Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Inverse Design of Multilayer Thin-Film Coatings with Neural Networks and Differentiable Transfer-Matrix Optics

Gurman Singh Dhaliwal
Indian Institute of Technology Roorkee
Mayank Goswami
mayankfph@iitr.ac.in

Indian Institute of Technology Roorkee https://orcid.org/0000-0001-8829-9980

Research Article

Keywords:

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11712099/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Inverse Design of Multilayer Thin-Film Coatings with Neural Networks and Differentiable Transfer-Matrix Optics

A physics-informed machine-learning approach: from a target reflectance spectrum to layer thicknesses

Gurman Singh Dhaliwal and Mayank Goswami [1, 2, #]
[1] Department of Physics, IIT Roorkee
[2] *Mehta Family School of Data Science and Artificial Intelligence*, IIT Roorkee, Roorkee, Uttarakhand-247667, India.
[#] mayank.goswami@ph.iitr.ac.in

Abstract. A thin-film coating is a stack of nanometre-thick layers whose reflectance spectrum is set by interference. Computing the spectrum from the thicknesses (forward problem) is easy with the transfer-matrix method (TMM); finding thicknesses that give a *desired* spectrum (inverse problem) is hard because many different stacks give similar spectra. We build a 200 000-sample dataset of 8-layer TiO_2/SiO_2 stacks on glass (400–800 nm), train a neural network that maps spectrum to thicknesses, show that the straightforward network fails because of non-uniqueness, fix this with a *tandem* network whose loss is computed through a differentiable TMM, and finally polish each prediction with an L-BFGS-B optimiser using an analytic gradient. On 300 unseen test spectra the mean spectral error falls from 0.114 (naive network) to 0.059 (tandem) to 0.013 (naive + refinement), i.e. about 9 lower than the naive network, at 0.05 s per target and with 92 % of targets matched to within 0.05 mean error.

1 Introduction

Almost every optical surface we meet is coated: spectacle lenses, camera lenses, solar cells, mirrors in lasers, architectural glass. A coating is a stack of thin dielectric layers (tens to hundreds of nanometres) and works by *interference*: light reflected at every interface adds up with a phase set by the layer thickness and refractive index. Choosing the thicknesses allows us to suppress reflection (anti-reflection coating), enhance it (mirror), or carve out a narrow band (filter).

The **forward problem** (thicknesses → spectrum) is solved exactly by the transfer-matrix method in microseconds. The **inverse problem** (spectrum → thicknesses) is what a coating engineer actually needs and has no closed-form solution. This project studies how machine learning can solve it, and, importantly, why the obvious machine-learning approach fails and how physics repairs it.

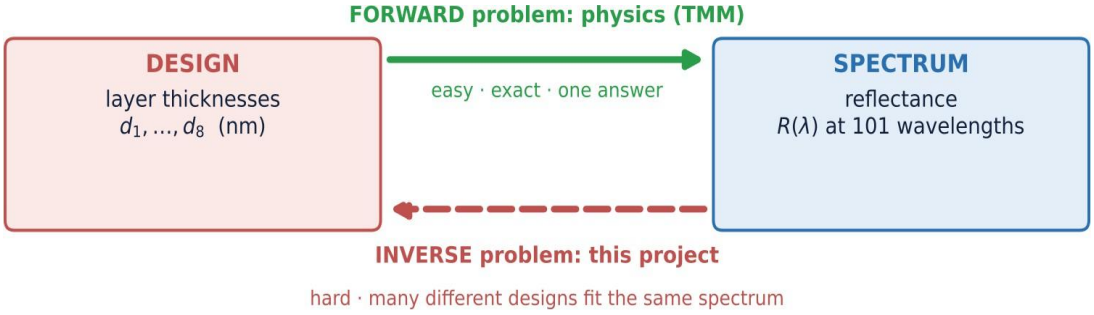


Figure 1: Forward problem (easy, unique) versus inverse problem (hard, one-to-many).

2 Problem Objectives and Need

2.1 Why the problem needs solving

- Coatings are everywhere and each application needs a different spectrum; designing by trial and error is slow.
- Classical design uses numerical optimisation from a starting guess; the landscape is full of local minima (Fig. 11), so results depend on the starting point and many restarts are needed.
- A trained network can propose a good design in milliseconds, which can be used directly or as a starting point for a local optimiser.

2.2 Objectives

1. Implement and validate a physically correct TMM forward model.
2. Generate a large, reproducible dataset of (spectrum, thicknesses) pairs.
3. Train a network for the inverse map and *measure its quality in spectrum space*, not thickness space.
4. Show and explain the non-uniqueness failure; fix it with a tandem (physics-in-the-loop) network.
5. Combine the network with gradient refinement and benchmark against classical random-restart optimization.

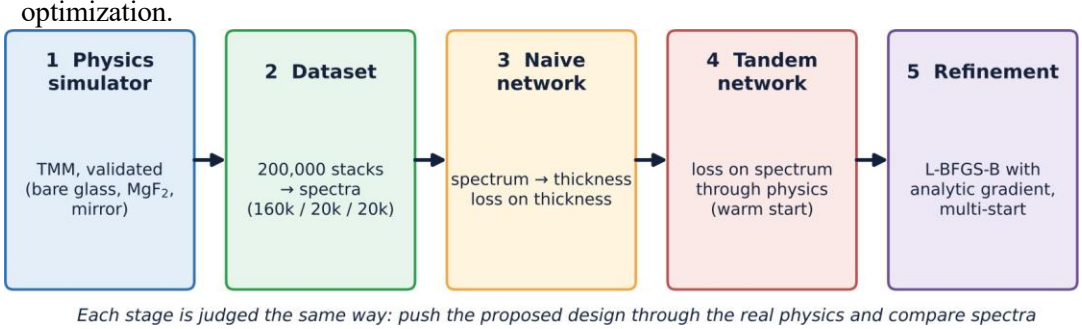


Figure 2: Overall pipeline of the project.

3 Theory

3.1 Interference in a thin film

Light hitting a film of index n_1 and thickness d is partly reflected at the top surface and partly at the bottom surface (Fig. 3). The two reflected waves differ in path by $2n_1 d\cos\theta_1$, giving the phase thickness

$$\delta = \frac{2\pi}{\lambda} n_1 d\cos\theta_1. \tag{1}$$

At an interface between indices n_a and n_b (normal incidence) the Fresnel amplitude coefficient is $r = \frac{n_a - n_b}{n_a + n_b}$ and the reflectance $R = |r|^2$. A bare glass surface ($n = 1.52$) therefore reflects $R = \left(\frac{0.52}{2.52}\right)^2 \approx 4.3\%$, which our code reproduces (0.0426).

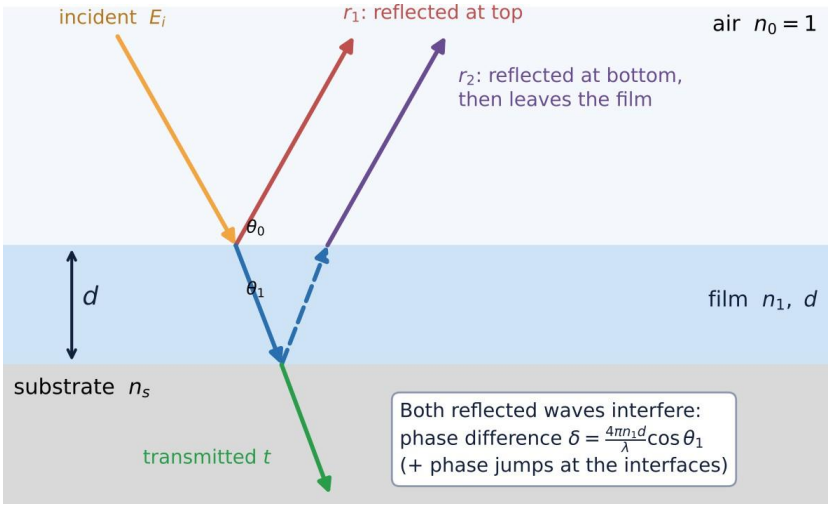


Figure 3: Two-beam interference in a single film: the reflected beams add constructively or destructively depending on δ.

3.2 Single layer: the quarter-wave anti-reflection coating

For a film on a substrate the reflectance depends on δ (Fig. 4). When $n_1 d = \lambda_0/4$ ($\delta = \pi/2$) the two reflected beams are in antiphase; they cancel completely if their amplitudes match, which requires

$$n_1 = \sqrt{n_0 n_s}. \tag{2}$$

For air on glass this is≈ 1.23; MgF_2 ($n = 1.38$) is the practical choice and lowers R from 4.3 % to 1.3 % at the design wavelength (our code: 0.0126, matching the analytic value).

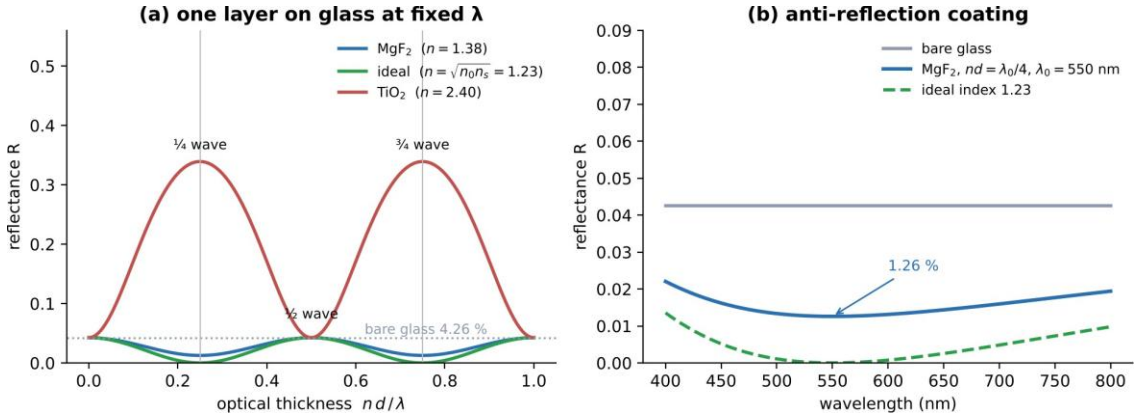


Figure 4: Single-layer coating: reflectance versus wavelength and the quarter-wave condition.

3.3 Multilayers: the Bragg (quarter-wave) mirror

Alternating high- and low-index quarter-wave layers make every reflected beam add in phase. For N layers (high index n_H outermost) and substrate n_s,

$$R = \left[\frac{1 - \frac{n_s}{n_0}\left(\frac{n_L}{n_H}\right)^{2N}}{1 + \frac{n_s}{n_0}\left(\frac{n_L}{n_H}\right)^{2N}}\right]^2 \quad (N \text{ even}), \tag{3}$$

which for 8 layers of TiO_2/SiO_2 on glass gives $R \approx 0.952$, again reproduced numerically. The high-

reflectance band is centred on λ_0 and widens with the index contrast n_H/n_L.

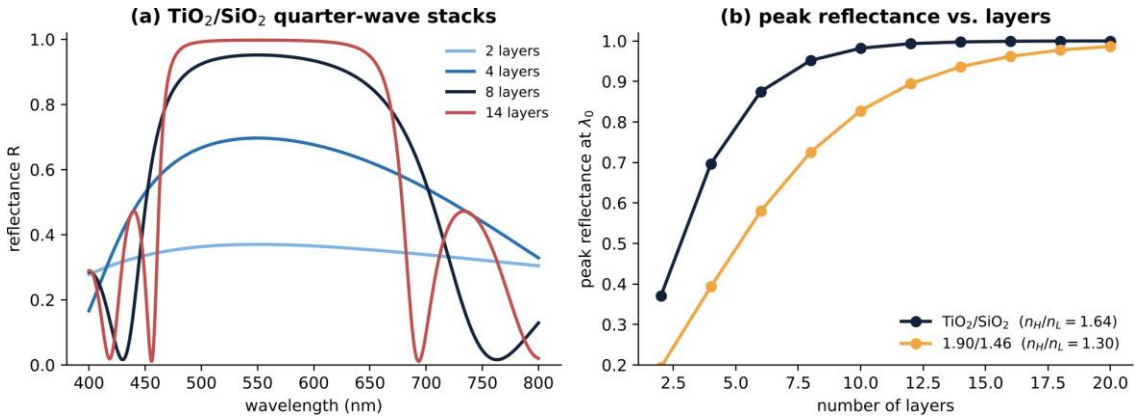


Figure 5: Quarter-wave stack: reflectance grows with the number of layer pairs; the stop band widens with index contrast.

3.4 Transfer-matrix method (TMM)

For a stack of N layers the interface bookkeeping is done with 2×2 matrices. For layer j with index n_j, thickness d_j and optical admittance η_j ($= n_j$ at normal incidence),

$$M_j = \begin{bmatrix} \cos\delta_j & -\dfrac{i\sin\delta_j}{\eta_j} \\ -i\eta_j \sin\delta_j & \cos\delta_j \end{bmatrix}, \qquad \delta_j = \frac{2\pi n_j d_j}{\lambda}. \tag{4}$$

The characteristic matrix of the whole stack is the ordered product $M = M_1 M_2 \cdots M_N$, and

$$\begin{bmatrix} B \\ C \end{bmatrix} = M \begin{bmatrix} 1 \\ \eta_s \end{bmatrix}, \quad r = \frac{\eta_0 B - C}{\eta_0 B + C}, \quad R = |r|^2, \quad T = \frac{4\eta_0 \eta_s}{|\eta_0 B + C|^2}. \tag{5}$$

For lossless layers $R + T = 1$. The method is exact for plane waves in homogeneous, isotropic layers and is evaluated for every wavelength separately.

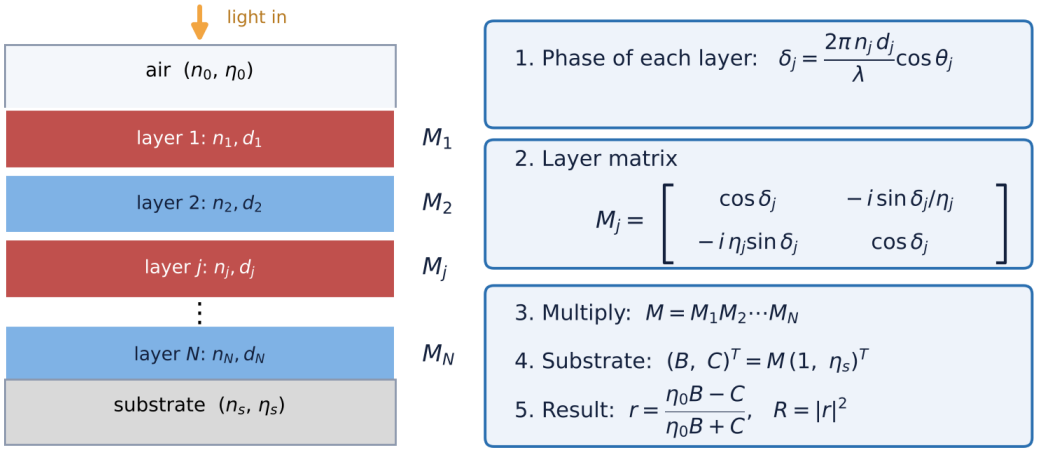


Figure 6: The transfer-matrix method: each layer contributes one 2×2 matrix; the product gives the stack response.

3.5 Oblique incidence and scope

For incidence angle θ_0, Snell's law gives θ_j in each layer and the admittances become $\eta_j = n_j \cos\theta_j$ (s-polarisation) or $n_j / \cos\theta_j$ (p-polarisation); δ_j contains $\cos\theta_j$. Our dataset uses **normal incidence only** and **non-dispersive, lossless** materials (constant n), which keeps the learning problem clean; extending to angles and dispersion is listed under future work.

4 Previous Work

- **Classical design:** Macleod's textbook and Born & Wolf formalise TMM and analytical designs (AR, mirror, filters) [1, 2]. Numerical refinement and "needle" optimisation insert new layers where they most reduce the merit function [3]; results depend on the starting design.
- **Neural networks for photonics:** Peurifoy et al. trained networks for scattering from multilayer nanoparticles and noted inverse-design difficulty [4]; Malkiel et al. designed plasmonic nanostructures with a bidirectional network [5].
- **Tandem networks:** Liu et al. showed that when the inverse map is one-to-many, a network trained directly on the design variables fails to converge, and cascading the inverse network with a pre-trained forward network solves this [6]. Our tandem network follows this idea, but uses an exact, differentiable TMM instead of a learned forward model.

- **Physics-driven optimisation:** Jiang and Fan combine a generator network with differentiable electromagnetic solvers [7].



Figure 7: Where this project sits: classical optimisation, learned inverse networks, and physics-in-the-loop training.

5 Method

5.1 Design space

Eight alternating layers of TiO_2 ($n = 2.4$) and SiO_2 ($n = 1.46$) on a glass substrate ($n = 1.52$) in air. Reflectance is computed at 101 wavelengths from 400 to 800 nm (4 nm step, fixed for the whole project so that the network input has a fixed meaning). Thicknesses lie in $[20, 200]$ nm. The material order is also fixed; otherwise the same input spectrum would correspond to different designs for different orderings.

5.2 Dataset generation

We generated **200 000** samples with seed 42 using four samplers so that the data contains both structured and generic designs (Fig. 8): uniformly random (40 %), quarter-wave around a random centre wavelength with ±5 % jitter (25 %), chirped stacks with smoothly varying thickness (20 %), and perturbed quarter-wave stacks with ±25 % jitter (15 %). Data was split 160 000 / 20 000 / 20 000 (train / validation / test); thicknesses are scaled by 1/200 and saved in `train.npz`, `val.npz`, `test.npz`, plus a 1 000-row `.csv`/`.xlsx` sample for inspection.

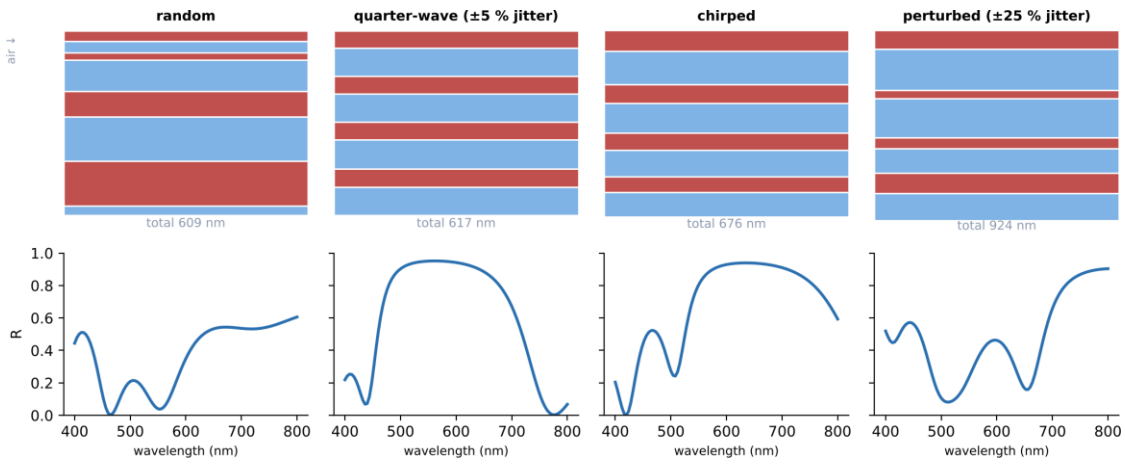


Figure 8: The four sampler types and an example of each.

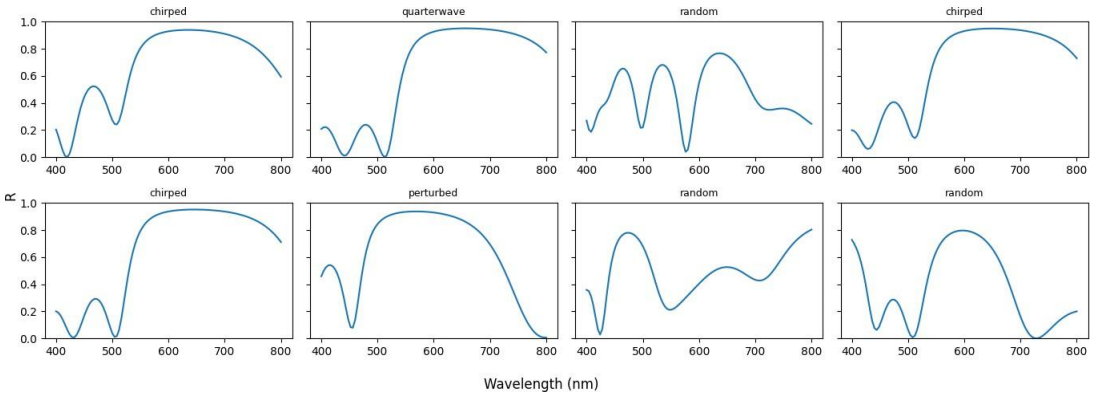


Figure 9: Random samples from the generated dataset (notebook output).

5.3 Why the naive network fails: non-uniqueness

Fig. 10 shows two real stacks from our data with very different thicknesses and almost identical spectra. A network trained with a thickness loss is then pulled towards both answers and outputs their *average*, which is generally not a valid design (Fig. 19).

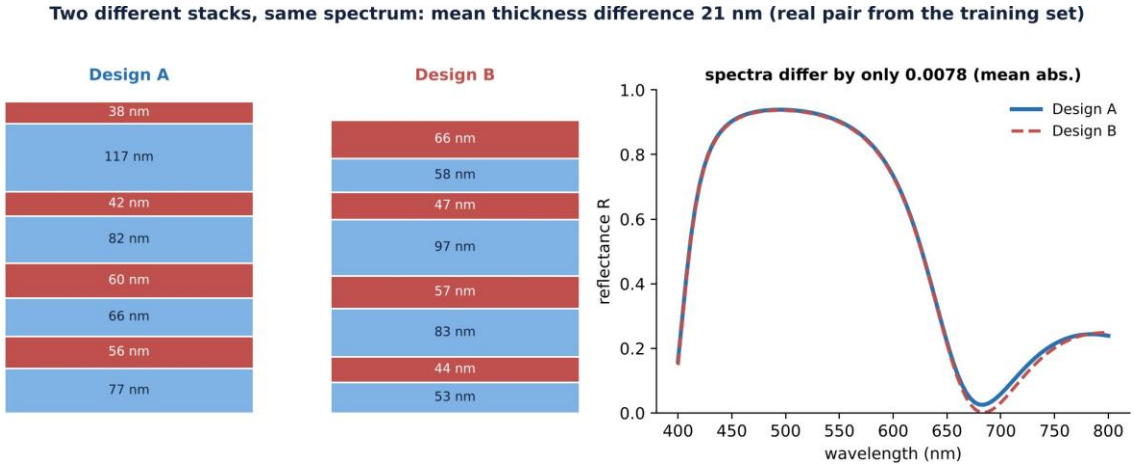


Figure 10: Two different stacks, nearly the same spectrum (real pair from the dataset).

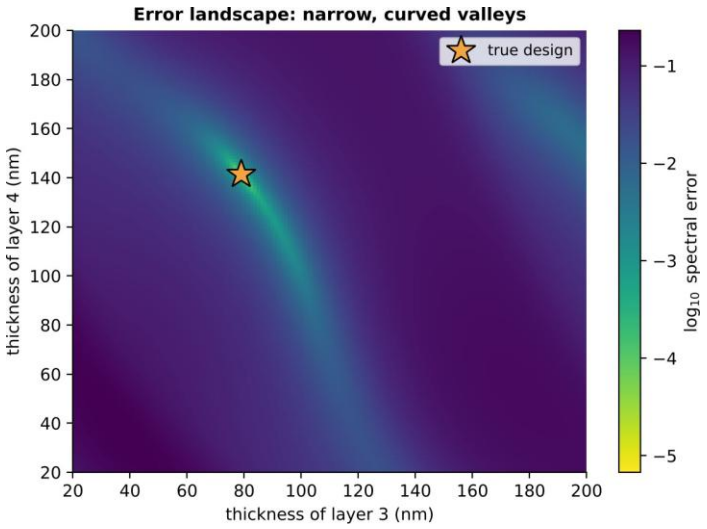


Figure 11: Cartoon of the error landscape: many local minima, so a single starting point is unreliable.

5.4 Naive inverse network

A fully connected network 101→ 512→512 →256 8→(ReLU hidden layers, sigmoid output, 448 264 parameters) maps the spectrum to scaled thicknesses (Fig. 12). It is trained with Adam (lr 10^{-3}, batch 512, ReduceLROnPlateau, early stopping) using mean-squared error on thickness; 40 epochs take 24 s on a Colab GPU.

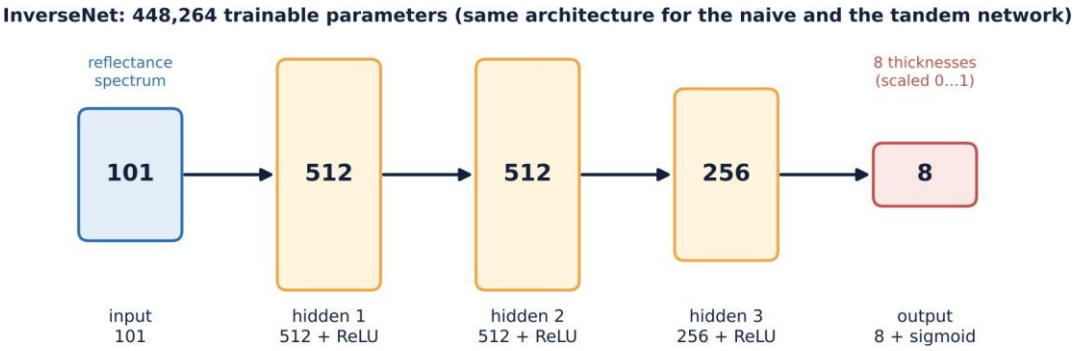


Figure 12: Network architecture.

```python
class InverseNet(nn.Module):
    """spectrum (101 numbers) -> thicknesses (8 numbers, scaled 0..1)"""
    def __init__(self, n_in=N_LAM, n_out=N_LAYERS, hidden=(512, 512, 256)):
        super().__init__()
        layers, prev = [], n_in
        for h in hidden:
            layers += [nn.Linear(prev, h), nn.ReLU()]
            prev = h
        layers += [nn.Linear(prev, n_out), nn.Sigmoid()]
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)

model = InverseNet().to(device)
```

```python
for epoch in range(1, EPOCHS + 1):
    model.train()
    perm = torch.randperm(len(Xtr_d), device=device)
    total = 0.0
    for i in range(0, len(perm), BATCH):
        idx = perm[i:i + BATCH]
        opt.zero_grad()
        loss = loss_fn(model(Xtr_d[idx]), ytr_d[idx])
        loss.backward()
        opt.step()
        total += loss.detach() * len(idx)
    tr_loss = (total / len(perm)).item()

    model.eval()
    with torch.no_grad():
        va_loss = loss_fn(model(Xva_d), yva_d).item()
    sched.step(va_loss)
```

Figure 13: Code: network definition and training loop (screenshots of the project notebook).

5.5 Tandem network with a differentiable TMM

The key change is the loss: the network's predicted thicknesses are passed through a TMM written in PyTorch and the loss is the error between the *re-computed spectrum* and the target spectrum. Any thickness set that reproduces the target is then a correct answer, so non-uniqueness stops being a problem. The real-valued recurrences (with $a = \cos\delta$, $b = -\sin\delta/n$, $c = -n\sin\delta$, $d = \cos\delta$) are

$$A' = Aa - Bc, \quad B' = Ab + Bd, \quad C' = Ca + Dc, \quad D' = -Cb + Dd, \tag{6}$$

and agree with the NumPy TMM to 8×10^{-7} (max difference over the test set). Training from random weights stalled, so the tandem network is *warm-started* from the naive network and fine-tuned with lr 3×10^{-4} for 30 epochs (92 s).

```python
def reflectance_xp(d, n_list, lam, n_air, n_sub, xp):
    """Real-valued TMM. Works with numpy (xp=np) and torch (xp=torch). d:(B,N), lam:(L,) -> R:(B,L)"""
    base = d[:, :1] / lam[None, :]
    A = xp.ones_like(base); B = xp.zeros_like(base); C = xp.zeros_like(base); D = xp.ones_like(base)
    for j, n in enumerate(n_list):
        delta = 6.283185307179586 * n * d[:, j:j+1] / lam[None, :]
        c, s = xp.cos(delta), xp.sin(delta)
        a, b, cc, dd = c, -s / n, -n * s, c
        A, B, C, D = A*a - B*cc, A*b + B*dd, C*a + D*cc, -C*b + D*dd
    nr, ni = n_air*A - D*n_sub, n_air*B*n_sub - C
    dr, di = n_air*A + D*n_sub, n_air*B*n_sub + C
    return (nr*nr + ni*ni) / (dr*dr + di*di)

def reflectance_torch(d_nm):
    return reflectance_xp(d_nm, N_LIST, LAM_T, float(N_AIR), float(N_SUB), torch)
```

```python
tandem = InverseNet().to(device)                    # same architecture as the naive network
# WARM START: begin from the trained naive network (training from random weights gets stuck in a poor solution)
tandem.load_state_dict(torch.load(f"{SAVE_DIR}/naive_inverse.pt", map_location=device))
opt = torch.optim.Adam(tandem.parameters(), lr=3e-4)   # smaller learning rate for fine-tuning
sched = torch.optim.lr_scheduler.ReduceLROnPlateau(opt, factor=0.5, patience=3)

def to_nm(out):                                     # sigmoid output (0..1) -> thickness in [D_MIN, D_MAX] nm
    return D_MIN + (D_MAX - D_MIN) * out

for epoch in range(1, EPOCHS + 1):
    tandem.train()
    perm = torch.randperm(len(Xtr_d), device=device)
    total = 0.0
    for i in range(0, len(perm), BATCH):
        idx = perm[i:i + BATCH]
        x = Xtr_d[idx]
        opt.zero_grad()
        loss = ((reflectance_torch(to_nm(tandem(x))) - x) ** 2).mean()   # spectrum vs target, through physics
        loss.backward()
        opt.step()
        total += loss.detach() * len(idx)
```

Figure 14: Code: differentiable TMM and the tandem fine-tuning.

5.6 Gradient refinement

Since the forward model is cheap and differentiable, any predicted design can be polished by a local optimizer. We minimize the mean-squared spectral error with SciPy's L-BFGS-B, bounds $[20, 200]$ nm, supplying an **analytic gradient** computed from prefix and suffix matrix products (verified against finite differences to 10^{-9}). The network provides the starting point; this removes the optimiser's main weakness, the unlucky start (Fig. 15).

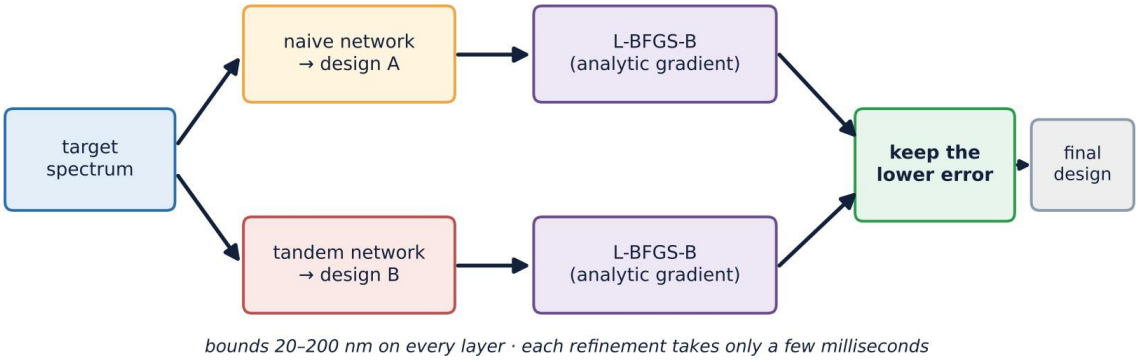


Figure 15: Network prediction as the starting point of L-BFGS-B refinement.

```python
def refine(x0_nm, target, maxiter=200):
    def fun(d):
        l, g, _ = loss_and_grad(d[None, :], target[None, :], N_LIST, wavelengths, N_AIR, N_SUB)
        return l[0], g[0]
    res = minimize(fun, np.asarray(x0_nm, np.float64), jac=True, method="L-BFGS-B",
                   bounds=[(D_MIN, D_MAX)] * N_LAYERS, options={"maxiter": maxiter})
    return res.x, res.fun
```

Figure 16: Code: the `refine` function.

5.7 Evaluation principle

We never compare thicknesses. Each predicted design is pushed through the *real* TMM and the **spectral MAE** $\frac{1}{101}\sum|R_{\text{pred}} - R_{\text{target}}|$ is reported, together with the fraction of targets matched below 0.02 and 0.05. Baselines: a random design and "always the average design".

6 Results

6.1 Naive network

Table 1: Naive network on the 20 000 test spectra.

Method	thickness MAE (nm)	spectral MAE
random design	–	0.3294
always the average design	30.8	0.3801
naive network	16.8	**0.1173**

The network beats both baselines by a wide margin (median spectral error 0.053; 34.2 % of targets below 0.02, 49.4 % below 0.05) but its error is highly uneven: 0.016 for chirped and 0.016 for quarter-wave targets, but 0.240 for random targets, which are the most ambiguous.

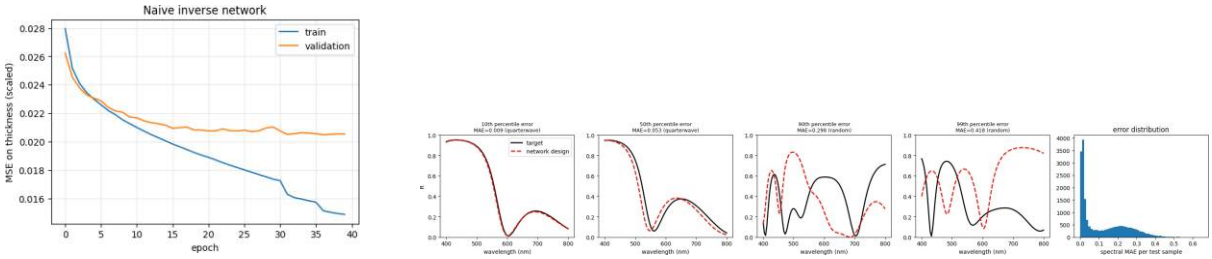


Figure 17: Naive network: training curve (left) and test evaluation (right), from the notebook.

6.2 Tandem network

Table 2: Naive versus tandem on the full 20 000-sample test set (spectral MAE).

	mean	median	< 0.02	< 0.05
naive	0.1173	0.0533	34.2 %	49.4 %
tandem	0.0589	0.0246	45.9 %	59.0 %

Table 3: Mean spectral MAE by sample type.

type	naive	tandem
chirped	0.0160	0.0118
perturbed	0.0853	0.0365
quarter-wave	0.0156	0.0112
random	0.2400	0.1192

The tandem loss halves the mean error, and the gain is largest where non-uniqueness is strongest (random and perturbed targets).

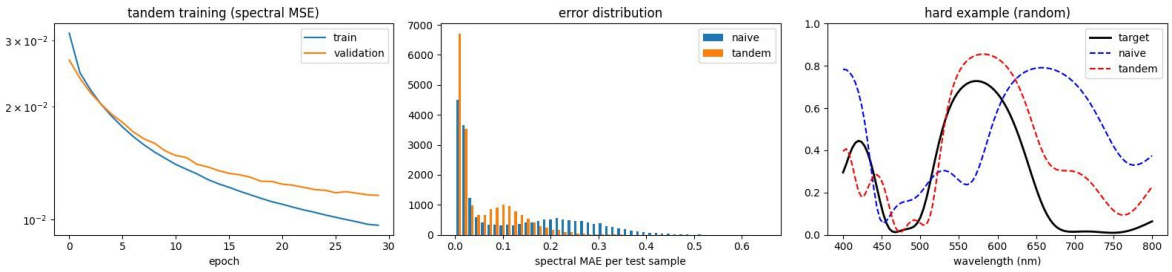


Figure 18: Tandem training curve, error distribution and a hard test example (notebook output).

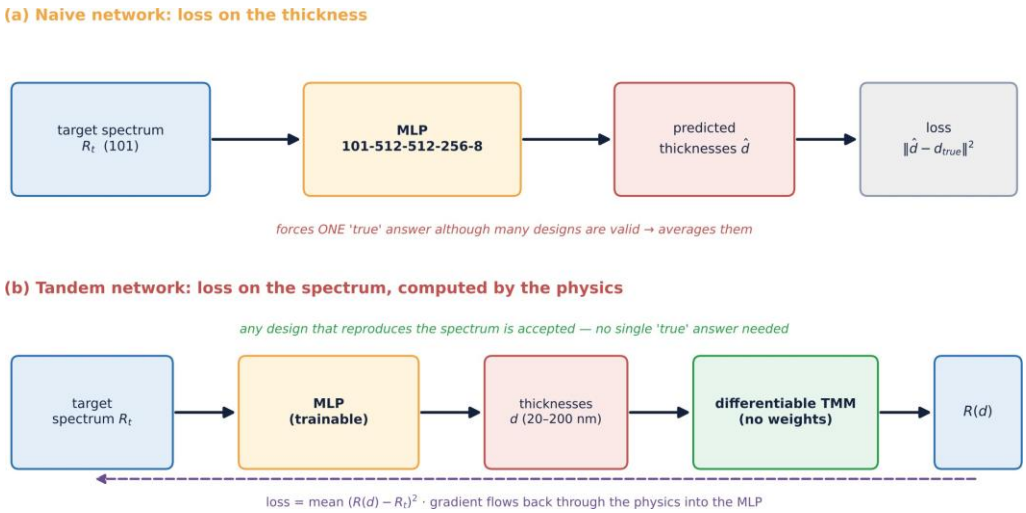


Figure 19: Why the tandem loss helps: averaging versus matching the spectrum.

6.3 Refinement and comparison with classical optimisation

Table 4 compares six strategies on the same 300 test targets.

Table 4: Strategies on 300 test targets ("sec/target" measured in Colab).

method	mean MAE	median	< 0.02	< 0.05	sec/target
naive	0.1140	0.0385	38 %	52 %	≈ 0
tandem	0.0592	0.0201	50 %	61 %	≈ 0
naive + refine	**0.0131**	**0.0035**	**80 %**	**92 %**	0.05
tandem + refine	0.0225	0.0052	63 %	80 %	0.04
random start + refine	0.0933	0.0883	9 %	22 %	0.08
best of 5 random + refine	0.0446	0.0406	26 %	61 %	0.41

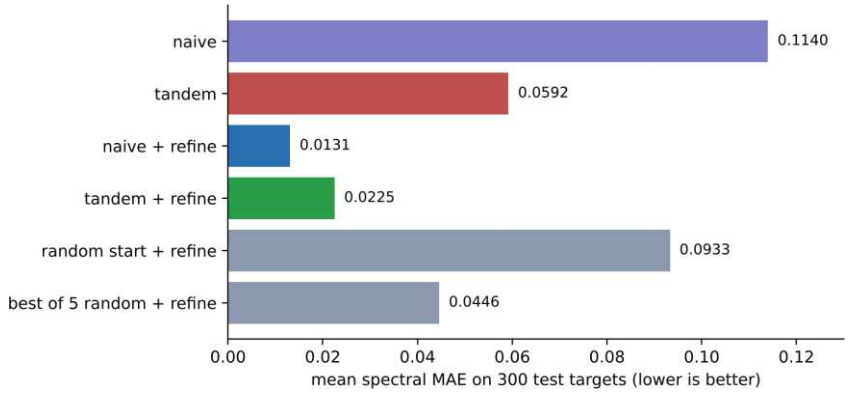


Figure 20: Mean spectral MAE of the six strategies (lower is better).

Main findings:

- Refinement from the **naive** network's prediction is best: mean error 0.0131, about 9× lower than the naive network alone, 4.5× lower than the tandem alone, and 3.4×lower than the best of five random restarts, while taking 8× less time (0.05 vs. 0.41 s).
- Refining from a random start is poor (0.0933): the optimiser gets stuck in local minima, which is the failure mode the network start avoids.
- Surprisingly, naive + refine beats tandem + refine. A plausible explanation (not tested here) is that the tandem network gives a good spectrum but a design that already sits in a narrow local minimum, whereas the naive network's averaged design lies in a broad basin from which L-BFGS-B can descend; further experiments would be needed to confirm this.

6.4 Hand-made target spectra

To test generalisation beyond the dataset we asked for three spectra drawn by hand: a mirror over 480–650 nm, a broadband anti-reflection ($R = 0.02$) and a dip at 600 nm.

Table 5: Spectral MAE on hand-made targets.

target	naive	tandem	tandem + refine
mirror 480–650 nm	0.309	0.321	0.073
broadband AR ($R = 0.02$)	0.358	0.207	0.114
dip at 600 nm	0.141	0.095	0.057

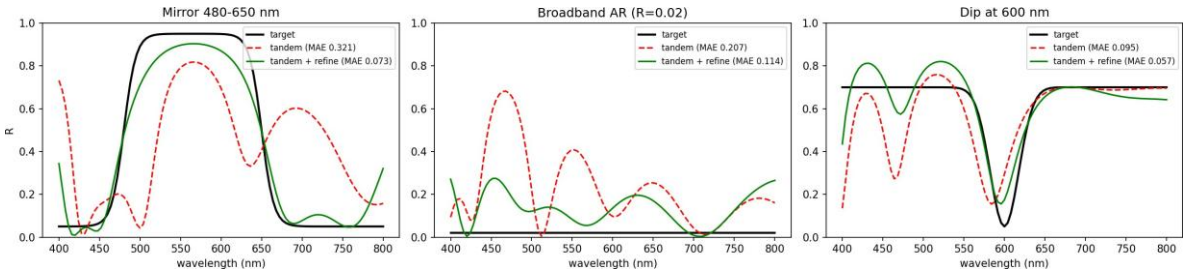


Figure 21: Hand-made targets: target (black), tandem (red dashed), tandem + refine (green). Refined thicknesses in nm: mirror [168, 86, 65, 96, 170, 96, 97, 179]; AR [20, 44, 200, 20, 200, 20, 119, 193]; dip [55, 82, 59, 200, 73, 153, 145, 181].

Refinement improves every case, but hand-made spectra are harder than test spectra because they may not be reachable by an 8-layer, two-material stack: the broadband AR design hits the 20 nm and 200 nm bounds several times, a sign that the target is at the edge of what this stack can do.

7 Conclusion

The inverse design problem for an 8-layer TiO_2/SiO_2 coating was solved to good accuracy by combining three ideas: (i) a physically validated TMM and a large reproducible dataset; (ii) a tandem network whose loss is evaluated in spectrum space through a differentiable TMM, which halves the error of a naive network by removing the non-uniqueness problem; (iii) network-initialised L-BFGS-B refinement with an analytic gradient, which brings the mean spectral error to 0.0131, with 92 % of 300 unseen targets matched within 0.05 and about 0.05 s per design, clearly better than random-restart optimisation at comparable or lower cost. The practical lesson is that machine learning and classical optimisation are complementary: the network supplies a good starting basin, the optimiser supplies precision.

8 Limitations and Future Work

- **Fixed structure:** 8 layers, fixed material order, fixed thickness bounds. A variable number of layers could be handled with a maximum of N_{max} layers where zero thickness means "absent".
- **Materials:** constant refractive indices, no dispersion, no absorption. Future work: Cauchy dispersion and a discrete material choice via Gumbel-softmax.
- **Normal incidence and one polarisation state only;** oblique incidence needs only the admittance change but multiplies the target size.
- **Fabrication:** no tolerance analysis for thickness errors and no minimum-layer-thickness constraint beyond 20 nm.
- **Statistics:** refinement results use 300 test targets and a single training run; no confidence intervals. Timing was measured on Colab and varies with hardware.
- **Remaining one-to-many ambiguity:** generative models (conditional VAE, mixture-density networks) could return several candidate designs; a small multi-hypothesis experiment we tried gave only a modest gain.
- Code was validated by the author in Colab (GPU); the project was not independently reproduced on other hardware.

Digital Resources: Video: https://youtu.be/ESs2XLpi84w

References

[1] H. A. Macleod, *Thin-Film Optical Filters*, 4th ed., CRC Press (2010).
[2] M. Born and E. Wolf, *Principles of Optics*, 7th ed., Cambridge University Press (1999).
[3] A. V. Tikhonravov, M. K. Trubetskov and G. W. DeBell, "Application of the needle optimization technique to the design of optical coatings," *Appl. Opt.* **35**, 5493 (1996).
[4] J. Peurifoy et al., "Nanophotonic particle simulation and inverse design using artificial neural networks," *Science Advances* **4**, eaar4206 (2018).
[5] I. Malkiel et al., "Plasmonic nanostructure design and characterization via deep learning," *Light: Science & Applications* **7**, 60 (2018).
[6] D. Liu, Y. Tan, E. Khoram and Z. Yu, "Training deep neural networks for the inverse design of nanophotonic structures," *ACS Photonics* **5**, 1365 (2018).
[7] J. Jiang and J. A. Fan, "Global optimization of dielectric metasurfaces using a physics-driven neural network," *Nano Letters* **19**, 5366 (2019).
[8] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.* **16**, 1190 (1995).
[9] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," arXiv:1412.6980 (2014).

A Code listings (screenshots of the notebook)

```python
def reflectance(d, n_list=None, lam=None):
    """TMM. d: (B, N) thicknesses in nm  ->  R: (B, N_LAM). Normal incidence."""
    n_list = layer_index if n_list is None else n_list
    lam = wavelengths if lam is None else lam
    d = np.atleast_2d(d)
    B = d.shape[0]
    m11 = np.ones((B, lam.size), dtype=complex)
    m12 = np.zeros((B, lam.size), dtype=complex)
    m21 = np.zeros((B, lam.size), dtype=complex)
    m22 = np.ones((B, lam.size), dtype=complex)
    for j in range(d.shape[1]):
        n = n_list[j]
        delta = 2 * np.pi * n * d[:, j:j+1] / lam[None, :]
        c, s = np.cos(delta), np.sin(delta)
        a11, a12, a21, a22 = c, -1j * s / n, -1j * n * s, c
        m11, m12, m21, m22 = (m11*a11 + m12*a21, m11*a12 + m12*a22,
                              m21*a11 + m22*a21, m21*a12 + m22*a22)
    Bv = m11 + m12 * N_SUB
    Cv = m21 + m22 * N_SUB
    r = (N_AIR * Bv - Cv) / (N_AIR * Bv + Cv)
    return (np.abs(r) ** 2).astype(np.float32)
```

Figure 22: NumPy transfer-matrix forward model.

```python
def validate():
    lam0 = np.array([550.0])
    bare = reflectance(np.zeros((1, 0)), np.array([]), lam0)[0, 0]
    mgf2 = reflectance(np.array([[550/(4*1.38)]]), np.array([1.38]), lam0)[0, 0]
    qw   = reflectance(np.array([[550/(4*n) for n in layer_index]]), layer_index, lam0)[0, 0]
    print(f"bare glass {bare:.4f} (0.0426) | MgF2 {mgf2:.4f} (0.0126) | mirror {qw:.4f} (~0.952 if N=8)")
    assert abs(bare - 0.0426) < 1e-3 and abs(mgf2 - 0.0126) < 1e-3
    if N_LAYERS == 8: assert abs(qw - 0.952) < 5e-3
    assert np.allclose(reflectance(np.zeros((1, N_LAYERS))), 0.0426, atol=1e-3)
    print("TMM validation passed")

validate()
```

Figure 23: Physics validation: bare glass, MgF_2 and 8-layer mirror against analytic values.

```python
def sample_random(n, rng):
    return rng.uniform(D_MIN, D_MAX, (n, N_LAYERS))

def sample_quarterwave(n, rng):
    lam0 = rng.uniform(LAM_MIN, LAM_MAX, (n, 1))
    d = lam0 / (4 * layer_index[None, :])
    return d * (1 + rng.normal(0, 0.05, d.shape))

def sample_chirped(n, rng):
    a = rng.uniform(LAM_MIN, LAM_MAX, (n, 1))
    b = rng.uniform(LAM_MIN, LAM_MAX, (n, 1))
    lam_j = a + (b - a) * np.linspace(0, 1, N_LAYERS)[None, :]
    d = lam_j / (4 * layer_index[None, :])
    return d * (1 + rng.normal(0, 0.03, d.shape))
```

```python
def generate(n_total, seed=SEED):
    rng = np.random.default_rng(seed)          # same seed -> same data every run
    ds, kinds = [], []
    for k, frac in MIX.items():
        n = int(round(n_total * frac))
        ds.append(np.clip(SAMPLERS[k](n, rng), D_MIN, D_MAX)); kinds += [k] * n
    d = np.vstack(ds).astype(np.float32)
    kinds = np.array(kinds)
    X = np.concatenate([reflectance(d[i:i+5000]) for i in range(0, len(d), 5000)])
    perm = rng.permutation(len(d))             # shuffle
    return X[perm], d[perm], kinds[perm]
```

Figure 24: Samplers and dataset generation.

```python
else:
    X, d, kinds = generate(N_SAMPLES)
    y = (d / D_MAX).astype(np.float32)          # thicknesses normalised to [0.1, 1]
    assert np.isfinite(X).all() and X.min() >= 0 and X.max() <= 1 + 1e-6
    n = len(X); i1, i2 = int(0.8*n), int(0.9*n)  # 80 / 10 / 10 split
    for name, sl in {"train": slice(0, i1), "val": slice(i1, i2), "test": slice(i2, n)}.items():
        np.savez_compressed(f"{SAVE_DIR}/{name}.npz", X=X[sl], y=y[sl], kinds=kinds[sl])
    np.savez(f"{SAVE_DIR}/meta.npz", wavelengths=wavelengths, layer_index=layer_index,
             D_MAX=D_MAX, N_SUB=N_SUB, N_AIR=N_AIR)
    print("generated and saved:"
```

Figure 25: Train / validation / test split.

```python
def loss_and_grad(d, X, n_list, lam, n_air, n_sub):
    """per-sample loss = mean_lambda (R-X)^2 and its analytic gradient wrt thicknesses (numpy, float64)."""
    d = np.asarray(d, np.float64); Bn, N = d.shape; L = lam.size
    nl = np.array(n_list, float)[None, :, None]
    k = 2*np.pi*nl/lam[None, None, :]
    delta = k * d[:, :, None]; c, s = np.cos(delta), np.sin(delta)
    mul = lambda P, Q: (P[0]*Q[0]-P[1]*Q[2], P[0]*Q[1]+P[1]*Q[3], P[2]*Q[0]+P[3]*Q[2], -P[2]*Q[1]+P[3]*Q[3])
    Ls  = [(c[:, j], -s[:, j]/nl[0, j], -nl[0, j]*s[:, j], c[:, j]) for j in range(N)]
    dLs = [(-s[:, j]*k[0, j], -c[:, j]*k[0, j]/nl[0, j], -nl[0, j]*c[:, j]*k[0, j], -s[:, j]*k[0, j]) for j in range(N)]
    one, zero = np.ones((Bn, L)), np.zeros((Bn, L)); I = (one, zero, zero, one)
    pre = [I]
    for j in range(N): pre.append(mul(pre[-1], Ls[j]))
    suf = [None]*N + [I]
    for j in range(N-1, -1, -1): suf[j] = mul(Ls[j], suf[j+1])
    A, B, C, D = pre[N]
    Bv = A + 1j*B*n_sub; Cv = D*n_sub + 1j*C
    Nn, Dn = n_air*Bv - Cv, n_air*Bv + Cv; r = Nn/Dn; R = np.abs(r)**2
    err = R - X; loss = (err**2).mean(axis=1); grad = np.zeros((Bn, N))
    for j in range(N):
        dA, dB, dC, dD = mul(mul(pre[j], dLs[j]), suf[j+1])
        dBv = dA + 1j*dB*n_sub; dCv = dD*n_sub + 1j*dC
        dN, dDn = n_air*dBv - dCv, n_air*dBv + dCv
        dr = (dN*Dn - Nn*dDn)/Dn**2; dR = 2*np.real(np.conj(r)*dr)
        grad[:, j] = (2*err*dR).mean(axis=1)
    return loss, grad, R
```

Figure 26: Loss and analytic gradient for refinement.

```python
best_x, t_total = [], 0.0
for i in sub:
    tgt = R_target[i].astype(np.float64)
    t = time.perf_counter()
    cands = [refine(pred_naive[i], tgt), refine(pred_tandem[i], tgt)]      # each returns (thicknesses, final loss)
    t_total += time.perf_counter() - t
    best_x.append(min(cands, key=lambda c: c[1])[0])
```

Figure 27: Multi-start refinement.