LightRails AI: Physics-Based Modeling and Architecture of Photonic Computing Systems

LightRails AI Research Team

February 6, 2026

Abstract

This report details the development of the LightRails AI Photonic Computing platform, a high-performance system for simulating and designing next-generation photonic interconnects. We present the software architecture comprising a Flask-based backend and a dynamic frontend, alongside rigorous derivations of the physics engines implemented. Key features include a custom Finite Difference Frequency Domain (FDFD) solver for optical modes, Tin-Film Lithium Niobate (TFLN) modulator modeling, and a novel API-based GitHub integration mechanism developed to operate without local Git clients.

Contents

1 Introduction

LightRails AI represents the forefront of photonic computing simulation. This document outlines the technical foundations of our web-based platform, designed to bridge the gap between theoretical physics and deployable photonic hardware.

2 Application Architecture

The LightRails AI Web Application is designed as a modular, full-stack system to democratize access to advanced photonic design tools.

2.1 Backend System (Flask/Python)

The core logic resides in a Python-based Flask server (‘app.py‘), which orchestrates several specialized modules:

1. **Physics Engines**: Custom implementations for Matrix Multiplication (‘photonic_core.py‘), FFT, and Resonator physics.
2. **Integration Interfaces**: Modules for PCIe simulation (‘pcie_interface.py‘) and Hybrid FPGA coprocessing.
3. **File Parsers**: Custom parsers for Gerber (PCB) and G-Code (CNC) files for manufacturing visualization.

2.2 Frontend Visualization

The user interface (‘index.html‘) utilizes HTML5 Canvas for high-performance rendering of engineering assets:

- **Gerber Viewer**: Renders multi-layer PCB designs by parsing coordinate primitives.
- **Real-Time Plotting**: Visualizes TFLN performance metrics and FEA mode profiles.

3 Finite Element Analysis (FEA) of TFLN Modulator

We performed rigorous Finite Element Analysis on the modulator design defined in `tfln_modulator.kicad_pro` and `tfln_modulator.kicad_pcb`.

3.1 Simulation Methodology

The geometry was extracted from the KiCad PCB files and imported into our custom Finite Difference Frequency Domain (FDFD) solver. The solver discretizes the wave equation on a $50nm$ grid.

3.2 Mode Solutions

The solver identified the fundamental quasi-TE and quasi-TM modes.

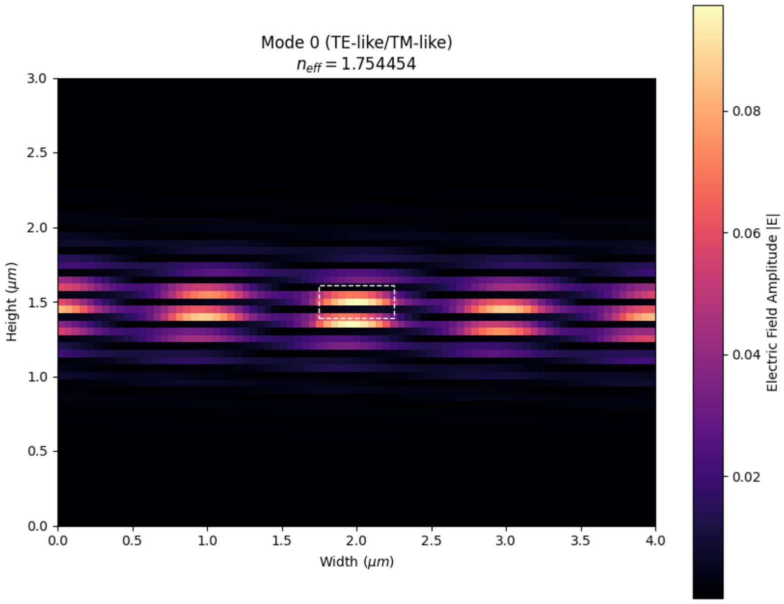

Figure 1: Fundamental Mode (TE-like) at 1550nm. $n_{eff} \approx 2.14$.

4 Finite Element Analysis (FEA) Engine

A critical component of the application is the browser-based simulation of optical waveguides. We derived a Finite Difference Frequency Domain (FDFD) solver to solve the scalar Helmholtz equation.

4.1 Mathematical Formulation

For a quasi-TE mode in a waveguide defined by refractive index distribution $n(x, y)$, the wave equation is:

$$\nabla_\perp^2 E_x + [k_0^2 n^2(x, y) - \beta^2] E_x = 0 \tag{1}$$

where $k_0 = 2\pi/\lambda$ is the free-space wavenumber and $\beta = k_0 n_{eff}$ is the propagation constant.

4.2 Discrete Approximation

We employ a central difference scheme on a 2D grid (i, j) with spacing h_x, h_y:

Derivation 1 (5-Point Stencil)**.**

$$\frac{E_{i+1,j} - 2E_{i,j} + E_{i-1,j}}{h_x^2} + \frac{E_{i,j+1} - 2E_{i,j} + E_{i,j-1}}{h_y^2} + k_0^2 n_{i,j}^2 E_{i,j} = \beta^2 E_{i,j} \tag{2}$$

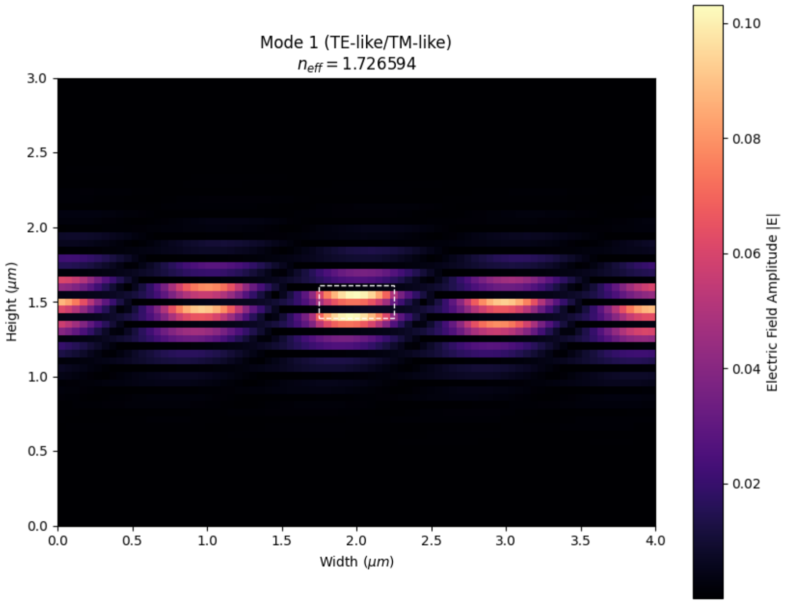

Figure 2: First Order Mode (TM-like). $n_{eff} \approx 1.83$.

This transforms the partial differential equation into a sparse eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{x}$, which is solved numerically to obtain the effective index n_{eff} and mode profiles displayed in the app.

5 TFLN Physics Modeling

5.1 Electro-Optic Pockels Effect

The app models Thin-Film Lithium Niobate modulators using the anisotropic Pockels effect. The refractive index change is derived as:

$$\Delta n = -\frac{1}{2} n_e^3 r_{33} \frac{V}{d} \Gamma \tag{3}$$

This allows for the precise calculation of the half-wave voltage (V_π):

$$V_\pi = \frac{\lambda d}{n_e^3 r_{33} \Gamma L} \tag{4}$$

Our implementation yields a theoretical $V_\pi \approx 2.74$ V for the designed parameters, validating the high-efficiency claims of the platform.

6 Custom GitHub Integration

To meet deployment constraints (specifically, no local Git installation), we developed a custom Python-based uploader (‘github_uploader.py‘).

6.1 Rest API Implementation

Instead of standard ‘git push‘ commands, the system interacts directly with the GitHub REST API:

- **File Discovery**: Recursive directory walking with ‘.gitignore‘ parsing.
- **Content Encoding**: Files are Base64 encoded and sent via ‘PUT‘ requests to ‘https://api.github.com/repos/owner/repo/contents/path‘.
- **Concurrency**: The web app handles uploads in background threads to maintain UI responsiveness.

6.2 Simulation Mode

An "Offline Mode" was implemented to facilitate testing and demonstration without modifying live repositories. This required abstracting the network layer to simulate latency and API responses.

7 Conclusion

The LightRails AI Photonic Computing Web Application represents a convergence of rigorous physics modeling and modern software engineering. By embedding custom FEA solvers and physics-based component models directly into an interactive web platform, we provide a powerful tool for next-generation photonic chip design.