TMMax: High-performance modeling of multilayer thin-film structures using transfer matrix method with JAX

Bahrem Serhat Danis[1] and **Esra Zayim**[2]

1 Department of Electrical and Electronics Engineering, Koç University, Istanbul, 34450, Turkey **2** Physics Engineering Department, Istanbul Technical University, Istanbul, 34469, Turkey

DOI: 10.21105/joss.09088

Software

- Review
- Repository
- Archive

Editor: Owen Lockwood

Reviewers:

- @HectorMozo3110
- @CrazeXD

Submitted: 15 July 2025
Published: 27 October 2025

Summary

Optical multilayer thin-films are fundamental components that enable the precise control of reflectance, transmittance, and phase shift in the design of photonic systems. Rapid and accessible simulation of these structures holds critical importance for designing and analyzing complex coatings. While researchers commonly use the traditional transfer matrix method for designing these structures, its scalar approach to wavelength and angle of incidence causes redundant recalculations and inefficiencies in large-scale simulations. Furthermore, traditional method implementations do not support automatic differentiation, which limits their applicability in gradient-based inverse design approaches. Here, we present TMMax, a Python library that fully vectorizes and accelerates transfer matrix method using the high-performance machine learning library JAX. TMMax supports CPU, GPU, and TPU hardware, and includes a publicly available material database. Our approach, demonstrated through benchmarking, allows us to model thin-film stacks with hundreds of layers within seconds. This illustrates that our method speeds up simulations by two orders of magnitude over a baseline NumPy implementation, enabling optical engineers and thin-film researchers in optics and photonics to efficiently design complex dielectric multilayer structures through rapid and scalable simulations.

Statement of need

The Transfer Matrix Method (TMM) models multilayer optical thin films by applying Snell's law for light propagation and Fresnel equations to compute interface transmittance and reflectance.

$$\mathbf{M} = \prod_{i=0}^{N-2} \mathbf{M}_i \tag{1}$$

In TMM, the optical behavior of an N-layer multilayer structure composed of dielectric materials is obtained by computing the system matrix $\mathbf{M}$, as shown in Equation (1). This matrix calculation, commonly referred to as the Abeles TMM (Abelès, Florin, 1950), results from the successive multiplication of the transfer matrices of each layer ($\mathbf{M}_i$) (Katsidis & Siapkas, 2002).

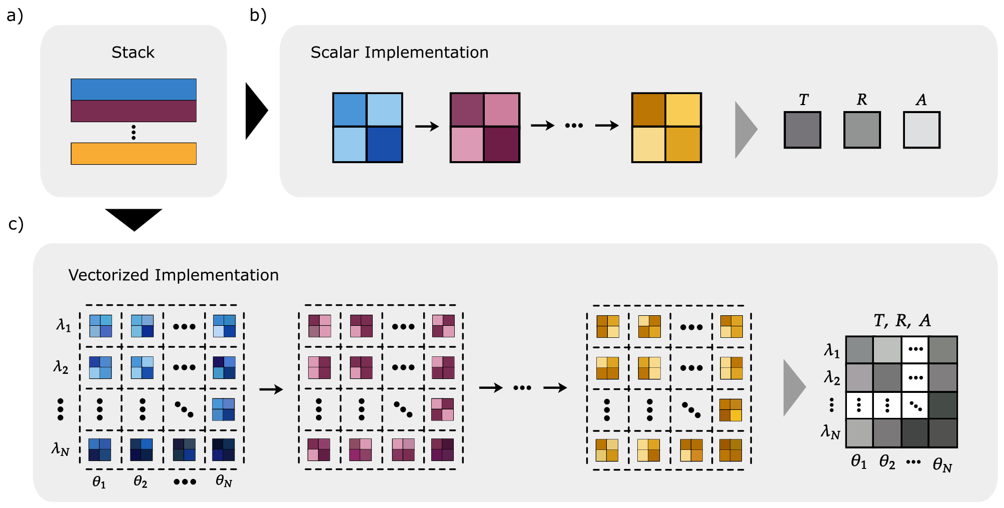

Figure 1: Schematic of two strategies for calculating transmission, reflection, and absorption in multilayer thin-film simulations. The system (a) is modeled either by sequentially multiplying 2×2 transfer matrices for each wavelength and incidence angle (b) or by vectorizing these operations across both axes (c).

In traditional TMM implementations, the stack of layers in Figure 1a is simulated using a single wavelength and angle of incidence, as shown in Figure 1b, and nested loops over wavelengths and angles lead to redundant calculations (Byrnes, 2020). TMMax removes these redundancies by vectorizing wavelengths and angles and all intermediate TMM operations via JAX (Bradbury et al., 2018). As seen in the schematic of the vectorized implementation in Figure 1c, we vectorize all intermediate operations in TMM and subsequently apply JAX's just-in-time (JIT) decorator. Instead of running the mapped TMM code sequentially over each batch element of wavelength and angle of incidence, `jax.jit` fuses all operations across the batch into a single XLA-compiled (OpenXLA Team, 2023) kernel. This reduces function call overhead and provides a faster TMM implementation. TMMax replaces the conventional for-loop system-matrix calculation (Nishida et al., 2011) with JAX's lax.scan, enabling JIT compilation and eliminating interpreter bottlenecks, while running efficiently on CPUs, GPUs, and TPUs without code changes.

TMMax supports deep learning–based inverse design by keeping all computations on the GPU, avoiding costly CPU–GPU data transfers (Hegde, 2019). Whereas NumPy-based (Harris et al., 2020) TMM packages that lack native gradients and require Autograd (Maclaurin et al., 2015), TMMax natively computes gradients. Additionally, TMMax integrates a curated database of 30 extensively used dielectric materials, sourced from `refractiveindex.info` (Polyanskiy, 2024), thereby enabling optical engineers and thin-film researchers in optics and photonics to efficiently simulate complex multilayer structures through a scalable, JAX-accelerated implementation.

Benchmarks

Runtime in TMM scales naturally with the number of layers, as well as the lengths of the wavelength and incidence-angle arrays, due to the increased number of transfer matrix multiplications. To benchmark TMMax, we used `tmm` library (Byrnes, 2020) as a reference.

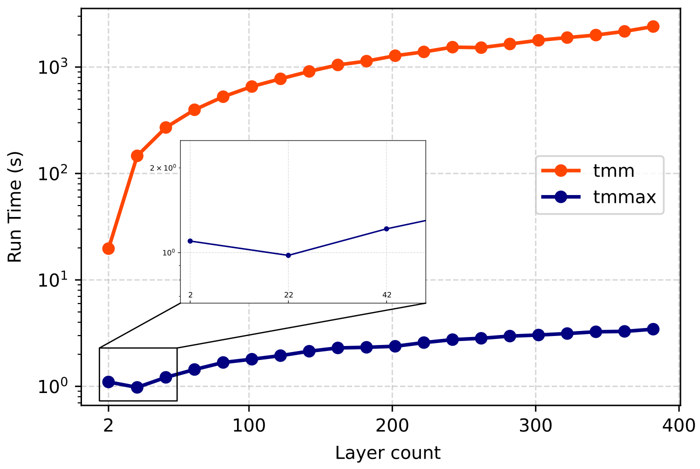

Figure 2: Run time vs. layer count comparing 'tmm' (orange) and TMMax (blue).

To assess how layer count affects computational performance, we sampled 20 multilayer structures ranging from 2 to 400 layers, with each layer randomly assigned one of seven materials and thicknesses between 100–500 nm. Spectral and angular domains were fixed at 20 points each, spanning 500–1000 nm and 0–π/2 radians, respectively. Figure 2 shows that while `tmm` runtime grows rapidly, TMMax scales efficiently, remaining nearly constant (~1.0–1.2 s) for low-layer structures and achieving speedups from 18× (2 layers) to 700× (400 layers).

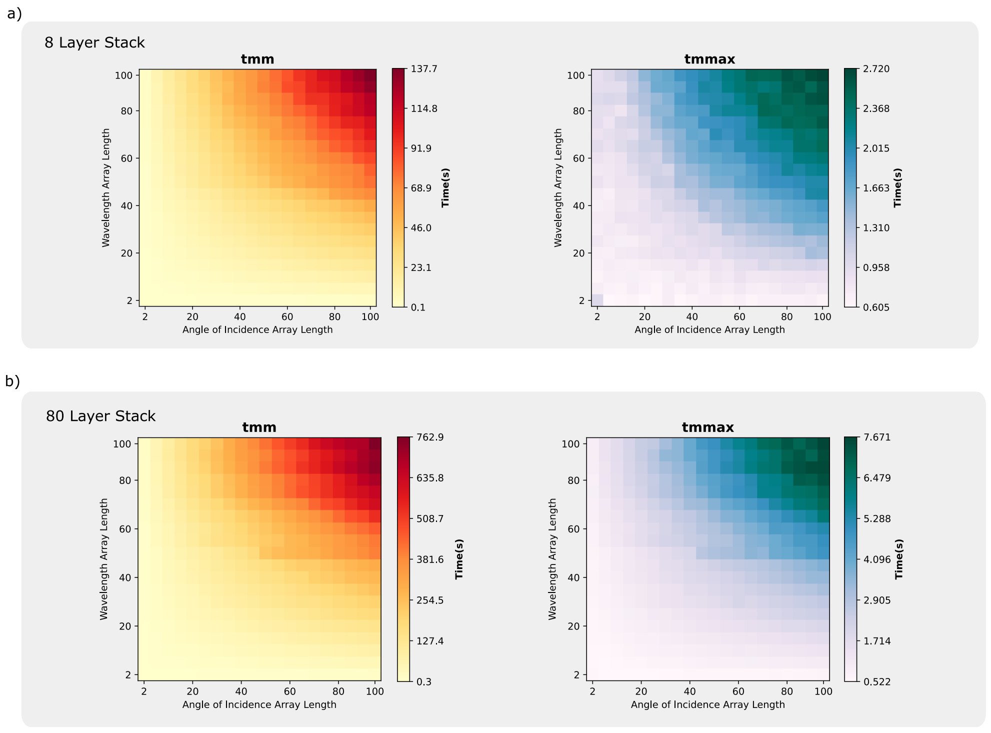

Figure 3: The colormaps show the runtime performance of 'tmm' and TMMax across varying simulation grid sizes, comparing 8- and 80-layer stacks in (a) and (b), respectively.

We benchmarked the effects of wavelength and incident angle array sizes by sampling 20 values from 2 to 100, generating simulation grids from 2×2 to 100×100 for an 8-layer structure (Figure 3a). `tmm` runtime rises sharply with grid size, reaching ~138 s for 100×100, whereas TMMax remains below 3 s. For the smallest 2×2 grid, `tmm` is faster (~0.1 s vs. ~0.6 s) due to NumPy's low overhead, while JAX incurs higher initialization costs. As layers increase to 80 (Figure 3b), `tmm` exceeds 760 s, but TMMax stays under 8 s, demonstrating superior efficiency and stability against both problem size and structural complexity.

We used Python's timeit module to benchmark each simulation 50 times, with all comparisons run on a single Intel Core i9 core without GPU or multicore use for fairness.

Installation

TMMax can be readily installed from the Python Package Index using `pip install tmmax`, which automatically handles all dependencies. For detailed installation instructions and platform compatibility, please refer to the TMMax Documentation.

Acknowledgements

This work was supported by the Scientific and Technological Research Council of Türkiye (TUBITAK) under the 2209-A Research Project Support Programme for Undergraduate Students, 2022 First-Term Call.

References

Abelès, Florin. (1950). Recherches sur la propagation des ondes électromagnétiques sinusoïdales dans les milieux stratifiés - application aux couches minces. *Ann. Phys.*, *12*(5), 596–640. https://doi.org/10.1051/anphys/195012050596

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Byrnes, S. J. (2020). *Multilayer optical calculations*. https://doi.org/10.48550/arXiv.1603.02720

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hegde, R. S. (2019). Accelerating optics design optimizations with deep learning. *Optical Engineering*, *58*(6), 065103. https://doi.org/10.1117/1.OE.58.6.065103

Katsidis, C. C., & Siapkas, D. I. (2002). General transfer-matrix method for optical multilayer systems with coherent, partially coherent, and incoherent interference. *Applied Optics*, *41*(19), 3978–3987. https://doi.org/10.1364/AO.41.003978

Maclaurin, D., Duvenaud, D., & Adams, R. P. (2015). Autograd: Effortless gradients in numpy. *ICML 2015 AutoML Workshop*, *238*.

Nishida, K., Ito, Y., & Nakano, K. (2011). Accelerating the dynamic programming for the matrix chain product on the GPU. *2011 Second International Conference on Networking and Computing*, 320–326. https://doi.org/10.1109/ICNC.2011.62

OpenXLA Team. (2023). *XLA: Accelerated Linear Algebra Compiler for Machine Learning*. https://github.com/openxla/xla. https://github.com/openxla/xla

Polyanskiy, M. N. (2024). Refractiveindex.info database of optical constants. *Scientific Data*, *11*(1), 94. https://doi.org/10.1038/s41597-023-02898-2