Lens Optimisation Using the LLaMEA Framework Specifically for Optical Lens Design

Yotam Lev (s4045513)

Supervisors:
Niki van Stein & Anna V. Kononova

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 13, 2026

Abstract

Optical lens design represents a highly complex Mixed-Integer Non-Linear Programming (MINLP) challenge that traditionally relies heavily on manual domain expertise to navigate rugged, non-convex evaluation landscapes. Optimising architectures such as the 24-dimensional Double-Gauss configuration which combines continuous geometric parameters with discrete material selections frequently causes conventional solvers to converge prematurely on local optima. This thesis investigates the application of the Large Language Model Evolutionary Algorithm (LLaMEA) framework to autonomously discover and generate optimisation heuristics specifically tailored for macroscopic optical lens design. By interfacing LLaMEA with a high-performance, JAX-based ray-tracing simulation acting as a strict black-box evaluator, the system iteratively evolves executable Python optimisation strategies without requiring prior optical knowledge. Empirical results demonstrate that the LLM-driven framework successfully synthesised novel algorithms capable of matching and exceeding the exploitation capabilities of state-of-the-art domain-specific benchmarks (LDG-EA) and reinforcement learning approaches (RLEMMO). The leading generated optimiser achieved a highly competitive best scalar loss of 1.18×10^{-4}. However, it was vastly outperformed when comparing the diversity of candidate solutions discovered. The thesis looks at algorithms generated with just the first-order derivative and those with both the first and second derivative in the LLM's context window, investigating not only the LLM's ability to generate state of the art optimisation algorithms but also how the input context impacts the generated output. This research builds a complete automated pipeline (Github) for LLM generated optimisation algorithms for optical lens design and validates LLaMEA as a highly effective, computationally efficient tool for automated algorithm discovery in complex, physically constrained environments.

Contents

1 Introduction

Optical lens systems represent a foundational infrastructure layer for modern technological advancement. Their widespread presence spans from consumer electronics to highly complex scientific apparatuses. Fundamentally, optical lens optimisation has long been a challenging problem and the improvement of optical lenses has enabled much of the technology we see today. High precision lenses are the critical enabling components in the development of modern lasers, advanced digital cameras, and the intricate photolithography systems used in semiconductor manufacturing. In these domains, there is a requirement for aberration free, physics bound optical resolution and nanoscale [CLT21]. Consequently, improving the optical loss function, minimising the root mean square (RMS) spot size while rigorously penalising physical aberrations is vital.

The design of these advanced optical systems is an inherently rugged, high-dimensional and mixed-variable optimisation challenge. The search space consists of highly non-convex continuous manifolds (defining surface curvatures and lens element distances) which are strictly bounded by non-differentiable, categorical decisions (glass materials from discrete catalogues) [CTS+23]. Historically, traditional design has been based on refinement of already known optical lens configurations. As standard commercial solvers and conventional gradient-based methods frequently converge prematurely on a single local optimum [CSJ+19], they struggle to navigate the landscape effectively, making the field heavily reliant on domain knowledge in optical physics and advanced mathematical proficiency.

The Double-Gauss lens configuration is used to systematically benchmark solutions, it serves as a foundational architecture for the majority of standard wide lenses used today [COM]. From a computational perspective, optimising a six-element Double Gauss lens is a complex Mixed-Integer Non-Linear Programming (MINLP) problem [Kid01].

To understand the dimensionality of this search space, see Figure 1, the standard configuration is characterised by its symmetry around a central aperture stop. The physical optical path is dictated by the precise interaction of light across multiple refracting surfaces. For a computational solver, every visual feature in this diagram represents a dimension that must be simultaneously tuned, the curvature of each individual glass surface, the physical thickness of each lens element, and the exact axial air gaps separating them [HL19].

The algorithm must navigate a state vector comprised of both 18 continuous variables and 6 categorical integers representing the glass/material id, selected from the Schott[45] catalogue which consists of 120 materials. The objective function, typically evaluating the root-mean-square (RMS) spot size of simulated light rays acts as a complex black box evaluator. The resulting error landscape is non convex, highly multi-modal, and sharply bounded by "infeasible cliffs" where

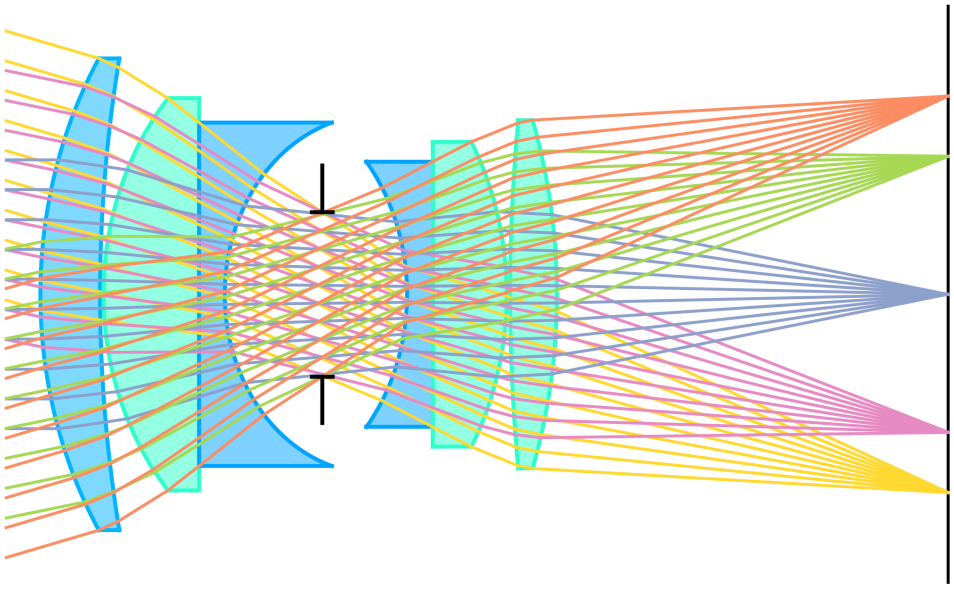


Figure 1: Highly optimised Double-Gauss lens design problem configuration. The system comprises six lens elements with twenty-four optimisation variables image from Antonov Kirill et al., 2026 [ATB+26].

physical impossibilities, such as total internal reflection or negative edge thicknesses, instantly invalidate the state [SO90].

This thesis uses the Large Language Model Evolutionary Algorithm (LLaMEA) framework for black-box optimisation of the Double-Gauss problem [vSB25]. As a targeted research investigation, the primary aim is to methodically explore the underlying thought process, design choices, and structural mechanics involved in adapting an automated algorithm discovery framework to a complex physical simulation environment. Specifically, the central research question guiding this study asks, to what extent the Large Language Model Evolutionary Algorithm framework can be adapted to automatically discover and generate highly competitive optimisation algorithms for macroscopic optical lens design. To address this overarching objective, the investigation is driven by the following sub-questions:

1. How does the performance of algorithms generated by LLaMEA without contextual domain knowledge compare to those generated using detailed, domain specific optical promtps?

2. What structural patterns characterise the highest performing algorithms generated by LLaMEA?

3. How do the generated algorithms allocate their computational budget when navigating the highly multi modal landscapes inherent to optical lens configurations?

Addressing these sub-questions requires situating LLaMEA's architecture within the established

paradigms of optical engineering. Because this automated solver relies on passing variables through prompting and environment initialisation rather than manual derivation, establishing the historical bottlenecks of conventional lens design provides the necessary contrast to evaluate LLaMEA's efficacy.

2 Background and Motivation

The traditional approach to solving highly complex, domain-specific challenges such as optical lens design relies heavily on extensive training and iterative human intuition. Conventionally, optimising these physical systems presents a structural bottleneck, as it demands an understanding of the underlying physics to manually construct effective mathematical heuristics. The primary motivation for deploying the LLaMEA [vSB25] framework in this context is its capacity to abstract away this domain expertise requirement. By treating the complex optical simulation as a black-box evaluator, LLaMEA shifts the burden of algorithmic discovery from the human to an automated, generative process.

An automated approach is significant due to the computational history of optical optimisation. Historically, optical lens systems design has been a rigorous manual challenge. Because the performance landscape is riddled with deep local minima, traditional gradient-based optimisation engines are sensitive to initialisation. To achieve viable results, one typically had to begin with an existing system and optimise continuously from that specific local basin [YFH24].

Additionally to escape local optima autonomously, past computational advancements focused on augmenting the geometry of the design space. A major leap was the *Saddle Point Construction* method. Instead of relying purely on random stochastic perturbations, this technique systematically expands the dimensionality of the optical system—for example, by mathematically inserting a non physical "null lens" of zero thickness. This artificially converts a local minimum into a saddle point, allowing the optimisation engine to slide down into entirely new local minima basins and discover novel configurations [BvTM07].

More recently, the computational focus has shifted toward Quality-Diversity (QD) Evolutionary Algorithms. Algorithms such as the Hill-Valley Evolutionary Algorithm (HillVallEA) were introduced into optical design to identify a diverse set of highly comparable lens topologies in a single run, rather than collapsing onto a single "optimal" design. Generating this diversity is critical for downstream engineering, where manufacturers must weigh theoretical optical performance against practical physical tolerances, weight, and production costs [ATB+26].

While foundational evolutionary approaches may successfully optimise continuous geometric variables (surface curvatures, lens thicknesses, and spacing), they falter when combined with discrete variables. Glass materials represent discrete, categorical variables. Shifting an optimisation engine from one discrete glass type to another introduces massive, chaotic discontinuities in the refractive index landscape. These sudden jumps easily break the covariance-matrix adaptation (CMA) that evolutionary algorithms rely on to dynamically learn and navigate the search space.

2.1 The LLaMEA Orchestration Framework

The first major piece of the thesis codebase is a customised version of the LLaMEA framework, originally built by Niki van Stein et al. [vSB25]. At its core, this framework acts as an automated experiment orchestrator designed to benchmark LLM-assisted algorithm evolution. Rather than optimising numerical weights, the "individuals" evolving within this system are raw, executable Python scripts representing distinct optimisation algorithms.

As depicted in the overarching system architecture Figure 2, the framework operates through a continuous, closed loop evolutionary cycle that modifies the standard LLaMEA pipeline to accommodate physical simulations. The workflow begins in the Initiation phase, which establishes the Double Gauss problem, role specifications, and prompt strategies then ensues a role playing iterative loop, with a "Client" - the feedback system, and the "Coder" - the LLM synthesising the feedback and generating the optimisation classes. Exclusively Ollama models were used throughout the experiments (Qwen2.5-code:14b or Mistral:latest when running locally and Qwen3-coder:30b, Qwen3.6:35b or Gemma4:e4bo when running on the hpc cluster).

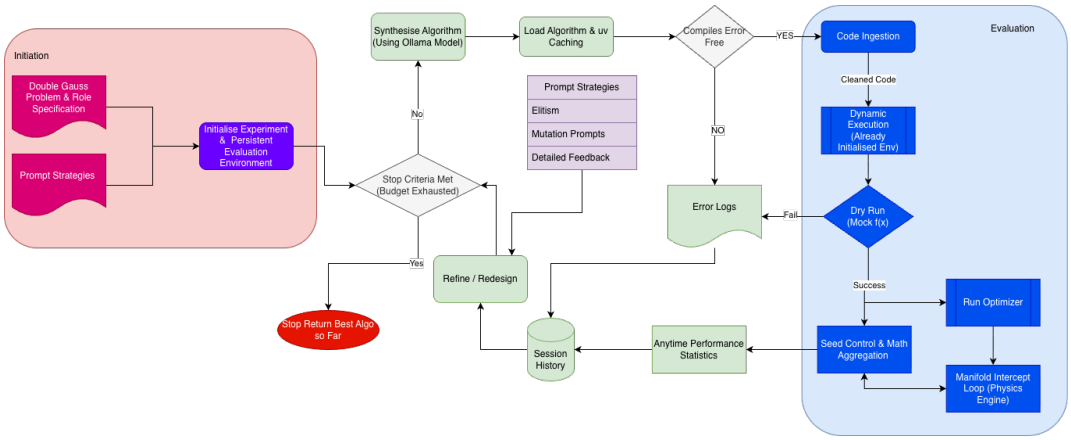


Figure 2: Altered LLaMEA global architecture adapted for this thesis, highlighting the novel evaluation loop for the optical lens design use case.

To ensure systemic stability before evaluation, the generated code is immediately passed through a rigorous caching and validation layer. This stage utilises a 'uv' package manager for rapid dependency resolution and a compilation checker to intercept syntax errors, routing failures to an Error Logs module for feedback upon termination of the iteration. Once validated, the algorithm enters the specialised Evaluation boundary which differs from the standard LLaMEA which uses the IOH Experimenter. The code undergoes Code Ingestion and is dynamically executed within a pre-initialised persistent sandbox. The optimiser is first subjected to a Dry Run (mock $f(x)$) to ensure runtime stability before entering the main Manifold Intercept Loop, where it directly queries the physics engine.

Throughout this execution, a Seed Control and Math Aggregation module gathers real-time performance statistics saving experimental logs to a dedicated `results` sub folder. The cycle continues until the algorithm triggers the Stop Criteria, determined by the exhaustion of a defined computational budget—which was scaled from 500 to 50,000 evaluations depending on the experimental parameters. Upon completion or failure, the logged feedback, alongside the previous successful coding output (depending on elitism) and targeted mutation prompts, is fed back into the Refine / Redesign node, prompting the LLM to generate the next generation of algorithmic optimisation classes until the `generation` count is reached.

2.2 Computational Architecture of the Optical Simulation Engine

The experimental framework of this thesis also relies on a high-performance camera lens simulation engine, by Antonv Kirill et al. to evaluate the fitness of candidate optical designs [ATB+26]. Specifically, this engine provides the loss function landscape that guides the optimisation classes dynamically generated by the LLaMEA loop. To accommodate the number of objective function evaluations required by evolutionary algorithms, the engine is designed around modern vectorisation and automatic differentiation paradigms, which allow this codebase to circumvent traditional computational bottlenecks that rely on iterative ray-tracing by implementing sequential geometric ray tracing natively in JAX.

JAX compiles the Python-based mathematical operations into highly optimised machine code via Accelerated Linear Algebra (XLA), enabling execution on both CPUs and GPUs. Furthermore, the engine utilises `jax.vmap` to vectorise the ray-tracing physics over thousands of parallel rays across multi-angle and multi-wavelength grids. This single-instruction, multiple data (SIMD) approach eliminates slow Python control-flow loops to ensure the strict numerical stability required for optical surface refractions (e.g., Snell's Law). Empirical benchmarks on the Double-Gauss objective demonstrate that this JAX-compiled forward pass executes nearly an order of magnitude faster than equivalent standard vectorised baselines.

2.2.1 Loss Formulation and Aberration Quantification

The primary utility of the simulation engine within the LLaMEA pipeline is to yield a continuous loss landscape representing optical fidelity. The loss function executes a forward trace of ray bundles from the object plane, through parameterised optical surfaces (curvatures, thicknesses, and refractive indices mapped from glass catalogues), to a virtual sensor plane.

The core optical quality metric is the RMS spot size. The simulation generates a grid of rays representing different field angles and spectral wavelengths as seen in Figure 1. At the sensor plane, the engine computes the variance of ray intersection coordinates relative to the chief ray (the centroid of the bundle). This RMS calculation inherently quantifies both geometric and chromatic aberrations without needing abstract analytical approximations.

Because optical systems must also be physically manufacturable, the loss formulation incorporates a set of soft penalty constraints. These include edge and centre thickness penalties to prevent unmanufacturable lens geometries, a free working distance constraint to avoid sensor collisions, and an Effective Focal Length (EFFL) penalty to ensure the system meets macroscopic design targets. The summation of the RMS spot size and these physical penalties forms the scalar fitness value returned to the LLaMEA loop.

A significant advantage of formulating the ray-tracing engine purely in JAX is the availability of exact automatic differentiation (autodiff). While the LLaMEA loop drives the outer optimisation architecture, local continuous variable tuning (e.g., precise surface curvatures) can be augmented with gradient-based methods.

The codebase exposes analytical derivatives by tracing the computational graph of the forward pass. Utilising reverse-mode automatic differentiation, the engine provides exact first-order gradients (`jax.grad` and `jax.value_and_grad`), Jacobians (`jax.jacrev`), and second-order Hessians (`jax.hessian`). By replacing traditional finite-difference approximations with exact autodiff, the engine avoids truncation and subtractive cancellation errors, providing robust, high-dimensional gradients necessary for complex continuous parameter tuning.

2.2.2 Experimental Execution Cost

The LLaMEA generative experiments were executed on a high-performance cluster node (`duranium.liacs.nl`) running Rocky Linux 9. The hardware configuration features 256 GB RAM, 20 Intel Xeon E5-2650v3 cores (2.30 GHz), and a mixed array of NVIDIA GPUs (six GTX 980 Ti and two Titan X) utilised for local LLM inference. Resource allocation was capped at 10 CPU cores which enabled the running of max two experiments concurrently. Generations were capped at a hard 1,500-second timeout, with successful algorithmic iterations generally completing within 12 to 18 minutes.

2.3 Mathematical Formulation and Topographical Constraints

The computational optimisation involves the simultaneous configuration of twenty-four distinct dimensional variables. This mixed-integer optimisation problem fundamentally violates the assumptions of conventional local search techniques, which generally require continuous, smooth manifolds to reliably calculate gradients and guarantee convergence. The variables are categorically divided to define the physical geometry and the material properties of the refractive paths.

Specifically, the continuous variables dictate the topological structure of the lens system. While a separated six-element configuration implies twelve independent surfaces, cementing and topological constraints reduce this. Typically, one might expect to optimise 10 distinct active curvature values however, in this constrained Double-Gauss formulation, 4 values are strictly pinned to either flat or what the inverse of the lens beside them because they maintain a fixed structural distance from one another. Locking these parallel planar boundaries substantially reduces the dimensionality of the continuous search space, restricting the degrees of freedom the optimisation algorithm must navigate.

The objective function $F(\Theta)$ utilised to score candidate designs is engineered to penalise any deviation from theoretical diffraction-limited imaging. While the absolute mathematical optimal value is zero, this is practically unachievable due to fundamental physical limits; a 'state-of-the-art' (SOTA) optimiser achieves scores of $\sim 0.5 \times 10^{-5}$. Consequently, the composite merit function balances the core imaging fidelity against an array of distinct physical penalty terms. The mathematical formulation is defined as:

$$F(\Theta) = \mathrm{RMS}(\Theta) + \sum_{k=1}^{5} w_k P_k(\Theta) \tag{1}$$

In this equation, $\mathrm{RMS}(\Theta)$ represents the Root-Mean-Square spot size of the image on the focal plane, serving as the primary metric for image sharpness and geometric/chromatic aberration control. The variables $P_k(\Theta)$ are specifically designed physical penalty functions scaled by empirically derived weights w_k as described in table 1. These terms mathematically constrain the computational search space to ensure that the resulting optical architectures are both physically manufacturable and functionally viable. By imposing these penalties, the fitness landscape is transformed, introducing steep gradients and boundaries that restrict the feasible search volume.

2.4 Benchmarking Baselines

To rigorously evaluate the performance of the proposed LLaMEA algorithm within the Camera Lens Simulation landscape, its optimisation trajectory was benchmarked against three classes of algorithms. These baselines provide a comparative spectrum ranging from naive exploration to

Penalty Term	**Weight** (w_k)	**Physical Implication and Fitness Landscape Effect**
P_1: Ray Failure / TIR	10	Imposes scalar penalty if light rays undergo Total Internal Reflection (TIR) or are vignetted by the lens housing. Aggressively terminating invalid ray paths and preventing the optimiser from exploring dead-end geometries.
P_2: Edge Thickness	1	Prevents lenses from having infinitely thin edges (which are physically unmanufacturable and prone to fracturing) or excessively thick overlapping edges.
P_3: On-Axis Thickness	1	Constrains the central thickness of optical elements to match targeted material budgets and transmission limits. Acts as a regularisation term driving element dimensions toward nominal manufacturing baselines, preventing arbitrary thickening.
P_4: Free Working Distance	1	Ensures a strict minimum spatial gap is maintained between the terminal optical surface and the sensor plane to accommodate mechanical shutters and filters.
P_5: EFFL	1	Forces the simulated macro-focal length to match the designated system specifications (e.g., field of view). Confines valid solutions to a lower-dimensional manifold within the search space, forcing the optimiser to trade-off parameters along the specific iso-surface where EFFL is constant.

Table 1: Formulation of the topographical penalty terms and their impact on the optimisation landscape [ATB+26].

state-of-the-art heuristic and learning-based policies.

2.4.1 Random Search

As a foundational baseline, a pure Random Search algorithm was employed. Random Search operates without memory or heuristic guidance, uniformly sampling the continuous and discrete parameter bounds of the search space. It serves primarily to validate that the intelligent search mechanics of the more advanced algorithms provide statistically significant convergence advantages over stochastic brute force.

2.4.2 Reinforcement Learning: RLEMMO Benchmark

Reinforcement learning initiated the integration of machine learning to dynamically control algorithm behaviour in multi-modal landscapes. A pivotal benchmark in this transitional era is the Reinforcement Learning-based Evolutionary Multimodal Optimisation (RLEMMO) framework [LMG+24]. RLEMMO exemplifies the shift from hard-coded exploration strategies toward learned, state-dependent search policies.

RLEMMO Overview RLEMMO operates as a Meta-Black-Box optimisation architecture. It maintains a population of candidate solutions but delegates the granular control of individual search operators to a centralised deep reinforcement learning agent. The guiding philosophy underpinning RLEMMO is the replacement of fixed mathematical adaptation rules with a flexible learned policy. This policy dynamically adjusts searching strategies at the individual level to correspond with the real-time, up-to-date topographical status of the optimisation process. To empirically validate this framework, several distinct RLEMMO implementations were deployed: a Multi-Armed Bandit variant (MAB-RLEMMO) driven by a rolling average heuristic, a Proximal Policy Optimisation configuration (PPO-RLEMMO) utilising a flat observation array, and an Attention RLEMMO leveraging graph attention alongside niche archiving. To ensure thorough convergence within the loss landscape, experimental runs were completed with extensive evaluation budgets ranging between 1,000,000 to 2,500,000.

Limitations of RL The efficacy of the RLEMMO methodology was exhaustively validated against the Multimodal Optimisation Problem benchmark suite [LMG+24]. Through its learned policy, the algorithm demonstrated highly competitive search performance against robust established baselines. By automatically inferring the optimal inflection points at which to exploit known basins versus injecting exploratory diversity to traverse uncharted topologies, RLEMMO showcased significant advancements. Applying it to the complex Double-Gauss problem exposes the structural limitations

of standard reinforcement learning approaches, which often require extensive hyperparameter tuning and struggle to autonomously interpret explicit structural or semantic constraints.

2.4.3 State-of-the-Art: LDG-EA

The final comparative benchmark in this investigation is the Lens-Descriptor Guided Evolutionary Algorithm (LDG-EA), coupled with the HILLVALLEA backend [ATB+26]. As the native, state-of-the-art optimiser designed specifically for the Camera Lens Simulation engine, LDG-EA serves as the highly specialised algorithmic upper bound. It represents the domain-engineered evolutionary optimisation against which the generalised LLaMEA approach will be measured.

Optical design inherently suffers from a complex discrete-continuous conflict: the need to simultaneously optimise continuous surface curvatures while selecting discrete glass materials. LDG-EA cleanly resolves this bottleneck through two key mechanisms:

- **Behaviour Descriptors:** The algorithm explicitly maps and partitions the design space into specific niches, defined by curvature-sign patterns and discrete material indices.
- **Localised Continuous Search:** By confining the evolutionary search strictly within a single descriptor niche at a time, the algorithm is shielded from the catastrophic fitness discontinuities that typically occur when dynamically swapping glass types.

The effectiveness of LDG-EA in navigating these discrete-continuous conflicts establishes it as an exceptional benchmark. Traditional evolutionary strategies and standard gradient-based optimisers often struggle with the 24-variable complexity of the Double-Gauss problem, typically collapsing into a narrow set of solutions or becoming trapped in a single local minimum. The leverage of LDG-EA is that it acts as a comprehensive topological mapper. Given a practical one-hour computational budget, the framework can autonomously generate tens of thousands of diverse, physically viable candidate designs, vastly exceeding the exploratory reach of standard baselines. More importantly, this massive volume of structural diversity does not necessarily compromise precision, experimental evaluations demonstrate that the best designs discovered autonomously by LDG-EA achieve optical performance scores that rival refined reference lenses [ATB+26].

Ultimately, LDG-EA represents a highly specialised, manually engineered meta-heuristic tailored specifically to overcome the strict domain constraints of optical physics. Rather than attempting to fundamentally redesign this established architecture, the objective of this thesis is to ascertain whether the LLaMEA framework can autonomously synthesise a competitive algorithm from scratch. By treating the complex physical environment as a strict black-box evaluator, this investigation aims to determine if an LLM-driven evolutionary approach can independently discover optimisation strategies that match, or closely approach, the exceptional performance scores set by the LDG-EA baseline.

3 Methodology

This section outlines the technical approach and architectural framework utilised to automatically generate, evaluate, and refine mixed-variable optimisation heuristics for the Double-Gauss optical lens design problem. The system builds upon the LLaMEA framework, which has been adapted to support complex physical simulations, dynamic prompt injection, and secure parallel processing. The approach relies on the seamless integration of two decoupled domains: the algorithmic discovery framework (LLaMEA) and the optimised evaluation engine (Camera Lens Simulation).

3.1 Problem Formulation: The Double-Gauss Model

The optical design of a Double-Gauss camera lens is formulated as a Mixed-Integer Non-Linear Programming (MINLP) problem. The target is to minimise a loss function $F(\Theta)$ representing optical aberrations, distortions, and physical manufacturability constraints. The optimisation search space is defined in twenty-four dimensions, partitioned into a continuous geometric subspace and a discrete material catalogue subspace:

$$\Theta = [\Theta_{geom}, \Theta_{mat}] \in \mathbb{R}^{24} \tag{2}$$

3.1.1 Continuous Geometric Subspace (Θ_{geom})

The first eighteen dimensions (θ_i for $i \in \{0, \ldots, 17\}$) parameterise the physical geometry of the lens elements. This continuous subspace is defined as:

$$\Theta_{geom} \in \mathbb{R}^{18} \tag{3}$$

It comprises:

- **Curvatures (10 parameters):** Optimisation parameters determining the radius of curvature for each spherical refracting surface.
  - $\theta_0 \in [0, 1000]$ (The initial boundary curvature)
  - $\theta_i \in [-1000, 1000]$ for $i \in \{1, \ldots, 9\}$
- **Thickness and Air Spaces (8 parameters):** Parameters governing the axial thickness of the glass elements and the physical air gaps separating them.
  - Glass Thickness (6 parameters) $\in [t_{min}, 20]$
  - Air Gaps (2 parameters) $\in [t_{min}, 50]$

Note: t_{min} represents the structurally required minimum target axis thickness set to 0.10 by default.

3.1.2 Discrete Material Subspace (Θ_{mat})

The remaining six dimensions (θ_i for $i \in \{18, \ldots, 23\}$) determine the specific glass materials selected for each individual lens element. This subspace is purely discrete and categorical:

$$\Theta_{mat} \in \{0, \ldots, |\mathcal{C}| - 1\}^6 \subset \mathbb{Z}^6 \tag{4}$$

where $|\mathcal{C}|$ represents the total number of available glasses in the chosen material catalogue (for the Schott[45] optical catalogue utilised in this study, $|\mathcal{C}| = 120$).

Each discrete coordinate θ_i acts as a direct index mapping to this physical catalog, seamlessly injecting the corresponding real-world refractive indices (n_d) and Abbe numbers (V_d) into the simulation for ray-tracing evaluation.

3.1.3 Search Space Normalisation

To insulate the optimiser from varying physical scales (e.g., millimeter thicknesses versus integer catalogue IDs) and to provide a standardised landscape for the generated algorithms, the search space is normalised to a hypercube:

$$\Theta_{norm} \in [-1.0, 1.0]^{24} \tag{5}$$

The evaluation framework maps the normalized search space Θ_{norm} back to the physical parameter space $\Theta_{real} \in [lb, ub]^{24}$ using a linear projection on every evaluation call:

$$\Theta_{real,i} = lb_i + \frac{\Theta_{norm,i} + 1.0}{2.0} \cdot (ub_i - lb_i) \tag{6}$$

where $lb \in \mathbb{R}^{24}$ and $ub \in \mathbb{R}^{24}$ represent the lower and upper bounds of the physical design envelope, respectively.

3.2 System Architecture and Execution Pipeline

The foundation of the evaluation pipeline is a highly modular, object-oriented framework designed specifically to isolate untrusted, LLM-generated code from the core simulation engine. As illustrated in the workflow diagram 15, the pipeline orchestration follows a strict, self-correcting cyclical structure:

1. **Initialisation and Hardware Routing:** The pipeline begins by parsing command-line arguments and setting up the local execution directories. It dynamically detects the available hardware topology (such as SLURM allocations on HPC clusters, Apple Silicon, or NVIDIA GPUs) to safely optimise the parallel worker count. Concurrently, it initialises the designated LLM routing backend (e.g., Gemini API or Ollama).

2. **Sandbox Creation and Prompting:** Before generation begins, the system reads the specific experimental configurations and defines the system prompts. A secure, short-lived sandbox environment is created using isolated Python subprocesses. Necessary scientific packages (such as JAX) and JIT-compilation caches are injected directly into this sandbox to mitigate computational overhead.

3. **LLaMEA Generation and Validation:** Inside the generational loop, the LLM yields a string containing a novel `python` optimisation strategy. This code is executed via Python's `exec` command within the sandbox. The framework implements resilient scripting to extract the targeted `Optimiser` class. If the class is missing, or fails to parse, an invalid fitness penalty ($-\infty$) is returned to the LLM along with the appropriate feedback.

4. **Dry-Testing:** To prevent the expensive physics engine from hanging on malformed algorithms, successfully parsed optimisers are subjected to a "Dry Test" against a fast, mock objective. Fatal execution errors caught here map generated tracebacks directly terminating the run.

5. **Bounded Simulation and Feedback Aggregation:** Upon passing the dry test, the full `DoubleGaussObjective` simulation is initialised. The LLM-generated optimiser instance is evaluated across multiple training seeds. The framework records the mean optical loss and transparently intercepts internal debugging (stored in the generated log files). This aggregated final fitness score, alongside the textual execution feedback, is passed back and the next generation is triggered.

3.3 Environment Sealing and Constraint Handling

Mixed-variable optimisation environments are highly susceptible to "reward hacking," especially when continuous optimisers operate over discrete categorical boundaries. In early iterations of this pipeline, algorithms exploited the lack of strict rounding constraints to evaluate fractional material indices, resulting in mathematically superior but physically impossible "phantom glasses."

To ensure the physical realizability of the lens designs and establish a true baseline, a strict "sealing" architecture is implemented inside the JAX evaluation sandbox. Any arbitrary real-valued search vector Θ_{real} proposed by the LLM is forcibly mapped into a strictly valid MINLP design vector Θ_{proj} via a boundary projection operator:

$$\Theta_{proj,i} = \begin{cases} \text{clip}(\Theta_{real,i}, lb_i, ub_i) & \text{for } i \in \{0, \dots, 17\} \\ \text{rint}(\text{clip}(\Theta_{real,i}, lb_i, ub_i)) & \text{for } i \in \{18, \dots, 23\} \end{cases} \tag{7}$$

where $\text{rint}(\cdot)$ is the round to nearest integer operator. This snapping guarantees that physical thicknesses and curvatures stay within physical boundaries (preventing structural element intersections), and that the categorical glass IDs are snapped exactly to physical catalogue indices. Additionally, the exact JAX analytical gradients and Hessian matrices dynamically injected into the sandbox are calculated relative to these strictly bounded spaces.

4 Results and Discussion

4.1 CodeBLEU Implementation and Structural Analysis

To quantify the structural and syntactic evolution of the generated optimisation algorithms, an incremental CodeBLEU [RGL+20] evaluation pipeline was implemented. CodeBLEU measures the overlap of n-grams, Abstract Syntax Trees (ASTs), and data flows, yielding a unified similarity score in the range $[0, 1]$. The pipeline traverses the `output-analysis` directory to aggregate log files, incrementally computing an $N \times N$ pairwise similarity matrix. The algorithm handles LLM-generated syntax errors by assigning a baseline similarity score of 0.0 to AST parse failures and computes a symmetric distance metric by averaging directional similarities.

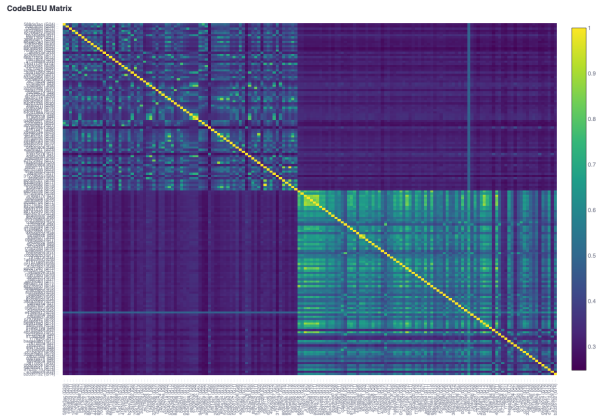
Analysis of the CodeBLEU matrices reveals two primary structural phenomena: the `lens_v4` cohort exhibits high internal similarity and strict structural separation between Elitism configurations (Figure 3b), whereas `lens_v5` displays lower overall similarity and weaker configuration distinction (Figure 4a). This divergence is driven by the following factors:

1. Mutation Prompts: In `lens_v4`, prompts request incremental improvements (Figure 16b), encouraging localised structural adjustments. Conversely, `lens_v5` mutation prompts (Figure 16d) instruct global paradigm shifts, reducing syntactic lineage and lowering similarity scores.

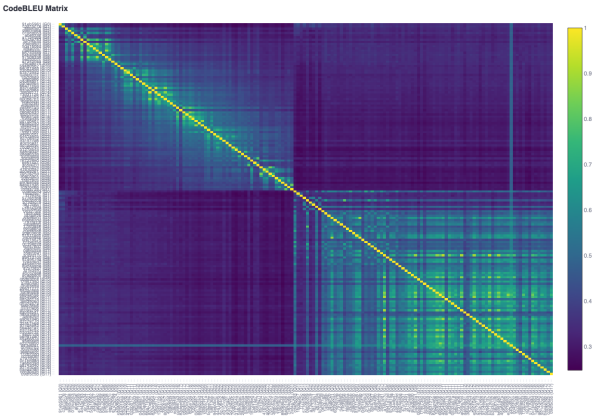
2. Elitism: The `lens_v4` matrix shows clear demarcation between Elitism configurations 3b, with dense clusters of parent-child resemblance when active. In `lens_v5`, aggressive mutation prompts override the stabilising effect of Elitism, maintaining low CodeBLEU scores punctuated only by faint generational lineage traces.

3. Algorithmic Complexity: Integrating the exact Hessian matrix (`hess_func`) in `lens_v5` necessitates complex boilerplate for regularisation and `trust-constr` solver integration. The variety of valid linear algebra implementations further dilutes structural similarity compared to the rigid L-BFGS-B loops in `v4`.

4. Inter-Experiment Correlation: A higher inter-experiment correlation exists between the

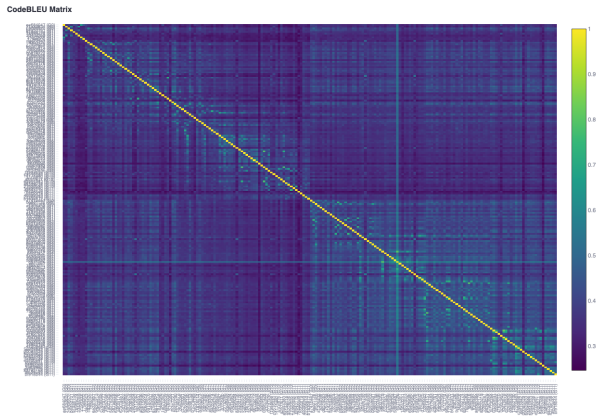


(a) `v4` CodeBLEU similarity matrix, separated by experiment and fitness.

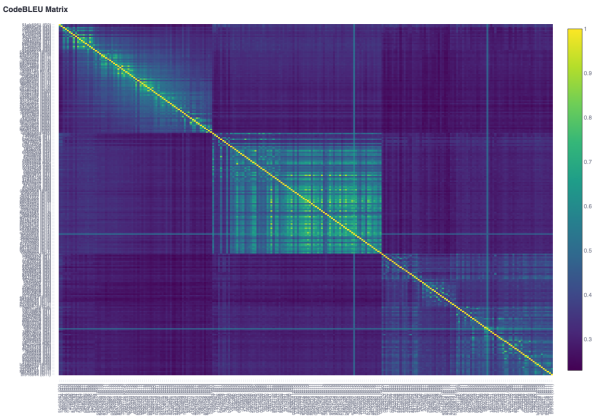


(b) `v4` CodeBLEU similarity matrix, grouped by experiment and generation.

Figure 3: CodeBLEU similarity matrices for the `v4` cohort. Top Left: `v4 False`, Bottom Right: `v4 True` (invalid scoring classes removed).



(a) `5` cohotCodeBLEU matrix separated by experiment and generation. Top Left: `v5_false`, Bottom Right: `v5 True`.



(b) Full 4 experiment correlation matrix, top left:`v4 False`, middle:`v4 True` bottom right:`v5 False` tov`5 True`

Figure 4: CodeBLEU similarity matrices experiment wide(b) and v5 cohort(a),(invalid scoring classes removed).

`v5 True` and `False` branches compared to `v4`. While partially attributable to equalised generation counts, this baseline similarity is primarily driven by the mandatory inclusion of complex Hessian processing boilerplate in all `v5` algorithms, heightening the baseline correlation and eliminating the high similarity clusters due to the algorithmic complexity.

4.2 Overall Score Analysis and Benchmark Comparison

To evaluate exploitation capabilities, the top-performing optimisation classes from each experimental cohort underwent extended standalone local evaluations. These optimisers were rerun with a heightened computational budget of 200,000 evaluations 5. Figure 9 shows a 50,000 parameter rerun of top-performing algorithms from each experiment. The `v4 True` experimental branch consistently produced the highest-performing classes, this is also reflected in Table 2, which presents the overall experiment statistics. While `v4 True` had a better mean and median score compared to the other cohorts, when running these `v4 True` algorithms with the enhanced budget, they all achieved 'high quality' scores in the range of $[0.2, 5] \times 10^{-3}$. Thereby demonstrating that optimisers relying strictly on first-order continuous gradients navigated the mixed-integer topology more effectively than those equipped with the exact analytical Hessian matrix in the `v5` cohort.

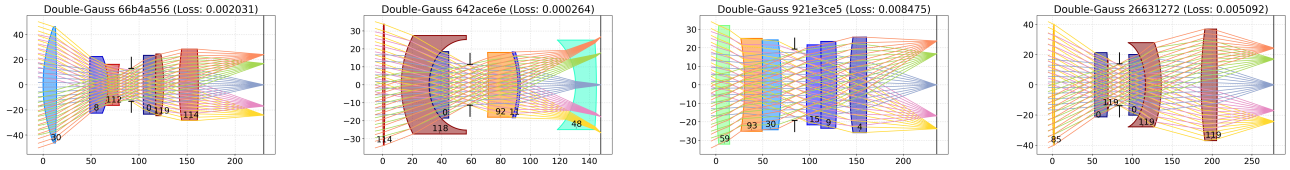


Figure 5: Top 4 performing from `v4 True` generated algorithms with class ID listed as header along with loss function, ran on the stand alone evaluation environment with seed = 25, and budget factor 200,000

Experiment	Mean Score	Median Score	Total Invalid	Failed Dry Run	Timed Out	Syntax Errors	Valid Classes
v4 False	-0.1974	-0.1179	74	22	18	0	76
v4 True	-0.0961	-0.0291	66	1	38	1	84
v5 False	-58.0941	-0.1739	63	15	6	1	37
v5 True	-450.5637	-0.1925	52	2	0	2	48

Table 2: Experiment Statistics showing performance and invalid execution breakdown across LLaMEA variants.

The leading optimisation class from the `v4 True` cohort (ID: `642ace6e`) converged to a scalar loss of $2 * 10^{-4}$, proving highly competitive with the state-of-the-art benchmark LDG-EA [ATB+26].

As established, the true strength of the LDG-EA lies in its ability to discover a wide variety of candidate solutions for subsequent local refinement. Kirill et al. demonstrate that a single computationally expensive LDG-EA run produces an average of 14,741 local-minimum candidates distributed across 636 distinct behaviour descriptors [ATB+26]. However, achieving this scale of exploration required executing LDG-EA on a high-performance cluster node (512 GB RAM, 128 CPU cores) utilising up to 100 hardware threads concurrently for one hour. In contrast, the LLM-generated optimisers were evaluated locally on an 18 GB MacBook, requiring approximately 120 seconds per seed for a standard budget of 50,000 evaluations.

Because the LLM-generated algorithms inherently converge on a single local optimum per run, assessing their exploratory viability required sequential execution across multiple seeds. A standalone experiment executing the `642ace6e` algorithm across 20 independent seeds (with a budget factor of 50,000 evaluations each) successfully produced a viable candidate solution in 12 of the 20 cases. Here, a "candidate solution" is defined as any configuration achieving a loss function $< 10^{-2}$, consistent with the unofficial threshold established by Kirill et al. Furthermore, an analysis of the generated convergence graphs reveals that the `642ace6e` algorithm consistently reaches these "candidate" threshold values within a mere 10,000 budget factor as seen in 7.

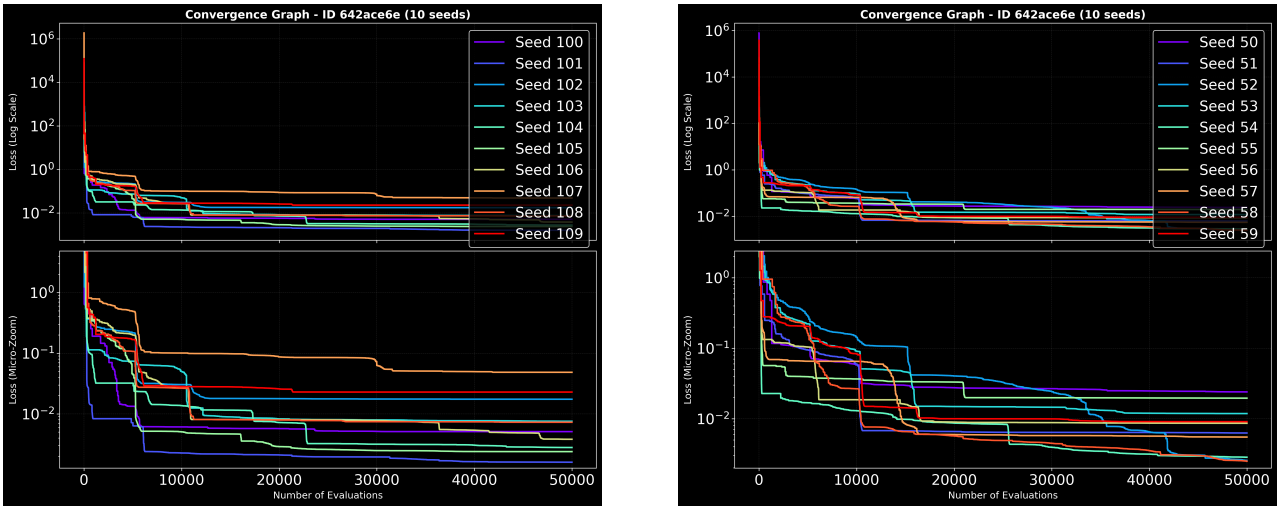
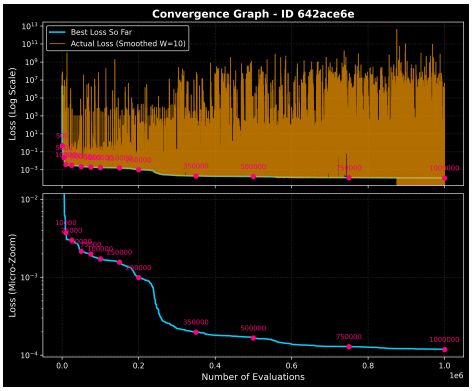


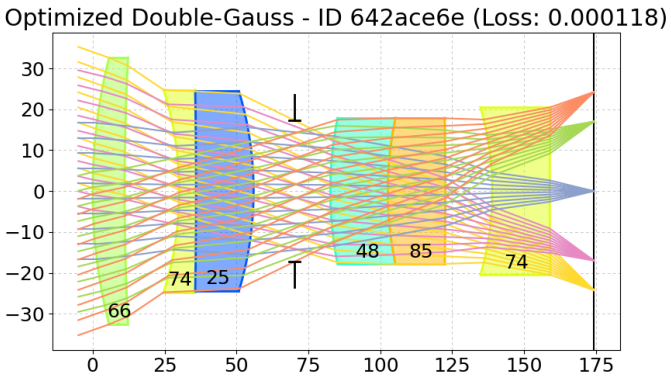
Figure 6: Convergence graphs for standalone class `642ace6e` multi-seed runs with 50,000 budget factor showing SOTA scores. Left: Seeds 100-109, Right: Seeds 50-59.

When evaluating absolute peak exploitation, the LDG-EA algorithm achieved a single best score of 3×10^{-4} [ATB+26]. The `642ace6e` algorithm outperformed this benchmark when allocated an extended computational budget, converging to a scalar loss of 2.64×10^{-4} at 200,000 evaluations, 1.33×10^{-4} at 500,000 evaluations, and 1.18×10^{-4} at 1,000,000 evaluations7a, although the single best score is quite insignificant in optical lens design this presents an interesting insight on the local

refinement capabilities of the generated algorithms. This high yield of valid candidate solutions from sequential seeding, combined with local refinement, suggests that extrapolating the LLM-generated heuristic across thousands of parallel seeds could generate a wide, highly competitive variety of candidate lenses. While this computational scaling poses a compelling direction for future research, the current results already suggest the LLM heuristic is capable of achieving single state of the art scores, while vastly outperforming all non state-of-the-art optimisation benchmark algorithms. As expected, the Random Search algorithm performed poorly, achieving average scores of $\sim 5 \times 10^{-1}$. Similarly, while the RLEMMO baseline also converged on a single solution, it failed to even reach the 10^{-2} candidate threshold despite an exhaustive 1,000,000 evaluation budget (see convergence plot, Figure 14), decisively underscoring the search power of both the LLM-generated heuristic and LDG-EA.



(a) `642ace6e` 1m budget factor convergence graph



`642ace6e` 1m budget factor optimised lens

Figure 7: Convergence and lens design achieved by class `642ace6e` with a budget factor of 1m

Because the LLM-generated algorithms inherently converge on a single local optimum per run, assessing their exploratory viability required sequential execution across multiple seeds. A standalone experiment executing the `642ace6e` algorithm across 20 independent seeds (with a budget factor of 50,000 evaluations each) successfully produced a viable candidate solution in 12 of the 20 cases. Here, a "candidate solution" is strictly defined as any configuration achieving a loss function $< 10^{-2}$, consistent with the threshold established by Kirill et al. Furthermore, an analysis of the generated convergence graphs reveals that the `642ace6e` algorithm consistently reaches these "candidate" threshold values within a mere 10,000 budget factor.

When evaluating absolute peak exploitation, the LDG-EA algorithm achieved a single best score of 3×10^{-4}. The `642ace6e` algorithm decisively outperformed this benchmark when allocated an extended computational budget, converging to a scalar loss of 2.64×10^{-4} at 200,000 evaluations, 1.33×10^{-4} at 500,000 evaluations, and 1.18×10^{-4} at 1,000,000 evaluations. This high yield of valid candidate solutions from sequential seeding, combined with unparalleled local refinement, suggests

that extrapolating the LLM-generated heuristic across thousands of parallel seeds could generate a wide, highly competitive variety of candidate lenses with extreme computational efficiency. While this computational scaling poses a compelling direction for future research, the current results already establish the LLM heuristic as vastly superior to standard baselines. As expected, the Random Search algorithm performed poorly, achieving average scores of $\sim 5 \times 10^{-1}$. Similarly, while the RLEMMO baseline also converged on a single solution, it failed to even reach the 10^{-2} candidate threshold despite an exhaustive 1,000,000 evaluation budget, decisively underscoring the superior search power of both the LLM-generated heuristic and LDG-EA.

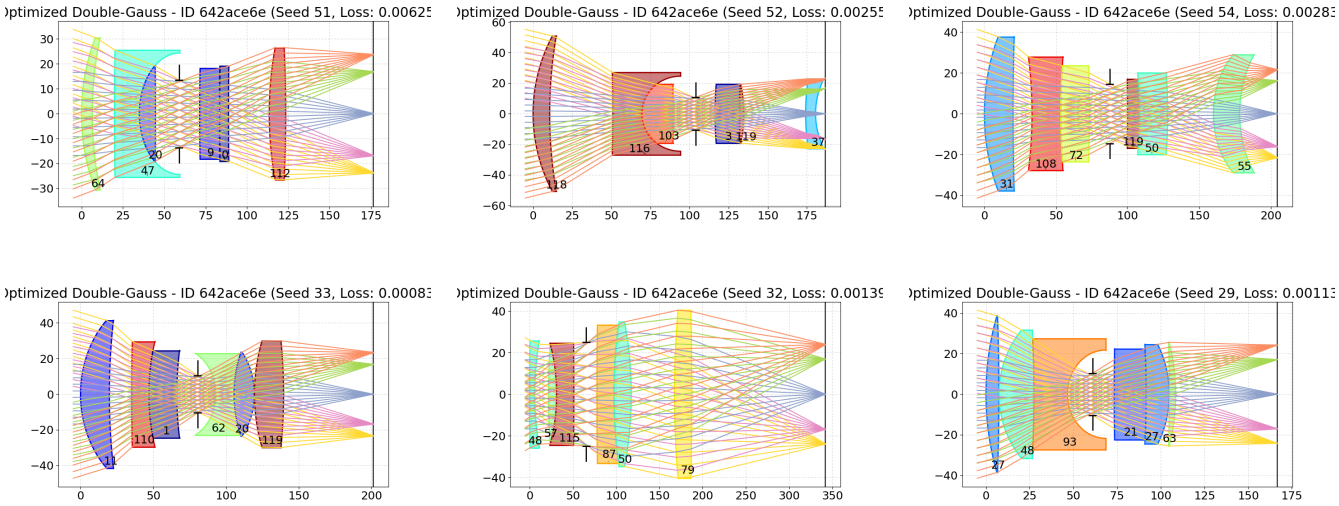


Figure 8: Evaluation of the top-performing generated optimiser (`642ace6e`) executed via the standalone evaluation script. **Top Row:** Top candidate lenses extrapolated over 20 independent seeds with a 50,000 evaluation budget factor. **Bottom Row:** The best candidates out of 10 independent seeds with an extended 200,000 evaluation budget factor (all achieving a scalar loss $\leq 0.5 \times 10^{-2}$). The header denotes the specific optimiser class ID, the random seed used, and the final optical loss achieved.

Despite this computational disparity, the best lenses generated by the `642ace6e` optimiser outperform the top five candidate lenses presented in the LDG-EA study. While the LLM-generated optimiser is limited by its tendency to converge on a single local optimum per run, executing it sequentially across 10 different seeds yielded diverse discrete compositions with consistent scores (≈ 0.002) comparable to the LDG-EA baseline. The LLM algorithm matched LDG-EA's absolute best reported score of 0.0005 given a budget factor of 200,000, and surpassed it at a 1,000,000 budget factor. Convergence graphs indicate that the LLM optimiser consistently secures competitive scores (0.009 or better) within a budget factor of just 25,000. Furthermore, the LLaMEA-generated optimiser outperformed the Reinforcement Learning baseline; the RLEMMO architecture stalled at a loss of ~ 0.011 even with a budget of 2.5 million evaluations.

Beyond scalar fitness, visual inspection of the resulting elite lens architectures (Figure 5)

demonstrates significant morphological diversity across the 24-dimensional search space. Despite achieving similar optical fidelity thresholds, the topological structures—encompassing refractive material selections, surface curvatures, and inter-element distances—vary considerably. This indicates that numerous disparate, physically viable parameter combinations can achieve competitive results across the optical lens design landscape.

4.2.1 Similarities within the Elite `v4 True` Cohort

Top performing classes within the `v4 True` branch reveals high mutual programmatic similarity (CodeBLEU ~ 0.7). These heuristics form a densely correlated cluster within the global similarity matrix. The algorithms share an architectural boilerplate featuring robust initialisation routines, standardised Latin Hypercube Sampling (LHS), and identical continuous-discrete boundary handling loops. However, their core algorithmic implementations diverge, deploying different mutation formulations, crossover logic, and triggers for localised gradient-descent (memetic) integration. This indicates that the LLM identified a stable programmatic framework for the physics evaluation loop while exploring distinct mathematical search strategies.

Despite implementational differences, the `v4 True` optimisers exhibit a macroscopic search strategy tailored to the MINLP environment, partitioning their computational budget into global discrete exploration followed by local continuous refinement. The algorithms prioritise valid discrete parameter combinations (glass material IDs) by implementing heavy-tailed recovery mechanics, such as Cauchy jumps [EST26] or Lévy flights [TNN14], to escape invalid domains caused by Total Internal Reflection (TIR) and boundary constraints. Following discrete stabilisation, the optimisers deploy localised, gradient-based refinement (e.g., memetic L-BFGS-B integration) using the continuous gradient. Furthermore, these algorithms implement stagnation detection, culling under performing subsets (20%) of the population and injecting genetic diversity via LHS to avoid deep local minima. Convergence traces for the leading class `642ace6e` show continuous granular refinement, with the loss function minimising consistently up to and possibly beyond 1,000,000 evaluations7a.

4.2.2 Structural Divergence and Early Termination in the `v4 False` Cohort

In contrast to the `v4 True` optimisers, the `v4 False` branch exhibits a fragmented algorithmic landscape (CodeBLEU ~ 0.4). The lack of Elitism prevented the LLM from converging upon a unified architectural boilerplate, resulting in algorithms with widespread evolutionary dispersion and lower structural similarity. This evolutionary fragmentation is vividly corroborated by the t-SNE diagrams tracking algorithmic lineage across evaluations (Figure 11). As seen in Figure 11b, the `v4 True` cohort forms dense, centralised hubs (represented by large nodes) where elite

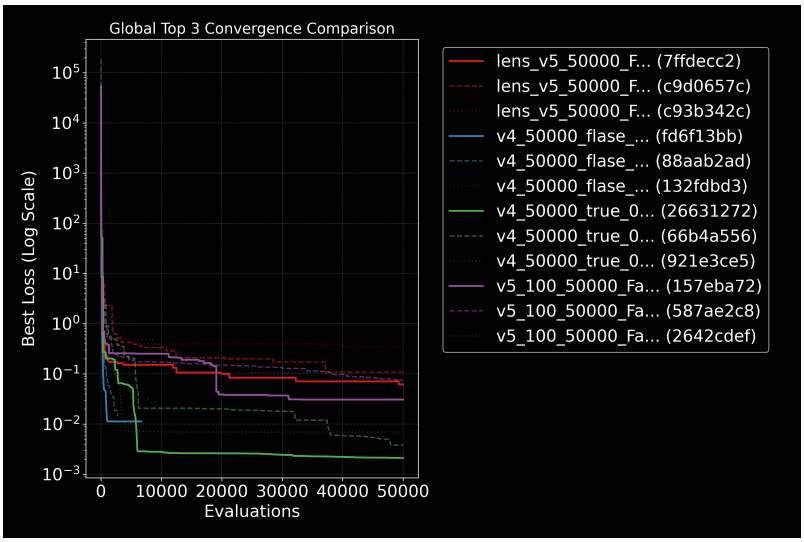
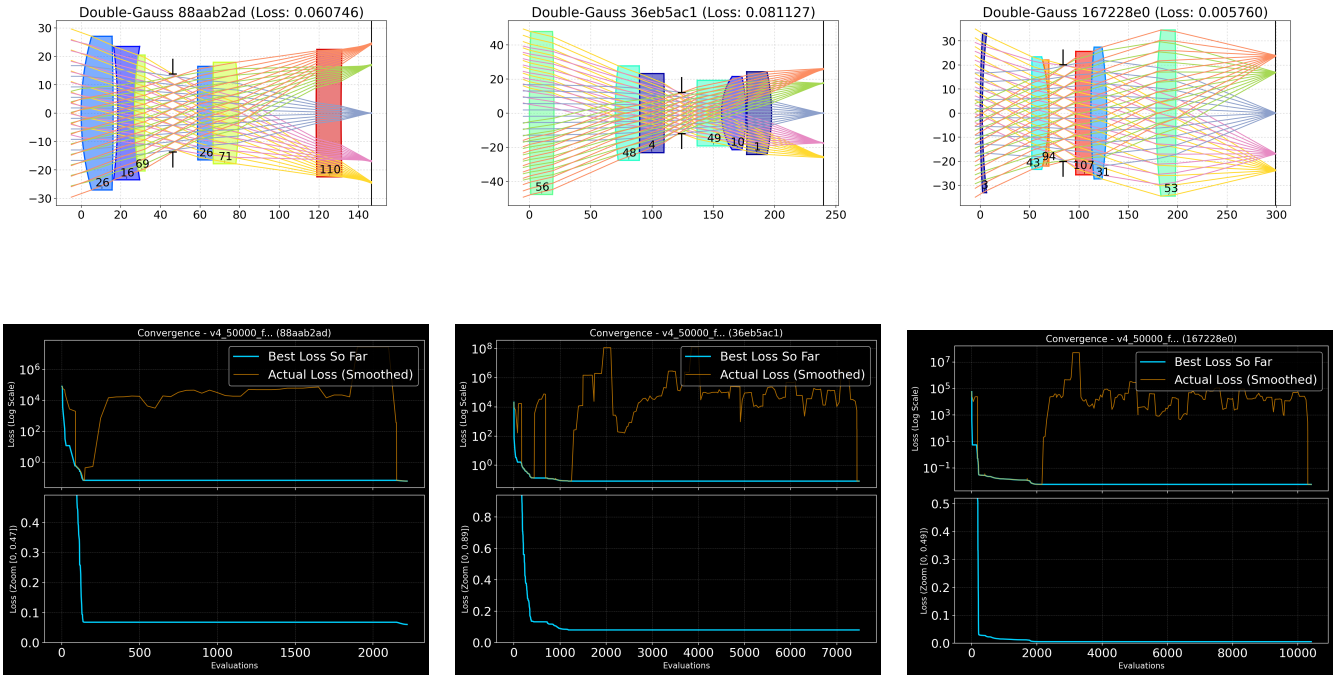


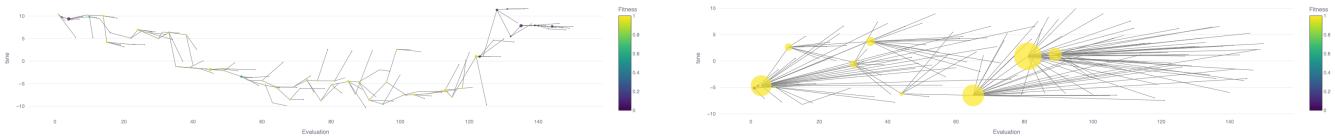
Figure 9: Experiment convergence plot showing convergence rates of top 3 experiments from all experiments.



(a) v4 False, 50k, s=25

Figure 10: **Top Row:** Visualisation of top 3 performing lens `v4 False` lens configurations with their class ID listed as header along with the loss function. **Bottom Row:** Said optimisation algorithm's convergence graphs top graph on a logarithmic scale, bottom graph on a linear scale. All evaluation runs were run through the stand alone evaluation script on seed 25, with a budget factor of 50,000.

algorithms are preserved and iteratively refined, spawning numerous structurally similar descendants. Conversely, the t-SNE projection for the `v4 False` cohort (Figure 11a) reveals a chaotic, wandering trajectory devoid of central anchor points. This erratic structural drift is the consequence of a combined compounding experimental configurations. The absence of Elitism (`elitism=False`) prevents the preservation of high-performing architectural frameworks, but the `v4 False` branch was also restricted to a smaller generational population size of 4 (compared to 9 in `v4 True`). This narrow evolutionary width, combined with the inability to anchor on past successes, forced the LLM into continuous, undirected architectural oscillations. Consequently, the optical fidelity achieved by the `v4 False` cohort is generally an order of magnitude inferior to the `v4 True` branch ($\sim [6, 0.5] * 10^{-2}$vs.$\sim 2 * 10^{-3}$ at 50,000 budget factor).



(a) `v4 False` t-SNE diagram

(b) `v4 True` t-SNE diagram

Figure 11: `v4` t-SNE diagrams generated through the `iohblade-webapp` [vSB25], visualising the evolutionary structural drift across evaluations.

This performance degradation stems primarily from acute budget mismanagement. Several top `v4 False` optimisers employ rigid stopping criteria, artificially halting the optimisation process after utilising only 1% to 5% (2,000 - 10,000) of the allocated budget. For example, Class 1 (`88aab2ad`) managed glass variables with a gradient-aware mutation core but terminated via a strict 30-generation cap. Class 2 (`36eb5ac1`) implemented an Adaptive Differential Evolution approach with a Simulated Annealing sequence but prematurely converged near 10,000 evaluations. Class 3 (`167228e0`), a Multi-Strategy Differential Evolution ensemble, utilised Gaussian noise and Lévy flights but similarly terminated via a 100-generation cap. Their reliance on rigid stopping criteria prevented the cohesive discrete-locking and sequential refinement observed in the `v4 True` cluster.

4.2.3 Mechanics in the Elite `v5 True` Cohort

The `v5 True` cohort integrated exact second-order continuous derivatives, yielding top-performing classes that abandoned standard Differential Evolution in favour of Particle Swarm Optimisation (PSO) coupled with a `trust-constr` local search. To manage the discrete dimensions where exact gradients fail, these algorithms actively freeze categorical variables before triggering Hessian-guided local refinement. Additionally, they autonomously enforce positive-definiteness on the local (18×18) Hessian matrix by extracting the absolute values of its eigenvalues, preventing backward steps in non-convex regions, while adhering to designated evaluation budgets.

Methodological differences emerged in Hessian and swarm management. Class A resizes its swarm adaptively and applies a Hessian whitening matrix to transform particle velocity direction. Class B utilises a rigid swarm, scales scalar inertia weight via the Hessian condition number to dampen velocity in unstable basins, and governs categorical jumps with Simulated Annealing. Class C synthesises these approaches, utilising Hessian whitening and condition-based inertia throttling while explicitly controlling discrete mutation probability using curvature dampeners and Simulated Annealing. Despite generating mathematically robust boilerplate, the `v5 True` optimisers were outperformed by the `v4` branch.

4.2.4 Methodological Divergence in the `v5 False` Cohort

The `v5 False` cohort produced disparate mathematical philosophies, though all algorithms utilised a memetic `trust-constr` local solver and computed the exact Hessian matrix to map local curvature. Class 1 and Class 3 precondition velocity vectors via eigen-decomposition of the Hessian matrix within a PSO framework, sharing a relatively high similarity score (0.669). However, Class 1 manages stagnation actively via anisotropic Lévy flights, whereas Class 3 relies on pure convergence, executing local search on every iteration. Conversely, Class 2 implements Differential Evolution (DE), modifying spatial difference vectors using the inverse Hessian. Comparing the PSO classes to the DE-based architecture of Class 2 drops the CodeBLEU similarity to 0.39.

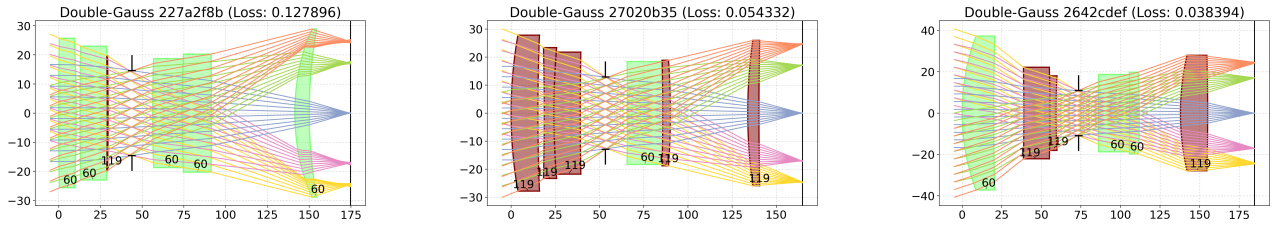


Figure 12: Top performing `v5 False` lens combinations from showing the unfavourable choice of lens material configurations with class ID and loss function as header (all generated with a 200,000 budget factor with seed 42 on the stand alone evaluation environment)

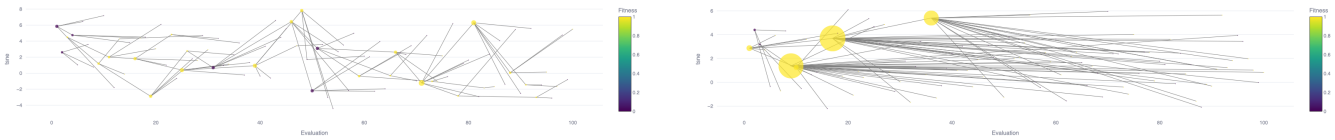
4.2.5 Categorical Entrapment and the "Hessian Trap" in `v5`

Despite their sophisticated integration of second-order continuous derivatives, visual inspection of the `v5` experiments reveals a severe shortcoming: the algorithms frequently converged upon highly sub-optimal, homogeneous discrete glass configurations (e.g., assigning identical materials to all lenses, or splitting the elements into two uniform material blocks of three) as seen in Figure 12.

This phenomenon can be diagnosed as a symptom of over-exploitation, creating a "Hessian Trap." Because the `v5` algorithms utilised exact second-order solvers like `trust-constr`, they rapidly and aggressively descended into the nearest continuous geometric minimum for whatever glass combination was initially sampled. Once the surface curvatures and thicknesses were perfectly optimised for that specific, flawed glass combination, any subsequent mutation to a categorical glass variable instantly destroyed the optical geometry, resulting in massive penalty spikes. Consequently, the algorithm's acceptance criteria rejected all discrete mutations, permanently trapping the optimiser in the early sub-optimal glass basin. With few stagnation releavers the algorithms got stuck in a local optima.

By effectively commanding the algorithm to freeze categorical exploration whenever a deep continuous geometric minimum was found, the LLM mathematically guaranteed premature discrete convergence, entirely failing to replicate the wide, heavy-tailed topographical exploration successfully employed by the `v4` cohort.

Additionally, much like the `v4` cohort, the `v5 True` branch (Figure 13b) exhibits a heavily centralised, hub-and-spoke evolutionary architecture, indicative of Elitism successfully anchoring and iteratively refining high-performing boilerplate. Conversely, the `v5 False` projection (Figure 13a) displays a fragmented, wandering trajectory lacking central hubs. However, compared to the chaotic structural oscillations observed in the `v4 False` branch, the evolutionary drift in `v5 False` is visibly more constrained. This dampened random walk is most likely attributable to the larger generational population size allocated to the `v5` experiments (a generation size of 9, compared to 4 in `v4 False`). While the absence of Elitism inherently prevents convergence upon a unified architectural framework, the wider evolutionary population provides a marginal stabilising effect on the structural trajectory.



(a) `v5 False` t-SNE diagram

(b) `v5 True` t-SNE diagram

Figure 13: `v5` t-SNE diagrams illustrating algorithmic lineage. Note the dampened wandering in `v5 False` compared to `v4 False` seen in figure 11a, a consequence of the larger generation size.

4.3 Inter-Experiment Divergence

Comparing the `v4` and `v5` cohorts highlights an evolutionary paradigm shift driven by the mathematical environment and system prompts. The elite `v4` optimisers relied on first-order Memetic Dif-

ferential Evolution with decoupled Cauchy jumps. While in `v5` experiments the LLM leaned towards Hybrid Particle Swarm Optimisation, generating eigenvalue regularisation for the `trust-constr` solver and actively coupling search spaces via the continuous Hessian trace to govern discrete mutation probabilities.

While the simpler first-order heuristics of `v4` achieved superior absolute scalar losses, the `v5` outcomes illustrate both the LLM's capacity to interpret and exploit specific mathematical tools, as well as its limitations in effectively deploying them across a discontinuous search space. Visual analysis of the resulting lens architectures confirms that the `v5` algorithms consistently struggled with the mixed-variable landscape, prematurely locking into underwhelming glass material configurations that severely restricted subsequent continuous fine-tuning.

These architectural divergences correlate directly with variations in the system prompts. The `lens_v4` prompts instructed incremental, exploitative logic using first-order gradients, fostering highly conserved code structures. Where as, the `lens_v5` prompts detailed exact analytical Hessian availability and required specific second-order solvers (`trust-constr`), inherently forcing complex regularisation boilerplate and generating wider syntactic variance.

4.4 Challenges in LLM-Driven Code Execution

Bridging natural language processing and high-performance physics computation introduced several distinct architectural challenges, spanning from environment configuration to mathematical constraint enforcement:

- **Untrusted Code Isolation:** Because the LLM arbitrarily generates Python code, the system required a robust sandboxing mechanism to prevent memory leaks, infinite `while` loops, or destructive system calls. By enforcing hard timeout constraints and isolating execution states across independent `multiprocessing` workers, system stability was preserved.

- **Virtual Environment Integrity:** Ensuring persistent library availability across the decoupled LLaMEA and physics engine codebases proved more complex than anticipated. Attempts to construct a single overarching `uv` environment frequently resulted in incorrect path resolutions, systematically isolating the execution environment and cutting off access to necessary LLaMEA `uv` plugins. Maintaining strict boundary definitions and isolated environments between the host orchestrator and the execution sandbox ultimately resolved these dependency collisions.

- **Prompt Engineering Guardrails:** To streamline the optimisation task for the LLM, the system was designed to present the entire 24-dimensional search space as a normalised,

continuous manifold $[-1.0, 1.0]^{24}$. This prevents the LLM from attempting manual, error-prone discretisation or scaling logic inside its own generated code, shifting the bounding responsibility entirely to the framework wrapper. Strict prompt guardrails were essential, as change of input (prompts) consistently caused complete invalid runs.

- **Library Hallucination:** Early LLM outputs often hallucinated dependencies or imported heavy, unauthorised packages (like `pyDOE`). The sandbox establishes secure global dictionaries, explicitly restricting imported libraries to safe mathematical tooling and directly injecting necessary helper functions (e.g., Latin Hypercube Sampling initialisers) to circumvent unauthorised imports.

4.4.1 Reward Hacking and the "Fictional Glass" Exploit

LLMs naturally favour continuous search landscapes, as they allow the employment of standard gradient descent or continuous random search strategies. In early, unaligned evaluation environments, the wrapper passed the optimiser's 24-dimensional vector directly into the JAX-based physics simulation without strictly enforcing integer constraints for the discrete variables. When an optimiser proposed a fractional glass ID (e.g., 45.7), the JAX engine did not crash; instead, it inadvertently interpolated the physical properties (such as refractive index and Abbe number) between the adjacent physical glasses.

This allowed the optimiser to effectively "cheat" the physics engine. By fine-tuning floating-point glass IDs, the algorithm invented mathematically perfect "fictional glasses" that possessed optical properties non-existent in any real-world catalogue. Because these phantom glasses perfectly corrected optical aberrations in simulation, the loss function returned exceptionally low scores. However, this was a severe structural discrepancy: when forcing the design to a physically realisable state (by rounding to the nearest real glasses), the optical performance instantly collapsed.

4.4.2 Standardisation and Gradient Masking

To permanently eliminate the "fictional glass" exploit and ensure that the LLaMEA environment evaluates lenses exactly as a strict, isolated solver would, comprehensive mathematical boundaries were implemented within the evaluation wrapper:

- **Strict Integer Enforcement (Lattice Snapping):** Before any vector reaches the optical physics simulation, the objective wrapper intercepts the normalised $[-1.0, 1.0]^{24}$ input. It translates the inputs to true physical bounds and explicitly applies a round-to-nearest-integer operation (`np.rint`) to the final 6 material indices. This enforces evaluation strictly on real, available catalogue glasses, immediately neutralising the interpolation exploit.

- **Gradient Masking:** Providing a full 24-dimensional analytical gradient back to the optimiser naturally encouraged it to continue treating the discrete glass IDs as continuous, differentiable variables. To counteract this, the analytical gradient function (`grad_func`) dynamically injected into the sandbox was explicitly truncated. It computes and returns only an 18-dimensional vector corresponding exclusively to the continuous geometry variables. By denying gradient information for the discrete subspace, the optimiser is structurally forced to recognise the MINLP nature of the problem and deploy non-gradient (heuristic) search strategies to navigate the glass catalogue space.

4.4.3 Data Volume and Black Box Opaqueness

Beyond structural and mathematical obstacles, the execution pipeline presented profound operational challenges related to observability and scale. The continuous, multi-threaded generational loop of LLaMEA generates a massive volume of raw data encompassing thousands of variations of generated Python code, standard output execution traces, multi-seed fitness evaluations, and traceback errors. Systematically parsing, simplifying, and extracting meaningful insights from this sheer amount of unstructured data proved to be a significant logistical hurdle. Furthermore, the core philosophy of a Black Box optimisation framework intrinsically abstracts the internal physics calculations away from the executing optimiser. While this strict encapsulation securely isolates the evaluation environment, it creates an extreme level of opaqueness when attempting to diagnose algorithmic failures. Due to the physics engine operating entirely behind the boundary of the JAX sandbox wrapper, interpreting the root causes of systemic errors was immensely difficult. For instance, when LLM-generated algorithms consistently triggered hard sandbox timeouts, the black box abstraction completely obscured the underlying mechanism. This was a persistent and highly disruptive issue, it was difficult to determine whether a timeout was caused by suboptimal utilisation of computational resources on the HPC cluster, the volume of sequential calculations demanded by the physics simulation, or a structural flaw within the LLM-generated optimisation class failing to terminate effectively. Similarly, when algorithms reliably collapsed to return $-\infty$ fitness scores, it was unclear if the failure stemmed from unstable mathematical divergence inside the optimisers custom heuristic or a fundamental array broadcasting error, substantially complicating the debugging and prompt-refinement process.

5 Conclusions and Further Research

5.1 Conclusions

This study investigated the adaptation of the LLaMEA framework to autonomously discover and generate optimisation heuristics for a macroscopic optical lens design problem. By treating the complex, 24-dimensional Double-Gauss simulation as a strict black-box evaluator, the framework successfully abstracted away the traditional requirement for domain-specific optical expertise, synthesising highly competitive mixed-variable optimisation algorithms from scratch.

The empirical analysis yielded several critical insights into algorithmic architecture and performance:

- **Efficacy of LLM-Generated Heuristics:** The leading generated optimiser (Class `642ace6e`) outperformed standard algorithmic baselines, including Random Search and the reinforcement learning-based RLEMMO architecture. When allocated an extended computational budget, it proved highly competitive with the domain-specific state-of-the-art benchmark (LDG-EA), converging to a scalar loss of 1.18×10^{-4} and demonstrating exceptional continuous refinement capabilities.

- **Contextual Calculus Trap:** Counterintuitively, algorithms relying exclusively on first-order continuous gradients (the `v4` cohort) navigated the mixed-integer topology far more effectively than those equipped with the exact analytical Hessian matrix (the `v5` cohort). The integration of exact second-order solvers induced a rapid over exploitation of continuous geometric minima that subsequently froze categorical exploration and caused premature convergence on sub-optimal glass configurations.

- **Elitism Induced Stability:** Structural analysis via CodeBLEU and t-SNE projections suggested that Elitism is paramount for algorithmic stability. Cohorts utilising Elitism (`v4 True`) successfully anchored and refined high-performing architectural boilerplates. In contrast, removing Elitism (`v4 False`) resulted in severe evolutionary fragmentation, chaotic structural drift, and budget mismanagement, leading to order-of-magnitude degradations in optical fidelity.

Ultimately, the results confirm that the LLaMEA framework is a highly viable mechanism for algorithm discovery in complex physical simulations, capable of autonomously generating heuristics that match or exceed the peak exploitation of manually engineered, state-of-the-art solvers.

5.2 Further Research

While the generated algorithms demonstrated exceptional local refinement, their inherent tendency to converge on a single optimal basin per run presents clear avenues for future exploration and structural improvement:

- **Mass Parallelisation and Computational Scaling:** The true leverage of the established LDG-EA benchmark lies in its ability to discover vast arrays of diverse candidate solutions. Because the LLM-generated heuristics consistently yield viable candidates within a modest budget, extrapolating these algorithms across thousands of parallel seeds—matching the high-performance computational scale of LDG-EA—could rapidly map an immense volume of structurally diverse, physically viable optical layouts.

- **Mitigating the Hessian Trap:** Further research must address the categorical entrapment observed in the `v5` cohort. Investigating architectural modifications—such as dynamically throttling trust-region bounds, implementing aggressive temperature-based categorical resets, or temporally decoupling continuous exploitation from discrete exploration—could allow LLMs to safely leverage exact second-order derivatives without sacrificing topographical diversity.

- **Contextual Prompt Engineering:** While this study demonstrated success relying on purely mathematical, zero-shot system prompts, future investigations should explore the impact of domain-aware prompt injection. Analysing whether providing explicit optical physics context alters the LLM's structural decision-making could further enhance the efficiency of the generative loop.

- **Generalisation to Asymmetric Architectures:** Validating the highest-performing Elitism cohorts on more complex, asymmetric optical design challenges (e.g., freeform lenses or systems with significantly higher discrete-to-continuous variable ratios) will determine the broader generalisability of these LLM-generated heuristics.

Abstract

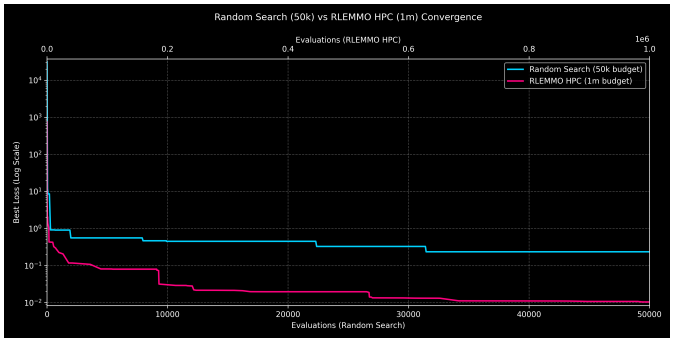


Figure 14: Convergence graph of the RLEMMO implementation(red) with a budget factor of 1m and a random search implementation(blue) with a budget factor of 50,000

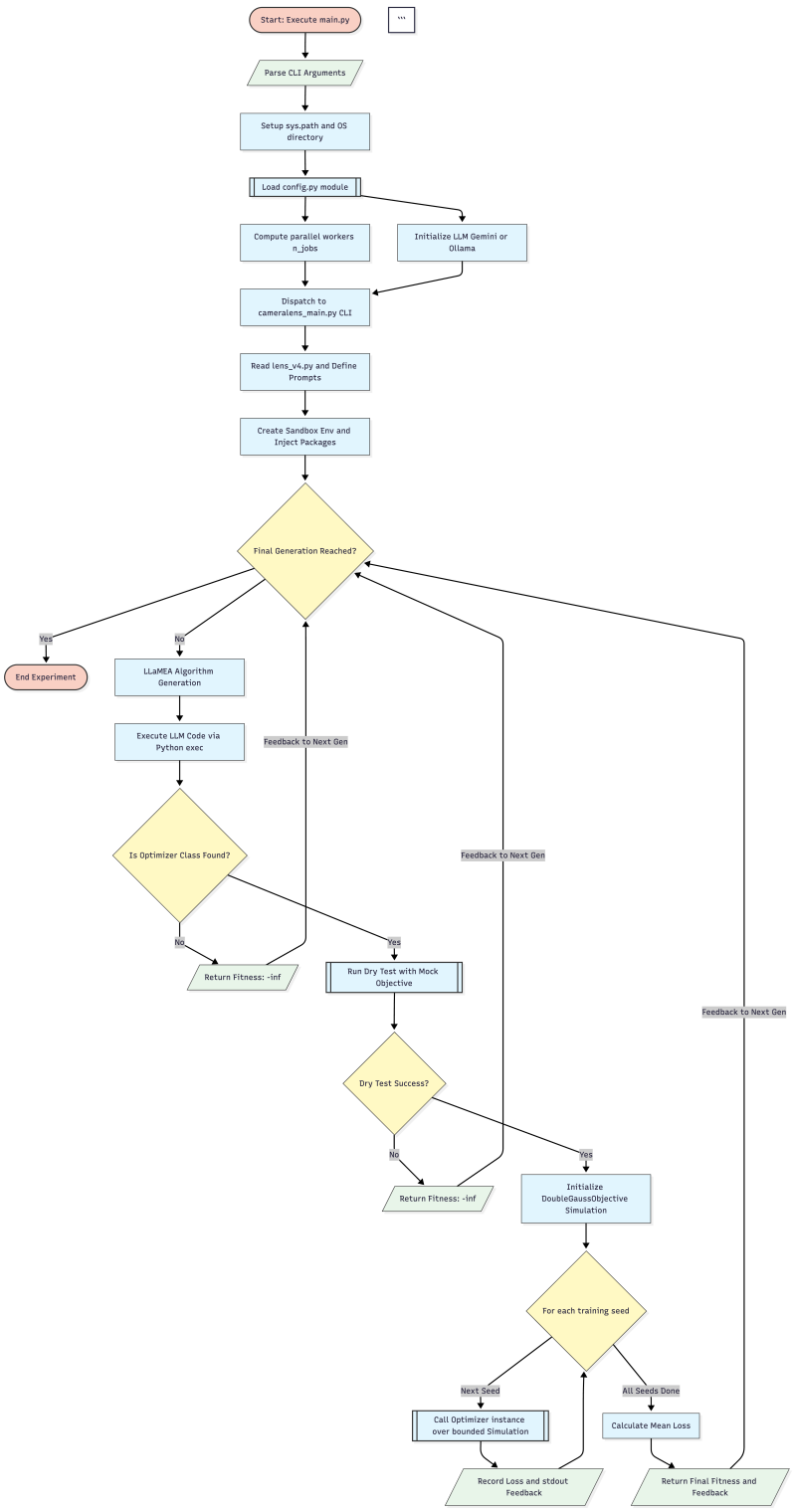


Figure 15: Workflow of the execution pipeline.

```
{
    "role_prompt": "You are a senior applied physicist with extensive background in software engineering.",
    "task_prompt": "You are an elite algorithm designer specializing in mixed-variable, black-box optimization.\n\n
     ### Problem Physics & Landscape:\nMINIMIZE a 24-dimensional camera lens design loss function.\nThe entire
     search space is normalized. EVERY dimension MUST be bounded strictly within [-1.0, 1.0].\n- Indices `x[0:18]`:
      18 Continuous geometry parameters (curvatures/distances).\n- Indices `x[18:24]`: 6 Categorical glass material
      IDs.\nThe landscape is highly non-convex and contains 'cliffs' where invalid lenses return `inf`.\n\n###
     STRICT CODING STANDARDS (CRITICAL) ###\n1. NO MANUAL DISCRETIZATION: Treat all 24 dimensions as continuous
     floats in [-1.0, 1.0]. Do NOT attempt to round, bin, or discretize indices [18:24] yourself. The target
     environment automatically handles the mapping and integer projection of the glass catalogs internally. Just
     pass floats in [-1.0, 1.0].\n2. GRADIENT DIMENSIONALITY: `grad_func(full_24d_x)` requires a 24D array but
     returns an 18D gradient array. This gradient ONLY applies to the first 18 continuous variables.\n3. SCIPY
     MINIMIZE: If using `scipy.optimize.minimize` (like L-BFGS-B) on the continuous variables, you MUST create
     wrapper functions that accept an 18D array, concatenate it with the fixed 6D categorical array, and pass the
     full 24D array to the main `func` and `grad_func`.\n4. LHS SAMPLING: Always use keyword arguments: `lhs(
     n_samples=N, n_dim=24)`.\n5. WRAPPER: Always use your internal `self._evaluate(x, func)` to track the budget.
     If `evals >= budget`, return `inf`.\n",
    "example_prompt": "Write a completely self-contained Python class named `Optimizer`.\n```python\nimport numpy
     as np\nfrom scipy.optimize import minimize\n\nclass Optimizer:\n    def __init__(self, budget: int, dim: int):\
     n        self.budget = budget\n        self.dim = dim\n        self.evals = 0\n        self.best_f = float('
     inf')\n        self.best_x = np.random.uniform(-1, 1, dim)\n\n    def _evaluate(self, x, func):\n        if
     self.evals >= self.budget: return float('inf')\n        # Ensure strictly bounded within [-1, 1]\n
     x_clean = np.clip(x, -1.0, 1.0)\n        f = func(x_clean)\n        self.evals += 1\n        if f < self.
     best_f:\n            self.best_f = f\n            self.best_x = x_clean.copy()\n        return f\n\n    def
     __call__(self, func, grad_func=None):\n        # Example: Local gradient refinement on the 18D continuous
     subspace\n        if grad_func is not None:\n            fixed_cats = self.best_x[18:24].copy()\n            \
     n            def cost_wrap(x_cont):\n                full_x = np.concatenate([x_cont, fixed_cats])\n
           return self._evaluate(full_x, func)\n                \n            def grad_wrap(x_cont):\n
        full_x = np.concatenate([x_cont, fixed_cats])\n                return grad_func(full_x) # Returns 18D
     gradient\n                \n            # Scipy strictly requires 1D float64 arrays\n            x0 = self.
     best_x[:18].astype(np.float64)\n            minimize(cost_wrap, x0, jac=grad_wrap, method='L-BFGS-B', bounds
     =[(-1.0, 1.0)] * 18)\n\n        # Add Global / Categorical search logic here...\n        \n        return self.
     best_f, self.best_x\n```\n",
    "output_format_prompt": "STRICT FORMATTING: # Description: <Description, max 2 sentences>\n # Code:\n```python\
     n<code>\n```"
}
```


(a) v4 Task and Role Prompts


```
{
    "mutation_prompts": [
        "Refine the strategy of the selected solution to improve its performance and robustness.",
        "Propose structural changes to the algorithm to better explore the search space.",
        "Optimize the internal logic and parameters of the algorithm for faster convergence.",
        "Identify potential weaknesses in the current optimization approach and address them.",
        "Refine the current algorithm by introducing more sophisticated local search or mutation operators."
    ]
}
```


(b) v4 Mutation Prompts

Figure 16: System prompts for lens experiments (Part 1: v4)

```
{
    "role_prompt": "You are a senior applied physicist with extensive background in software engineering.",
    "task_prompt": "You are an elite algorithm designer specializing in mixed-variable, second-order optimization.\
     n\n### CRITICAL ENVIRONMENT & SYNTAX GUARDRAILS:\n1. ALLOWED LIBRARIES: Use ONLY ‘numpy‘, ‘scipy‘, and ‘cma‘.
     NEVER import ‘sklearn‘ or ‘scipy.stats.qmc‘.\n2. LHS SYNTAX: For Latin Hypercube Sampling, you must use
     standard numpy: ‘pop = np.random.uniform(-1, 1, size=(n_samples, self.dim))‘.\n3. SIGNATURES: Your ‘__call__‘
     MUST exactly match this signature to pass framework dry-runs:\n   ‘def __call__(self, func, grad_func=None,
     hess_func=None, **kwargs):‘ \n4. STATE & BUDGET: Initialize ‘self.evals = 0‘ and ‘self.best_f = float(’inf’)‘
     in ‘__init__‘. Always check ‘if self.evals >= self.budget: break‘ before calling ‘func‘, ‘grad_func‘, or ‘
     hess_func‘.\n5. BOUNDARY ENFORCEMENT (CRITICAL): Inside your evaluation wrapper, you MUST strictly clip values:
      ‘x = np.clip(x, -1.0, 1.0)‘. The categorical variables x[18:24] MUST be mapped to valid glass integers [0, 5]
      using np.clip(np.round(x[18:24]), 0, 5).astype(int).\n\n### PROBLEM STRUCTURE:\nMinimize a 24-dimensional
     highly non-convex optical loss function. Bounds are strictly ‘[-1, 1]‘.\n- ‘x[0:18]‘: Continuous geometry.\n- ‘
     x[18:24]‘: Categorical material IDs.\n\n### THE HESSIAN ADVANTAGE (‘hess_func‘) ###\nYou are provided with ‘
     hess_func(full_24D_x)‘. It returns an exact ‘(18, 18)‘ symmetric matrix of second derivatives for the
     continuous parameters.\n- **CRITICAL SOLVER RULE**: If using ‘scipy.optimize.minimize‘, you MUST use ‘method=’
     trust-constr’‘ because it is the ONLY exact-Hessian solver that supports ‘bounds‘. Do NOT use ‘trust-exact‘ or
      ‘Newton-CG‘ with bounds.\n- **REGULARIZATION**: The exact Hessian will often be indefinite. You MUST ensure
     it is positive-definite by taking the absolute value of its eigenvalues or adding a sufficiently large
     identity matrix before passing it to any solver or Newton step.\n\n### INSPIRATION & BASELINE STRATEGIES (For
     Initial Guidance) ###\nTo get started, consider these proven strategies for this specific landscape:\n- **
     Memetic Hybrids:** Use Differential Evolution (or CMA-ES) for global exploration of the 24D space.
     Periodically trigger local search (‘scipy.optimize.minimize‘ with ‘trust-constr‘) on the continuous subspace
     for the best individuals.\n- **Second-Order Mutations:** Instead of random noise, perturb the 18 continuous
     dimensions using a damped, regularized Newton step: ‘np.linalg.solve(H_reg, -grad)‘.\n- **Categorical
     Refinement:** Use specialized crossover for dimensions 18-24 that strictly sample from valid ID steps ({0, 1,
     2, 3, 4, 5}).\n",
    "example_prompt": "Write a completely self-contained Python class named exactly ‘Optimizer‘.\n‘‘‘python\nimport
      numpy as np\nfrom scipy.optimize import minimize\n\nclass Optimizer:\n    def __init__(self, budget: int, dim:
      int):\n        self.budget = budget\n        self.dim = dim\n        self.evals = 0\n        self.best_f =
     float(’inf’)\n        self.best_x = np.zeros(dim)\n\n    def _evaluate(self, x, func):\n        if self.evals
     >= self.budget: return float(’inf’)\n        # Strict Boundary and Casting Enforcement\n        eval_x = np.
     clip(x.copy(), -1.0, 1.0)\n        eval_x[18:24] = np.clip(np.round(eval_x[18:24]), 0, 5).astype(int)\n
      \n        f = func(eval_x)\n        self.evals += 1\n        if f < self.best_f:\n            self.best_f = f\
     n            self.best_x = eval_x.copy()\n        return f\n\n    def __call__(self, func, grad_func=None,
     hess_func=None, **kwargs):\n        # Initialize population using standard numpy\n        pop = np.random.
     uniform(-1, 1, (10, self.dim))\n        for x in pop: self._evaluate(x, func)\n\n        # Example of using
     the exact Hessian in a local search:\n        if hess_func is not None and grad_func is not None:\n
      res = minimize(\n                lambda x_c: func(np.concatenate([x_c, self.best_x[18:]])),\n
     self.best_x[:18],\n                jac=lambda x_c: grad_func(np.concatenate([x_c, self.best_x[18:]])),\n
              hess=lambda x_c: hess_func(np.concatenate([x_c, self.best_x[18:]])),\n                method=’trust-
     exact’\n            )\n        while self.evals < self.budget:\n            # Optimization logic here...\n
             pass\n        return self.best_f, self.best_x\n‘‘‘\n",
    "output_format_prompt": "STRICT FORMATTING: # Description: <Description, max 2 sentences>\n # Code:\n‘‘‘python\
     n<code>\n‘‘‘"
}
```

(c) v5 Task and Role Prompts

```
{
    "mutation_prompts": [
        "Critically analyze the algorithmic structure you just generated. Propose a novel exploration strategy that
      better escapes deep, infeasible local minima in the continuous subspace, while still utilizing the exact
     Hessian for rapid convergence when trapped in a promising basin.",
        "Design a more sophisticated mechanism for handling the mixed-variable nature of this problem. Instead of
     treating the categorical and continuous variables as entirely separate pipelines, how can the discrete choices
      (glass materials) dynamically dictate the continuous geometric optimization, or vice versa?",
        "Introduce a self-tuning or adaptive mechanism. For example, dynamically adjust the frequency of local
     search, the mutation scale, or the population size based on runtime feedback like the improvement rate, budget
      remaining, or the condition number of the Hessian matrix.",
        "Take a completely different evolutionary or meta-heuristic paradigm (e.g., Swarm Intelligence, Simulated
     Annealing, or a Multi-Armed Bandit selector for operators) and adapt it specifically to exploit this 24D,
     second-order-enabled landscape."
    ]
}
```

(d) v5 Mutation Prompts

Figure 16: System prompts for lens experiments (Part 2: v5)

Reproducibility

5.3 Code Availability

The source code for this research is provided in the accompanying repository. The implementation integrates the BLADE framework (utilising the LLaMEA algorithm) with a custom physics engine located in the camera-lens-simulation/ directory, which models the complex Double-Gauss objective function. The blade-framework/ directory manages the LLM-driven evolutionary loop, prompt generation, and local inference execution.

5.4 Hardware

All LLaMEA generative experiments were executed on a high-performance cluster node (duranium.liacs.nl) through the Leiden Institute of Advanced Computer Science (LIACS), Leiden University. The hardware configuration features:

CPU: 20 Intel Xeon E5-2650v3 cores @ 2.30GHz GPU: Mixed array of NVIDIA GPUs (six GTX 980 Ti and two Titan X) utilised for local LLM inference RAM: 256 GB OS: Rocky Linux 9

5.5 Software Environment

The project utilises modern Python managed via uv for fast dependency resolution. Core mathematical, simulation, and evolutionary dependencies include jax, numpy, scipy, and cma. The exact environment can be reliably reproduced using the project's dependency manifest by running uv sync or through standard package installation.

5.6 Running Experiments

The LLM-driven optimization experiments are defined as modular runs within the BLADE framework. The primary generative loop can be initiated using the main entry point:

```
uv run python main.py
```

To validate, benchmark, and visualise the best-performing optimisers generated by the LLM, the standalone solver is used. This allows for rigorous testing across different budgets and random seeds:

```
uv run python solve_lens_aligned.py
```

A Streamlit web application (uv run iohblade-webapp) is also provided to interactively trace the Code Evolution Graphs and fitness progression.

5.7 Data Generation

No external datasets are required for training or evaluation. The fitness data is generated procedurally by querying the mathematical DoubleGaussObjective simulation, which evaluates the physical optics loss function (incorporating ray tracing, focal lengths, and glass indices) dynamically given a set of lens parameters.

A.6 Computational Architecture and Execution Cost To manage computational overhead efficiently across the cluster, resource allocation was deliberately throttled to max out 10 CPU cores when running two experiments concurrently.

Because the Double-Gauss simulation is highly mathematically intensive, strict temporal limits were enforced within the BLADE framework to prevent the generational loop from stalling on poorly formulated or infinite-looping Python code generated by the LLMs. Algorithm evaluation generations were capped at a hard 1,500-second timeout. Under normal circumstances, successful algorithmic iterations generated by the LLM generally completed their evaluation budget within 12 to 18 minutes.

A.7 Default Hyperparameters and Budgets The evaluation budget for the final optimiser validation against the Double-Gauss problem was heavily scaled up (ranging from 50,000 to 200,000 function evaluations) to test convergence properties comprehensively. LLM generation temperatures, mutation strategies, and population limits were defined dynamically on a per-experiment basis within the run configurations.

A.8 Random Seeds Due to the stochastic nature of evolutionary algorithms and local search heuristics, optimal algorithm validation was executed across multiple random seeds (e.g., 8 independent runs) to ensure statistical significance. Results and convergence plots report the achieved milestones and final configurations across these varying seeds.

References

[ATB+26] Kirill Antonov, Teus Tukker, Tiago Botari, Thomas H. W. Bäck, Anna V. Kononova, and Niki van Stein. Lens-descriptor guided evolutionary algorithm for optimization of complex optical systems with glass choice. *arXiv preprint arXiv:2601.22075*, 2026. doi:10.48550/arXiv.2601.22075.

[BvTM07] Florian Bociort, Maarten van Turnhout, and Oana Marinescu. Practical guide to saddle-point construction in lens design. *SPIE Proceedings*, 6667:666708, 2007. doi:10.1117/12.732477.

[CLT21] Geoffroi Côté, Jean-François Lalonde, and Simon Thibault. Deep learning-enabled framework for automatic lens design starting point generation. *Opt. Express*, 29(3):3841–3854,

Feb 2021. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-29-3-3841, doi:10.1364/OE.401590.

[COM] COMSOL AB. *Double Gauss Lens*. COMSOL AB, Stockholm, Sweden. COMSOL Multiphysics Application Library: models.roptics.double_gauss_lens. URL: https://www.comsol.com/model/double-gauss-lens-54571.

[CSJ+19] Sawyer D. Campbell, David Sell, Ronald P. Jenkins, Eric B. Whiting, Jonathan A. Fan, and Douglas H. Werner. Review of numerical optimization techniques for meta-device design. *Opt. Mater. Express*, 9(4):1842–1863, Apr 2019. URL: https://opg.optica.org/ome/abstract.cfm?URI=ome-9-4-1842, doi:10.1364/OME.9.001842.

[CTS+23] Qiao Chen, Xiongxin Tang, Feijun Song, Jiacheng Zhao, Yuanlin Zhang, Xiao Chen, Qiuyan Tang, and Fanjiang Xu. Optical design algorithm utilizing continuous-discrete variables grounded on stochastic processes. *Opt. Express*, 31(25):41428–41444, Dec 2023. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-31-25-41428, doi:10.1364/OE.504443.

[EST26] Anton V. Eremeev, Dmitri V. Silaev, and Valentin A. Topchii. Generalized heavy-tailed mutation for evolutionary algorithms, 2026. URL: https://arxiv.org/abs/2604.00502, arXiv:2604.00502.

[HL19] Thomas Houllier and Thierry Lépine. Comparing optimization algorithms for conventional and freeform optical design. *Opt. Express*, 27(13):18940–18957, Jun 2019. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-27-13-18940, doi:10.1364/OE.27.018940.

[Kid01] Michael J. Kidger. *Fundamental Optical Design*. SPIE Press, Bellingham, WA, 2001.

[LMG+24] Hongqiao Lian, Zeyuan Ma, Hongshu Guo, Ting Huang, and Yue-Jiao Gong. Rlemmo: Evolutionary multimodal optimization assisted by deep reinforcement learning. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 683–693, 2024. doi:10.1145/3638529.3653995.

[RGL+20] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. 2020. URL: https://arxiv.org/abs/2009.10297, arXiv:2009.10297.

[SO90] Doron Sturlesi and Donald C. O'Shea. Global view of optical design space. *Optical Engineering*, 29(2):207–218, 1990.

[TNN14] Truyen Tran, Trung Thanh Nguyen, and Hoang Linh Nguyen. Global optimization using lévy flights, 2014. URL: https://arxiv.org/abs/1407.5739, arXiv:1407.5739.

[vSB25] Niki van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. 2025. URL: https://arxiv.org/abs/2405.20132, arXiv:2405.20132.

[YFH24] Xinge Yang, Qiang Fu, and Wolfgang Heidrich. Curriculum learning for ab initio deep learned refractive optics. *Nature Communications*, 15(1):6572, 8 2024. doi:10.1038/s41467-024-50835-7.