete material classes using neural networks. *Scientific Reports* 12, 19019 (2022).
- <sup>4</sup> Peurifoy, J. et al. Nanophotonic particle simulation and inverse design using artificial neural networks. *Science Advances* 4, eaar4206 (2018).
- <sup>5</sup> So, S., Mun, J. & Rho, J. Simultaneous Inverse Design of Materials and Structures via Deep Learning: Demonstration of Dipole Resonance Engineering Using Core–Shell Nanoparticles. *ACS Applied Materials & Interfaces* 11, 24264–24268 (2019). 1904.02848.
- <sup>6</sup> Estrada-Real, A., Khairah-Walih, A., Urbaszek, B. & Wiecha, P. R. Inverse design with flexible design targets via deep learning: Tailoring of electric and magnetic multipole scattering from nano-spheres. *Photonics and Nanostructures - Fundamentals and Applications* 52, 101066 (2022). 2207.03431.
- <sup>7</sup> Soun, D., Azéma, A., Roach, L., Drisko, G. L. & Wiecha, P. R. Gradient-based optimization of core-shell particles with discrete materials for directional scattering. *Optics Express* 33, 25945–25958 (2025). 2502.13338.
- <sup>8</sup> Li, Y. & Bowler, N. Computation of Mie derivatives. *Applied Optics* 52, 4997–5006 (2013).
- <sup>9</sup> Prahl, S. Miepython: A Python library for Mie scattering calculations (2025).
- <sup>10</sup> de Sivry-Houle, M. P., Godbout, N. & Boudoux, C. PyMieSim: An open-source library for fast and flexible far-field Mie scattering simulations. *Optics Continuum* 2, 520–534 (2023).
- <sup>11</sup> Ladutenko, K., Pal, U., Rivera, A. & Peña-Rodríguez, O. Mie calculation of electromagnetic near-field for a multi-layered sphere. *Computer Physics Communications* 214, 225–230 (2017).
- <sup>12</sup> Wiecha, P. R. Pymiecs - a simple python Mie toolbox · GitLab. <https://gitlab.com/wiechapeter/pymiecs> (2024).
- <sup>13</sup> Wengert, R. E. A simple automatic derivative evaluation program. *Commun. ACM* 7, 463–464 (1964).
- <sup>14</sup> Hughes, T. W., Williamson, I. A. D., Minkov, M. & Fan, S. Forward-Mode Differentiation of Maxwell's Equations. *ACS Photonics* 6, 3010–3016 (2019).
- <sup>15</sup> Minkov, M. et al. Inverse Design of Photonic Crystals through Automatic Differentiation. *ACS Photonics* 7, 1729–1741 (2020).
- <sup>16</sup> Colburn, S. & Majumdar, A. Inverse design and flexible parameterization of meta-optics using algorithmic differentiation. *Communications Physics* 4, 1–11 (2021).
- <sup>17</sup> Wang, C., Chen, N. & Heidrich, W. dO: A Differentiable Engine for Deep Lens Design of Computational Imaging Systems. *IEEE Transactions on Computational Imaging* 8, 905–916 (2022).
- <sup>18</sup> Luce, A., Alaee, R., Knorr, F. & Marquardt, F. Merging automatic differentiation and the adjoint method for photonic inverse design. *Machine Learning: Science and Technology* 5, 025076 (2024).
- <sup>19</sup> Kim, Y. et al. Meent: Differentiable Electromagnetic Simulator for Machine Learning (2024). 2406.12904.
- <sup>20</sup> Ponomareva, S. et al. TorchGDM: A GPU-accelerated Python toolkit for multi-scale electromagnetic scattering with automatic differentiation. *SciPost Physics Codebases* 060 (2025).
- <sup>21</sup> Liu, D., Tan, Y., Khoram, E. & Yu, Z. Training Deep Neural Networks for the Inverse Design of Nanophotonic Structures. *ACS Photonics* 5, 1365–1369 (2018).
- <sup>22</sup> Khairah-Walih, A. et al. A newcomer's guide to deep learning for inverse design in nano-photonics. *Nanophotonics* 12, 4387–4414 (2023). 2307.08618.

- <sup>23</sup> Dinsdale, N. J. et al. Deep learning enabled design of complex transmission matrices for universal optical components. *ACS Photonics* 8, 283–295 (2021). 2009.11810.
- <sup>24</sup> Cai, L.-W. On the computation of spherical Bessel functions of complex arguments. *Computer Physics Communications* 182, 663–668 (2011).
- <sup>25</sup> Polyanskiy, M. N. Refractiveindex.info database of optical constants. *Scientific Data* 11, 94 (2024).
- <sup>26</sup> Jiang, J. & Fan, J. A. Multiobjective and categorical global optimization of photonic structures based on ResNet generative neural networks. *Nanophotonics* 10, 361–369 (2020).
- <sup>27</sup> Gillman, E. & Fiebig, H. R. Accurate recursive generation of spherical Bessel and Neumann functions for a large range of indices. *Computer in Physics* 2, 62–72 (1988).
- <sup>28</sup> Johnson, P. B. & Christy, R. W. Optical Constants of the Noble Metals. *Physical Review B* 6, 4370–4379 (1972).
- <sup>29</sup> Edwards, D. F. Silicon (Si)\*. In Palik, E. D. (ed.) *Handbook of Optical Constants of Solids*, 547–569 (Academic Press, Burlington, 1997).
- <sup>30</sup> Beutel, D., Fernandez-Corbaton, I. & Rockstuhl, C. Treams – a T-matrix-based scattering code for nanophotonics. *Computer Physics Communications* 297, 109076 (2024).
- <sup>31</sup> Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs] (2014). 1412.6980.
- <sup>32</sup> Nocedal, J. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation* 35, 773–782 (1980).
- <sup>33</sup> Majic, M. & Ru, E. C. L. Numerically stable formulation of Mie theory for an emitter close to a sphere. *Applied Optics* 59, 1293–1300 (2020).
- <sup>34</sup> Zhang, J. A modified recursive transfer matrix algorithm for radiation and scattering computation of multilayer spheres (2024). 2409.10877.
- <sup>35</sup> Bonev, B. et al. Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere (2023). 2306.03838.
- <sup>36</sup> Lentz, W. J. Continued fraction calculation of spherical Bessel functions. *Computer in Physics* 4, 403–407 (1990).
- <sup>37</sup> Asadova, N. et al. T-matrix representation of optical scattering response: Suggestion for a data format. *Journal of Quantitative Spectroscopy and Radiative Transfer* 333, 109310 (2025).