

LLM4Laser: Large Language Models Automate the Design of Lasers

Renjie Li^{1,2*}, Ceyao Zhang^{1*}, Sixuan Mao¹, Xiyuan Zhou¹,
Feng Yin¹, Sergios Theodoridis³, Zhaoyu Zhang^{1†}

¹The Chinese University of Hong Kong, Shenzhen

²University of Illinois Urbana-Champaign, ³University of Athens

Abstract

With the rapid evolution of global autonomous driving technology, the demand for its core sensing hardware, Light Detection and Ranging (LiDAR), is escalating. As the light source part of the LiDAR system, lasers, particularly the cutting-edge Photonic Crystal Surface Emitting Lasers (PCSEL), have correspondingly attracted extensive research attention. The conventional manual design and optimization of PCSEL typically require expertise in semiconductor physics and months of dedicated effort to achieve satisfactory results. While AI-driven approaches can expedite this process, laser designers still need to invest time in learning the AI algorithms involved. Meanwhile Large Language Models (LLMs), leveraging their powerful reasoning abilities, can effectively comprehend natural language and provide constructive feedback in multi-turn dialogues. They have already demonstrated potential to assist humans in scientific fields such as robotics design and chemical discovery. A question naturally arises is: Can LLMs transform the lasers design process? This paper proposes a novel **human-AI co-design** paradigm to show that LLMs can guide the laser design and optimization process both conceptually and technically. Specifically, by simply having conversations, GPT assisted us with writing both Finite Difference Time Domain (FDTD) simulation code and deep reinforcement learning (RL) code to acquire the optimized PCSEL solution, spanning from the proposition of ideas to the realization of algorithms. Given that GPT will perform better when given detailed and specific prompts, we break down the PCSEL design problem into a series of sub-problems and converse with GPT by posing open-ended heuristic questions rather than definitive commands. We achieved a significant milestone towards *self-driving laboratories*, that is, a fully automated AI-driven pipeline, for laser design and production.

Introduction

Trends of design automation (i.e., human out of the loop) in the integrated circuit (IC), nanotechnology, and semiconductor industries (Mirhoseini et al. 2021; Zhang et al. 2020b; Chen et al. 2021b) are emerging rapidly. Aided by artificial intelligence (AI), machine automation is beginning to replace conventional IC design and fabrication processes involving humans that had existed for over half a century.

The nanophotonics industry (Almeida et al. 2004; Altug, Englund, and Vučković 2006; Thomson et al. 2016), however, has not experienced comparable level of automation due to its unique fabrication precision requirements (Hocevar et al. 2012; Kim et al. 2014) and relatively complex theoretical models (Xie et al. 2021; Zeng et al. 2020). As an important example of nanophotonics, the design of Photonic Crystal Surface Emitting Lasers (PCSELs; Hirose et al. 2014; Noda et al. 2017; Yoshida et al. 2019), demands rigorous physical modeling and calculations with Finite Difference Time Domain (FDTD) or Finite Element Analysis (FEA) simulation tools. Moreover, the inverse design and optimization (Ma et al. 2021; Molesky et al. 2018; So et al. 2020) of PCSEL typically suffers from three aspects: the one-to-many mapping and non-convexity nature, a high demand for expert knowledge in semiconductor physics and theoretical modeling/simulation (and thus human involvement), and a lack of ready-to-use machine learning algorithms/packages. The above challenges prohibit an end-to-end automated design pipeline for PCSELs and other advanced laser devices alike.

Luckily, recent advancements in machine learning (Goodfellow, Bengio, and Courville 2016; Mnih et al. 2015; Theodoridis and Koutroumbas 2006) and optimization algorithms (Luo et al. 2010; Hale, Yin, and Zhang 2008; Milzarek and Ulbrich 2014) have propelled the progress of automated nanophotonics design. Early in the 90s, heuristic, evolutionary (Hegde 2019), and gradient-based (Zhang et al. 2020a) optimization algorithms began to emerge prolifically. Key algorithms include Newton’s method (Milzarek and Ulbrich 2014), particle swarm (Ma and Li 2020), genetic algorithm (Ren et al. 2021), Bayesian optimization (Shahriari et al. 2015), and simulated annealing (Bertsimas and Tsitsiklis 1993) etc. These algorithms provide a new way of thinking when facing these non-convex optimization problems and lay a solid foundation for continued research. But the challenge remains with heavy human involvement. To solve this, around 2012, researchers proposed deep learning (DL; Krizhevsky, Sutskever, and Hinton 2012; Goodfellow, Bengio, and Courville 2016) frameworks that take advantage of an abundance of training data and neural network’s inference ability. These DL models greatly boosted the efficiency of nanophotonic inverse design, pushing the possibility of automated design into a new stage (Jiang, Chen, and Fan 2021; So et al. 2020; Li et al. 2022; Asano and Noda 2018; Li et al.

*These authors contributed equally.

†Corresponding author: zhangzy@cuhk.edu.cn

Copyright © 2025, Authors All rights reserved.

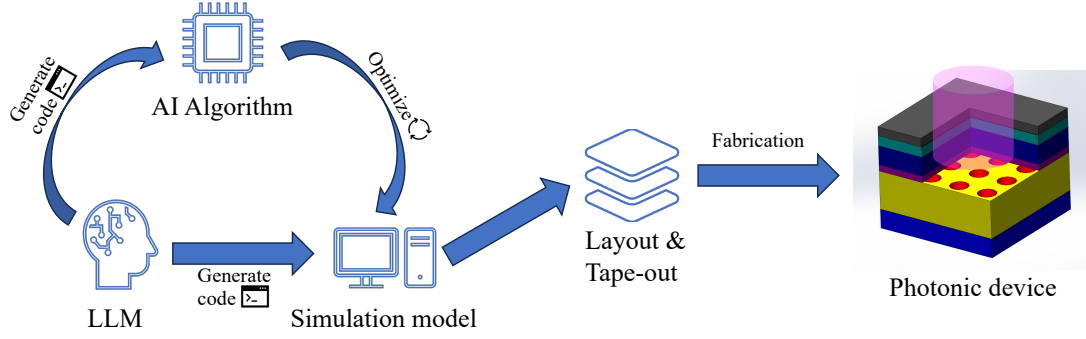


Figure 1: Long-term vision of this work: LLMs for automated PCSEL design and optimization pipeline that enables *self-driving* laboratories. The human facilitator prompts the LLM to generate FDTD code for simulating the PCSEL structure and AI (e.g., reinforcement learning (RL)) code for subsequent optimizations of the PCSEL model. The FDTD code is written with the MIT meep (Oskooi et al. 2010) package. The AI optimization process with RL is built upon an earlier work’s L2DO framework (Li et al. 2023b). The final optimized PCSEL design (shown on far right) is then converted to CAD layout and prepared for tape-out and foundry fabrication.

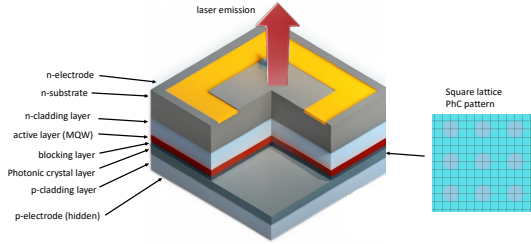


Figure 2: Photonic Crystal Surface Emitting Laser (PCSEL), with abundant applications in sensing, LiDAR, and telecommunications.

2021). Circa 2023, a new DL framework based on RL (e.g., deep Q-learning (DQN) (Mnih et al. 2015)), called Learning to Design Optical-Resonators (L2DO; Li et al. 2023b), provides the solution for inverse design of photonic crystal nanocavities without human intervention. With two orders of magnitude higher sample efficiency compared to supervised learning, L2DO has preliminarily realized photonics design automation on an algorithmic level. However, since both the simulation code and DL code in L2DO were still created by the human designer, we were still a distance away from the fully automated photonic design.

Recently, as LLMs has become dominant in the field of AI, a limited handful of researchers have found the potential for using LLMs in hardware design and implementation at an early time. In 2020, researchers utilized an improved GPT-2 model called “DAVE” for Verilog code snippets generation and output evaluation (Pearce, Tan, and Karri 2020), which is a crucial component in the IC design pipeline.

A more recent model named “Chip-chat” (Blocklove et al. 2023) came out in 2023, which is an LLM-driven method for IC Verilog code generation and is one of the first wholly-AI-written Hardware Description Language (HDL) for chip tape-out. Meanwhile, LLMs have also contributed significantly to the design and control of robots. Researchers have shown the guidance value of LLM in a robotic gripper design process (Stella, Della Santina, and Hughes 2023), both conceptually and technically. By means of simply conversing with GPT, they successfully designed a robotic gripper capable of reaping the tomato plant. Last but not least, LLM-based agent also shows great planning ability in both game AI (Wang et al. 2023; Zhang et al. 2023) and embodied robotic AI (Ahn et al. 2022) tasks.

In this work, we propose a new **human-AI co-design** paradigm for PCSELs as illustrated in Figure 1 and demonstrate the practical implications of LLMs for laser design methodologies. Specifically, we explored and verified the potential of applying LLMs to machine learning-based design and optimization of PCSELs, during which we seek to maintain as little human involvement as possible. By simply having conversations spanning from the proposition of initial ideas to the implementation of final algorithms, GPT-4 assisted us with writing FDTD simulation code and deep RL (e.g. DQN) code to acquire the optimized PCSEL solution. The optimized PCSEL meets the following figure of merit (Hirose et al. 2014): single-mode, high-beam quality, large-area, and small-divergence angle. A high-level overview of the end-to-end design pipeline is illustrated in Figure 1. Given that GPT will perform better when given detailed and specific prompts, we break down the PCSEL design problem into a series of sub-problem modules and converse with GPT by strictly posing open-ended heuristic questions rather than definitive commands. These rules are summarized and

proposed as five golden tricks. This paper shows that LLMs, such as ChatGPT, can guide the laser design and optimization processes, on both the conceptual and technical level. All in all, we achieved a significant milestone towards an automated end-to-end laser design and optimization pipeline with minimal human intervention.

Background

Photonic Crystal Surface Emitting Lasers (PCSEL)

Among lasers, PCSELs (Hirose et al. 2014; Noda et al. 2017; Yoshida et al. 2019) represent an avant-garde technology that integrates the advantages of photonic crystals (PhC; Quan, Deotare, and Loncar 2010) and Vertical Cavity Surface Emitting Lasers (VCSELs; Chang-Hasnain 2000). PCSELs emit high-quality laser beam vertically and find broad applications in sensing, detection, and telecommunications.

Methods

Objective overview

In this article, we investigate the potential of an LLM-based automated PCSEL design and optimization pipeline, as shown in Figure 1. The target metrics or figures of merit (FOM) of the PCSEL to be satisfied are discussed in Appendix B and C, which were set according to target application specifications and scenarios with full consideration of the physical limits of PCSEL lasing. For example, the wavelength of 1310 nm is important for applications in telecommunications and satellite communications, a high beam quality and a suitably high Q-factor is important for applications like autonomous driving, metal machining/material processing, and medical surgeries, and a small divergence angle is important for achieving high beam quality, small focal spot, and long-distance light propagation. In this initial demonstration, actual tape-out will not be implemented. And due to computational limitations, our PCSEL device has a side length of $2.0\ \mu\text{m}$ with periodic boundary conditions.

Human-AI co-design paradigm

The *human-AI co-design* paradigm is more clearly illustrated in Figure 3. We divide the design process into three steps: 1) conceptualization, 2) code generation and debugging, and 3) simulation and optimization. Due to the limitation of the status quo of LLMs, the AI agent usually cannot give out the perfect solution all at once. Therefore, the human needs to act as a liaison to help guide/facilitate the design work while simultaneously bearing in mind that excessive human involvement could compromise the integrity of the AI agent’s decisions. So for example, if a large portion of the design task is dominated/controlled by humans, it does not reflect the human-AI co-design paradigm and thus should be avoided as much as possible. Overall, in our experience, the design flow should observe the following rules and practice to obtain optimal feedback from the LLM.

First, the whole design process should start with the human providing an open-ended question to GPT, rather than

giving definitive commands. This is the beginning of an important conceptualization process, where the human stimulates the LLM to brainstorm and generate creative ideas. For example, you can start a conversation by “What is the most sought-after laser nowadays?” or “What are some good optimization algorithms to use in hardware design?” Subsequent conversations will continue by gradually guiding GPT to arrive at a specific solution for this conceptual question. Sample Q&A rounds are demonstrated in the left column of Figure 3. Once we get to the code generation and debugging stage, questions can become more specific and technical, such as “Can you help me design a high-power and large-area PCSEL model using an FDTD algorithm written in Python?” or “Can you help me improve an existing deep-Q learning code implemented with experience replay that’s written with PyTorch?”

Second, humans should respect the self-correcting mechanism of GPT rather than directly pointing out the problems/errors it has. In the conversations, there might be times when the answers given by LLMs are self-contradictory or simply wrong. This is attributable to GPT’s insufficient understanding of the problem to be solved, which usually happens at the early stages of a conversation. To properly respect the self-correcting mechanism of GPT, one should report the error by responding “you have just mentioned XXX, and I hope you could think twice about this and regenerate your answer” or “are you sure XXX is the correct answer? Please elaborate” rather than “lines XXX and XXX of the code you generated are wrong” or “fix the XXX function/method in the code for me”. In the code-generating part, these kinds of mistakes will be especially common. In our experience, it usually takes several conversational iterations before the code finally becomes bug-free and ready to run. Additionally, when debugging the code it is not advised to point out the precise location of errors for GPT; the proper way is to copy the error message from the terminal and let AI do the modification itself. A demonstration of this interactive debugging process is shown in the middle column of Figure 3.

Based on the above practical rules, we propose and recommend to the readership the following five golden tricks for successful human-AI co-designs using LLMs:

1. **Open-ended Question Start:** Begin design by posing broad queries to GPT, sparking creative brainstorming and conceptualization. For instance, “What’s the latest sought-after laser?” or “Optimization algorithms for hardware design?”
2. **Divide and Conquer:** GPT performs better when given detailed questions and information, so break down the design problem into a series of sub-problems.
3. **Technical Advancement:** Transition to specific, technical queries during code generation and debugging, like “Develop a high-power PCSEL model using FDTD algorithm in Python.”
4. **Respect Self-Correction:** Honor GPT’s self-correcting capacity by prompting reconsideration of inaccurate answers. Instead of direct corrections, encourage reflection and elaboration for refinement.

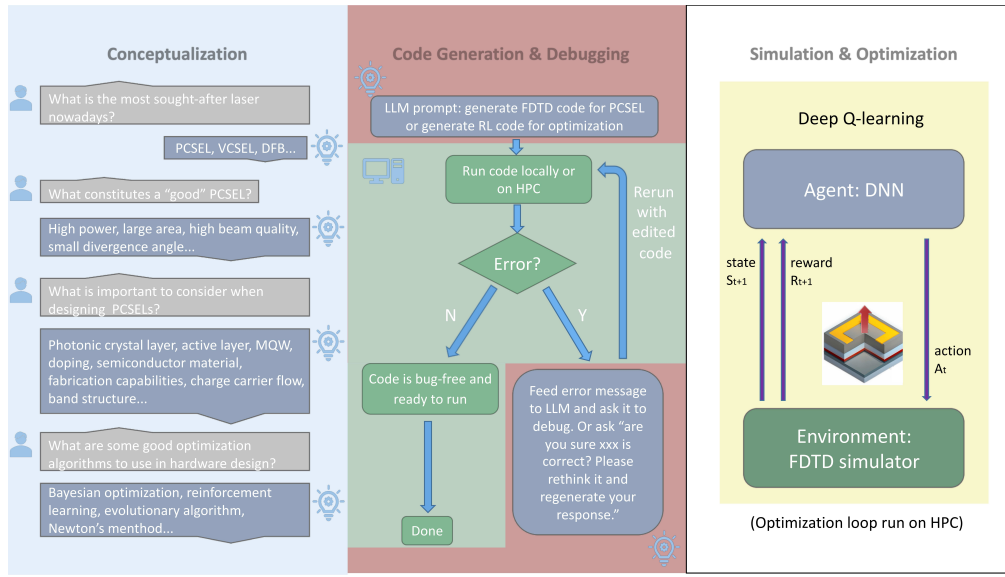


Figure 3: LLM4Laser: A novel Human-AI co-design paradigm for applying LLMs to PCSEL design and optimization. A pictorial overview of the discussions and interactions between the human facilitator and the LLM, with the questions prompted by the human and the answers/solutions provided by the LLM (GPT).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

5. **Effective Debugging Approach:** During code debugging, provide error messages for GPT to address. Refrain from pinpointing error locations; allow GPT to autonomously modify code based on terminal output.

Conceptualization with the LLM The conceptualization step is for the AI to have a general understanding of the underlying problem so that the AI can choose proper methods and algorithms for more detailed prompts down the road. We kick off the conversation by raising heuristic questions "What constitutes a "good" PCSEL laser?" and "Let us design a PCSEL together, How do you think we should start?", and the LLM provides us with a general design pipeline by saying "here is a general procedure to get started with" where the design problem is broken down into sub-modules such as "understanding the basics of PCSEL", "material selection", "designing photonic crystal structure" etc. Then we take a further step by asking: "I have understood the basics of PCSEL, now what are some important factors to consider when designing PCSELS?" The LLM then points out what to look out for when designing PCSELS. Some sample chats are shown in the left column of Figure 3. We then prompted another heuristic question about choosing optimization algorithms. GPT-4 eventually provides the answer: "In this case, reinforcement learning and Bayesian optimization might be the most suitable for your problem" after we've clarified our optimization objectives and constraints. When we think that GPT-4 has, for the most part, understood what we are trying to do, we ask it: "Could you generate an appropriate code skeleton according to the above conversation? Please note that the FDTD should be implemented with meep and RL should be with PyTorch." As a result, GPT-4 gives us a code

skeleton for FDTD simulation using meep (Oskooi et al. 2010) and DQN using PyTorch, respectively. The problem has now evolved to a matter of expanding these code skeletons to full-blown scripts, meaning that our work is moving to the next stage for code generation.

Code generation and improvement In previous step, we have divided the coding problem into two modules (FDTD and DQN) and obtained the initial code skeletons of both modules. In this step, we will complete, debug, and improve our existing code skeletons, which is an essential step of the whole PCSEL design process. This step is pictorially illustrated in the middle column of Figure 3.

For generating an FDTD simulation code from a skeleton script, we need to provide the LLM with more specific and concrete physical parameters concerning the PCSEL. For instance, we may require that our PCSEL has five layers, called "n-substrate layer", "n-cladding layer", "active layer", "photonic crystal layer", and "p-cladding layer" respectively, as well as 50×50 air holes in the PhC layer. It is worth mentioning that the value of parameters, such as the refractive index and the thickness of each layer, could be randomly initialized due to the DQN optimization process that we are going to implement subsequently. In the conversations, we assigned those values in line with the PCSEL model that we have built in the past. In addition, we also provided requirements for boundary conditions, meshing resolution, refractive indices, etc. After we provided the concrete physical parameters, GPT-4 expanded our code by completing the PCSEL geometry and simulation settings, as shown in Figure 4. Additionally, the flux and far-field monitors are added according to our requirement for calculating

```

# 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)

# Initialize the simulation
sim = mp.Simulation(cell_size=cell_size,
                    boundary_layers=pm1_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 = mp.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.fz, 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.

the emitting power, modal volume, and divergence angle; the Harminv monitor is added for calculating the Q-factor. Some of these monitors and their calculations are shown in Figure 4. Now that the FDTD simulation code has been written, the next step is to proceed with debugging and fine-tuning until the code finally runs successfully. As the middle column of Figure 3 and Figure 7 shows, we iteratively test-run the code on our local computer and transfer the error messages to GPT for debugging, repeating this process until the code becomes bug-free. In our experience, most bugs can be eliminated within five iterations.

Next, we generated the DQN code for RL-based optimization of PCSELS (Figure 6). As a core component of the DQN algorithm, we first need an environment to provide the feedback interface. We adopted OpenAI Gym (Brockman et al. 2016) as the wrapper class for our environment, which is the FDTD simulator that we have generated and fine-tuned. When letting GPT generate the code for the environment, we told GPT what the state space, action space, and reward function are. Further specifications such as the step size of actions, and the upper and lower bounds of state variables are given to GPT as well. Then, with the environment code, we could complete and implement our DQN code. A main DQN script is finished by GPT based on the code skeleton given earlier, considering requirements for the replay buffer, policy DNN, optimizer, loss function, etc. See Figure 6 for the core part of the completed DQN script. Note that the DQN script imports the environment class. The next step is to run the DQN code, letting it interact with our FDTD environment and continuously optimize the PCSEL device.

Just like the FDTD code, the DQN code is then debugged and fine-tuned by iteratively running the code and feeding error messages to GPT. An example of this process is demonstrated in Figure 7 in Appendix A.

Final optimization step with RL For the full optimization loop of PCSEL using DQN, refer to Figure 8. The main framework is built upon the one proposed in an earlier work (Li et al. 2023b). The objective here is to optimize the existing PCSEL structure such that the target metrics are met. Therefore at each iteration, computed optical attributes (lasing area, Q-factor ...) that are closer to the target metrics will earn a higher reward. The state is defined as the design parameters of PCSEL, whereas the action comprises the changes made to the state at each iteration. The environment, which is a core component of DQN, is realized with the FDTD simulation code that we generated earlier. A detailed explanation of the working theories of DQN, the state and action setup, the agent/policy net, and the reward definition is reserved in the supplementary material. The optimization loop is run on an HPC cluster that has 20 CPU cores and 2 GPUs with CUDA, where each trial takes up to 5 days to run.

Results and discussion

Due to the space limitations, we put the details about the optimization of PCSEL via DQN in Appendix B. Figure 5 showcases the learning curves (scores vs. episodes) of training the DQN algorithm to optimize our PCSEL according to the algorithm in Figure 8. Three representative trials have

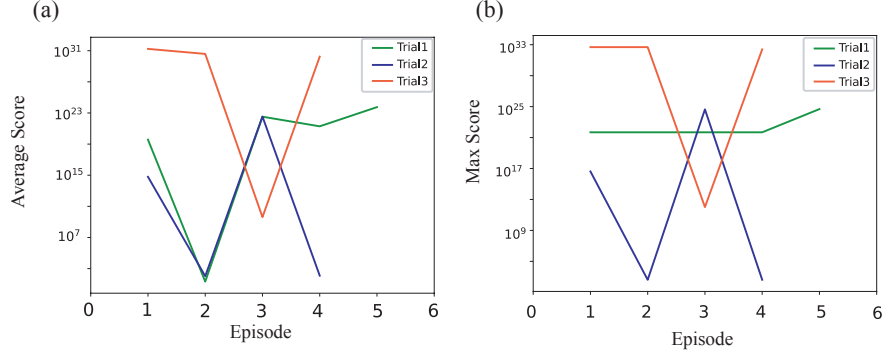


Figure 5: Learning curves of training the DQN to optimize PCSEL, plotted as scores vs. episodes. (a) Average score of each episode; (b) Maximum score of each episode. Each episode contains a horizon of 500 steps. Vertical axes are plotted in log scale.

Metrics	Optimized values	Literature data (Chen et al. 2021a)
Operating wavelength (nm) \uparrow	1383	948
Q factor \uparrow	37000	2900
Normalized loss (1/nm) \downarrow	4.3×10^{-7}	7.7×10^{-6}

Table 1: Optical attributes of the DQN-optimized PCSEL structure. The rightmost column reports the best literature data ((Chen et al. 2021a)) for a direct metric comparison, which has a operating wavelength of 945 nm. Normalized loss per unit length is additionally reported for a more fair and comprehensive comparison.

been selected to demonstrate the training convergence results. Figure 5(a) shows the average score of each episode, whereas (b) shows the max score of each episode. Each episode contains a maximum of 500 time steps. Each curve, which represents a complete trial, was trained for 5 days until cut off at the end of the 5th day due to limits on computing resources.

Using the optimization results illustrated in Figure 5, the optical attributes of the optimized PCSEL that satisfy the target metrics/figure of merit (FOM) are reported in Table 1. The corresponding set of solved design parameters of the optimized PCSEL is included in Table 3 in suppl. mat. To better illustrate the advantage of our results, the best PCSEL data from the literature are listed in the rightmost column of Table 1 for a direct comparison of metrics. The literature data were chosen after an exhaustive literature survey (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) of advancements in PCSEL over the past 10 years, which is fully reported in Table 5 of Appendix E.

The device size (side length) of the literature data (Chen et al. 2021a) and our PCSEL is $125.0 \mu\text{m}$ and $2.0 \mu\text{m}$, respectively. In this proof-of-concept work, we limited our PCSEL's side length to $2.0 \mu\text{m}$ due to insufficient computing resources, as larger models would exponentially increase the simulation time and stall the optimization process. In Table 1, the calculated wavelength of 1383 nm is within the acceptable tolerance of the target 1310 nm wavelength, being red shifted by 70 nm. The Q factor of our device is over

an order of magnitude larger than the literature data, leading to stronger resonance in the PhC layer. Also, the normalized loss per unit length (Kalapala et al. 2022) (radiation loss of the resonance mode) of our device is over one order of magnitude smaller than the literature data as seen in Table 1, which indicates that our device is more energy efficient and lossless in spite of its smaller size. This is also confirmed by the larger Q-factor of our device. However, since we used an infinitely large simulation model in FDTD (Bloch boundary conditions), divergence angle and lasing area are not comparable to the literature data at the moment and we will include these metrics in future endeavors. All in all, we can conclude that the optimized attributes shown in Table 1 have satisfied and even exceeded the target metrics set by us.

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.

Last not but least, we experimented with another latest LLM called Llama2 (Touvron et al. 2023), which was released by Meta AI 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, which are shown and analyzed in Appendix D. 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.

Conclusion

In this paper, we introduced a novel human-AI co-design paradigm for PCSELS, showcasing the broader relevance of LLMs in laser design scenarios. We systematically explored the application of LLMs in machine learning-driven design and optimization of PCSELS, aiming for minimal human intervention. Through ordinary conversational interactions, ranging from initial concept proposals to final algorithm implementation, GPT-4 aided in crafting FDTD simulation and deep reinforcement learning (e.g. DQN) code to achieve an optimized PCSEL solution meeting criteria like single-mode operation, high beam quality, large area, and narrow divergence angle. This paradigm successfully addressed three major challenges faced by state-of-the-art deep learning-enabled inverse design methods: 1) the fundamental one-to-many mapping or the non-convex issue; 2) heavy human involvement for technical input; and 3) shortage of ready-to-use machine learning methods. The design process involves breaking down the design problem into modular sub-problems and heuristically prompting GPT to answer open-ended questions, among several other golden tricks we summarized and recommended to the audience. We hope that these golden tricks can serve as a general guideline for anyone who wishes to benefit from LLMs' power in hardware design. Our results demonstrate that LLMs, such as ChatGPT, can effectively guide laser design and optimization, both conceptually and technically. In the end, we proposed several future research directions and showcased a comparison to meta's Llama2. We will subsequently pursue these future thrusts to deliver better optimization results. Overall, we mark a significant step toward an AI-empowered automated, end-to-end nanophotonic design and optimization pipeline.

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*.
- Almeida, V. R.; Barrios, C. A.; Panepucci, R. R.; and Lipson, M. 2004. All-optical control of light on a silicon chip. *Nature*, 431(7012): 1081–1084.
- Altug, H.; Englund, D.; and Vučković, J. 2006. Ultrafast photonic crystal nanocavity laser. *Nature physics*, 2(7): 484–488.
- Asano, T.; and Noda, S. 2018. Optimization of photonic crystal nanocavities based on deep learning. *Optics express*, 26(25): 32704–32717.
- Bertsimas, D.; and Tsitsiklis, J. 1993. Simulated annealing. *Statistical science*, 8(1): 10–15.
- Blocklove, J.; Garg, S.; Karri, R.; and Pearce, H. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. *arXiv preprint arXiv:2305.13243*.
- Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.
- Chang-Hasnain, C. J. 2000. Tunable vcsel. *IEEE Journal of Selected Topics in Quantum Electronics*, 6(6): 978–987.
- 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.
- Hale, E. T.; Yin, W.; and Zhang, Y. 2008. Fixed-point continuation for L1-minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3): 1107–1130.
- Hegde, R. S. 2019. Photonics inverse design: pairing deep neural networks with evolutionary algorithms. *IEEE Journal of Selected Topics in Quantum Electronics*, 26(1): 1–8.
- 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.
- Hocevar, M.; Immink, G.; Verheijen, M.; Akopian, N.; Zwiller, V.; Kouwenhoven, L.; and Bakkers, E. 2012. Growth and optical properties of axial hybrid III–V/silicon nanowires. *Nature communications*, 3(1): 1266.
- 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.
- 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. Continuous-wave lasing operation of 1.3- μm wavelength InP-based photonic crystal surface-emitting lasers using MOVPE regrowth. *Optics Express*, 28(24): 35483–35489.
- Jiang, J.; Chen, M.; and Fan, J. A. 2021. Deep neural networks for the evaluation and design of photonic devices. *Nature Reviews Materials*, 6(8): 679–700.
- 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.
- Kim, S.; Marelli, B.; Brenckle, M. A.; Mitropoulos, A. N.; Gil, E.-S.; Tsioris, K.; Tao, H.; Kaplan, D. L.; and Omenetto, F. G. 2014. All-water-based electron-beam lithography using silk as a resist. *Nature nanotechnology*, 9(4): 306–310.
- Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.
- 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.
- Luo, Z.-Q.; Ma, W.-K.; So, A. M.-C.; Ye, Y.; and Zhang, S. 2010. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 27(3): 20–34.
- 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.
- Ma, Z.; and Li, Y. 2020. Parameter extraction and inverse design of semiconductor lasers based on the deep learning and particle swarm optimization method. *Optics Express*, 28(15): 21971–21981.
- Milzarek, A.; and Ulbrich, M. 2014. A semismooth Newton method with multidimensional filter globalization for L1-optimization. *SIAM Journal on Optimization*, 24(1): 298–333.
- 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.
- Quan, Q.; Deotare, P. B.; and Loncar, M. 2010. Photonic crystal nanobeam cavity strongly coupled to the feeding waveguide. *Applied Physics Letters*, 96(20): 203102.
- 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.
- Ren, Y.; Zhang, L.; Wang, W.; Wang, X.; Lei, Y.; Xue, Y.; Sun, X.; and Zhang, W. 2021. Genetic-algorithm-based deep neural networks for highly efficient photonic device design. *Photonics Research*, 9(6): B247–B252.
- 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.
- Thomson, D.; Zilkie, A.; Bowers, J. E.; Komljenovic, T.; Reed, G. T.; Vivien, L.; Marris-Morini, D.; Cassan, E.; Viot, L.; Fédéli, J.-M.; et al. 2016. Roadmap on silicon photonics. *Journal of Optics*, 18(7): 073003.
- Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- 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.
- Xie, B.; Wang, H.-X.; Zhang, X.; Zhan, P.; Jiang, J.-H.; Lu, M.; and Chen, Y. 2021. Higher-order band topology. *Nature Reviews Physics*, 3(7): 520–532.
- Yoshida, M.; De Zoysa, M.; Ishizaki, K.; Tanaka, Y.; Kawasaki, M.; Hatsuda, R.; Song, B.; Gellera, 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.; Xiao, P.; Sun, R.; and Luo, Z. 2020a. A single-loop smoothed gradient descent-ascent algorithm for nonconvex-concave min-max problems. *Advances in neural information processing systems*, 33: 7377–7389.
- Zhang, W.; Gao, B.; Tang, J.; Yao, P.; Yu, S.; Chang, M.-F.; Yoo, H.-J.; Qian, H.; and Wu, H. 2020b. 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 Design of Lasers

Supplementary Material

Appendix A: The debugging demonstrations by GPT

The code generated by GPT for RL is shown in Figure 6 and the debugging iterations with GPT is shown in Figure 7.

```
# 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(self, state):
        """selects an action according 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) #
        # potentially 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 6: 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:

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


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,
                       50*lattice_spacing + 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 7: 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 6 were cleared out within 5 iterations.

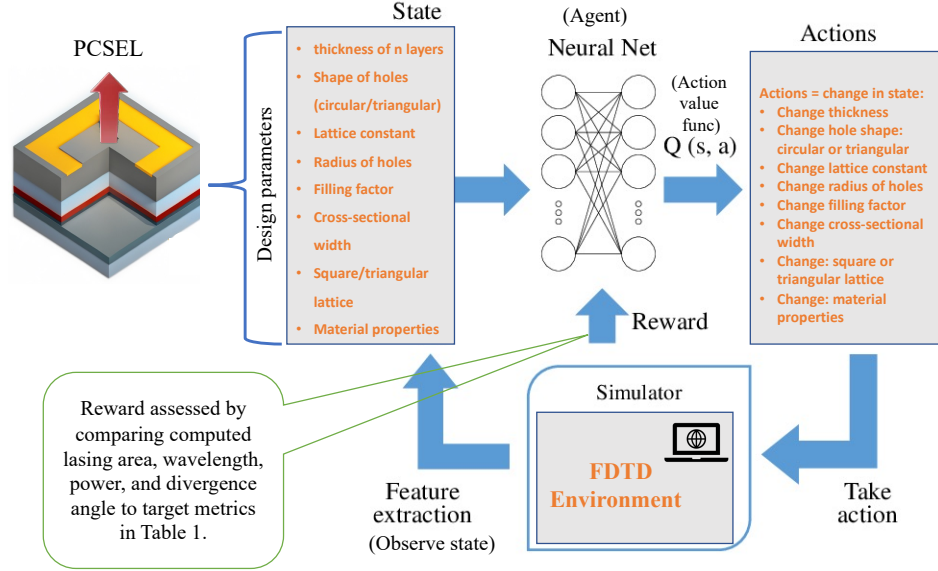


Figure 8: Optimization of PCSEL via reinforcement learning (DQN), where the objective is to meet those target metric/FOM listed in Table 4. This is a detailed expanded version of the right column of Figure 3, including the state and action parameters, the reward definitions, the FDTD environment, and the agent DNN. $Q(s, a)$ is action-value functions predicted by the agent DNN.

A concise introduction to the DQN algorithm is presented in 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 $Q(s, a)$, as outlined in Algorithm 1. 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:

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

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:

$$R = \sum_t^T \gamma^t r_t \quad (3)$$

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 2 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 2. 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 2, 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 \pm a step size	20	Discrete

Table 2: State space and action space of this DQN-based optimization of PCSEL. State variables are net changes in design parameters (i.e. state = Δ design 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:

$$rew_1 = 1 - |\lambda^* - \lambda|/\lambda^* \quad (4)$$

$$rew_2 = 1 - (area^* - area)/area^* \quad (5)$$

$$rew_3 = 1 - (Q^* - Q)/Q^* \quad (6)$$

$$rew_4 = 1 + (divergence^* - divergence)/divergence^* \quad (7)$$

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

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.

Solved design parameters	
State space	Solved Values
substrate layer thickness	0.1680 μm
n cladding layer thickness	0.1460 μm
active layer thickness	0.1920 μm
PhC layer thickness	0 μm
p cladding layer thickness	0 μm
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 μm
hole radius factor	-0.1892

Table 3: Solved design parameters of the optimized PCSEL. Hole radius = hole radius factor \times lattice spacing / 2

Metric/FOM	Target values
Operating wavelength	= 1310 nm
Lasing area	$\geq 0.36 \mu m^2$
Q factor	≥ 10000
Divergence angle	$\leq 3^\circ$
Beam quality M^2	≤ 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.

Optimization results The best design parameters of the optimized PCSEL structure is summarized in Table 3 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 Lumerical FDTD's near field power monitor, while the dipole power was set to $3.98265e-14$ w in Lumerical FDTD), or,

$$power_ratio = Poynting/dipole_power = Poynting/3.98265 \times 10^{-14} \quad (9)$$

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.

Computing resources and software packages used. The RL code was meticulously developed in Python, strictly adhering to the algorithmic model depicted in the Figure 8. 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

Meep and DQN code generated by meta Llama2 is shown in Figure 9 and 10, 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.

The values of most of the hyperparameters were selected by performing an informal search with the PCSEL. We did not perform a systematic or exhaustive grid search owing to the high computational cost, although it is understandable that even better results could be obtained by systematically tuning the hyperparameter values one-by-one.

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 5 below (only square lattice and circular holes are reported).

```

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 9: Meep code generated by Llama2.

```

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)
    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 10: DQN code generated by Llama2.

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 et al. (2020)	957	50*50	2500	/	/	/	/	/	/	/
Ohnishi et al. (2004)	959.44	50*50	2500	/	1.1°	/	2.83e-09	/	0.55	/
Sakai et al. (2005)	965	50*50	2500	286.25	1°	1700	/	/	0.5	1.29052e-05
Hsu, Lin, and Pan (2017)	1299	300*300	90000	390	≤ 2°	5000	1.76625e-08	/	1	3.22051e-06
Chen et al. (2017)	1260	300*300	90000	/	/	/	/	/	/	/
Chen et al. (2021a)	948	125*125	15625	281	0.75°	2900	6.22e-09	3.1	0.375	7.70647e-06
Wang et al. (2021)	935	340*340	115600	/	0.38°	/	/	6.5	0.19	/
Reilly et al. (2020)	1010	250*250	62500	/	/	/	/	/	/	/
Kalapala et al. (2022)	1040	2000*2000	4e+6	/	/	100000	3.14e-08	/	/	/
Ours	1310	2*2	4	400	1.2°	36400	9.92e-14	1.36	0.6	4.30787e-07

Table 5: Comparative assessment of SOTA PCSELS, where GaAs serves as the gain material and PhC lattices are characterized by square and circular holes. The “/” indicates the absence of data in the cited literature.