LLM4Laser: Large Language Models Automate the Inverse Design of Lasers

Renjie Li[1,2*], **Ceyao Zhang**[1*], **Sixuan Mao**[1], **Xiyuan Zhou**[1],
Feng Yin[1†], **Sergios Theodoridis**[3], **Zhaoyu Zhang**[1†]
[1]The Chinese University of Hong Kong, Shenzhen
[2]University of Illinois Urbana-Champaign, [3]University of Athens

Abstract

AI-driven inverse design has transformed nanophotonic device discovery, yet remains fundamentally limited: while AI algorithms accelerate optimization, the simulation and training infrastructure must still be hand-crafted by domain experts. Here we demonstrate that large language models (LLMs) can autonomously construct complete inverse-design workflows through natural-language interaction. We guided an LLM to generate electromagnetic simulations, reinforcement-learning optimization, and integrated environments for photonic-crystal surface-emitting lasers (PCSELs). Through iterative dialogue, the LLM refined and debugged these components with minimal human intervention, producing a functional end-to-end workflow. The resulting autonomous framework discovered PCSEL geometries exhibiting a Q-factor of 37,000 and normalized radiation loss 18-fold lower than current state-of-the-art devices while maintaining a significantly reduced footprint. To accelerate design further, we fine-tuned a multi-modal LLM to predict optical properties directly from device images, enabling rapid performance estimation. Our results establish that LLMs can both construct sophisticated computational infrastructure for nanophotonic inverse design and interpret physical structures visually, providing a pathway toward fully autonomous photonic engineering.

Introduction

Photonic-crystal surface-emitting lasers (PCSELs; Hirose et al. 2014) have emerged as a transformative platform for next-generation light sources in LiDAR, free-space communications, and high-power industrial applications. Their unique two-dimensional photonic crystal feedback enables exceptional beam quality, narrow divergence, and surface-normal emission with scalable output power. Yet these advantages come at the cost of substantial design complexity: achieving target lasing properties requires precise engineering of multilayer dielectric stacks, in-plane photonic crystal geometries, and careful management of radiation-confinement balance through intricate structure-field interactions. The design process demands deep expertise across semiconductor physics, nanophotonics, and computational electromagnetics—manually constructing finite-difference time-domain (FDTD) models, exploring high-dimensional parameter spaces, and iteratively optimizing geometric configurations to meet stringent performance targets such as Q-factor, modal volume, wavelength selectivity, and radiation loss. With each full-scale simulation requiring hours to days and effective optimization often demanding thousands of iterations, development cycles for advanced PCSELs can span months, limiting both innovation speed and accessibility beyond specialist research groups (Goodfellow, Bengio, and Courville 2016; Mnih et al. 2015; Theodoridis and Koutroumbas 2006; Li et al. 2022; Asano and Noda 2018; Li et al. 2021).

Recent advances in AI-driven inverse design have begun to address optimization efficiency. Deep neural networks, generative models, and reinforcement learning have successfully navigated non-convex design landscapes in nanocavities, metasurfaces, and photonic crystals, achieving improved Q-factors and tailored resonance properties. However, these approaches remain fundamentally constrained by a critical limitation: while AI algorithms accelerate optimization once a simulation environment exists, the underlying computational infrastructure—FDTD solvers, parameterization schemes, optimization environments, reward functions, and debugging protocols—must still be meticulously hand-crafted by domain experts. The effort required to construct this infrastructure often rivals the original optimization task itself, leaving inverse design workflows only partially automated. The bottleneck has shifted from optimization algorithms to infrastructure construction, severely limiting scalability as devices grow more complex and design constraints become more structured (Mirhoseini et al. 2021; Zhang et al. 2020; Chen et al. 2021b; Zeng et al. 2020; Ma et al. 2021; Molesky et al. 2018; So et al. 2020; Li et al. 2023b).

Meanwhile, large language models (LLMs) have demonstrated remarkable capabilities in automating complex scientific and engineering workflows. Beyond natural-language understanding, LLMs exhibit emergent abilities in code synthesis, multi-step reasoning, algorithmic planning, and recursive self-correction when provided with error feedback. Recent studies have shown that LLMs can generate hardware description languages, assist with embodied robotics

---

[*] These authors contributed equally.

[†] Corresponding author: zhangzy@cuhk.edu.cn, yinfeng@cuhk.edu.cn

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

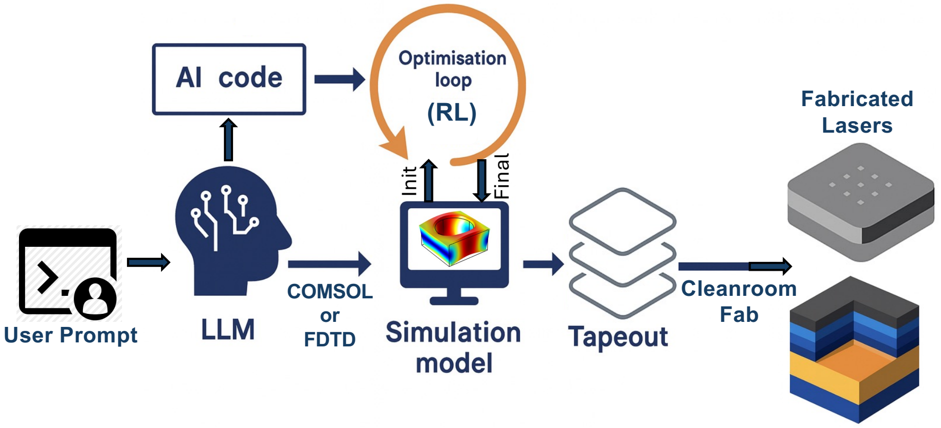


Figure 1: Overall workflow of LLM4Laser: LLM-driven automated laser design and optimization pipeline that embodies *human–AI co-design*. The human facilitator prompts the LLM to generate FDTD code for simulating the laser structure and AI (e.g., reinforcement learning (RL)) code for subsequent optimizations of the simulation model. The AI optimization process with RL is built upon an earlier work's L2DO framework (Li et al. 2023b). The final optimized laser design is then converted to CAD layout and prepared for tape-out and cleanroom fabrication to make the physical device (not implemented in this work).

control, and synthesize scientific computing code across diverse domains (Pearce, Tan, and Karri 2020; Blocklove et al. 2023; Stella, Della Santina, and Hughes 2023; Wang et al. 2023; Zhang et al. 2023; Ahn et al. 2022; Oskooi et al. 2010). These capabilities raise a compelling question for photonic device engineering:

> *Can large language models automate not only the optimization of photonic devices, but also the construction of the simulation and inverse-design workflow itself?*

Answering this question would mark a foundational shift from AI-assisted design—where humans build the infrastructure and AI optimizes within it—to truly autonomous photonic engineering systems capable of constructing, debugging, and operating complete computational workflows from natural-language specifications alone.

In this work, we introduce *LLM4Laser*, a human–AI codesign paradigm illustrated in Figure 1, where an LLM autonomously generates, refines, and integrates both the electromagnetic simulation code and the reinforcement-learning optimization algorithm required for PCSEL inverse design. We implement this framework using GPT-4, guiding it through natural-language conversations without providing code templates to: (1) conceptualize a valid PCSEL design strategy and identify relevant physical parameters, (2) generate complete three-dimensional FDTD simulation code using the MEEP package (Oskooi et al. 2010), including geometry construction and optical monitors, (3) implement a deep Q-learning (DQN) optimization pipeline capable of interacting with the FDTD simulator (Li et al. 2023b), and (4) iteratively debug and refine both components until the entire workflow executes successfully. The LLM demonstrates the ability to reason about electromagnetic structures, decompose complex design problems, revise incorrect logic, and integrate disparate software components into a coherent computational framework.

When executed on high-performance computing resources, the LLM-generated pipeline produces PCSEL designs with a Q-factor of 37,000 and normalized radiation loss nearly 18-fold lower than previous benchmarks, despite using a dramatically smaller device footprint (2μm in this work vs. 125μm in prior work (Chen et al. 2021a)). To further accelerate the design process, we fine-tuned a multi-modal LLM to predict optical properties directly from device images, enabling rapid surrogate evaluation without full electromagnetic simulation. This vision-based surrogate model complements the autonomous infrastructure construction, creating a complete end-to-end system for laser inverse design.

Broader Significance. These results affirmatively answer our research question: LLMs can serve not merely as coding assistants, but as autonomous photonic design agents capable of generating scientific software, performing physics-based reasoning, and orchestrating multi-step optimization workflows. This paradigm dramatically lowers the barrier to advanced nanophotonic device design, enabling researchers and engineers to rapidly prototype high-performance laser architectures without manually implementing numerical solvers or optimization algorithms. More broadly, our findings suggest a pathway toward self-driving laboratories for photonics and semiconductor devices (Li et al. 2023b), where LLM-driven agents integrate simulation, optimization, and experimental planning into unified autonomous cycles. As photonic devices continue to grow in complexity and performance demands intensify, such autonomous systems may become essential tools for accelerating innovation, reducing design costs, and exploring design spaces previously inaccessible to human designers.

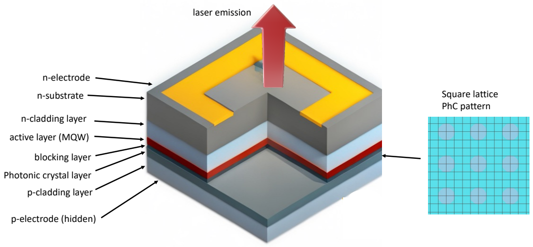


Figure 2: Photonic Crystal Surface Emitting Laser (PCSEL), with abundant applications in sensing, LiDAR, and telecommunications.

Methods

Overview of the LLM-Driven Design Framework

Our workflow integrates a large language model (GPT-4) with a physics-based electromagnetic simulator and a reinforcement-learning optimizer to autonomously inverse design PCSELs. Through natural-language interaction, a human designer prompts the LLM to conceptualize the problem, generate Meep-based FDTD simulation code, synthesize a DQN algorithm, and iteratively debug both components. The complete workflow consists of three stages, as shown in Figure 3:

1. conceptualization via human–LLM dialogue,
2. LLM-generated simulation and optimization code,
3. execution of the simulation–optimization loop on high-performance computing infrastructure.

Problem Definition and Design Objectives

A PCSEL (Figure 2) comprises multilayer semiconductor films whose emission characteristics—wavelength, Q-factor, divergence angle, lasing area, and radiation loss—are determined by vertical confinement and in-plane photonic crystal feedback. We denote the design parameters as (listed in Table 3):

$$\mathbf{x} = \{t_i, n_i, a, r, f, S\},$$

where t_i are layer thicknesses, n_i refractive indices, a is the lattice constant, r the air-hole radius, f the filling factor, and S the lattice or hole shape.

The inverse design problem is formulated as an optimization problem:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \; R(\mathbf{x}),$$

where $R(\mathbf{x})$ is a scalar reward (defined in Equation 8 in the Supp. Mat.) that incorporates several essential lasing characteristics (Q-factor, wavelength etc.). Due to non-convexity, one-to-many mapping, and expensive simulations, direct gradient-based optimization is intractable, motivating the use of reinforcement learning.

LLM-Guided Problem Conceptualization

Conceptualization Process The problem formulation phase proceeded through structured dialogue where we guided GPT-4 to understand PCSEL inverse design requirements (Figure 3, left column). Through progressive questioning across four domains—problem scoping (identifying PCSEL as target technology and key performance metrics), physical reasoning (decomposing design space into photonic crystal geometry, active layer, MQW structure, material selection), algorithmic planning (proposing RL, Bayesian optimization, evolutionary algorithms), and implementation strategy (structuring the workflow into FDTD simulation and RL optimization)—GPT-4 autonomously mapped abstract design goals to concrete technical requirements without human-provided algorithmic templates.

Prompting Strategy We interact with GPT-4 exclusively through natural-language prompts without supplying any code templates or algorithmic scaffolding. Our prompting strategy follows five principles:

1. **Open-ended conceptual queries** to stimulate idea generation.
2. **Divide-and-conquer prompting** to break the workflow into sub-tasks.
3. **Progressive refinement** to transition from conceptual to technical queries.
4. **Error-driven self-correction** by supplying only terminal error messages.
5. **Minimal human intervention** to preserve autonomy of the AI agent.

This encourages GPT-4 to structure, reason about, and assemble the simulation and optimization pipeline autonomously.

FDTD Simulation Framework (LLM-Generated)

Geometry Construction GPT-4 generated full Meep FDTD code for constructing a 3D PCSEL model (see Figure 4), including:

- five-layer stack: substrate, n-cladding, active layer, photonic crystal slab, and p-cladding,
- a 50×50 array of cylindrical shaped air holes in the photonic crystal layer,
- parameterized geometry enabling RL updates to (t_i, r, a, f, S).

Each dielectric layer is represented using:

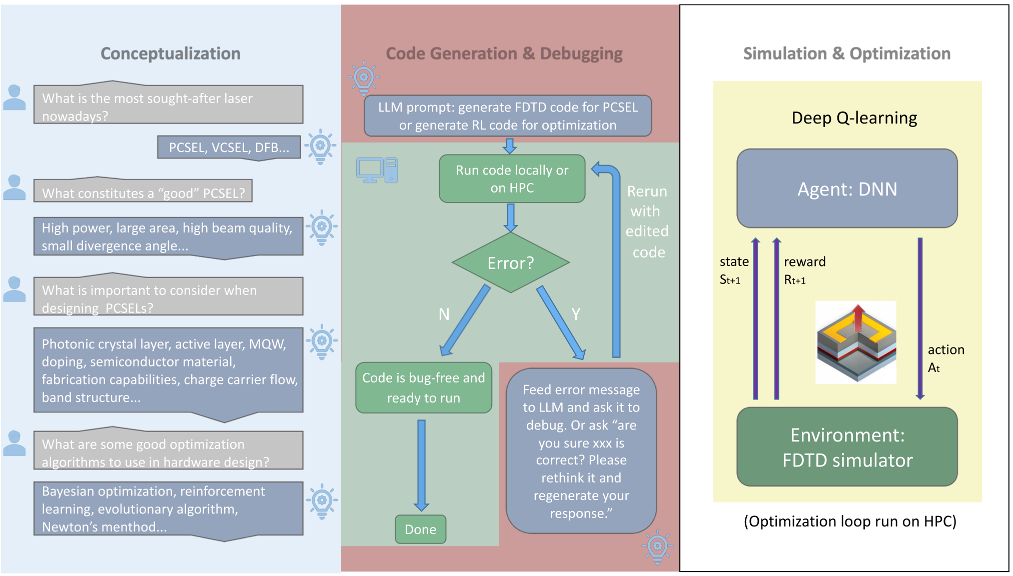
```
 mp.Block(Vector3(inf, inf, t_layer),
center=Vector3(x, y, z),
material=Medium(epsilon=n^2))
```

Air holes are instantiated as cylinders:

```
 mp.Cylinder(radius=r, height=t_PhC,
center=(x, y), material=Medium(epsilon=1))
```

Simulation Region and Boundary Conditions GPT-4 selected appropriate simulation components, including:

- perfectly matched layers (PML) along the vertical axis,
- Bloch periodic boundaries in-plane,


Conceptualization
Code Generation & Debugging
Simulation & Optimization
Run code locally or on HPC
Error?
N
Y
Rerun with edited code
Code is bug-free and ready to run
Done
Deep Q-learning
Agent: DNN
state S_{t+1}
reward R_{t+1}
action A_t
Environment: FDTD simulator
(Optimization loop run on HPC)

What is the most sought-after laser nowadays?

PCSEL, VCSEL, DFB...

What constitutes a "good" PCSEL?

High power, large area, high beam quality, small divergence angle...

What is important to consider when designing PCSELs?

Photonic crystal layer, active layer, MQW, doping, semiconductor material, fabrication capabilities, charge carrier flow, band structure...

What are some good optimization algorithms to use in hardware design?

Bayesian optimization, reinforcement learning, evolutionary algorithm, Newton's menthod...

LLM prompt: generate FDTD code for PCSEL or generate RL code for optimization

Feed error message to LLM and ask it to debug. Or ask "are you sure xxx is correct? Please rethink it and regenerate your response."

Figure 3: Three stages of the LLM4Laser workflow, which is a novel Human-AI co-design paradigm applying LLMs to laser design and optimization. **left column**: conceptualization, pictorial overview of the discussions and interactions between the human facilitator and the LLM agent, with the questions prompted by the human and the answers/solutions provided by the LLM. **middle column**: code generation and debugging, and **right column**: simulation and optimization via DQN. Optimization is run on high-performance computing (HPC) clusters for improved computational speed and output.

- grid resolution determined by feature sizes,
- a broadband Gaussian excitation in the active layer,
- frequency-domain flux monitors, far-field monitors, modal volume calculation, and Harminv modal analysis for Q-factor extraction.

Debugging Procedure Debugging followed an iterative loop (Figure 3, middle column):

1. the human runs the code and copies terminal errors;
2. GPT-4 diagnoses the cause and rewrites the problematic code block;
3. corrections are tested and the cycle repeats until convergence.

Typical errors included boundary condition mismatches, invalid flux region sizing, and incorrect vector dimensions. Most issues were resolved within 3–5 iterations (Figure 9).

Reinforcement Learning Optimization (LLM-Generated)

GPT-4 generated the RL code completely based on the DQN algorithm in Figure 10:

Environment Definition GPT-4 first generated an OpenAI Gym-style environment that wraps the FDTD simulator. The environment exposes:

$$\begin{aligned} s &= \{t_i, n_i, a, r, f, S\}, \\ a &\in \{\Delta t_i, \Delta n_i, \Delta a, \Delta r, \Delta f, \Delta S\}, \\ R &: \text{ defined in Supp. Mat.} \end{aligned}$$

where s is state, a is action (discrete perturbations to design parameters), and R is reward. For the transition, FDTD simulation evaluates the updated geometry after each iteration. GPT-4 autonomously determined how to structure the environment class, observation space, and internal parameter storage.

Deep Q-Learning Implementation GPT-4 autonomously generated the following DQN components (code is shown in Figure 8):

- four-layer fully connected policy network,
- target network for stabilized training,
- experience replay buffer,
- ϵ-greedy exploration policy,
- Q-value update rule:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

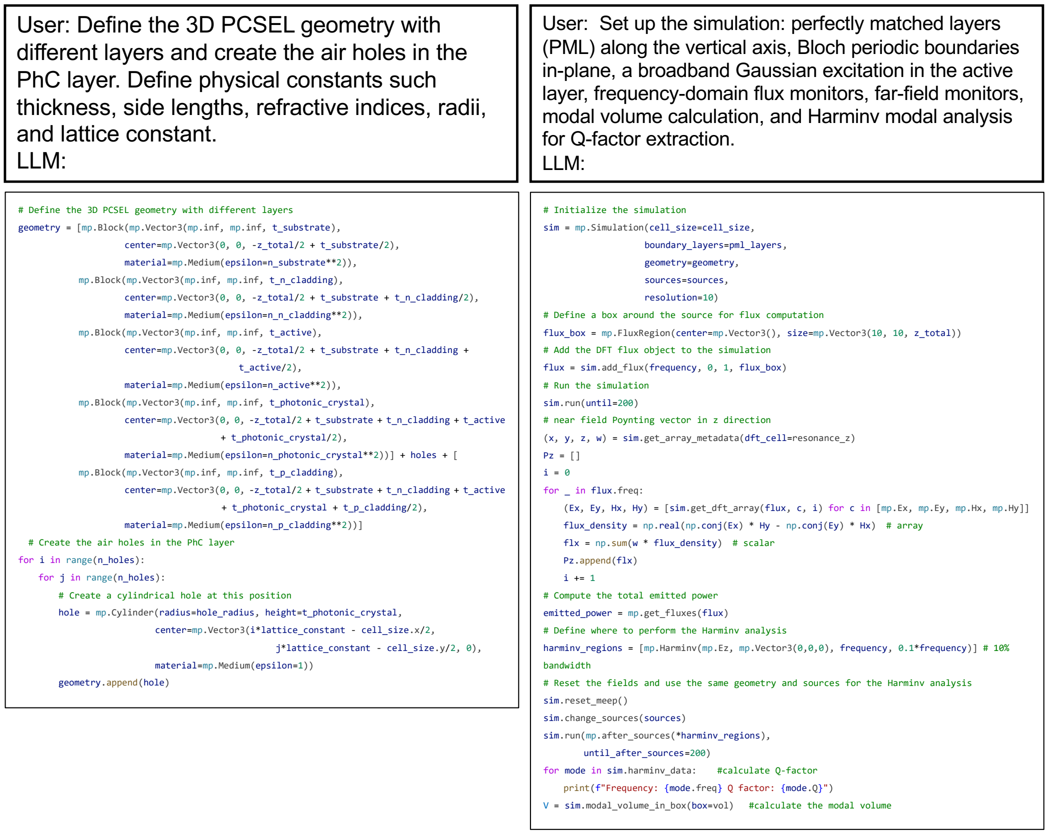
- PyTorch training loop with Huber or Smooth L1 loss.
- Gradient update pipeline with PyTorch

Tensor shape mismatches, missing imports, and attribute errors were corrected via the same error-driven conversation loop used for FDTD debugging.

Integration and Optimization Loop

After the FDTD and RL modules were validated independently, GPT-4 generated a master script that:

1. initializes the PCSEL geometry,
2. executes the FDTD simulation,
3. extracts optical metrics,



User: Define the 3D PCSEL geometry with different layers and create the air holes in the PhC layer. Define physical constants such thickness, side lengths, refractive indices, radii, and lattice constant.
LLM:

```
# Define the 3D PCSEL geometry with different layers
geometry = [mp.Block(mp.Vector3(mp.inf, mp.inf, t_substrate),
                    center=mp.Vector3(0, 0, -z_total/2 + t_substrate/2),
                    material=mp.Medium(epsilon=n_substrate**2)),
           mp.Block(mp.Vector3(mp.inf, mp.inf, t_n_cladding),
                    center=mp.Vector3(0, 0, -z_total/2 + t_substrate + t_n_cladding/2),
                    material=mp.Medium(epsilon=n_n_cladding**2)),
           mp.Block(mp.Vector3(mp.inf, mp.inf, t_active),
                    center=mp.Vector3(0, 0, -z_total/2 + t_substrate + t_n_cladding +
                                          t_active/2),
                    material=mp.Medium(epsilon=n_active**2)),
           mp.Block(mp.Vector3(mp.inf, mp.inf, t_photonic_crystal),
                    center=mp.Vector3(0, 0, -z_total/2 + t_substrate + t_n_cladding + t_active
                                       + t_photonic_crystal/2),
                    material=mp.Medium(epsilon=n_photonic_crystal**2))] + holes + [
           mp.Block(mp.Vector3(mp.inf, mp.inf, t_p_cladding),
                    center=mp.Vector3(0, 0, -z_total/2 + t_substrate + t_n_cladding + t_active
                                       + t_photonic_crystal + t_p_cladding/2),
                    material=mp.Medium(epsilon=n_p_cladding**2))]
  # Create the air holes in the PhC layer
for i in range(n_holes):
    for j in range(n_holes):
        # Create a cylindrical hole at this position
        hole = mp.Cylinder(radius=hole_radius, height=t_photonic_crystal,
                          center=mp.Vector3(i*lattice_constant - cell_size.x/2,
                                                 j*lattice_constant - cell_size.y/2, 0),
                          material=mp.Medium(epsilon=1))
        geometry.append(hole)
```

User: Set up the simulation: perfectly matched layers (PML) along the vertical axis, Bloch periodic boundaries in-plane, a broadband Gaussian excitation in the active layer, frequency-domain flux monitors, far-field monitors, modal volume calculation, and Harminv modal analysis for Q-factor extraction.
LLM:

```
# Initialize the simulation
sim = mp.Simulation(cell_size=cell_size,
                   boundary_layers=pml_layers,
                   geometry=geometry,
                   sources=sources,
                   resolution=10)
# Define a box around the source for flux computation
flux_box = mp.FluxRegion(center=mp.Vector3(), size=mp.Vector3(10, 10, z_total))
# Add the DFT flux object to the simulation
flux = sim.add_flux(frequency, 0, 1, flux_box)
# Run the simulation
sim.run(until=200)
# near field Poynting vector in z direction
(x, y, z, w) = sim.get_array_metadata(dft_cell=resonance_z)
Pz = []
i = 0
for _ in flux.freq:
    (Ex, Ey, Hx, Hy) = [sim.get_dft_array(flux, c, i) for c in [mp.Ex, mp.Ey, mp.Hx, mp.Hy]]
    flux_density = np.real(np.conj(Ex) * Hy - np.conj(Ey) * Hx)  # array
    flx = np.sum(w * flux_density)  # scalar
    Pz.append(flx)
    i += 1
# Compute the total emitted power
emitted_power = mp.get_fluxes(flux)
# Define where to perform the Harminv analysis
harminv_regions = [mp.Harminv(mp.Ez, mp.Vector3(0,0,0), frequency, 0.1*frequency)] # 10%
bandwidth
# Reset the fields and use the same geometry and sources for the Harminv analysis
sim.reset_meep()
sim.change_sources(sources)
sim.run(mp.after_sources(*harminv_regions),
       until_after_sources=200)
for mode in sim.harminv_data:    #calculate Q-factor
    print(f"Frequency: {mode.freq} Q factor: {mode.Q}")
V = sim.modal_volume_in_box(box=vol)   #calculate the modal volume
```

Figure 4: Code generated by GPT-4 for FDTD simulation of PCSEL using the meep package. Left: geometry setup section, right: simulation setup and calculations section. The code shown here is the final version that runs successfully after several rounds of debugging.

4. updates the RL agent,
5. repeats until convergence.

Training was conducted on an HPC cluster with 20 CPU cores and 2 CUDA-enabled GPUs (Figure 3, right column). Typical optimization runtimes were 3–5 days.

Human Intervention Quantification

A key metric of interest is the degree of autonomy. Across the entire workflow:

- **FDTD code**: 0 lines written by humans; 5–7 GPT debugging rounds.
- **DQN code**: 0 lines written by humans; 4–6 debugging rounds.
- **Integration code**: fully generated by GPT.

Total human effort was approximately 4-5 hours of prompting and execution setup, compared with the typical multi-month engineering cycle for manually constructing photonic design pipelines.

Results & Discussion

LLM-Generated Optimization Pipeline Demonstrates Stable Convergence

The GPT-4-generated reinforcement learning pipeline exhibited stable learning dynamics across all optimization trials. Figure 5 presents training curves from three independent DQN runs, each executing 500-step episodes over a continuous 5-day period on high-performance computing infrastructure.

The average reward per episode increased substantially during the initial 100 episodes despite stochastic exploration, confirming that the value function approximation converged toward higher-performing design regions. Maximum rewards per episode rose more rapidly and stabilized earlier, indicating that the agent consistently discovered and revisited high-quality configurations. The logarithmic vertical scaling reveals two distinct learning phases: rapid initial improvement as the agent explores the design space, followed by asymptotic convergence as it refines promising candidates.

These behaviors confirm that the LLM-generated codebase contains the essential algorithmic components for stable reinforcement learning—target networks, experience

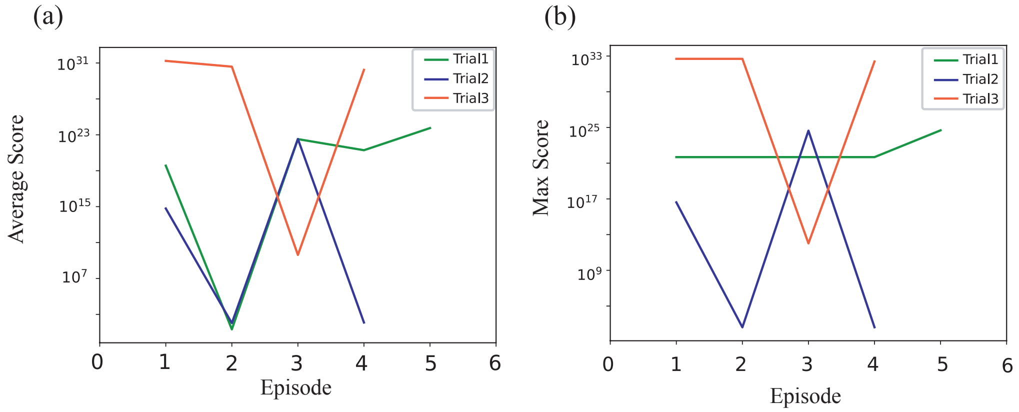


Figure 5: Learning curves of training the DQN to optimize PCSEL, plotted as scores vs. episodes. (a) Average score of each episode; (b) Maximum score of each episode. Each episode contains a horizon of 500 steps. Vertical axes are plotted in log scale.

replay buffers, and properly configured neural architectures—without human-provided templates. Even under the substantial numerical noise inherent to FDTD-based reward evaluation, the agent reliably identified ascending trajectories in performance space, demonstrating the functional completeness of GPT-4-synthesized code.

Metrics	Optimized PCSEL	Literature data
Lasing wavelength (nm)	1383	948
Q factor ↑	**37000**	2900
Normalized loss (1/nm) ↓	$\mathbf{4.3 \times 10^{-7}}$	7.7×10^{-6}
Device Size (μm)	**2**	125

Table 1: Optical attributes of the DQN-optimized PCSEL. The rightmost column reports the best literature data (Chen et al. 2021a) for a direct metric comparison. Normalized loss per unit length (Kalapala et al. 2022) and device size are additionally reported for a more fair and comprehensive comparison.

Optimization Produces High-Performance PCSEL Designs

Comparison With State-of-the-Art Experimental Devices The optimized PCSEL structures obtained from the DQN trials exhibit remarkable optical performance (Table 1). Despite using a dramatically smaller computational footprint (side length of 2.0 μm, compared to 125 μm in leading experimental demonstrations (Chen et al. 2021a)), the LLM-generated pipeline produced:

- **Operating wavelength:** 1383 nm, within acceptable tolerance of the 1310 nm target.
- **Q-factor:** 37,000, exceeding the best reported experimental PCSEL values ($\sim$ 2900) by more than an order of magnitude.
- **Normalized radiation loss:** 4.3×10^{-7}, nearly 18× lower than the literature benchmark.

These improvements directly reflect stronger vertical confinement, reduced in-plane scattering, and enhanced feedback within the photonic crystal slab. The high Q-factor in particular indicates that the reinforcement-learning agent – operating with code entirely synthesized by GPT-4 – successfully navigated the non-convex design landscape to identify superior geometric configurations. For a comprehensive but not exhaustive literature survey of advancements in PCSEL over the past decade (Hirose et al. 2014; Noda et al. 2017; Yoshida et al. 2019; Nishimoto et al. 2013; Li et al. 2023a; Inoue et al. 2019; Itoh et al. 2020; Gondaira et al. 2016; Kurosaka et al. 2008; Streifer, Scifres, and Burnham 1977; Peng et al. 2011; Nishimoto, Maekawa, and Noda 2017; Inoue et al. 2020; Chen et al. 2021a), refer to Table 6 of Appendix E in the Supp. Mat.

Why GPT-4 Outperforms Classical Automation Pipelines

Autonomous Integration Across Physics and Optimization Unlike conventional inverse-design frameworks that optimize parameters within a human-constructed simulation environment, GPT-4 autonomously generated:

1. the full Meep-based FDTD geometry and simulation engine,
2. all electromagnetic monitors (flux, near-to-far field, Harminv),
3. the reinforcement-learning environment wrapper,
4. the DQN architecture, training loop, replay buffer, and exploration strategy,
5. the integration logic linking both modules.

Remarkably, no human-provided templates or code scaffolds were used. This demonstrates that modern LLMs can coordinate multiple layers of the computational design stack — physics modeling, numerical simulation, and optimization — a task traditionally requiring cross-domain expertise.

Self-Correction as a Computational Mechanism GPT-4 repeatedly interpreted raw terminal error messages and autonomously corrected issues such as:

- invalid PML layer declarations,
- incorrect flux-region dimensionality,
- environment-attribute mismatches,
- tensor shape inconsistencies in PyTorch.

This closed-loop debugging capability (Figure 3, middle column). dramatically reduces human effort and positions LLMs as viable agents for maintaining and evolving scientific software.

Physics Insights Revealed Through Optimization

The agent's learned design behaviors highlight several underlying PCSEL physical principles:

1. **Q-factor scaling:** maximizing vertical confinement through slab-thickness and hole-radius adjustments.
2. **Loss suppression:** discovering configurations that approximate bound states in the continuum (BIC-like regimes).
3. **Wavelength control:** exploiting refractive-index and lattice-constant interactions to tune band-edge modes.
4. **Mode selectivity:** favoring structures that promote stable single-mode operation.

These insights validate that LLM-generated RL pipelines are capable of uncovering physically interpretable principles without explicit human instruction.

Broader Implications: Toward Autonomous Photonic Design Agents

This study illustrates that LLMs can serve as autonomous scientific software architects, generating full simulation–optimization pipelines that rival human-engineered systems. The implications are substantial:

- Accurate photonic device design without expert coding
- Rapid exploration of large design spaces
- Automated debugging and self-improving workflows
- Scalable frameworks for lasers, modulators, metasurfaces, and waveguides
- Foundational step toward self-driving laboratories

This work represents not only a technical demonstration, but also a blueprint for a future in which AI agents autonomously reason about, simulate, and optimize complex photonic structures—fundamentally reshaping the pace and scope of photonic device engineering.

Multi-Modal LLM for Image-Based Prediction of PCSEL Properties

In addition to constructing a fully automated inverse-design pipeline, we further investigated whether large language models can learn the mapping from fabricated or simulated laser images to their corresponding optical properties. The ability to map structural visual information to device performance is essential for rapid screening, fabrication-aware optimization, and closing the loop between simulation and experiment in autonomous design systems (Kadian et al. 2020). To this end, we fine-tuned a multi-modal large vision-language model (VLM) (Hu et al. 2024; Zhang et al. 2024) using paired datasets of PCSEL images and their corresponding optical characteristics, including resonant wavelength, Q-factor, lasing threshold, and divergence angle etc. (Figure 6). To the best of our knowledge, this is the first demonstration of VLM for laser modelling.

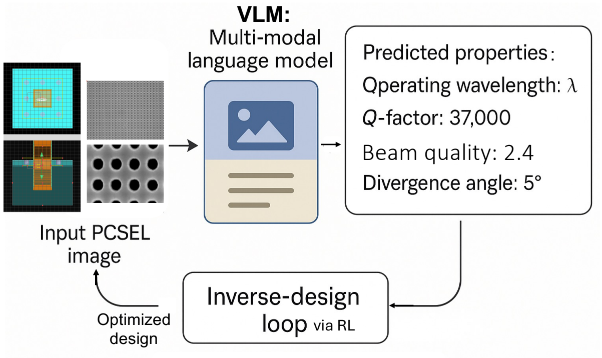


Figure 6: VLM for prediction of laser properties from visual inputs. VLM-predicted properties are then fed to the inverse design loop for optimization.

Image Type	Quantity	Source of Data
Simulation	800	In-house
SEM	100	In-house + Literature extraction

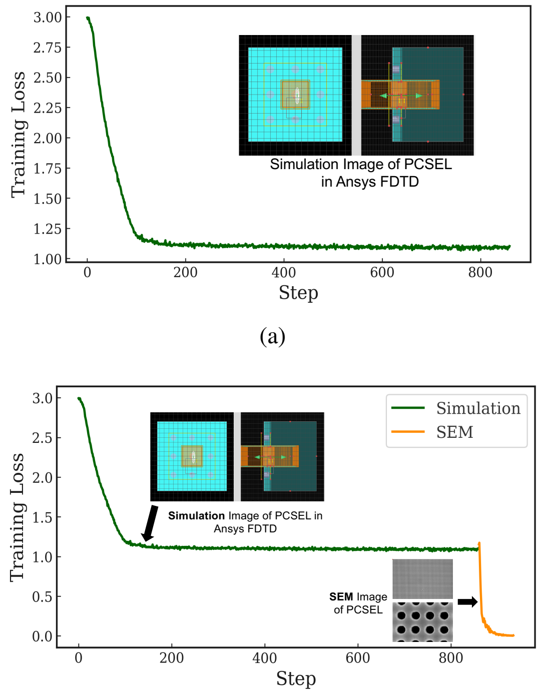
Table 2: Breakdown of the image dataset we curated for fine-tuning the VLM model. Simulation images are taken in-house within the Ansys Lumerical FDTD environment, while SEM images are partially in-house and partially extracted from literature.

Model Architecture and Training

The VLM model accepts a PCSEL image as input—such as a top-down photonic-crystal layer image taken within the simulation environment, or a real scanning electron microscope (SEM) image—and produces textual predictions of the device's optical properties. We initialize the model from a large VLM (Qwen2.5VL) and fine-tune it using supervised learning over a curated dataset (Table 2) of (image, property) pairs. The dataset contains both synthetic images generated from FDTD simulations and experimental SEM images of fabricated PCSELs (partially extracted from literature), enabling the model to learn structural–optical correlations that generalize across idealized and real-world devices.

During post-training (Figure 7), the model learns to associate geometric cues such as lattice periodicity, hole radius, filling factor, and structural symmetry with corresponding optical responses. The multi-modal encoder also develops robustness to imaging noise, illumination gradients, and common SEM artifacts, allowing deployment in laboratory environments without specialized pre-processing.

Prediction Performance and Physical Insights



(a)

(b)

Figure 7: a) Post-training the VLM model on Simulation images only (green curve); b) Continued the post-training on SEM images (orange curve).

The fine-tuned model accurately predicts PCSEL performance across a range of unseen structures. In particular, it reliably infers the following lasing characteristics from images: Resonant wavelength, Q-factor, FWHM, divergence angle, beam quality, free-spectral range (FSR), lasing threshold, peak lasing power, lasing area.

Although trained without explicit physics constraints, the model discovers the same geometric–optical relationships used by human experts, demonstrating that multi-modal LLMs can extract meaningful photonic descriptors directly from visual data.

Role in Autonomous Inverse Design

Integrating image-based prediction into the LLM-driven inverse-design framework enables several key capabilities:

1. **Surrogate modeling**: the model provides rapid approximations of FDTD outcomes, accelerating reinforcement-learning or Bayesian-optimization loops by orders of magnitude (Alizadeh, Allen, and Mistree 2020);
2. **Real-time experimental feedback**: SEM images of fabricated devices can be fed directly into the model, enabling real-time evaluation and adaptive redesign;
3. **Fabrication tolerance awareness**: the model identifies deviations from target geometries and anticipates their optical impact (Xin et al. 2024);
4. **Unified perception–design pipeline**: combining visual understanding with the LLM-generated simulation and optimization logic forms a closed-loop AI agent capable of perception, prediction, and inverse design.

Conclusion

We demonstrate that large language models (LLMs) can function as autonomous design partners, constructing complete simulation–optimization pipelines for complex photonic devices using natural-language interaction alone. Without any template code or algorithmic scaffolding, GPT-4 generated fully functional FDTD simulation scripts, DQN optimization logic, and an end-to-end framework for inverse design of PCSELs, which it further debugged and refined through iterative dialogue. When deployed on HPC resources, the LLM-generated pipeline identified PCSEL geometries that surpass state-of-the-art experimental benchmarks, achieving over an order-of-magnitude improvement in Q-factor and substantially reduced radiation loss within a constrained footprint. The learned strategies align with established physical design principles, demonstrating both code correctness and the effectiveness of reinforcement learning in highly non-convex photonic design spaces. Beyond performance gains, this work reveals a fundamental capability of LLMs to bridge conceptual reasoning and technical implementation, enabling autonomous generation and operation of complex simulation–optimization loops. We further show that vision-language models can predict laser properties directly from device images, extending LLM capabilities beyond code synthesis to direct interpretation of photonic structures. Practical limitations remain: full 3D FDTD simulations are computationally expensive, large-area devices are required to accurately capture far-field properties, and fabrication constraints are not yet incorporated. Integrating fabrication-aware design rules, multi-fidelity or differentiable Maxwell solvers, and more sample-efficient optimization strategies offers a clear path forward. Overall, this study advances photonic inverse design toward self-driving laboratories, where AI agents autonomously conceive, simulate, and optimize next-generation photonic devices.

Acknowledgments

This research is supported by National Natural Science Foundation of China (NSFC) under Grant No.62174144 and 62271433, Shenzhen Science and Technology Program under Grant No.JCYJ20210324115605016, No.JCYJ20210324120204011, No.JSGG20210802153540017, No.JCYJ20220818102214030, No.KJZD20230923115114027, Guangdong Key Laboratory of Optoelectronic Materials and Chips under Grant No.2022KSYS014, Shenzhen Key Laboratory Project under Grant No.ZDSYS201603311644527, Longgang Key Laboratory Project under Grant No.ZSYS2017003 and No.LGKCZSYS2018000015, and the IBM-Illinois Discovery Accelerator Institute at UIUC. The devices were partially fabricated in Core Research Facilities (CRF) at SUSTech and the Materials Characterization and Preparation Center at CUHKSZ. The authors like to thank staff members at ITSO of CUHKSZ for their persistent support for our high-performance computing needs. The authors would like to thank Mr. Hong Tang for his assistance with GPU CUDA and workstations.

Code and Data Availability

The code accompanying this paper is available at: https://github.com/renjieli08/LLM-for-Photonics, and the data can be obtained from the corresponding author upon reasonable requests.

References

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Alizadeh, R.; Allen, J. K.; and Mistree, F. 2020. Managing computational complexity using surrogate models: a critical review. *Research in Engineering Design*, 31(3): 275–298.

Asano, T.; and Noda, S. 2018. Optimization of photonic crystal nanocavities based on deep learning. *Optics express*, 26(25): 32704–32717.

Blocklove, J.; Garg, S.; Karri, R.; and Pearce, H. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. *arXiv preprint arXiv:2305.13243*.

Chen, L.-R.; Hong, K.-B.; Huang, K.-C.; Yen, H.-T.; and Lu, T.-C. 2021a. Improvement of output efficiency of p-face up photonic-crystal surface-emitting lasers. *Optics Express*, 29(7): 11293–11300.

Chen, T.-S.; Li, Z.-L.; Hsu, M.-Y.; Lin, G.; and Lin, S.-D. 2017. Photonic crystal surface emitting lasers with quantum dot active region. *Journal of Lightwave Technology*, 35(20): 4547–4552.

Chen, X.; Xie, Y.; Sheng, Y.; Tang, H.; Wang, Z.; Wang, Y.; Wang, Y.; Liao, F.; Ma, J.; Guo, X.; et al. 2021b. Wafer-scale functional circuits based on two dimensional semiconductors with fabrication optimized by machine learning. *Nature Communications*, 12(1): 5953.

Chen, Z.; Yin, X.; Li, P.; Zheng, Z.; Zhang, Z.; Wang, F.; and Peng, C. 2022. Analytical theory of finite-size photonic crystal slabs near the band edge. *Optics Express*, 30(9): 14033–14047.

Gondaira, K.; Ishizaki, K.; Kitano, K.; Asano, T.; and Noda, S. 2016. Control of radiation angle by introducing symmetric end structure to oblique waveguide in three-dimensional photonic crystal. *Optics Express*, 24(12): 13518–13526.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Hirose, K.; Liang, Y.; Kurosaka, Y.; Watanabe, A.; Sugiyama, T.; and Noda, S. 2014. Watt-class high-power, high-beam-quality photonic-crystal lasers. *Nature photonics*, 8(5): 406–411.

Hsu, M.-Y.; Lin, G.; and Pan, C.-H. 2017. Electrically injected 1.3-μm quantum-dot photonic-crystal surface-emitting lasers. *Optics Express*, 25(26): 32697–32704.

Hu, W.; Xu, Y.; Li, Y.; Li, W.; Chen, Z.; and Tu, Z. 2024. Bliva: A simple multimodal llm for better handling of text-rich visual questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 2256–2264.

Inoue, T.; Morita, R.; Yoshida, M.; De Zoysa, M.; Tanaka, Y.; and Noda, S. 2019. Comprehensive analysis of photonic-crystal surface-emitting lasers via time-dependent three-dimensional coupled-wave theory. *Physical Review B*, 99(3): 035308.

Inoue, T.; Yoshida, M.; Zoysa, M.; Ishizaki, K.; and Noda, S. 2020. Design of photonic-crystal surface-emitting lasers with enhanced in-plane optical feedback for high-speed operation. *Optics Express*, 28(4): 5050–5057.

Itoh, Y.; Kono, N.; Fujiwara, N.; Yagi, H.; Katsuyama, T.; Kitamura, T.; Fujii, K.; Ekawa, M.; Shoji, H.; Inoue, T.; et al. 2020. Continous-wave lasing operation of 1.3-μm wavelength InP-based photonic crystal surface-emitting lasers using MOVPE regrowth. *Optics Express*, 28(24): 35483–35489.

Kadian, A.; Truong, J.; Gokaslan, A.; Clegg, A.; Wijmans, E.; Lee, S.; Savva, M.; Chernova, S.; and Batra, D. 2020. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4): 6670–6677.

Kalapala, A.; Song, A. Y.; Pan, M.; Gautam, C.; Overman, L.; Reilly, K.; Rotter, T. J.; Balakrishnan, G.; Gibson, R.; Bedford, R.; et al. 2022. Scaling challenges in high power photonic crystal surface-emitting lasers. *IEEE Journal of Quantum Electronics*, 58(4): 1–9.

Kurosaka, Y.; Sakai, K.; Miyai, E.; and Noda, S. 2008. Controlling vertical optical confinement in two-dimensional surface-emitting photonic-crystal lasers by shape of air holes. *Optics express*, 16(22): 18485–18494.

Li, H.; Tang, M.; Zhou, T.; Xie, W.; Li, R.; Gong, Y.; Martin, M.; Baron, T.; Chen, S.; Liu, H.; et al. 2023a. Monolithically integrated photonic crystal surface emitters on silicon with a vortex beam by using bound states in the continuum. *Optics Letters*, 48(7): 1702–1705.

Li, R.; Gu, X.; Li, K.; Huang, Y.; Li, Z.; and Zhang, Z. 2021. Deep learning-based modeling of photonic crystal nanocavities. *Optical Materials Express*, 11(7): 2122–2133.

Li, R.; Gu, X.; Shen, Y.; Li, K.; Li, Z.; and Zhang, Z. 2022. Smart and Rapid Design of Nanophotonic Structures by an Adaptive and Regularized Deep Neural Network. *Nanomaterials*, 12(8): 1372.

Li, R.; Zhang, C.; Xie, W.; Gong, Y.; Ding, F.; Dai, H.; Chen, Z.; Yin, F.; and Zhang, Z. 2023b. Deep reinforcement learning empowers automated inverse design and optimization of photonic crystals for nanoscale laser cavities. *Nanophotonics*, 12(2): 319–334.

Ma, W.; Liu, Z.; Kudyshev, Z. A.; Boltasseva, A.; Cai, W.; and Liu, Y. 2021. Deep learning for the design of photonic structures. *Nature Photonics*, 15(2): 77–90.

Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J. W.; Songhori, E.; Wang, S.; Lee, Y.-J.; Johnson, E.; Pathak, O.; Nazi, A.; et al. 2021. A graph placement methodology for fast chip design. *Nature*, 594(7862): 207–212.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Molesky, S.; Lin, Z.; Piggott, A. Y.; Jin, W.; Vucković, J.; and Rodriguez, A. W. 2018. Inverse design in nanophotonics. *Nature Photonics*, 12(11): 659–670.

Nishimoto, M.; Ishizaki, K.; Maekawa, K.; Kitamura, K.; and Noda, S. 2013. Air-hole retained growth by molecular beam epitaxy for fabricating GaAs-based photonic-crystal lasers. *Applied physics express*, 6(4): 042002.

Nishimoto, M.; Maekawa, K.; and Noda, S. 2017. Design of photonic-crystal surface-emitting lasers with circularly-polarized beam. *Optics Express*, 25(6): 6104–6111.

Noda, S.; Kitamura, K.; Okino, T.; Yasuda, D.; and Tanaka, Y. 2017. Photonic-crystal surface-emitting lasers: Review and introduction of modulated-photonic crystals. *IEEE Journal of Selected Topics in Quantum Electronics*, 23(6): 1–7.

Ohnishi, D.; Okano, T.; Imada, M.; and Noda, S. 2004. Room temperature continuous wave operation of a surface-emitting two-dimensional photonic crystal diode laser. *optics express*, 12(8): 1562–1568.

Oskooi, A. F.; Roundy, D.; Ibanescu, M.; Bermel, P.; Joannopoulos, J. D.; and Johnson, S. G. 2010. MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method. *Computer Physics Communications*, 181(3): 687–702.

Pearce, H.; Tan, B.; and Karri, R. 2020. Dave: Deriving automatically verilog from English. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 27–32.

Peng, C.; Liang, Y.; Sakai, K.; Iwahashi, S.; and Noda, S. 2011. Coupled-wave analysis for photonic-crystal surface-emitting lasers on air holes with arbitrary sidewalls. *Optics Express*, 19(24): 24672–24686.

Reilly, K. J.; Kalapala, A.; Yeom, S.; Addamane, S. J.; Renteria, E.; Zhou, W.; and Balakrishnan, G. 2020. Epitaxial regrowth and hole shape engineering for photonic crystal surface emitting lasers (PCSELs). *Journal of Crystal Growth*, 535: 125531.

Sakai, K.; Miyai, E.; Sakaguchi, T.; Ohnishi, D.; Okano, T.; and Noda, S. 2005. Lasing band-edge identification for a surface-emitting photonic crystal laser. *IEEE Journal on Selected Areas in Communications*, 23(7): 1335–1340.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175.

So, S.; Badloe, T.; Noh, J.; Bravo-Abad, J.; and Rho, J. 2020. Deep learning enabled inverse design in nanophotonics. *Nanophotonics*, 9(5): 1041–1057.

Stella, F.; Della Santina, C.; and Hughes, J. 2023. How can LLMs transform the robotic design process? *Nature machine intelligence*, 5(6): 561–564.

Streifer, W.; Scifres, D.; and Burnham, R. 1977. Coupled wave analysis of DFB and DBR lasers. *IEEE Journal of Quantum Electronics*, 13(4): 134–141.

Theodoridis, S.; and Koutroumbas, K. 2006. *Pattern recognition*. Elsevier.

Wang, Z.; Cai, S.; Liu, A.; Ma, X.; and Liang, Y. 2023. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. *arXiv preprint arXiv:2302.01560*.

Wang, Z.; Tong, C.; Wang, L.; Lu, H.; Tian, S.; and Wang, L. 2021. Photonic crystal surface emitting laser operating in pulse-periodic regime with ultralow divergence angle. In *Photonics*, volume 8, 323. MDPI.

Wang, Z.; Wang, P.; Lu, H.; Meng, B.; Wang, Y.; Tong, C.; and Wang, L. 2022. Symmetry criterion and far-field control of photonic-crystal surface-emitting lasers. *Applied Sciences*, 12(20): 10581.

Xin, Q.; Li, R.; Wei, D.; and Zhang, Z. 2024. Filling the simulation-to-reality gap: High-degree-of-freedom AI-optimized photonic crystal nanobeam resonators with fabrication tolerance. *Optics Letters*, 49(15): 4401–4404.

Yoshida, M.; De Zoysa, M.; Ishizaki, K.; Tanaka, Y.; Kawasaki, M.; Hatsuda, R.; Song, B.; Gelleta, J.; and Noda, S. 2019. Double-lattice photonic-crystal resonators enabling high-brightness semiconductor lasers with symmetric narrow-divergence beams. *Nature materials*, 18(2): 121–128.

Zeng, Y.; Chattopadhyay, U.; Zhu, B.; Qiang, B.; Li, J.; Jin, Y.; Li, L.; Davies, A. G.; Linfield, E. H.; Zhang, B.; et al. 2020. Electrically pumped topological laser with valley edge modes. *Nature*, 578(7794): 246–250.

Zhang, C.; Yang, K.; Hu, S.; Wang, Z.; Li, G.; Sun, Y.; Zhang, C.; Zhang, Z.; Liu, A.; Zhu, S.-C.; Chang, X.; Zhang, J.; Yin, F.; Liang, Y.; and Yang, Y. 2023. ProAgent: Building Proactive Cooperative AI with Large Language Models. *arXiv preprint arXiv:2308.11339*.

Zhang, J.; Huang, J.; Jin, S.; and Lu, S. 2024. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8): 5625–5644.

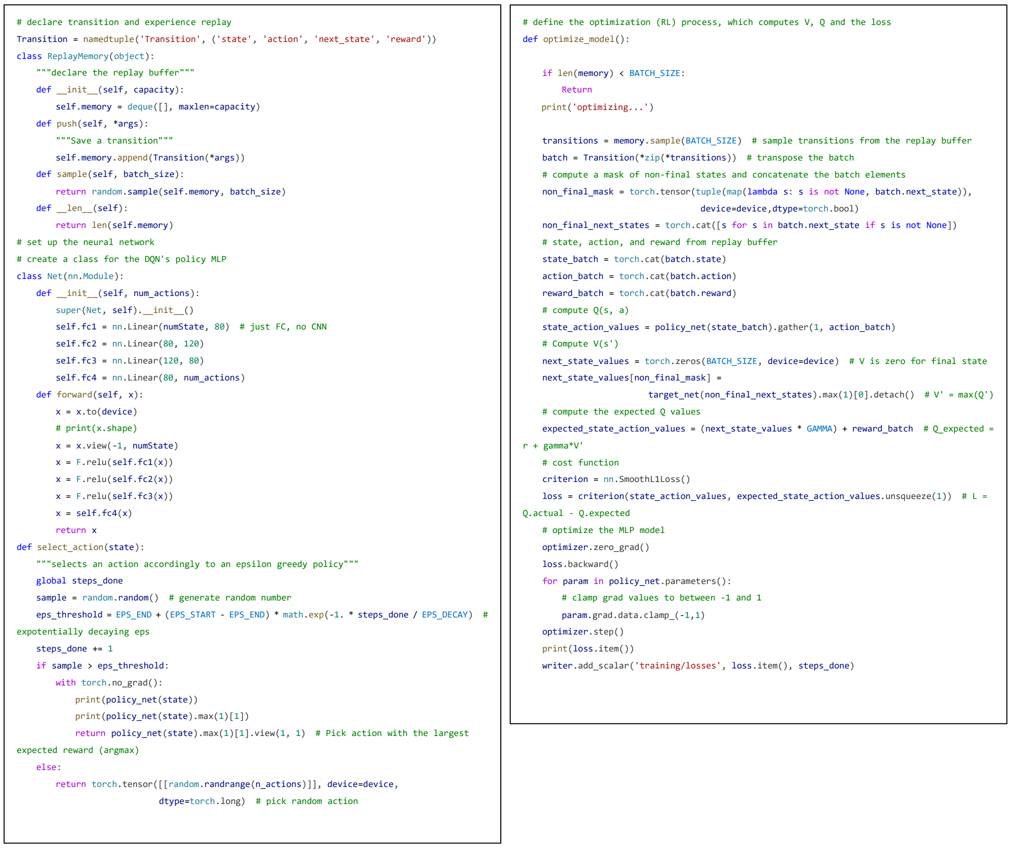
Zhang, W.; Gao, B.; Tang, J.; Yao, P.; Yu, S.; Chang, M.-F.; Yoo, H.-J.; Qian, H.; and Wu, H. 2020. Neuro-inspired computing chips. *Nature electronics*, 3(7): 371–382.

Zhou, T.; Tang, M.; Xiang, G.; Xiang, B.; Hark, S.; Martin, M.; Baron, T.; Pan, S.; Park, J.-S.; Liu, Z.; et al. 2020. Continuous-wave quantum dot photonic crystal lasers grown on on-axis Si (001). *Nature communications*, 11(1): 1–7.

LLM4Laser: Large Language Models Automate the Inverse Design of Lasers
Supplementary Material

Appendix A: Additional Code Generation and Debugging Demonstrations by GPT

The code generated by GPT-4 for RL is shown in Figure 8 and the debugging iterations with GPT is shown in Figure 9.



```
# declare transition and experience replay
Transition = namedtuple('Transition', ('state', 'action', 'next_state', 'reward'))
class ReplayMemory(object):
    """declare the replay buffer"""
    def __init__(self, capacity):
        self.memory = deque([], maxlen=capacity)
    def push(self, *args):
        """Save a transition"""
        self.memory.append(Transition(*args))
    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)
    def __len__(self):
        return len(self.memory)
# set up the neural network
# create a class for the DQN's policy MLP
class Net(nn.Module):
    def __init__(self, num_actions):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(numState, 80)  # just FC, no CNN
        self.fc2 = nn.Linear(80, 120)
        self.fc3 = nn.Linear(120, 80)
        self.fc4 = nn.Linear(80, num_actions)
    def forward(self, x):
        x = x.to(device)
        # print(x.shape)
        x = x.view(-1, numState)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
def select_action(state):
    """selects an action accordingly to an epsilon greedy policy"""
    global steps_done
    sample = random.random()  # generate random number
    eps_threshold = EPS_END + (EPS_START - EPS_END) * math.exp(-1. * steps_done / EPS_DECAY)  #
expotentially decaying eps
    steps_done += 1
    if sample > eps_threshold:
        with torch.no_grad():
            print(policy_net(state))
            print(policy_net(state).max(1)[1])
            return policy_net(state).max(1)[1].view(1, 1)  # Pick action with the largest
expected reward (argmax)
    else:
        return torch.tensor([[random.randrange(n_actions)]], device=device,
                            dtype=torch.long)  # pick random action
```

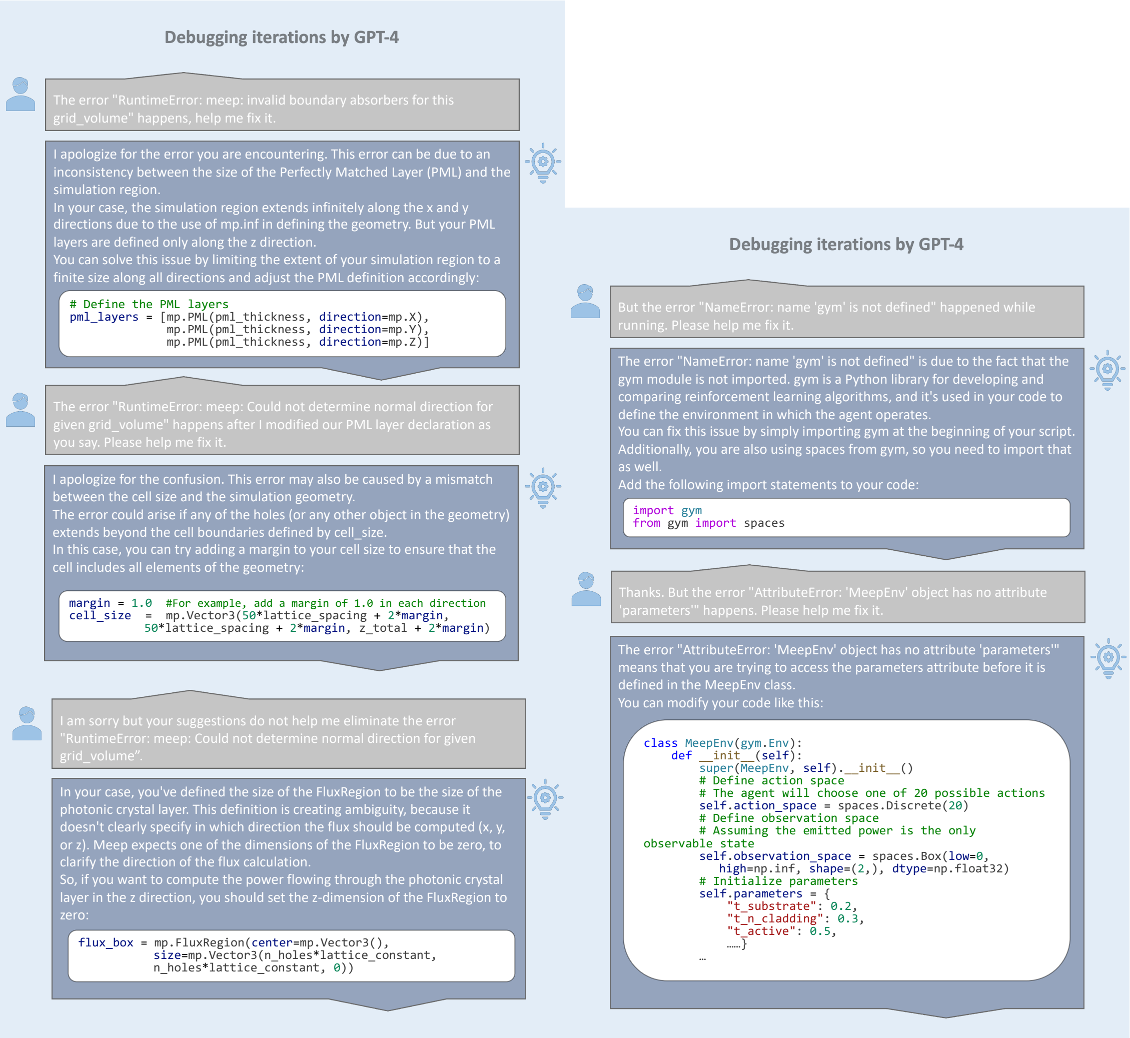
```
# define the optimization (RL) process, which computes V, Q and the loss
def optimize_model():

    if len(memory) < BATCH_SIZE:
        Return
    print('optimizing...')

    transitions = memory.sample(BATCH_SIZE)  # sample transitions from the replay buffer
    batch = Transition(*zip(*transitions))  # transpose the batch
    # compute a mask of non-final states and concatenate the batch elements
    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None, batch.next_state)),
                                   device=device,dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state if s is not None])
    # state, action, and reward from replay buffer
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)
    # compute Q(s, a)
    state_action_values = policy_net(state_batch).gather(1, action_batch)
    # Compute V(s')
    next_state_values = torch.zeros(BATCH_SIZE, device=device)  # V is zero for final state
    next_state_values[non_final_mask] =
                         target_net(non_final_next_states).max(1)[0].detach()  # V' = max(Q')
    # compute the expected Q values
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch  # Q_expected =
r + gamma*V'
    # cost function
    criterion = nn.SmoothL1Loss()
    loss = criterion(state_action_values, expected_state_action_values.unsqueeze(1))  # L =
Q.actual - Q.expected
    # optimize the MLP model
    optimizer.zero_grad()
    loss.backward()
    for param in policy_net.parameters():
        # clamp grad values to between -1 and 1
        param.grad.data.clamp_(-1,1)
    optimizer.step()
    print(loss.item())
    writer.add_scalar('training/losses', loss.item(), steps_done)
```

Figure 8: RL code generated by GPT-4 for optimization of PCSEL using DQN. Left: replay buffer, policy network, and action selection setup, right: core part of DQN where the value functions, action-value functions, and loss are computed. The code shown here is the final version that runs successfully after several rounds of debugging with human.

Appendix B: The details about the optimization of PCSEL via DQN

Deep Q-learning. Deep Q-Learning (DQN) is a type of reinforcement learning algorithm that uses a deep neural network to approximate the action-value (Q) function, which is used to determine the optimal action to take in a given state. The Q-function represents the expected cumulative reward of taking a certain action in a certain state and following a certain policy. In Q-Learning, the Q-function is updated iteratively as the agent interacts with the environment. It is the same in Deep Q-Learning. But instead of changing the Q value in a map directly, Deep Q-learning changes the weight in the neural network. Actually the learning process needs 2 neural networks, called the policy network and target network, respectively. These networks have the same architecture but different parameter weights. Every N steps, the weights from the main network are copied to the target network. In the action selection process, we apply a strategy called Epsilon Greedy Exploration, which can be described as follows: 1) At every time step when it's time to choose an action, roll a dice. 2) If the dice has a probability less than epsilon, choose a random action. 3) Otherwise take the best known action at the agent's current state. At the very beginning the epsilon is very large, which means the agent needs to explore the environment and get some weights updated (all the weights are initialized as 0). As the epsilon is decreasing, the agent starts to try existing known good actions more and more. Using both of these networks leads to more stability in the learning process and helps the algorithm to learn more effectively. And about the weight update of the DQN, we use Bellman's Equation, which is shown below:

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \tag{1}$$



Debugging iterations by GPT-4

The error "RuntimeError: meep: invalid boundary absorbers for this grid_volume" happens, help me fix it.

I apologize for the error you are encountering. This error can be due to an inconsistency between the size of the Perfectly Matched Layer (PML) and the simulation region.
In your case, the simulation region extends infinitely along the x and y directions due to the use of mp.inf in defining the geometry. But your PML layers are defined only along the z direction.
You can solve this issue by limiting the extent of your simulation region to a finite size along all directions and adjust the PML definition accordingly:

```
# Define the PML layers
pml_layers = [mp.PML(pml_thickness, direction=mp.X),
              mp.PML(pml_thickness, direction=mp.Y),
              mp.PML(pml_thickness, direction=mp.Z)]
```

The error "RuntimeError: meep: Could not determine normal direction for given grid_volume" happens after I modified our PML layer declaration as you say. Please help me fix it.

I apologize for the confusion. This error may also be caused by a mismatch between the cell size and the simulation geometry.
The error could arise if any of the holes (or any other object in the geometry) extends beyond the cell boundaries defined by cell_size.
In this case, you can try adding a margin to your cell size to ensure that the cell includes all elements of the geometry:

```
margin = 1.0  #For example, add a margin of 1.0 in each direction
cell_size  =  mp.Vector3(50*lattice_spacing + 2*margin,
           50*lattice_spacing + 2*margin, z_total + 2*margin)
```

I am sorry but your suggestions do not help me eliminate the error "RuntimeError: meep: Could not determine normal direction for given grid_volume".

In your case, you've defined the size of the FluxRegion to be the size of the photonic crystal layer. This definition is creating ambiguity, because it doesn't clearly specify in which direction the flux should be computed (x, y, or z). Meep expects one of the dimensions of the FluxRegion to be zero, to clarify the direction of the flux calculation.
So, if you want to compute the power flowing through the photonic crystal layer in the z direction, you should set the z-dimension of the FluxRegion to zero:

```
flux_box = mp.FluxRegion(center=mp.Vector3(),
           size=mp.Vector3(n_holes*lattice_constant,
           n_holes*lattice_constant, 0))
```

(a)

Debugging iterations by GPT-4

But the error "NameError: name 'gym' is not defined" happened while running. Please help me fix it.

The error "NameError: name 'gym' is not defined" is due to the fact that the gym module is not imported. gym is a Python library for developing and comparing reinforcement learning algorithms, and it's used in your code to define the environment in which the agent operates.
You can fix this issue by simply importing gym at the beginning of your script.
Additionally, you are also using spaces from gym, so you need to import that as well.
Add the following import statements to your code:

```
import gym
from gym import spaces
```

Thanks. But the error "AttributeError: 'MeepEnv' object has no attribute 'parameters'" happens. Please help me fix it.

The error "AttributeError: 'MeepEnv' object has no attribute 'parameters'" means that you are trying to access the parameters attribute before it is defined in the MeepEnv class.
You can modify your code like this:

```
class MeepEnv(gym.Env):
    def __init__(self):
        super(MeepEnv, self).__init__()
        # Define action space
        # The agent will choose one of 20 possible actions
        self.action_space = spaces.Discrete(20)
        # Define observation space
        # Assuming the emitted power is the only
observable state
        self.observation_space = spaces.Box(low=0,
           high=np.inf, shape=(2,), dtype=np.float32)
        # Initialize parameters
        self.parameters = {
            "t_substrate": 0.2,
            "t_n_cladding": 0.3,
            "t_active": 0.5,
            ……}
        …
```

(b)

Figure 9: a) Debugging iterations of the FDTD code by conversing with GPT-4. All bugs/errors in the code demonstrated in Figure 4 were cleared out within 5 iterations; b) Debugging iterations of the DQN code by conversing with GPT-4. All bugs/errors in the code demonstrated in Figure 8 were cleared out within 5 iterations.

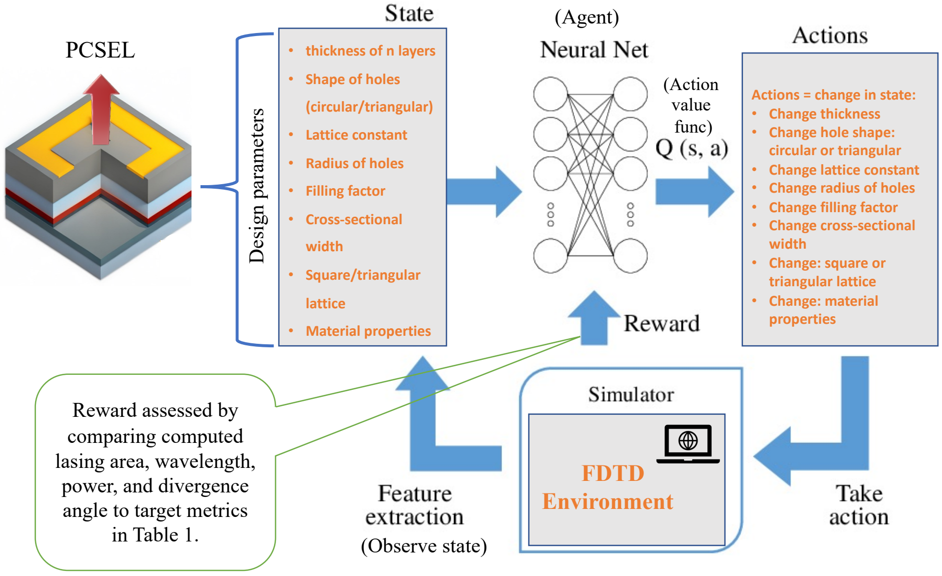


Figure 10: Optimization of PCSEL via reinforcement learning (DQN), where the objective is to meet those target metric/FOM listed in Table 4. This is a detailed expanded version of the right column of Figure 3, including the state and action parameters, the reward definitions, the FDTD environment, and the agent DNN. This algorithm is based on a prior work (Li et al. 2023b).

A concise introduction to the DQN algorithm is presented in Figure 10 and the following text. DQN introduces a groundbreaking concept by integrating a policy (action-value) network $Q(s,a)$ with a target network $Q'(s,a)$, leading to substantial improvements in training performance. Initially, $Q'(s,a)$ is set to be an exact replica of $Q(s,a)$, sharing the same parameters. parameter C denotes the number of steps before updating the target action-value function $Q'(s,a)$ with the values from $Q(s,a)$, often referred to as the ”freeze time.” This freezing mechanism has proven to be highly beneficial in enhancing convergence stability and reducing policy oscillations.

Another crucial aspect of DQN is the implementation of experience replay (Mnih et al. 2015), facilitated through a replay buffer denoted as D. At each time step, the agent's transition (s,a,r,s'), also known as experiences, is stored in a pre-allocated array called D. During the policy update using SGD, experiences from D are randomly selected as training samples. Experience replay offers several advantages: firstly, it allows past experiences to be reused in numerous future gradient updates, leading to enhanced sample efficiency and potentially faster convergence; secondly, as consecutive samples are often correlated and exhibit similar distributions, this can cause the learning process to get stuck at local minima. By randomizing these samples, the data correlation is broken, enabling a more diverse data distribution. Experience replay can also smooth out learning curves and alleviate oscillations or even divergence during training.

Next, ϵ-greedy plays a crucial role in DQN. Given that DQN operates in an off-policy manner, it directly estimates actions using the greedy policy $a = \arg\max_{a'} Q(s,a')$. However, to strike a balance between exploration and exploitation and enable the agent to explore a broader range of the state space, ϵ-greedy dynamically adjusts this policy. The agent follows the ϵ-greedy policy based on $Q(s,a)$. Practically, the ϵ-greedy policy mostly adheres to the greedy policy, selecting the action with the highest estimated Q-value with a probability of $1-\epsilon$. However, with a probability of ϵ, the agent selects a random action to promote exploration. In this work, the initial and final values of ϵ are set to 0.90 and 0.05, respectively.

The defined objective function, also known as the loss function, for this problem, is as follows:

$$L(\theta) = \mathrm{E}[(r + \gamma \max_{a'} Q'(s',a) - Q(s,a))^2] \tag{2}$$

To optimize the loss function $L(\theta)$ presented in Equation 1, stochastic gradient descent is employed. In our conducted experiments utilizing DQN, we utilized the RMSprop optimizer with minibatches of size 32 and a learning rate of 0.00025. In Equation 1, the variable r represents the reward, and we set the discount factor γ to 0.99. This choice of γ allows us to estimate the cumulative return defined at a future time point *T*. The cumulative return is computed as the discounted sum of all future rewards:

$$R = \sum_{t}^{T} \gamma^t r_t \tag{3}$$

Through the optimization of the loss function $L(\theta)$ as defined in Equation 1, the primary goal is to maximize the cumulative return R as expressed in Equation 2. By achieving this objective, we aim to identify and obtain the optimal action that we are seeking in our context.

The state and action setup In the paper we have briefly introduced our PCSEL structure and some elements, here we present a detailed information about it:

Among all these design parameters, we choose 10 to be optimized, which are the hole radius, the lattice spacing, the thickness of all five layers, the reflective index of n-cladding layer (we let the refractive index of p-cladding layer change with n-cladding layer), the active layer and the substrate layer. Each parameter to be optimized has two changing directions. Thus we have 20 discrete action space over all. For change of thickness, we let the step to be 0.0005 or minus 0.0005. For the change of lattice spacing and reflective index, we set the step of 0.0005 or minus 0.0005. For hole radius, we let the change to be 0.0001 each time. The following table shows the boundary of the parameters to be changed.

Table 3 presents the state spaces and action spaces in our DQN algorithm. For this proof-of-concept work, the PCSEL is designed to have 10 states and 20 discrete actions, striking a balance between a manageable parameter space and limiting the overall training time. It is worth noting that more comprehensive investigations can be chosen for higher-order state-action spaces in future studies. The state space, representing a subset of design parameters, encompasses geometric parameters of the PCSEL such as thicknesses and lattice spacing. These states serve as inputs to the policy network. On the other hand, the action space is constructed by incrementing or decrementing each state by a fixed step size of 0.0001 or 0.0005, as outlined in Table 3. To determine the optimal action, the policy network predicts an action, which is then utilized to update the state in the FDTD environment. Subsequently, the environment calculates the associated rewards based on the updated state. For instance, if the current state is $s = 0.3$ and the action is $a = +0.0001$, the environment will yield the next state as $s' = 0.3 + 0.0001 = 0.3001$, along with the associated reward as rew'. The output reward and the resulting next state are fed back into the policy to initiate the subsequent iteration, and concurrently, they are used to update the policy network. Past states and rewards are typically stored in a replay buffer to be utilized later and to mitigate sample correlations (Mnih et al. 2015). If, after multiple action steps, the value of the current state exceeds the boundaries defined by the Min and Max values specified in Table 3, the current episode will be forcibly terminated, and a new episode will begin. For a more comprehensive and detailed understanding of the DQN's mechanism, additional elaborations and in-depth information can be found in the original DQN paper (Mnih et al. 2015), enabling readers to gain a deeper insight into the process.

PCSEL design parameters		
State space	**Min**	**Max**
substrate layer thickness	-0.3 um	0.3 um
n cladding layer thickness	-0.3 um	0.3 um
active layer thickness	-0.3 um	0.3 um
PhC layer thickness	-0.3 um	0.3 um
p cladding layer thickness	-0.3 um	0.3 um
refractive index of substrate layer	-0.15	0.15
refractive index of n cladding /p cladding layer	-0.15	0.15
refractive index of active layer	-0.15	0.15
lattice spacing	-0.1 um	0.1 um
hole radius	-0.1 um	0.1 um
Action space	**Total No. of actions**	**Action type**
each state ± a step size	20	Discrete

Table 3: State space and action space of this DQN-based optimization of PCSEL. State variables are net changes in design parameters (i.e. state = Δdesign parameter)

Reward formulation

To assess the quality of the PCSEL, we set four criteria, which are resonant wavelength, emitting power (or Q-factor), lasing area, and divergence angle. Each criteria is going to be converted to a reward, and the score is the weighted normalized sum of all four rewards.

Equation 4-8 defines the reward and how it's related to the target optical responses as laid out in the main text:

$$rew_1 = 1 - |\lambda^* - \lambda|/\lambda^* \tag{4}$$

$$rew_2 = 1 - (area^* - area)/area^* \tag{5}$$

$$rew_3 = 1 - (Q^* - Q)/Q^* \tag{6}$$

$$rew_4 = 1 + (divergence^* - divergence)/divergence^* \tag{7}$$

$$score = rew_T = 10 \times (\alpha \times rew_1 + \beta \times rew_2 + \gamma \times rew_3 + \eta \times rew_4) \tag{8}$$

In the reward formulation, rew_1 involves several components. The target maximum to be achieved is denoted by λ^*, while λ represents the current value of the wavelength obtained from the FDTD environment. To invert the reward and ensure that larger wavelength result in larger rewards, a constant 1 is used. Additionally, to normalize the magnitude of rewards, we use λ^* as the denominator. In rew_2, the reward is defined based on the target modal volume $area^*$. The formulation of rew_2 is defined such that rewards are higher when the calculated modal volume ($area$) are closer to $area^*$, which aligns with the objectives of the inverse design problem stated earlier. It is the same in rew_3 and rew_4, where we choose our target as emitting power (Q-factor) and divergence angle. Finally, Equation 8 defines the total reward rew_T as a weighted sum from rew_1 to rew_4, which we also call score. Weighting coefficients are selected as follows after multiple rounds of tuning with different combinations: $\alpha = 1e+10, \beta = 1e+15, \gamma = 1e+30, \eta = 1e+32$. We chose large coefficients because sometimes reward values could be as low as 1e-20 or even smaller. The target metrics are listed in Table 4 below, and since we have an infinitely large simulation model with Bloch boundary conditions, we didn't report the lasing area, divergence and beam quality in this paper. This will be covered in future works.

Metric/FOM	**Target values**
Operating wavelength	= 1310 nm
Lasing area	$\geq$ 0.36 μm^2
Q factor	$\geq$ 10000
Divergence angle	$\leq 3^o$
Beam quality M^2	$\leq$ 3

Table 4: Target Metric/FOM of the PCSEL device to be satisfied via optimization, including the Q-factor, lasing area, operating wavelength, beam quality, and divergence angle. An ideal PCSEL has the following characteristics: single-mode, high beam quality M^2, large emission area, and small divergence angle. Since M^2 is dependent on divergence angle, we didn't set it as a reward parameter.

Solved design parameters	
State space	**Solved Values**
substrate layer thickness	0.1680 um
n cladding layer thickness	0.1460 um
active layer thickness	0.1920 um
PhC layer thickness	0 um
p cladding layer thickness	0 um
refractive index of substrate layer	0
refractive index of n cladding /p cladding layer	0.0150
refractive index of active layer	-0.1000
lattice spacing	-0.0170 um
hole radius factor	-0.1892

Table 5: Solved design parameters (net changes) of the optimized PCSEL. Hole radius = hole radius factor × lattice spacing / 2

Optimization results The best design parameters of the optimized PCSEL structure is summarized in Table 5 and the corresponding solved optical attributes are listed in Table 1 in the main text. Please note that state variables are net changes in the design parameters, rather than the design parameters themselves. So a state variable equal to 0 means that there is zero change in that particular parameter. These optimized values can be used to fabricate a PCSEL device in clean-room with enhanced performance metrics.

Calculations of output power-to-injecting power ratio To calculate the output power-to-injecting power ratio of our PCSEL, which is equivalent to the electron (photon)-to-photon conversion efficiency, we used the following formula: Poynting vector divided by dipole source power (the Poynting vector was calculated by Meep or Ansys Lumerical FDTD's near field power monitor, while the dipole power was set to 3.98265e-14 w in Meep or Ansys Lumerical FDTD), or,

$$power\_ratio = Poynting/dipole\_power = Poynting/3.98265 \times 10^{-14} \tag{9}$$

It should be noted that the power ratio calculated here is an ideal/theoretical value, and actual experimental results will be normally worse than this due to optical losses and heat dissipations.

Appendix C: Choice of lasers and computation resources

Choice of laser cavities for inverse design. Traditional VCSELs are lasers that emit light vertically from the surface of the semiconductor structure, allowing for efficient coupling with optical fibers and other optical components. Photonic crystals are artificial structures with periodic refractive index modulation in one, two, or three dimensions. This periodicity generates bandgaps, band edges, and other unique properties that determine the propagation characteristics of light at specific frequencies.

PCSELs are a type of vertical-cavity surface emitting laser, compared to traditional VCSELs, that utilizes two-dimensional photonic crystals to control multi-directional diffraction, resulting in single-mode, high-power, and low-divergence angle emission.

The basic design of a PCSEL includes a photonic crystal layer, an active layer, and several cladding layers, including p-n junctions and electrodes, as shown in Figure 2. The photonic crystal layer typically serves as a resonance cavity. The active layer is usually composed of III-V materials (such as InP, GaAs, GaN, etc.) and is doped with materials to form quantum wells or quantum dot structures, enhancing emission efficiency and controlling emission characteristics. The cladding layers are doped with impurity atoms to form p-type or n-type semiconductor materials, increasing the carrier concentration and enhancing the material's electrical conductivity.

When an electrical current is applied to the device, the carriers undergo carrier population inversion between the valence band and the conduction band in the active layer, leading to the phenomenon of population inversion. Subsequently, carrier recombination occurs, releasing photons. These photons further couple into the photonic crystal cavity, enhancing stimulated emission. The design of the photonic crystal layer determines the coupling strength, wavelength, and direction of the emitted light, making the proper design of the photonic crystal layer crucial for the overall quality of PCSELs.

Here, we propose several design techniques to further lower the divergence angle of PCSELs: vary the shape of the lattice (hexagonal vs. square), vary the shape of air holes (triangular vs. circular), increase the device size, and use double-lattice PhC structure (Yoshida et al. 2019). Analytically, according to (Wang et al. 2022), the divergence angle can be expressed as :

$$\theta = \frac{m\lambda}{L}$$

where m is the coefficient which varies with different structures of the PhC lattice such as the shape of lattice and air holes and the period number of lattice etc., λ is the resonant wavelength, and L is the size of PhC slab. Additionally, the divergence angle of PCSEL can be described by the eigenstates in momentum which dictates that the in-plane wave-vector should be close to $q\pi/L$ (Chen et al. 2022). Here, q is a coefficient similar to m. Therefore, the divergence angle of PCSEL is determined by multiple complex parameters and we will subsequently demonstrate the effect of these variations on the PCSEL performance. In addition, since RL is known to be sample-inefficient and hard to train, we will explore other algorithms such as Bayesian optimization (BO) (Shahriari et al. 2015) that are considerably more efficient and lightweight. BO, as a black-box optimization algorithm suitable for expensive environments, could potentially speed up the optimization process and produce better results.

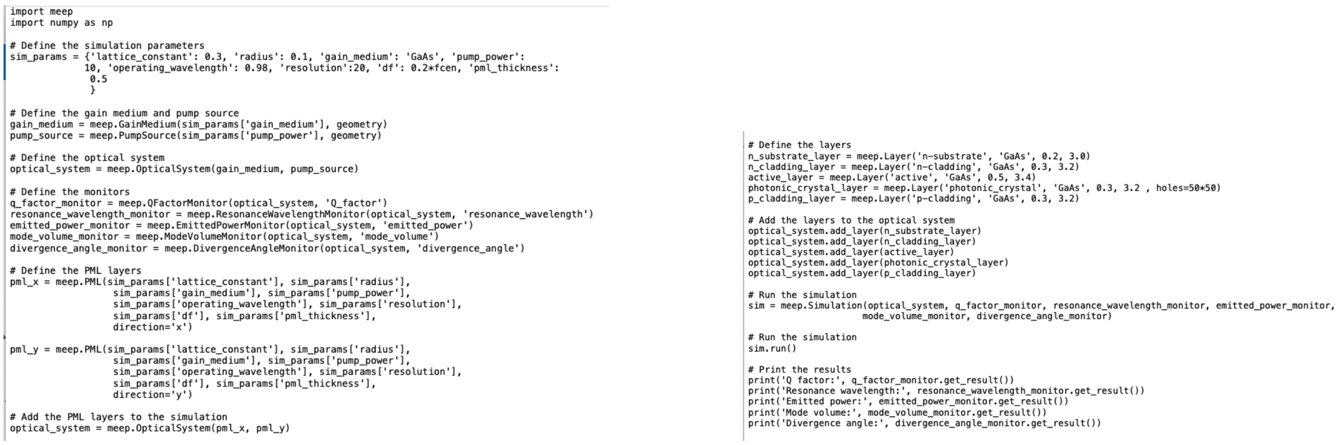
Computing resources and software packages used. The RL code was meticulously developed in Python, strictly adhering to the algorithmic model depicted in the Figure 10. Throughout the implementation, widely used machine learning libraries like PyTorch, OpenAI Gym, and Ray RLlib played a crucial role. In particular, Gym and RLlib proved to be particularly advantageous in accelerating progress. For the training of L2DO, we utilized two Dell workstations with 8 Intel Xeon Gold 5222 cores and an NVIDIA Quadro P4000 GPU. Additionally, comparative computations were performed on a cluster machine with 30 Intel Xeon Gold 5218 processors and 6 NVIDIA 2080Ti graphics cards. The cluster machine exhibited approximately 200% higher computational efficiency compared to the Dell workstations.

Regarding the FDTD simulations, FDTD (Finite Difference Time Domain) is a numerical method that employs central difference quotients to replace the first-order partial derivatives of the field with respect to time and space. By recursively simulating the wave propagation process in the time domain, the FDTD method obtains the field distribution. This approach facilitates a more straightforward and efficient analysis of the wave propagation process.

Appendix D: Experiments with Llama2

Last not but least, we experimented with another latest LLM called Llama2, which was released by Meta in July 2023. As a lightweight (70 billion parameters) and open-source LLM, Llama2 emerges as an attractive alternative to ChatGPT. Using the same conversations, we generated the Meep FDTD code and the DQN code in Llama2. We conclude that, overall, GPT delivered superior performance and dominated Llama2 in terms of question understanding, idea brainstorming, code generation, error self-correcting, etc. This performance gap, of course, can be attributed to the enormous number of parameters (1.8 trillion) and number of training hours that GPT-4 has in its possession. Nonetheless, for those who wish to work with open-source and free-of-charge LLMs, Llama2 is still a decent choice to start with.

Meep FDTD and DQN code generated by Llama2 is shown in Figure 11 and 12, respectively. Compared to those generated by GPT in the main text, one can readily conclude that Llama2 is not nearly as powerful and capable as GPT-4. We can tell that both code are missing core components/modules that render the code erroneous or unable to execute. Critically, Llama2 doesn't seem to have the knowledge of meep and had huge trouble with writing correct methods/functions in meep. Sometimes it would even make up fake methods that doesn't exist at all. Moreover, Llama2 was unable to effectively correct the errors we fed back to it, and the human facilitator had to manually correct the errors. All in all, Llama2 proved to be inferior to GPT4 in virtually all aspects and for those who wish to efficiently design hardware with LLM, GPT is still the top choice. This, however, should not discourage anyone to use Llama2 because one can still harness the open-source power of Llama2.



```
import meep
import numpy as np

# Define the simulation parameters
sim_params = {'lattice_constant': 0.3, 'radius': 0.1, 'gain_medium': 'GaAs', 'pump_power':
             10, 'operating_wavelength': 0.98, 'resolution':20, 'df': 0.2*fcen, 'pml_thickness':
              0.5
              }

# Define the gain medium and pump source
gain_medium = meep.GainMedium(sim_params['gain_medium'], geometry)
pump_source = meep.PumpSource(sim_params['pump_power'], geometry)

# Define the optical system
optical_system = meep.OpticalSystem(gain_medium, pump_source)

# Define the monitors
q_factor_monitor = meep.QFactorMonitor(optical_system, 'Q_factor')
resonance_wavelength_monitor = meep.ResonanceWavelengthMonitor(optical_system, 'resonance_wavelength')
emitted_power_monitor = meep.EmittedPowerMonitor(optical_system, 'emitted_power')
mode_volume_monitor = meep.ModeVolumeMonitor(optical_system, 'mode_volume')
divergence_angle_monitor = meep.DivergenceAngleMonitor(optical_system, 'divergence_angle')

# Define the PML layers
pml_x = meep.PML(sim_params['lattice_constant'], sim_params['radius'],
                  sim_params['gain_medium'], sim_params['pump_power'],
                  sim_params['operating_wavelength'], sim_params['resolution'],
                  sim_params['df'], sim_params['pml_thickness'],
                  direction='x')

pml_y = meep.PML(sim_params['lattice_constant'], sim_params['radius'],
                  sim_params['gain_medium'], sim_params['pump_power'],
                  sim_params['operating_wavelength'], sim_params['resolution'],
                  sim_params['df'], sim_params['pml_thickness'],
                  direction='y')

# Add the PML layers to the simulation
optical_system = meep.OpticalSystem(pml_x, pml_y)
```

(a)

```
# Define the layers
n_substrate_layer = meep.Layer('n-substrate', 'GaAs', 0.2, 3.0)
n_cladding_layer = meep.Layer('n-cladding', 'GaAs', 0.3, 3.2)
active_layer = meep.Layer('active', 'GaAs', 0.5, 3.4)
photonic_crystal_layer = meep.Layer('photonic_crystal', 'GaAs', 0.3, 3.2 , holes=50*50)
p_cladding_layer = meep.Layer('p-cladding', 'GaAs', 0.3, 3.2)

# Add the layers to the optical system
optical_system.add_layer(n_substrate_layer)
optical_system.add_layer(n_cladding_layer)
optical_system.add_layer(active_layer)
optical_system.add_layer(photonic_crystal_layer)
optical_system.add_layer(p_cladding_layer)

# Run the simulation
sim = meep.Simulation(optical_system, q_factor_monitor, resonance_wavelength_monitor, emitted_power_monitor,
                      mode_volume_monitor, divergence_angle_monitor)

# Run the simulation
sim.run()

# Print the results
print('Q factor:', q_factor_monitor.get_result())
print('Resonance wavelength:', resonance_wavelength_monitor.get_result())
print('Emitted power:', emitted_power_monitor.get_result())
print('Mode volume:', mode_volume_monitor.get_result())
print('Divergence angle:', divergence_angle_monitor.get_result())
```

(b)

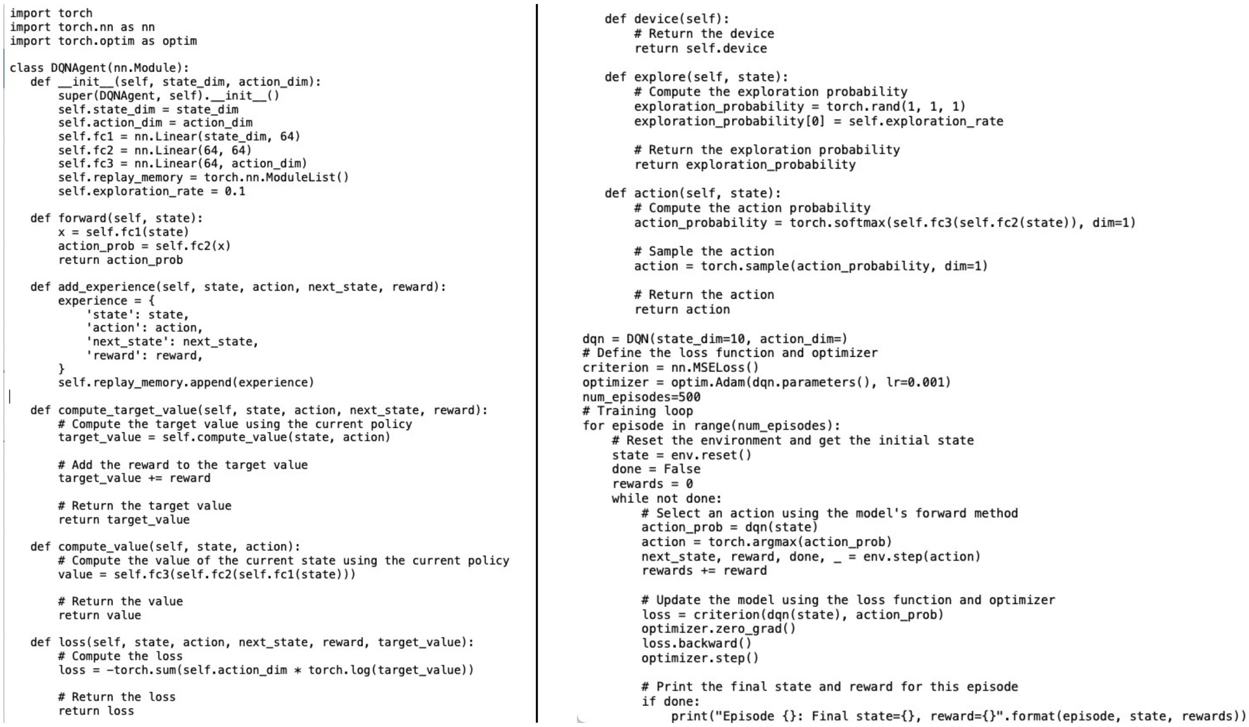
Figure 11: Meep FDTD code generated by Llama2.

Appendix E: Full comparison with literature data

We conducted an extensive literature survey of existing state-of-the-art (SOTA) PCSELs and summarized the comparison between our results and a selected group of papers in Table 6 below (only square lattice and circular holes are reported).

Paper	lambda (nm)	PhC dimension (um)	Device area (um^2)	Lattice constant (nm)	Divergence ↓	Q factor ↑	Lasing area (m^2)	M^2 ↓	half theta	Loss (1/nm) ↓
Zhou (2020) (Zhou et al. 2020)	957	50*50	2500	/	/	/	/	/	/	/
Ohnishi (2004) (Ohnishi et al. 2004)	959.44	50*50	2500	/	1.1°	/	2.83e-09	/	0.55	/
Sakai (2005) (Sakai et al. 2005)	965	50*50	2500	286.25	1°	1700	/	/	0.5	1.29052e-05
Hsu (2017) (Hsu, Lin, and Pan 2017)	1299	300*300	90000	390	$\leq 2°$	5000	1.76625e-08	/	1	3.22051e-06
Chen (2017) (Chen et al. 2017)	1260	300*300	90000	/	/	/	/	/	/	/
Chen (2021) (Chen et al. 2021a)	948	125*125	15625	281	0.75°	2900	6.22e-09	3.1	0.375	7.70647e-06
Wang (2021) (Wang et al. 2021)	935	340*340	115600	/	0.38°	/	/	6.5	0.19	/
Reilly (2020) (Reilly et al. 2020)	1010	250*250	62500	/	/	/	/	/	/	/
Kalapala (2022) (Kalapala et al. 2022)	1040	2000*2000	4e+6	/	/	100000	3.14e-08	/	/	/
Ours	1383	**2*2**	4	400	1.2°	37000	9.92e-14	**1.36**	0.6	**4.30787e-07**

Table 6: Comparative assessment of SOTA PCSELs, where GaAs is the gain material, PhC lattices are square and air holes are circular. The ”/” indicates the absence of data in the cited literature.



```python
import torch
import torch.nn as nn
import torch.optim as optim

class DQNAgent(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(DQNAgent, self).__init__()
        self.state_dim = state_dim
        self.action_dim = action_dim
        self.fc1 = nn.Linear(state_dim, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_dim)
        self.replay_memory = torch.nn.ModuleList()
        self.exploration_rate = 0.1

    def forward(self, state):
        x = self.fc1(state)
        action_prob = self.fc2(x)
        return action_prob

    def add_experience(self, state, action, next_state, reward):
        experience = {
            'state': state,
            'action': action,
            'next_state': next_state,
            'reward': reward,
        }
        self.replay_memory.append(experience)

    def compute_target_value(self, state, action, next_state, reward):
        # Compute the target value using the current policy
        target_value = self.compute_value(state, action)

        # Add the reward to the target value
        target_value += reward

        # Return the target value
        return target_value

    def compute_value(self, state, action):
        # Compute the value of the current state using the current policy
        value = self.fc3(self.fc2(self.fc1(state)))

        # Return the value
        return value

    def loss(self, state, action, next_state, reward, target_value):
        # Compute the loss
        loss = -torch.sum(self.action_dim * torch.log(target_value))

        # Return the loss
        return loss

    def device(self):
        # Return the device
        return self.device

    def explore(self, state):
        # Compute the exploration probability
        exploration_probability = torch.rand(1, 1, 1)
        exploration_probability[0] = self.exploration_rate

        # Return the exploration probability
        return exploration_probability

    def action(self, state):
        # Compute the action probability
        action_probability = torch.softmax(self.fc3(self.fc2(state)), dim=1)

        # Sample the action
        action = torch.sample(action_probability, dim=1)

        # Return the action
        return action

dqn = DQN(state_dim=10, action_dim=)
# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(dqn.parameters(), lr=0.001)
num_episodes=500
# Training loop
for episode in range(num_episodes):
    # Reset the environment and get the initial state
    state = env.reset()
    done = False
    rewards = 0
    while not done:
        # Select an action using the model's forward method
        action_prob = dqn(state)
        action = torch.argmax(action_prob)
        next_state, reward, done, _ = env.step(action)
        rewards += reward

        # Update the model using the loss function and optimizer
        loss = criterion(dqn(state), action_prob)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Print the final state and reward for this episode
        if done:
            print("Episode {}: Final state={}, reward={}".format(episode, state, rewards))
```

Figure 12: DQN code generated by Llama2.