EngiBench: A Framework for Data-Driven Engineering Design Research

Florian Felten[1] **Gabriel Apaza**[2,*] **Gerhard Bräunlich**[1,*] **Cashen Diniz**[1,*]
Xuliang Dong[2,*] **Arthur Drake**[2,*] **Milad Habibi**[2,*] **Nathaniel J. Hoffman**[2,*]
Matthew Keeler[1,*] **Soheyl Massoudi**[1,*] **Francis G. VanGessel**[2,*] **Mark Fuge**[1]
[1]ETH Zürich [2]University of Maryland, College Park
ffelten@mavt.ethz.ch mafuge@ethz.ch

Abstract

Engineering design optimization seeks to automatically determine the shapes, topologies, or parameters of components that optimize performance under given conditions. This process often depends on physics-based simulations, which are difficult to install, computationally expensive, and require domain-specific expertise. To mitigate these challenges, we introduce EngiBench, the first open-source library and datasets spanning diverse domains for data-driven engineering design. EngiBench provides a unified API and a curated set of benchmarks—covering aeronautics, heat conduction, photonics, and more—that enable fair, reproducible comparisons of optimization and machine learning algorithms, such as generative or surrogate models. We also release EngiOpt, a companion library offering a collection of such algorithms compatible with the EngiBench interface. Both libraries are modular, letting users plug in novel algorithms or problems, automate end-to-end experiment workflows, and leverage built-in utilities for visualization, dataset generation, feasibility checks, and performance analysis. We demonstrate their versatility through experiments comparing state-of-the-art techniques across multiple engineering design problems, an undertaking that was previously prohibitively time-consuming to perform. Finally, we show that these problems pose significant challenges for standard machine learning methods due to highly sensitive and constrained design manifolds.[2]

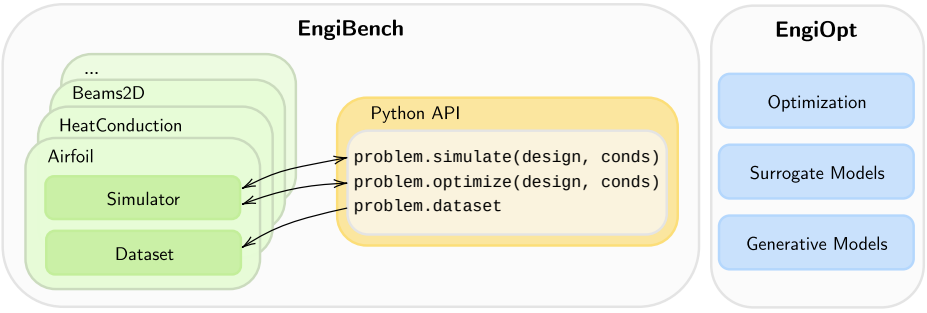


FIGURE 1: Overview of ENGIBENCH and ENGIOPT components.

[*]Alphabetically ordered.
[2]Documentation: https://engibench.ethz.ch.
Benchmarks code: https://github.com/IDEALLab/EngiBench.
Datasets: https://huggingface.co/IDEALLab.
Learning and optimization library code: https://github.com/IDEALLab/EngiOpt.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

https://doi.org/10.52202/085713-0003

1 Introduction

Designing complex engineering systems has traditionally relied on iterative optimization processes and computationally expensive simulations. Recent advances in machine learning (ML)—particularly in inverse design (ID) and surrogate modeling (SM)—offer promising alternatives. Rather than repeatedly evaluating candidate designs through costly simulations, ID methods aim to generate designs directly from desired conditions and performance specifications, while surrogate models approximate simulation outputs. By significantly reducing the number of simulation calls, these approaches enable more efficient exploration of design spaces and lower computational costs. Engineering design researchers have explored ML methods for accelerating design exploration and optimization in many different directions: approximating optimal designs [9], comparing neural operators or surrogate models [15], exploring design spaces/configurations [35], studying transfer learning [4], among many others.

However, progress in data-driven engineering design has been significantly slowed by the lack of standardized simulation environments and publicly available, diverse datasets [60]. Most existing datasets are not shared, making it difficult to reproduce results or compare methods fairly. Even for datasets that do exist, different problem formulations often use different conventions or metrics, making it difficult or time consuming to compare algorithms across problems. Lastly, generating new datasets is not only computationally expensive, but also requires configuring and deploying complex simulation software—such as for computational fluid dynamics (CFD), finite element analysis (FEA), or circuit simulation—which can take weeks or even months and demands considerable domain expertise. This complexity presents a barrier to entry for ML researchers without an engineering background, narrowing research in these domains. As a result, researchers often focus on a single application domain, limiting broader generalization and hindering cross-domain benchmarking of algorithms.

To address this gap, we introduce ENGIBENCH—an open-source library supporting multiple domains in data-driven engineering design.[3] It provides an intuitive and extensible Python API that unifies the modeling of diverse design problems and includes a suite of rigorously tested problem implementations. Each ENGIBENCH problem comprises a simulation engine and an associated dataset—encapsulating designs, objective values, attributes, and conditions—accessible via a standardized interface (see Fig. 1). To showcase the possibilities of using our framework, we also open-source a CleanRL-style [31] companion library called ENGIOPT, which contains a collection of ML and optimization algorithms integrated with ENGIBENCH.

The key contributions of this paper are:

1. Novel challenging ML tasks that include hard constraints on what defines a feasible solution, and domain-specific metrics going further than statistical similarity, see Section 2.1. We show that the proposed problems are far from trivial, as these manifolds are highly sensitive to slight changes in the design space (*e.g.*, keeping the continuity of an airfoil spline), making them difficult to learn with traditional methods, see Section 4.
2. A unified API which can be used in various optimization or ML paradigms, including inverse design, surrogate-based optimization, physics-informed neural networks (PINNs), etc. This API eases testing algorithms on a diverse set of problems for the engineering design community, see Section 3.1.
3. Easy access to a diverse collection of real-world engineering design problems and associated physics simulators modeling different physical laws. The datasets collected on these problems are also made publicly available. These include problems across aerodynamics, heat transfer, power electronics design, photonics, etc., see Section 3.3.
4. A curated set of well-known generative and optimization algorithms implementations compatible with our API. This allows for benchmarking of multiple baseline algorithms across various design problems. In our proof-of-concept experiments, we observe that, contrary to popular belief, unconditional generative adversarial networks (GANs) may outperform their conditional counterparts and diffusion models on specific metrics relevant to engineering design; see Section 4.

[3] ENGIBENCH builds on the Maryland Inverse Design Benchmark (MIDBench) project (`https://ideal.umd.edu/midbench/`), which served as an initial prototype but did not reach the maturity needed for publication or widespread adoption.

2 Preliminaries and Related Work

ENGIBENCH aims at easing the work of researchers in ML for engineering design, but also proposes new challenges for the ML community. This section first defines the problem, its solutions, and how progress is generally measured in the field, it then discusses existing related works and how ENGIBENCH differs from those.

2.1 Problem setting

The problem of finding optimal designs in engineering, often approached computationally, can be formulated as the following optimization problem. Let $x \in \mathcal{X} \subseteq \mathbb{R}^d$ represent the design variables (*e.g.*, spline parameters of an airfoil), $a : \mathcal{X} \mapsto \mathcal{A} \subseteq \mathbb{R}^k$ be attributes computed for each design (*e.g.*, mass), $c \in \mathcal{C} \subseteq \mathbb{R}^l$ represent the environmental conditions (*e.g.*, cruising velocity), $f : \mathcal{X} \times \mathcal{C} \mapsto \mathcal{F} \subseteq \mathbb{R}^o$ be a (multi-)objective function (*e.g.*, drag and lift), σ represent simulation assumptions and limitations (*e.g.*, numerical approximations), and g and h represent inequality and equality constraints, respectively (*e.g.*, a constraint ensuring the lift coefficient must exceed a given minimum). Then, without loss of generality,

$$\underset{x \in \mathcal{X}}{\text{minimize}}\ \tilde{f}(x, c, \sigma) \quad \text{subject to} \quad \begin{aligned} g_i(x, a(x), c, \sigma) &\leq 0, \quad i = 1, \dots, q, \\ h_j(x, a(x), c, \sigma) &= 0, \quad j = 1, \dots, r, \end{aligned}$$

where $\tilde{f}$ only approximates the true objective f due to the simulation assumptions & limitations σ.[4]

The solutions to this problem depend on the specific context and can take different forms: a single optimal design, a Pareto set of optimal designs when multiple objectives are considered, or a diverse set of near-optimal designs, allowing the human designer to select the most suitable one when exploring the design space.

Solving methods In engineering design, optimization methods such as genetic algorithms (GAs) or adjoint solvers have traditionally been used to search for the optimal design variables $x^* \in \mathcal{X}$. However, this process is computationally expensive, as it requires running simulation software to evaluate each design. ML approaches, such as inverse design and surrogate-based optimization, have emerged as popular alternatives. These methods leverage datasets $\mathcal{D} = \{(x_i, a(x_i), c_i, \tilde{f}(x_i, c_i))\}_{i=1}^N$ of existing designs, their corresponding properties, and performance outcomes to reduce the computational burden of traditional optimization techniques.

Inverse design aims to learn a (generative) model $m : \mathcal{A} \times \mathcal{C} \mapsto \mathcal{X}$ that can predict the optimal design variables x^* given desired attributes a and conditions c. The output of such a model is rarely optimal but is often used as an initial guess for the optimization process to speed up its convergence. The model can also be used as an operator in a classical search algorithm [60].

Surrogate-based optimization approximates the objective function $\tilde{f}(x, c)$ using a surrogate model $m : \mathcal{X} \times \mathcal{C} \mapsto \mathcal{F}$. This model is then used to guide the optimization process (*e.g.*, Bayesian optimization, GAs, or gradient-based methods when m is differentiable), replacing the computationally expensive simulations with faster approximations.

Performance metrics Depending on the context and application, engineers prioritize different aspects of a model's output and often rely on performance metrics that go beyond mean squared error. In some cases, they may even prefer models that produce less detailed but easier-to-optimize outputs (*e.g.*, blurry images) over sharper ones [25]. Below, we outline several metrics used in our experiments. For a more comprehensive overview, we refer the reader to Regenwetter et al. [61].

Similarity: A model should generate results similar to the dataset. In inverse models, this means matching the dataset distribution, while surrogate models aim for accurate predictions of the objective values. A common metric is *maximum mean discrepancy* (MMD, [23]), which measures the distance between two distributions. Given a dataset $\mathcal{D} = \{z_i\}_{i=1}^N$ and generated samples $\mathcal{D}_g = \{\hat{z}_j\}_{j=1}^M$, MMD is defined as $\text{MMD}^2(k, \mathcal{D}, \mathcal{D}_g) = \mathbb{E}[k(z, z')] + \mathbb{E}[k(\hat{z}, \hat{z}')] - 2\mathbb{E}[k(z, \hat{z})]$, where $k(\cdot, \cdot)$ is a

[4] For conciseness, we often omit σ and denote $\tilde{f}(x, c, \sigma)$ as $\tilde{f}(x, c)$.

https://doi.org/10.52202/085713-0003

kernel function (*e.g.*, Gaussian).[5] A lower MMD indicates better alignment between generated and real distributions.

Diversity: In inverse problems, to allow for exploration of the design space, models should generate a diverse set of designs. *Determinantal point processes* (DPP, [37]) quantify diversity by favoring well-spread-out samples. Given a similarity kernel $K \in \mathbb{R}^{M \times M}$ with elements $K_{ij} = k(\hat{x}_i, \hat{x}_j)$, diversity is measured as $\text{DPP}(\mathcal{D}_g) = \det(K)$. A higher determinant value indicates greater diversity.

Optimality: Engineers prefer models that generate high-performance designs. The *cumulative optimality gap* (COG) measures how far a generated design lags behind the best known objective value *throughout* the optimization process. Let $f^* = \max_{x \in \mathcal{X}} \tilde{f}(x, c)$ denote the optimal objective value under condition c, typically estimated using an adjoint solver. For a generated design $\hat{x} = x_0$ that is refined through T optimization steps yielding a sequence $x_0, \ldots, x_T$, the COG is defined as $\text{COG}(x_0) = \sum_{t=0}^{T} (\tilde{f}(x_t, c) - f^*)$. A lower COG implies the optimization path remained close to optimal throughout, and is thus preferred in practical settings.

Feasibility: Designs must be physically feasible and avoid constraint violations. The *ratio of violated constraints* (RVC) measures how many generated samples fail to satisfy constraints on the designs. It is defined as

$$\text{RVC}(\mathcal{D}_g) = \frac{1}{M} \sum_{k=1}^{M} \mathbb{I}\left(\exists i \in [1, q] \text{ s.t. } g_i(\hat{x}_k, a(\hat{x}_k), c) > 0 \text{ or } \exists j \in [1, r] \text{ s.t. } h_j(\hat{x}_k, a(\hat{x}_k), c) \neq 0\right),$$

where $\mathbb{I}(\cdot)$ is an indicator function. A lower RVC indicates that the model tends to generate designs that satisfy physical and problem-specific constraints more reliably. Another indicator used in this work is the *ratio of failed simulations* (RF), which measures the number of generated designs that threw errors when trying to simulate.

2.2 Related work

Datasets and Benchmarks: A wealth of well-known datasets have been used over the years to assess the performance of ML algorithms. Examples such as MNIST [39] and CIFAR-10 [36] have streamlined the work of ML practitioners, enabling fair comparisons and reducing publication workloads. However, most of these tasks primarily focus on maximizing the similarity between model outputs and dataset labels, and are often unconstrained. In contrast, our problems require running heavyweight simulations (*e.g.*, CFDs) to validate designs. Recently, a few general benchmarks have bridged new gaps in ML for complex physical systems. For example, PDEBench [66] and LagrangeBench [68] propose a wide range of new datasets and metrics for ML applications in Partial Differential Equation (PDE) flow problems. While some valuable concepts are presented that may be extended to other domains, these works are limited to the single-physics domain of fluid flows. In contrast, ENGIBENCH provides a much wider variety of engineering datasets encompassing both single- and multi-physics domains. A few novel works that *do* include significant multi-physics data are BubbleML [28], which benchmarks different neural operators and UNets for capturing phase change phenomena, and The Well [54], a large collection of numerical simulations for various spatiotemporal physical systems to train and evaluate surrogate ML models. Although the scale and variety of simulation data are impressive in both works, little emphasis is placed on generalized metrics or analysis across different problems, and no APIs or extensions to generative modeling are provided. In ML specifically for engineering design, most studies rely on custom datasets, often kept private, limiting reproducibility and benchmarking. Fortunately, this trend is changing, and recent efforts have made datasets publicly available for specific domains such as airfoils [10], beams [65], bicycles [59], car bodies [15], among many others. However, these datasets remain problem-specific, making cross-domain benchmarking difficult and slowing the study of the generalization of ML methods across engineering tasks.

Simulation software in engineering design: Unlike conventional ML tasks, engineering design requires evaluating solutions via physics-based simulations. Simulation software has been developed for decades to evaluate designs, with open-source examples such as MachAero for CFD in airplane design [72, 62, 43], Dolfin/FEniCS for FEAs [41, 42], and NgSpice [1] in circuit simulation, along

[5] Note that z would denote designs (x) for inverse design models and objective values ($\tilde{f}(x, c)$) for surrogate models.

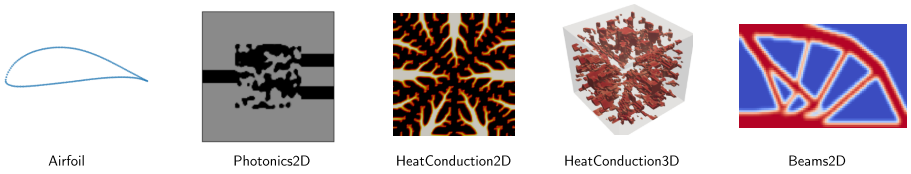


FIGURE 2: Visualization of some of the implemented problems.

with many closed-source/commercial solver packages. However, these tools often present significant barriers to use: they can be difficult to install and parameterize, frequently involve long setup times, may produce inconsistent results due to differences in simulator versions or simulation configurations, or, in the case of commercial tools, cannot be easily adopted or scaled without significant cost. Moreover, their heterogeneous interfaces and problem representations further complicate integration into ML or optimization workflows, making it difficult to apply the same codebase across different design problems to conduct cross-domain studies. Collectively, these challenges create a high barrier to entry for ML practitioners, slow down research progress, and hinder reproducibility in the field.

3 EngiBench

To address these challenges, we propose ENGIBENCH, an open-source library designed for data-driven engineering design across multiple domains. It provides an intuitive and extensible Python API that unifies the modeling of diverse design problems. Conceptually, each ENGIBENCH problem consists of a simulation engine and an associated dataset containing designs, objectives, attributes, and conditions, all accessible through a standardized interface; see Fig. 1.

ENGIBENCH abstracts away simulation complexities behind its user-friendly API. This allows ML and engineering researchers to focus on modeling and optimization rather than spending weeks configuring simulation environments. Additionally, ENGIBENCH provides a repository of ready-to-use problems and datasets generated from the supported simulators, see Fig. 2. Researchers can directly leverage these datasets for training and validating models without needing to generate their own, significantly accelerating experimentation and enhancing reproducibility in ML for engineering design.

3.1 Application Programming Interface

```python
from engibench.problems.beams2d.v0 import Beams2D
problem = Beams2D(seed=42) # Instantiate problem
# Inspect problem
problem.design_space  # Box(0.0, 1.0, (50, 100), float64)
problem.objectives  # (("compliance", "MINIMIZE"),)
problem.conditions  # (("volfrac", 0.35), ("forcedist", 0.0),...)
problem.dataset # A HuggingFace Dataset object
# inverse_model = train_inverse(problem.dataset)
desired_conds = {"volfrac": 0.7, "forcedist": 0.3}
# generated_design = inverse_model.predict(desired_conds)
random_design, _ = problem.random_design()
# check constraints on the design, config pair
violated_constraints = problem.check_constraints(random_design, desired_conds)
if not violated_constraints:
    # Only simulate to get objective values
    objs = problem.simulate(random_design, desired_conds)
    problem.reset(seed=41)
    # Or run a gradient-based optimizer to polish the design
    opt_design, history = problem.optimize(random_design, desired_conds)
    problem.render(opt_design)
```

LISTING 1: API example.

https://doi.org/10.52202/085713-0003

Listing 1 illustrates a high-level usage example of our API. Users can instantiate a problem (lines 1–2), extract relevant training data (lines 4–7), generate new designs (lines 8–11), check constraints for those designs (lines 13–14), validate or optimize these designs using a simulation engine (lines 15–19), and render a given design (line 20). A key feature of the API is its flexibility: switching to a different problem only requires modifying the first two lines of code, such as importing *HeatConduction2D* instead of *Beams2D*.

3.2 Features

ENGIBENCH introduces several features ensuring good reproducibility and ease of use.

Versioning: Similar to popular reinforcement learning libraries, such as Gymnasium and its variants [69, 17, 19], our problems' implementations and datasets are versioned (see line 1 in Listing 1), ensuring traceability when changes affect results.

Problem metadata: Problems expose attributes such as *design_space*, *objectives*, and *conditions*, containing information about the size, bounds, and shapes of the design components, the number of conditions and objectives. These allow automatic extraction of information like input and output dimensions for ML models, but also provide a generic way to extract relevant columns from the datasets (lines 4–7 in Listing 1).

Constraint checking: The library can verify constraint violations for a given design and conditions (line 13 in Listing 1). Constraints are categorized as theoretical (from the mathematical problem definition) or implementation-based (arising from simulation assumptions). Severity levels include errors (preventing simulation) and warnings (indicating possible inaccuracies). Although our implementations do not capture all possible kinds of errors, this allows users to pinpoint obvious issues with designs or configurations; see Appendix B.2 for more information regarding error handling.

Virtualized environment: Some simulation software requires a native installation to run. To simplify the deployment and execution of our library in such cases, we support integration with Docker, Podman, and Apptainer. This drastically reduces the setup time for users as they can just *pip install and run*.

Adjoint optimization: Most problems feature an adjoint optimizer (line 19 of Listing 1), enabling gradient-based optimization starting from generated designs. These state-of-the-art methods scale well to high-dimensional design spaces, though they may converge to local optima [45]. This is notably useful to compute optimality-related metrics, such as COG.

Integration with ML/optimization libraries: ENGIBENCH seamlessly integrates with tools like HuggingFace Datasets and Diffusers [70], PyTorch [56], BoTorch [3], and Pymoo [8], streamlining ML and optimization workflows. We provide several examples of ML and optimization algorithms in ENGIOPT.

Distributed computing: ENGIBENCH supports distributed execution, enabling large-scale dataset generation through parallel optimizations or simulations on high-performance computing clusters using Slurm [76]—*e.g.*, we generated our Photonics2D dataset in under 30 minutes using such a tool.

3.3 Implemented problems

As mentioned earlier, ENGIBENCH includes a collection of implemented problems conforming to its API, listed in Table 1 and partly illustrated in Fig. 2. These were selected to reflect the diversity of real-world engineering design scenarios. They vary in design representation—ranging from vectors (*e.g.*, electrical circuit parameters) to 2D images (*e.g.*, pixel-based topology optimization for beams) and 3D tensors (*e.g.*, voxel-based topology optimization for heat conduction)—as well as in physics domain, including aerodynamics, structural elasticity, thermal diffusion, and power electronics. The benchmark also covers both single- and multi-objective settings, and includes problems with a wide range of computational costs, from fast prototyping tasks to more demanding simulations. This range in simulation complexity is intentional: our goal is to provide a continuum of tasks that can serve both small labs with limited resources and large-scale teams who wish to address more complex, difficult, and realistic problems. Finally, several problems (and datasets) replicate setups from prior publications [2, 24, 25, 13], facilitating reproducibility and comparison with existing work. Each problem is briefly described below and detailed explanations can be found in Appendix C.

TABLE 1: Problems currently implemented in EngiBench. Simulation times have been averaged across 10 runs on a MacBook Pro M3 Max (on the CPU cores).

Problem	Design Repr.	Design Space	Opt. class	Physics	Simu. time (seconds)	Num. Objs.
Airfoil	Vector + Scalar Tuple	Cont.	Shape	Navier-Stokes	9.85	1
Heat-Conduction2D	Image	Disc.	Topology	Therm. diffusion	10.22	1
Heat-Conduction3D	3D Tensor	Disc.	Topology	Therm. diffusion	31.57	1
Thermo-Elastic-Beams2D	Image	Disc.	Topology	Therm. diffusion & Lin. elasticity	0.22	3
Beams2D	Image	Disc.	Topology	Lin. elasticity	0.19	1
Photonics2D	Image	Disc.	Topology	Maxwell's eqs.	0.3	1
PowerElectronics	Vector	Mixed	Tabular	Electr. dynamics	0.66	2

Airfoil: The airfoil problem involves aerodynamic shape optimization based on the Reynolds-averaged Navier–Stokes (RANS) equations. The design is parameterized by 192 control points defining the 2D airfoil geometry, along with a scalar representing the angle of attack. The objective is to match a prescribed lift coefficient while minimizing drag. We use MachAero [72, 62, 43] to perform the CFD simulations. A description of this problem and dataset is provided in Diniz and Fuge [13].

HeatConduction2D/HeatConduction3D: These problems are a specific subset of topology optimization (TO) problems aimed at placing highly conductive material to minimize thermal compliance within a unit square (2D) or unit cube (3D), subject to: a constraint on the volume of material used, and given conditions, particularly the location of the adiabatic region. Our code implementation is a modified version of the open-source Dolfin-adjoint example to solve the 2D and 3D heat conduction topology optimization problems [52, 41, 42].

ThermoElasticBeams2D: The ThermoElasticBeams2D problem specifies a multi-physics TO problem governed by linear elasticity and steady-state heat conduction. The goal is to find an optimal placement of material such that both structural compliance and thermal compliance are minimized while respecting a volume fraction constraint. Our implementation is based on the well-known 88-line MATLAB code for compliance minimization [2], adapted to Python and extended to the thermoelastic multi-physics case.

Beams2D: Beams2D is a structural TO problem that optimizes a beam under bending. The beam has a force applied at the top, which can be parameterized through conditions. The goal is to optimize the distribution of solid material within a rectangular grid to minimize compliance while satisfying constraints on material usage and minimum feature size. Our implementation is also based on the 88-line MATLAB code for compliance minimization [2], adapted to Python.

Photonics2D: Photonics2D is an electromagnetic waveguide optimization problem where the goal is to demultiplex an incoming port such that light under two different wavelengths exits the device at different locations [32, 57]. The goal is to maximize the electromagnetic field present at the desired exit location for the target wavelength. Our implementation is based on the examples from the `ceviche` finite-difference frequency-domain (FDFD) library [33].

https://doi.org/10.52202/085713-0003

PowerElectronics: This problem models a DC-DC power converter circuit with a fixed circuit topology. The circuit comprises 5 switches, 4 diodes, 3 inductors, and 6 capacitors. We vary circuit parameters, such as capacitance, then use the NgSpice simulator [1] to approximate the circuit and compute performance metrics including *DcGain* and *Voltage Ripple*. Despite the fixed topology, optimizing the circuit parameters to minimize the objectives remains a challenging task for surrogate models, as we show in Section 4.

3.4 Implemented algorithms in EngiOpt

We provide baselines that are well understood by ML practitioners—Generative Adversarial Networks (GANs) [22, 51, 58] and diffusion models [30, 70] for inverse design, and MLP+NSGA-II [12] for surrogate-assisted optimization. The implementations are single-file (CleanRL-style [31]), easy to debug, and avoid heavy simulator-specific dependencies or complex physics priors to maximize reproducibility and pedagogical value. They also provide a baseline framework that can easily extend to more complex algorithms in future versions of EngiOpt.

For harder cases (Sec. 4.2), we also include more specialized algorithms when warranted: (i) Bézier-GAN uses an airfoil-specific Bézier parameterization that enforces smoothness and reduces meshing failures [11]; (ii) our surrogate stack combines robust preprocessing, Bayesian hyperparameter search, implicit deep ensembles, and NSGA-II for multi-objective search. Notably, these models have matched simulator Pareto fronts and were experimentally validated in turbomachinery [46, 47, 48]. Even with these advanced models, several problems remain challenging (*e.g.*, PowerElectronics), underscoring the intrinsic difficulty of many engineering design tasks.

4 Proof-Of-Concept Experiments

In this section, we use EngiBench and EngiOpt to conduct a series of proof-of-concept empirical experiments and discuss the corresponding results. Our aim is to illustrate the types of experiments, comparisons, and analyses—across different algorithms, problem domains, and performance metrics—that our framework supports. Given the potentially high computational cost of training, optimization, and evaluation, we limit each algorithm-problem pair to 10 independent runs, reporting the mean and standard deviation for each metric. While more extensive experiments with reduced variance could be conducted, our focus here is not on providing a comprehensive benchmark of existing algorithms, but rather on demonstrating the practical capabilities and flexibility of our framework. A complete description of the conducted experiments, additional results, hyperparameters, discussions, and illustrations is available in Appendix D.

4.1 Cross-domain study

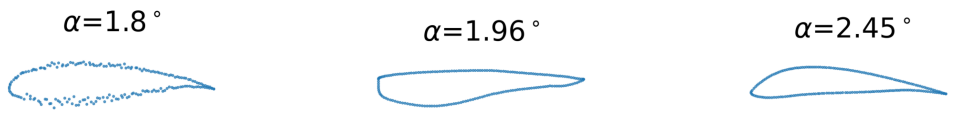
We showcase a new type of experiment that was previously difficult to conduct: a comparative study of different algorithms across multiple engineering design problems using a variety of performance metrics. With our tools, this evaluation became straightforward—different problems could be explored simply by changing the `problem_id` argument from the command line for each algorithm. We trained several generative models for inverse design over 100 epochs, including a deep convolutional generative adversarial network (GAN2D), a conditional deep convolutional GAN (CGAN2D), and a conditional diffusion model (CDiffusion2D).

Results are shown in Table 2. Interestingly, the unconditioned GAN model performs well overall in terms of COG and MMD, despite not producing well-defined shapes (see Figs. 12 to 14 in Appendix D). In terms of RVC—specifically, maintaining the volume fraction below the specified threshold for B2D and HC2D (P2D has no such constraint)—all algorithms perform rather poorly. The CGAN model achieves the best performance, whereas the CDiffusion model performs worst, which is somewhat unexpected. Regarding design diversity, the GAN model performs best on the first two problems, likely because it does not restrict itself to condition-specific regions. In contrast, the diffusion model excels on the photonics task, suggesting that the "best" generative model may depend on the nature of the design problem at hand. Finally, raw model outputs are often not directly usable in practice (*e.g.*, disconnected beam members, Fig. 12) and typically require post-processing or the injection of domain-specific priors, as illustrated in the next section—highlighting a common limitation of current generative approaches.

TABLE 2: Averaged metrics (mean ± std) per problem and model over 10 seeds. Gradient colors indicate best by linear interpolation between the min. and max. values across algorithms for a given metric on a given problem.

Each cell displays $\frac{\text{COG } (\downarrow) \quad \text{RVC } (\downarrow)}{\text{MMD } (\downarrow) \quad \text{DPP } (\uparrow)}$. B2D=Beams2D, HC2D=HeatConduction2D, P2D=Photonics2D.

	GAN2D		CGAN2D		CDiffusion2D	
B2D	1.56e+08 ± 6.92e+07	9.44e-01 ± 3.50e-02	1.51e+08 ± 1.74e+08	6.76e-01± 1.50e-01	2.45e+08 ± 1.09e+08	9.98e-01 ± 6.32e-03
	1.07e-01 ± 1.95e-02	8.08e-07 ± 2.56e-06	1.25e-01 ± 1.02e-01	3.62e-19 ± 1.14e-18	1.90e-01 ± 4.17e-02	3.50e-19 ± 7.87e-19
HC2D	2.56e-03 ± 2.77e-04	9.60e-01 ± 1.89e-02	1.38e-03 ± 3.15e-04	7.52e-01 ± 1.68e-01	6.44e-03 ± 3.03e-03	1.00e+00 ± 0.00e+00
	8.43e-02 ± 1.06e-02	7.73e-01 ± 1.30e-01	3.68e-01 ± 4.23e-02	2.65e-23 ± 7.57e-23	1.83e-01 ± 3.52e-02	3.70e-03 ± 6.58e-03
P2D	9.44e+02 ± 1.82e+01	N/A	9.74e+02 ± 7.72e+00	N/A	9.19e+02 ± 2.38e+01	N/A
	3.11e-01 ± 1.22e-01	5.11e-07 ± 1.52e-06	9.00e-01 ± 4.61e-02	6.24e-89 ± 1.97e-88	5.30e-02 ± 6.58e-03	8.23e-01 ± 1.46e-01



GAN Diffusion BézierGAN

FIGURE 4: Example outputs of our generative models on the Airfoil problem.

4.2 Hard cases: Airfoils and PowerElectronics

Surrogate models for PowerElectronics: We trained two robust-scaled MLP surrogates—one for *DcGain* and one for *Voltage Ripple*—using Bayesian hyperparameter search and implicit deep ensembles [74, 20]. We then ran NSGA-II using our surrogate models to find Pareto optimal designs. Despite careful tuning and the variance reduction afforded by the ensembles, all the surrogate-estimated Pareto fronts diverge sharply from NgSpice when re-evaluated (*e.g.*, see Fig. 3); across our 10 runs, all squared MMD tests rejected distributional equality.

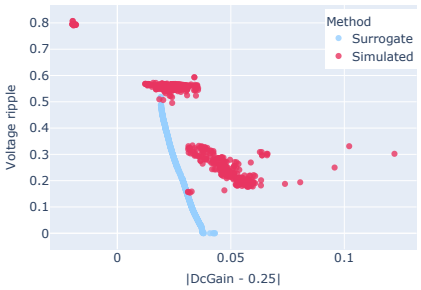


FIGURE 3: Difference between the Pareto fronts based on the surrogate models and the simulator.

We attribute the failure largely to the stiff, outlier-prone *Voltage Ripple* response (see Appendix D for more details), which remains difficult to approximate even after log transformation. In contrast, *DcGain* exhibits smoother behavior. These results underscore the difficulty of substituting physical simulators with black-box surrogates in systems with mixed time scales and sparse, high-variance regimes. Richer experimental design and physics guided feature transformations may be essential for reliable optimization.

Generative models for Airfoil: We trained 3 generative models to perform inverse design for airfoils: a standard GAN, a diffusion model, and a BézierGAN, which is specifically tailored to airfoil parameterization. Each model was trained for 2500 epochs. For every algorithm and random seed, we generated 50 candidate designs and attempted to simulate them, reporting the resulting ratio of failed simulations (RF). The GAN yielded an RF of 0.304 ± 0.167, the Diffusion model 0.034 ± 0.038, and the BézierGAN 0.014 ± 0.027. Notably, the Diffusion models exhibited mode collapse, generating highly similar designs across samples, which impacts RF (see Appendix D). These results support the intuition that domain-informed generative models improve performance. Simulation failures were

https://doi.org/10.52202/085713-0003

typically caused by issues during meshing. Such failures were often due to violations of geometric continuity in the point-based spline representation (see Fig. 4).

5 Use Cases, Limitations, and Future Work of ENGIBENCH

While our experiments highlight only a few possibilities, ENGIBENCH supports a broader range of research workflows. For the ML community, these contributions can serve as new testbeds for assessing ML model performance on problems that differ from traditional image- or text-based datasets since our problems come from engineering applications, are backed by real-world physics, and are constrained. For the engineering design community, it enables cross-domain algorithm evaluation via a simple `problem_id` switch. Conversely, engineers can contribute new problems conforming to the API and immediately benefit from existing algorithm implementations in ENGIOPT. Several datasets include not only input–output pairs but also full field data (*e.g.*, flow fields), making them suitable for PINNs and neural operators. The framework also supports multi-resolution or multi-fidelity studies (*e.g.*, we provide multiple Beams2D datasets with different resolutions), enabling training on cheap data and generalization to higher-fidelity scenarios. Additionally, it facilitates transfer learning across problems and the development of a "foundation design model." Finally, the API can support latent-space optimization via autoencoders and integration with reinforcement learning to guide optimization or data generation.

Despite its breadth, ENGIBENCH does not yet support unstructured meshes or grammar-based representations. Additionally, our benchmarks focus on static simulations, whereas real-world components are often dynamic—morphing or changing position during operation. Moreover, it may be beneficial to make problem configurations more flexible, such as varying the number of heat sources or sinks in the heat conduction problem. We have begun addressing this in the Beams2D problem by providing datasets at multiple image resolutions. Currently, ENGIBENCH does not include multi-part or assembly-level design tasks, such as the joint optimization of interdependent components within a mechanical system. Finally, simulator assumptions can introduce biases that propagate to learned models, and quantifying these biases without physical experiments remains challenging—a common limitation in simulation-based ML. However, ENGIBENCH's diversity of problems and simulators helps reveal such issues: models exploiting simulator-specific artifacts tend to regress to the mean in cross-task evaluations, exposing overfitting. Creating multiple versions of problems with different simulators and fidelities could further enable systematic bias assessment. Addressing these gaps will guide future developments. We do not anticipate any negative societal impacts of this work.

6 Conclusion

We introduced ENGIBENCH and ENGIOPT, two modular, open-source libraries for reproducible research in engineering design. ENGIBENCH provides diverse physics-driven benchmarks and datasets under a unified interface, while ENGIOPT offers compatible implementations of ML algorithms including GANs, diffusion models, and surrogate-based optimization. Through experiments, we demonstrate how these tools enable rigorous comparisons of algorithms across domains for different performance metrics and introduce new challenges for ML models when applied to constrained, highly-sensitive, real-world design problems.

Acknowledgments and disclosure of funding

We would like to thank the open-source community, notably the developers of NumPy [27], PyTorch [56], HuggingFace Datasets and Diffusers [40, 70], Weights and Biases [7], BoTorch [3], Pymoo [8], Gymnasium [69], MachAero [72, 62, 43], Dolfin/FEniCS [52, 41, 42], Slurm [76], Docker [50], Singularity/Apptainer [38]. We also acknowledge useful discussions with both Dr. Jun Wang and Dr. Quiyi Chen who helped refine core concepts that led to MIDBench, an earlier precursor of ENGIBENCH. The individual problem implementations in Table 1 as well as baseline models in ENGIOPT were developed over 5-6 years at UMD and were funded in part by the following grants: DARPA-16-63-YFA-FP-059, NSF CAREER 1943699, ARPA-E DE-AR0001216, ARPA-E DE-AR0001200, & ARL CA W911NF2320040.

References

[1] Ngspice: A mixed-level/mixed-signal circuit simulator. http://ngspice.sourceforge.net, 2025. Accessed: 2025-03-31. 4, 8, 32

[2] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S. Lazarov, and Ole Sigmund. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, January 2011. ISSN 1615-1488. doi: 10.1007/s00158-010-0594-7. URL https://doi.org/10.1007/s00158-010-0594-7. 6, 7, 27, 28, 29

[3] Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, 2020. 6, 10, 42

[4] Mohammad Mahdi Behzadi and Horea T Ilieş. Gantl: Toward practical and real-time topology optimization with conditional generative adversarial networks and transfer learning. *Journal of Mechanical Design*, 144 (2):021711, 2022. 2

[5] Martin P Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4): 193–202, 1989. 28

[6] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013. 24

[7] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com. 10

[8] J. Blank and K. Deb. pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020. 6, 10, 42

[9] Qiuyi Chen, Jun Wang, Phillip Pope, Wei Chen, and Mark Fuge. Inverse design of two-dimensional airfoils using conditional generative models and surrogate log-likelihoods. *Journal of Mechanical Design*, 144(2): 021712, 2022. 2

[10] Wei Chen, Kevin Chiu, and Mark Fuge. *Aerodynamic Design Optimization and Shape Exploration using Generative Adversarial Networks*. 2019. doi: 10.2514/6.2019-2351. URL https://arc.aiaa.org/doi/abs/10.2514/6.2019-2351. 4, 19

[11] Wei Chen, Kevin Chiu, and Mark D. Fuge. Airfoil Design Parameterization and Optimization Using Bézier Generative Adversarial Networks. *AIAA Journal*, 58(11):4723–4735, 2020. ISSN 0001-1452. doi: 10.2514/1.J059317. URL https://doi.org/10.2514/1.J059317. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J059317. 8, 39

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002. ISSN 1941-0026. doi: 10.1109/4235.996017. 8, 42

[13] Cashen Diniz and Mark Fuge. Optimizing diffusion to diffuse optimal designs. In *AIAA SCITECH 2024 Forum*. American Institute of Aeronautics and Astronautics, 2024. doi: 10.2514/6.2024-2013. URL https://arc.aiaa.org/doi/abs/10.2514/6.2024-2013. 6, 7, 19, 22

[14] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in Reinforcement Learning and How To Tune Them. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, June 2023. URL https://openreview.net/forum?id=0Vm8Ghcxmp. 42

[15] Mohamed Elrefaie, Florin Morar, Angela Dai, and Faez Ahmed. Drivaernet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks. In *Advances in Neural Information Processing Systems*, volume 37, pages 499–536, 2024. 2, 4

[16] Ahmad Fawaz, Yuchao Hua, Steven Le Corre, Yilin Fan, and Lingai Luo. Topology optimization of heat exchangers: A review. *Energy*, 252:124053, 2022. 23

[17] Florian Felten, Lucas Nunes Alegre, Ann Nowe, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno Castro da Silva. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023. 6

[18] Florian Felten, Daniel Gareev, El-Ghazali Talbi, and Grégoire Danoy. Hyperparameter Optimization for Multi-Objective Reinforcement Learning, October 2023. URL http://arxiv.org/abs/2310.16487. arXiv:2310.16487 [cs]. 42

[19] Florian Felten, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M. Roijers, Jordan K. Terry, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, and Roxana Rădulescu. MOMAland: A Set of Benchmarks for Multi-Objective Multi-Agent Reinforcement Learning, July 2024. URL http://arxiv.org/abs/2407.16312. arXiv:2407.16312 [cs]. 6

[20] M. A. Ganaie, Minghui Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, October 2022. ISSN 0952-1976. doi: 10.1016/j.engappai.2022.105151. 9, 42

[21] Oliver Giraldo-Londoño, Lucia Mirabella, Livio Dalloro, and Glaucio H Paulino. Multi-material thermo-mechanical topology optimization with applications to additive manufacturing: Design of main composite part and its support structure. *Computer Methods in Applied Mechanics and Engineering*, 363:112812, 2020. 25

[22] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. URL `https://proceedings.neurips.cc/paper_files/paper/2014/hash/f033ed80deb0234979a61f95710dbe25-Abstract.html`. 8

[23] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012. 3, 42

[24] Milad Habibi, Jun Wang, and Mark Fuge. When is it actually worth learning inverse design? In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 87301, page V03AT03A025. American Society of Mechanical Engineers, 2023. 6, 24

[25] Milad Habibi, Shai Bernard, Jun Wang, and Mark Fuge. Mean squared error may lead you astray when optimizing your inverse design methods. *Journal of Mechanical Design*, 147(2):021701, 2025. 3, 6, 24

[26] Hannah M. Hajdik, Anil Yildirim, Neil Wu, Benjamin J. Brelje, Sabet Seraj, Marco Mangano, Joshua L. Anibal, Eirikur Jonsson, Eytan J. Adler, Charles A. Mader, Gaetan K. W. Kenway, and Joaquim R. R. A. Martins. pyGeo: A geometry package for multidisciplinary design optimization. *Journal of Open Source Software*, 8(87):5319, 2023. doi: 10.21105/joss.05319. 21

[27] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825): 357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`. Publisher: Springer Science and Business Media LLC. 10

[28] Sheikh Md Shakeel Hassan, Arthur Feeney, Akash Dhruv, Jihoon Kim, Youngjoon Suh, Jaiyoung Ryu, Yoonjin Won, and Aparna Chandramowlishwaran. Bubbleml: A multi-physics dataset and benchmarks for machine learning. *arXiv preprint arXiv:2307.14623*, 2023. 4

[29] Xiaolong He, Jichao Li, Charles A. Mader, Anil Yildirim, and Joaquim R. R. A. Martins. Robust aerodynamic shape optimization—from a circle to an airfoil. *Aerospace Science and Technology*, 87:48–61, April 2019. doi: 10.1016/j.ast.2019.01.051. 19

[30] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html`. 8

[31] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. ISSN 1533-7928. URL `http://jmlr.org/papers/v23/21-1342.html`. 2, 8

[32] Tyler W Hughes, Momchil Minkov, Ian AD Williamson, and Shanhui Fan. Adjoint method and inverse design for nonlinear nanophotonic devices. *ACS Photonics*, 5(12):4781–4787, 2018. 7, 30, 31

[33] Tyler W Hughes, Ian AD Williamson, Momchil Minkov, and Shanhui Fan. Forward-mode differentiation of maxwell's equations. *ACS Photonics*, 6(11):3010–3016, 2019. 7, 31

[34] Gaetan K. W. Kenway, Charles A. Mader, Ping He, and Joaquim R. R. A. Martins. Effective adjoint approaches for computational fluid dynamics. *Progress in Aerospace Sciences*, 110:100542, October 2019. doi: 10.1016/j.paerosci.2019.05.002. 21

[35] Shahroz Khan, Kosa Goucher-Lambert, Konstantinos Kostas, and Panagiotis Kaklis. Shiphullgan: A generic parametric modeller for ship hull design using deep convolutional generative model. *Computer Methods in Applied Mechanics and Engineering*, 411:116051, 2023. 2

[36] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 4

[37] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012. ISSN 1935-8237. doi: 10.1561/2200000044. URL `http://dx.doi.org/10.1561/2200000044`. 4

[38] Gregory M. Kurtzer, cclerget, Michael Bauer, Ian Kaneshiro, David Trudgian, and David Godlove. hpcng/singularity: Singularity 3.7.3, April 2021. URL https://doi.org/10.5281/zenodo.4667718. 10

[39] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 4

[40] Quentin Lhoest, Albert Villanova del Moral, Patrick von Platen, Thomas Wolf, Mario Šaško, Yacine Jernite, Abhishek Thakur, Lewis Tunstall, Suraj Patil, Mariama Drame, Julien Chaumond, Julien Plu, Joe Davison, Simon Brandeis, Victor Sanh, Teven Le Scao, Kevin Canwen Xu, Nicolas Patry, Steven Liu, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Nathan Raw, Sylvain Lesage, Anton Lozhkov, Matthew Carrigan, Théo Matussière, Leandro von Werra, Lysandre Debut, Stas Bekman, and Clément Delangue. Datasets: A Community Library for Natural Language Processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184. Association for Computational Linguistics, November 2021. URL https://aclanthology.org/2021.emnlp-demo.21. 10

[41] Anders Logg and Garth N. Wells. Dolfin: Automated finite element computing. *ACM Trans. Math. Softw.*, 37(2), April 2010. ISSN 0098-3500. doi: 10.1145/1731022.1731030. URL https://doi.org/10.1145/1731022.1731030. 4, 7, 10

[42] Anders Logg, Kent-Andre Mardal, and Garth Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-23098-1 978-3-642-23099-8. doi: 10.1007/978-3-642-23099-8. URL https://link.springer.com/10.1007/978-3-642-23099-8. 4, 7, 10

[43] Charles A. Mader, Gaetan K. W. Kenway, Anil Yildirim, and Joaquim R. R. A. Martins. ADflow—an open-source computational fluid dynamics solver for aerodynamic and multidisciplinary optimization. *Journal of Aerospace Information Systems*, 2020. doi: 10.2514/1.I010796. 4, 7, 10, 21

[44] Joaquim Martins. Aerodynamic design optimization: Challenges and perspectives. *Computers & Fluids*, 239:105391, 03 2022. doi: 10.1016/j.compfluid.2022.105391. 19

[45] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, Cambridge, UK, January 2022. ISBN 9781108833417. doi: 10.1017/9781108980647. URL https://mdobook.github.io. 6, 19

[46] Soheyl Massoudi and Jürg Schiffmann. Dimensionless group-driven ensemble neural networks for robust design optimization in engineering. *Journal of Computational Design and Engineering*, 12(7):61–95, July 2025. ISSN 2288-5048. doi: 10.1093/jcde/qwaf056. 8

[47] Soheyl Massoudi, Cyril Picard, and Jürg Schiffmann. An Integrated Approach to Designing Robust Gas-Bearing Supported Turbocompressors Through Surrogate Modeling and Constrained All-At-Once Multi-Objective Optimization. *Journal of Mechanical Design*, 146(121706), July 2024. ISSN 1050-0472. doi: 10.1115/1.4065823. 8

[48] Soheyl Massoudi, Cameron Bush, and Jürg Schiffmann. Robust design optimization of gas-lubricated herringbone grooved journal bearings: Surrogate modeling and experimental validation. *Tribology International*, 204:110429, April 2025. ISSN 0301-679X. doi: 10.1016/j.triboint.2024.110429. 8

[49] Liang Meng, Weihong Zhang, Dongliang Quan, Guanghui Shi, Lei Tang, Yuliang Hou, Piotr Breitkopf, Jihong Zhu, and Tong Gao. From topology optimization design to additive manufacturing: Today's success and tomorrow's roadmap. *Archives of Computational Methods in Engineering*, 27(3):805–830, 2020. 23

[50] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014. ISSN 1075-3583. 10

[51] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, November 2014. URL http://arxiv.org/abs/1411.1784. arXiv:1411.1784 [cs]. 8

[52] Sebastian Mitusch, Simon Funke, and Jørgen Dokken. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4(38):1292, 2019. 7, 10, 24

[53] Xiaobao Mo, Hui Zhi, Yizhi Xiao, Haiyu Hua, and Liang He. Topology optimization of cooling plates for battery thermal management. *International Journal of Heat and Mass Transfer*, 178:121612, 2021. 23

[54] Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina Agocs, Miguel Beneitez, Marsha Berger, Blakesly Burkhart, Stuart Dalziel, Drummond Fielding, et al. The well: a large-scale collection of diverse physics simulations for machine learning. *Advances in Neural Information Processing Systems*, 37: 44989–45037, 2024. 4

[55] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *Journal of Artificial*

Intelligence Research, 74, September 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13596. URL https://dl.acm.org/doi/10.1613/jair.1.13596. 42

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. 6, 10

[57] Alexander Y Piggott, Jesse Lu, Konstantinos G Lagoudakis, Jan Petykiewicz, Thomas M Babinec, and Jelena Vučković. Inverse design and demonstration of a compact and broadband on-chip wavelength demultiplexer. *Nature photonics*, 9(6):374–377, 2015. 7, 30, 31

[58] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06434. 8

[59] Lyle Regenwetter, Brent Curry, and Faez Ahmed. BIKED: A Dataset for Computational Bicycle Design With Machine Learning Benchmarks. *Journal of Mechanical Design*, 144(031706), October 2021. ISSN 1050-0472. doi: 10.1115/1.4052585. URL https://doi.org/10.1115/1.4052585. 4

[60] Lyle Regenwetter, Amin Heyrani Nobari, and Faez Ahmed. Deep generative models in engineering design: A review. *Journal of Mechanical Design*, 144(7), 2022. 2, 3

[61] Lyle Regenwetter, Akash Srivastava, Dan Gutfreund, and Faez Ahmed. Beyond Statistical Similarity: Rethinking Metrics for Deep Generative Models in Engineering Design. *Computer-Aided Design*, 165: 103609, December 2023. ISSN 0010-4485. doi: 10.1016/j.cad.2023.103609. URL https://www.sciencedirect.com/science/article/pii/S0010448523001410. 3

[62] Ney Secco, Gaetan K. W. Kenway, Ping He, Charles A. Mader, and Joaquim R. R. A. Martins. Efficient mesh generation and deformation for aerodynamic shape optimization. *AIAA Journal*, 2021. doi: 10.2514/1.J059491. 4, 7, 10, 21

[63] Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4):401–424, 2007. 26, 28

[64] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 42

[65] Ivan Sosnovik and Ivan Oseledets. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 34(4):215–223, August 2019. ISSN 1569-3988. doi: 10.1515/rnam-2019-0018. URL https://www.degruyter.com/document/doi/10.1515/rnam-2019-0018/html. Publisher: De Gruyter. 4

[66] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022. 4

[67] Lei Tang, Tong Gao, Longlong Song, Liang Meng, Chengqi Zhang, and Weihong Zhang. Topology optimization of nonlinear heat conduction problems involving large temperature gradient. *Computer Methods in Applied Mechanics and Engineering*, 357:112600, 2019. 23

[68] Artur Toshev, Gianluca Galletti, Fabian Fritz, Stefan Adami, and Nikolaus Adams. Lagrangebench: A lagrangian fluid mechanics benchmarking suite. *Advances in Neural Information Processing Systems*, 36: 64857–64884, 2023. 4

[69] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments, November 2024. URL http://arxiv.org/abs/2407.17032. arXiv:2407.17032 [cs]. 6, 10

[70] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, Steven Liu, William Berman, Yiyi Xu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. URL https://github.com/huggingface/diffusers. 6, 8, 10

[71] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006. 24

[72] Ella Wu, Gaetan Kenway, Charles A. Mader, John Jasa, and Joaquim R. R. A. Martins. pyoptsparse: A python framework for large-scale constrained nonlinear optimization of sparse systems. *Journal of Open Source Software*, 5(54):2564, 2020. doi: 10.21105/joss.02564. 4, 7, 10

[73] Neil Wu, Gaetan Kenway, Charles A. Mader, John Jasa, and Joaquim R. R. A. Martins. pyoptsparse: A python framework for large-scale constrained nonlinear optimization of sparse systems. *Journal of Open Source Software*, 5(54):2564, 2020. doi: 10.21105/joss.02564. 21

[74] Jie Xue, Zhuo Wang, Deting Kong, Yuan Wang, Xiyu Liu, Wen Fan, Songtao Yuan, Sijie Niu, and Dengwang Li. Deep ensemble neural-like P systems for segmentation of central serous chorioretinopathy lesion. *Information Fusion*, 65:84–94, January 2021. ISSN 1566-2535. doi: 10.1016/j.inffus.2020.08.016. 9, 42

[75] Anil Yildirim, Gaetan KW Kenway, Charles A Mader, and Joaquim RRA Martins. A jacobian-free approximate newton–krylov startup strategy for rans simulations. *Journal of Computational Physics*, 397: 108741, 2019. 21

[76] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, pages 44–60, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39727-4. doi: 10.1007/10968987_3. 6, 10, 17

 https://doi.org/10.52202/085713-0003

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [NA]
   (b) Did you include complete proofs of all theoretical results? [NA]
3. If you ran experiments (*e.g.*for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendices A and D.
   (b) Did you specify all the training details (*e.g.*, data splits, hyperparameters, how they were chosen)? [Yes] See Appendices A and D.
   (c) Did you report error bars (*e.g.*, with respect to the random seed after running experiments multiple times)? [Yes] We report mean and standard deviation over 10 seeds for all experiments.
   (d) Did you include the total amount of compute and the type of resources used (*e.g.*, type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.
4. If you are using existing assets (*e.g.*, code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Appendix A.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Appendix A.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [NA]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [NA]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [NA]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [NA]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [NA]

A Useful Information

A.1 Links

The project's parts can be found at the following links:

- Documentation website: `https://engibench.ethz.ch`.
- ENGIBENCH code: `https://github.com/IDEALLab/EngiBench`. The code used for this paper was tagged with `v0.0.1`.
- ENGIOPT code: `https://github.com/IDEALLab/EngiOpt`. The code used for this paper was tagged with `v0.0.1`.
- Datasets: `https://huggingface.co/IDEALLab`.
- All hyperparameters, learning curves, Python version, OS version, hardware specifications, and run commands for all our experiments are available at `https://wandb.ai/engibench/engiopt`.

A.2 Licenses

Both our codebases are released under the GPL-3.0 license. All the datasets are released under the CC-BY-NC-SA license.

A.3 Maintenance And External Contributions

The IDEAL Lab and ETHZ's Scientific IT Services (SIS) are committed to the long-term maintenance of the project.

We also hope the open-source community will contribute to its ongoing development and improvement. To support this, we provide detailed instructions for making contributions to ENGIBENCH on our website, *e.g.*, `https://engibench.ethz.ch/tutorials/new_problem/`.

A.4 Experimental setup

Our experiments were conducted on the Euler cluster from ETH Zurich's high-performance computer. The compute nodes equipped with GPUs contain NVidia GeForce RTX4090 (24GB) and AMD EPYC 9554 CPUs. We used Python 3.11.6, CUDA 12.5, and PyTorch 2.6. Each training job has been allocated one GPU and 4 cores using Slurm [76]. All datasets have been generated on CPU nodes of the UMD's high-performance computing cluster Zaratan equipped with 128 AMD EPYC 7763 CPUs, except for photonics, which was generated on Euler.

Training all algorithms across all problems with multiple random seeds for our experiments required approximately 80 GPU-hours. Evaluations of the generated or optimized designs for our experiments took 55 hours. Generating the datasets, however, was significantly more time-consuming and took place over the past few years, as we reused existing simulation data. We estimate that generating the full set of datasets required several thousand CPU-hours in total. A per-problem estimation of the dataset generation time is given below.

- Airfoil: ≈ 5000 hours.
- HeatConduction2D: ≈ 1 hour.
- HeatConduction3D: ≈ 300 hours.
- ThermoElasticBeams2D: ≈ 10 hours.
- Beams2D: ≈ 12 hours (4 hours per dataset).
- Photonics2D: ≈ 200 hours.
- PowerElectronics: ≈ 20 hours.

B Features

In this section, we go through some of the features and design choices we have made for the library.

https://doi.org/10.52202/085713-0003

B.1 Example usage

```python
from engibench.problems.beams2d.v0 import Beams2D

# Create a problem
problem = Beams2D(seed=42)

# Inspect problem
problem.design_space  # Box(0.0, 1.0, (50, 100), float64)
problem.objectives  # (("compliance", "MINIMIZE"),)
problem.conditions  # (("volfrac", 0.35), ("forcedist", 0.0),...)
problem.dataset # A HuggingFace Dataset object

# Train your inverse design model or surrogate model
conditions = problem.dataset["train"].select_columns(problem.conditions_keys)
designs = problem.dataset["train"].select_columns("optimal_design")
cond_designs_keys = problem.conditions_keys + ["optimal_design"]
cond_designs = problem.dataset["train"].select_columns(cond_designs_keys)
objs = problem.dataset["train"].select_columns(problem.objectives_keys)

# Train your models
inverse_model = train_inverse(inputs=conditions, outputs=designs)
surr_model = train_surrogate(inputs=cond_designs, outputs=objs)

# Use the model predictions, inverse design here
desired_conds = {"volfrac": 0.7, "forcedist": 0.3}
generated_design = inverse_model.predict(desired_conds)

violated_constraints = problem.check_constraints(generated_design,
↪  desired_conds)
if not violated_constraints:
    # Only simulate to get objective values
    objs = problem.simulate(design=generated_design, config=desired_conds)
    problem.reset(seed=41)
    # Or run a gradient-based optimizer to polish the generated design
    opt_design, history = problem.optimize(generated_design, desired_conds)
```

LISTING 2: API usage with automated column extraction for training.

Listing 2 presents a longer version of our API usage, showing how to automatically extract relevant columns from the datasets (lines 13—17) to perform ID or surrogate-based optimization (lines 20—25).

B.2 Error handling

ENGIBENCH handles errors at multiple levels while preserving flexibility for advanced users. The framework includes several mechanisms:

- **Constraint checking:** Each benchmark problem includes standardized validation for configuration and constraints (e.g., Listing 1, line 14). For instance, in the Beams2D task, setting `volfrac = 2.0` and checking for constraints returns:
  - `Config.volfrac: 2.0` $\notin$ `[0.0, 1.0] (Theory, error)`
  - `Config.volfrac: 2.0` $\notin$ `[0.1, 0.9] (Implementation, warning)`

  These pre-simulation checks catch common issues early.
- **User control:** While errors are flagged, users can still choose to simulate invalid configurations—for instance, to explore failure regions or generate negative data.
- **Failure tracking:** If a solver crashes (*e.g.*, due to meshing errors or NaNs), the exception is caught and surfaced, even in containerized runs.
- **Failure rates and reporting:** Simulation failure rates vary across tasks and methods. We often track these to compare robustness—for example, the failure ratio for generative airfoil models is reported in Section 4.2.

C Extensive Description of Problems

This section describes the implemented problems in more details.

C.1 Airfoil

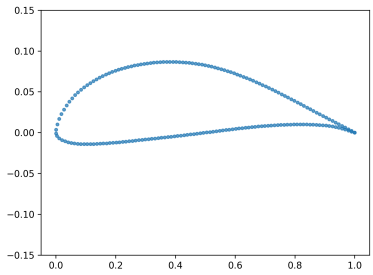


FIGURE 5: An Airfoil design, plotted via `problem.render`.

Motivation

The field of aerodynamics has always been a challenging testbed for engineering problems. In fact, many optimization and design methodologies were originally developed or perfected specifically for aerodynamic design applications [45]. Part of the reason for this is that even relatively simple aerodynamics problems can be complex, with slight changes in design parameters typically resulting in large changes in performance. In addition, the potential applications derived from solving these problems are quite practical, ranging from fixed-wing aircraft to hydrofoils used in ships, and wind turbine blades [44]. Here, we present Airfoil, a simple yet sufficiently realistic 2-dimensional aerodynamics benchmark problem.

The airfoil problem presents a simple aerodynamic shape optimization routine based on Reynolds' averaged Navier-Stokes equations (RANS). In this problem, the solver attempts to indirectly morph the initial geometry to achieve a certain prescribed lift coefficient (which could correspond to a hypothetical loading requirement) while minimizing the amount of drag generated by the design.

Design Space

The design space is represented as a tuple containing 192 2D points describing the airfoil coordinates and a scalar describing the rotation of the coordinates relative to the chord line needed to achieve a certain incoming direction of flow (the angle of attack, α): $\mathcal{X} = \left\{(x, y)^{192}, \alpha\right\}$. This specific coordinate parameterization (192) and rotational scalar design parameterization have been previously used in [10]. Another benchmark airfoil optimization problem, mentioned in [29],[6] was also used to internally validate the meshing and design parameterization. All training data, as well as the original and complete formulation of this problem can be found in [13].

Objectives

The objective is to minimize the coefficient of drag, c_d, and the optimization problem is defined as follows:

[6]Transonic RAE2822

https://doi.org/10.52202/085713-0003

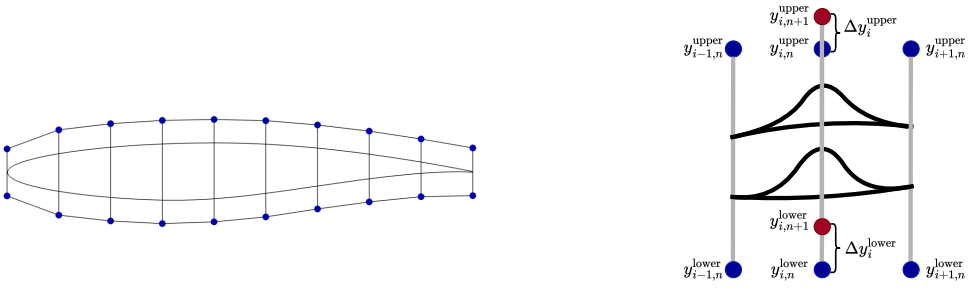
$$
\begin{aligned}
\min_{\Delta y_i, \alpha} \quad & c_d \\
\text{s.t.} \quad & c_l = c_l^{con} \\
& -0.025 \leq \Delta y_i \leq 0.025 \\
& 0.0 \leq \alpha \leq 10.0 \\
& \left(\frac{A}{A_{init}}\right)_{min} \leq \frac{A}{A_{init}} \leq 1.2
\end{aligned}
$$

The terms used in the definition of the problem are described in Table 3. Note that some variables are defined relative to a required initial design input.

TABLE 3: Optimization Problem Parameters

Category	Parameter	Quantity	Lower	Upper	Units	Description
Objective	c_d	1	-	-	Non-Dim./Counts	Coefficient of drag
Variable	Δy_i	20	-0.025	0.025	m	Change from initial FFD cage y value: $\Delta y_i = y_i - y_{init}$
	α	1	0.0	10.0	Degrees	Angle of Attack
Constraint	$c_l = c_l^{con}$	1	0.0	0.0	Non-Dim.	Coefficient of lift
	$\frac{A}{A_{init}}$	1	$\left(\frac{A}{A_{init}}\right)_{min}$	1.20	Non-Dim.	Area fraction; relative to initial

Instead of directly parameterizing all (192) coordinates of the airfoil, a smaller set of 20 control points is used. The collection of these control points forms what is known as a free-form deformation cage (FFD). Changes in the values of the FFD cage smoothly deform the underlying coordinates. In the airfoil problem, modifications to the geometry are parameterized by changes in the FFD control point y coordinates, Δy_i. Figure 6b shows how these changes in i^{th} FFD y coordinate in the n^{th} optimization iteration results in a smooth morphing of the underlying coordinates in the next $n + 1^{\text{th}}$ iteration (in red). Figure 6a shows a sample FFD cage for an airfoil from the dataset, with the 20 different FFD Δy_i design variables (in blue).



(A) Sample airfoil FFD cage

(B) FFD cage morphing process

FIGURE 6: Airfoil FFD design variables

Note that, for simplicity, in this definition we have omitted certain constraints pertaining to thickness as well as those concerned with the shearing of the leading (front) and trailing (tail end) edges.

Conditions

The conditions for the airfoil problem are described in Table 4.

TABLE 4: Airfoil Conditions

Category	Condition	Description
Flow Condition	M	Mach number
	Re	Reynold's number
Constraint	c_l^{con}	Coefficient of lift
	$\left(\frac{A}{A_{init}}\right)_{min}$	Minimum area fraction

Note that while it may be possible to constrain the design's area ahead of time, this is not typically possible for the prescribed coefficient of lift. In addition it may not be possible to achieve certain combinations of prescribed area and coefficient of lift constraints.

Constraints

Theoretical constraints (error)

$$\begin{aligned} \mathcal{X} &\in \left\{(x,y)^N, \alpha\right\} \\ \alpha &\in [0, 10] \\ \left(\frac{A}{A_{init}}\right)_{min} &\in [0, 1.2) \end{aligned}$$

Theoretical constraints (warning)

$$\left(\frac{A}{A_{init}}\right) \in \left[\left(\frac{A}{A_{init}}\right)_{min}, 1.2\right)$$

Implementation constraints (error)

$$\begin{aligned} M &\in (0, \infty) \\ Re &\in (0, \infty) \end{aligned}$$

Implementation constraints (warning)

$$\begin{aligned} M &\in [0.1, 1.0] \\ Re &\in [10^5, 10^9] \end{aligned}$$

Simulator

All simulations used the open source and differentiable ADflow solver [43, 34] as part of the MACH-Aero framework.[7] ADflow was configured to run RANS simulations with the Spalart-Allmaras model for turbulence effects. Furthermore, within ADflow, the approximate Newton-Krylov method was used to improve convergence and robustness [75]. pyHyp, a hyperbolic mesh generator [62] was used to generate volume meshes automatically. For the optimization problem itself, we use the sequential least squares programming algorithm as implemented in the sparse optimization framework, pyOptSparse [73]. For geometry parameterization and deformation, module, we used the pyGeo and IDWarp frameworks [26, 62].

[7] `https://github.com/mdolab/MACH-Aero`.

https://doi.org/10.52202/085713-0003

Dataset

A dataset, originally described in [13], is integrated within our framework. The limits for the parameters in the data set are listed in Table 5. 1400 parameter combinations were chosen using Latin hypercube sampling (LHS) in the 4-dimensional parameter space. The training, testing and validation sets were randomly split into 748, 140, and 47 airfoil samples, respectively. Finally, Table 6 describes each variable in the data set available through the ENGIBENCH API.

TABLE 5: Sampled Parameter Bounds

Category	Parameter	Lower	Upper	Description
Flow Condition	M	0.4	0.9	Mach number
	Re	1E6	10E6	Reynold's number
Constraint	C_l^{con}	0.5	1.2	Coefficient of lift
	$\left(\frac{A}{A_{init}}\right)_{min}$	0.75	1.0	Minimum area ratio

TABLE 6: Airfoil dataset variables

Category	Variable Name	Description
Flow Condition	mach	Mach number
	reynolds	Reynold's number
Constraint	cl_target	Coefficient of lift (targeted constraint during optimization)
	area_ratio_min	Minimum area ratio
Design Value	area_initial	Area of the initial design
	cd	Coefficient of drag for the current design
	cl	Coefficient of lift for the selected design
	area_ratio	Area ratio of the selected design

C.2 HeatConduction2D/HeatConduction3D

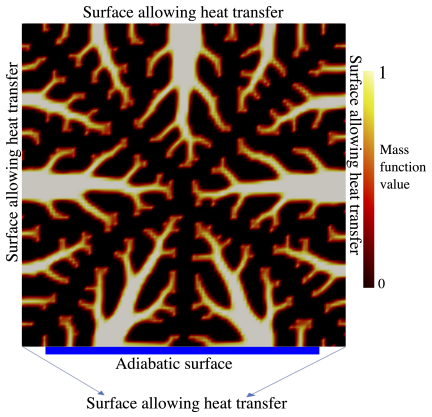


FIGURE 7: The dendrite-like topology emerging from optimizing heat conduction.

Motivation

Heat conduction problems serve as fundamental benchmarks for the development and evaluation of design optimization methods, with applications ranging from thermal management in electronic

devices to insulation systems and heat exchangers in industrial applications [53, 16]. As thermal management has become critical in fields such as aerospace, automotive, and consumer electronics, both industry and academia have shown growing interest in advanced thermal design systems [67]. In response to this demand, topology optimization has become popular as a powerful approach for improving heat dissipation while minimizing material usage. In addition, the development of additive manufacturing technologies has made the complex geometries produced by topology optimization more feasible to fabricate in real-world applications [49].

Design Space

These problems are a specific subset of topology optimization problems aimed at minimizing thermal compliance within a unit square (2D) or unit cube (3D), subject to: a constraint on the *volume* of highly conductive material used, and given boundary conditions, particularly the location of the adiabatic region. The adiabatic region refers to a symmetric *length* on the bottom side of the 2D problem space or a prescribed symmetric *area* on the bottom surface of the 3D problem space. The design space for the 2D problem consists of a 2D array representing solid densities, which is parametrized by resolution, that is, $\mathcal{X} = [0, 1]^{\text{resolution} \times \text{resolution}}$. By default, a 101×101 space is used for the 2D problem. The 3D design space is similarly represented as a 3D tensor of densities. By default, a $51 \times 51 \times 51$ space is used for the 3D problem.

Objectives

In this problem, we aim to minimize the thermal compliance (minimize C_T)

$$C_T = \int_\Omega hT + \alpha \int_\Omega \nabla x \cdot \nabla x,$$

subject to the Poisson equation with mixed Dirichlet–Neumann conditions:

$$\begin{aligned} \nabla \cdot (k(a)\nabla T) + h = 0 \quad & \text{in } \Omega, \\ T = 0 \quad & \text{on } \Gamma_D, \\ (k(a)\nabla T) \cdot n = 0 \quad & \text{on } \Gamma_N. \end{aligned}$$

The design variable $x \in$ [0,1] represents the spatial distribution of thermally conductive material, constrained by a total volume budget:

$$\int_\Omega x \leq V$$

wherein Ω denotes the design domain (unit square in 2D, unit cube in 3D), h is the heat source term, T is the temperature field, α a regularization parameter, $k(a)$ is the material conductivity interpolated from x. The boundaries Γ_D and Γ_N correspond to fixed-temperature (Dirichlet) and insulated (Neumann) regions, respectively.

Conditions

The conditions ($\mathcal{C}$) include two key factors: the specified volume fraction of highly conductive material (*volume*), and the location of the adiabatic region. The adiabatic region refers to a specified length on the bottom side of the 2D problem space or a prescribed symmetric area on the bottom surface of the 3D problem space.

Constraints

This section outlines the constraints relevant to the 2D and 3D heat conduction problems.

https://doi.org/10.52202/085713-0003

Theoretical constraints (error)

$$\begin{aligned} \mathcal{X}^{2D} &\in [0,1]^{\text{resolution}\times\text{resolution}} \\ \mathcal{X}^{3D} &\in [0,1]^{\text{resolution}\times\text{resolution}\times\text{resolution}} \\ resolution &\in [1,\infty) \\ volume &\in [0,1] \\ length &\in [0,1] \\ area &\in [0,1] \end{aligned}$$

Implementation constraints (warning)

$$\begin{aligned} resolution &\in [10,1000] \\ volume &\in [0.3,0.6] \end{aligned}$$

Simulator

We employ a modified version of the open-source Dolfin-adjoint example to solve the 2D and 3D heat conduction topology optimization problems [52]. Rather than formulating the problem as an integer optimization task, we use a continuous relaxation approach, which is the standard technique in topology optimization [6]. This allows the material distribution to change smoothly between solid and void, enabling efficient gradient-based optimization. The solver relies on the interior-point method implemented in Ipopt [71], which handles the nonlinear constraints.

Dataset

The datasets for the 2D and 3D heat conduction problems have been originally introduced in Habibi et al. [24, 25]. Each dataset entry contains the optimal design solution (`optimal_design`) along with its corresponding set of input conditions. To generate our 2D/3D dataset, we sampled two input conditions: volume and the adiabatic region's length (2D) or area (3D). We divided each parameter range into 20 equal intervals, yielding 21 values per parameter (*e.g.*, area: 0.0, 0.05, ..., 1.0). By combining all pairs, we generated 441 optimized topologies. For test-train splitting, we randomly excluded two unique values of the volume limit and two unique values of the adiabatic region from the training set. This exclusion resulted in 361 designs for training, with the remaining 80 designs evenly split into 40 for validation and 40 for testing. The parameter sampling bounds used to generate the datasets are summarized in Table 7.

TABLE 7: Sampled Parameter Bounds

Problem	Parameter	Lower	Upper
HeatConduction2D	volume	0.3	0.6
	length	0	1
HeatConduction3D	volume	0.3	0.6
	area	0	1.0

C.3 ThermoElasticBeams2D



FIGURE 8: Dendrite-like topology emerging when optimizing a thermo-elastic beam for both structural stiffness and thermal diffusion.

Motivation

As articulated in their respective sections, both the Beams2D and HeatConduction2D problems are fundamental engineering design problems that have historically served as benchmarks for the development and testing of optimization methods. While their relevance is supported by needs in real engineering design scenarios (aerospace, automotive, consumer electronics, *etc.*), their mono-domain nature ignores the reality that coupling between domains exists, and should be accounted for in scenarios where performance in one domain significantly impacts performance in another [21]. To address this distinction, a multi-physics topology optimization problem is developed that captures the coupling between structural and thermal domains.

Design Space

This multi-physics topology optimization problem is governed by linear elasticity and steady-state heat conduction with a one-way coupling from the thermal domain to the elastic domain. The problem is defined over a square 2D domain, where load elements and support elements are placed along the boundary to define a unique elastic condition. Similarly, heatsink elements are placed along the boundary to define a unique thermal condition. The design space is then defined by a 2D array representing density values (parameterized by $\mathcal{X} = [0, 1]^{\text{nelx} \times \text{nely}}$, where *nelx* and *nely* denote the x and y dimensions).

Objectives

The objective of this problem is to minimize total compliance C under a volume fraction constraint V by placing a thermally conductive material. Total compliance is defined as the sum of thermal compliance (C_T) and structural compliance (C_S):

$$C = C_T + C_S$$

Above, thermal compliance is defined by:

$$C_T = \int_\Omega hT + \alpha \int_\Omega \nabla x \cdot \nabla x,$$

where the domain assumes uniform heat generation. The corresponding thermal boundary conditions are defined by mixed Dirichlet-Neumann conditions, namely:

https://doi.org/10.52202/085713-0003

$$\begin{aligned} \nabla \cdot (k(a)\nabla T) + h &= 0 \quad \text{in } \Omega, \\ T &= 0 \quad \text{on } \Gamma_D, \\ (k(a)\nabla T) \cdot n &= 0 \quad \text{on } \Gamma_N. \end{aligned}$$

where the terms are defined as follows: Ω represents the 2D domain, h denotes uniform heat generation, T is the temperature field, α is a regularization parameter, and $k(a)$ is material conductivity interpolated from design variable x. Finally, Γ_D and Γ_N represent the Dirichlet and Neumann boundary conditions, respectively.

As the problem has a one-way coupling from the thermal domain to the elastic domain, structural compliance is defined by two components: compliance due to external loads imposed on the system C_{S1}, and compliance due to the forces induced by element-wise thermal expansion C_{S2}. Both components are captured in the following calculation as long as element displacement vector $\mathbf{u}_e$ captures displacements due to both external loads and thermal expansion forces:

$$\underset{\mathbf{x}\in[0,1]^{\text{nelx}\times\text{nely}}}{\text{minimize}} \; C_S(\mathbf{x}) = \sum_{e=1}^{N} E_e(x_e)\, \mathbf{u}_e^{\mathrm{T}} \mathbf{k}_0 \mathbf{u}_e \quad \text{where} \quad \mathbf{KU} - \mathbf{F} = 0.$$

where N is the number of total elements ($nelx * nely$), $E_e(x_e)$ is the penalized Young's Modulus based on the Solid Isotropic Material with Penalization method (SIMP) formulation [63], $\mathbf{u}_e$ is the element displacement vector, and $\mathbf{k}_0$ is the reference stiffness matrix.

For the equation above, the displacement vector is calculated using the relation $\mathbf{KU} - \mathbf{F} = 0$, where F captures the forces due to both external loads and the thermal expansion of elements. The forces due to thermal expansion (f^{th}) are calculated as follows for each element:

$$f_e^{th} = x_p^e C_e(\mathbf{T_e} - T_{ref})$$

where T_e represents the nodal temperatures of the element, T_{ref} represents the stress-free reference temperature, x_p^e represents the SIMP density-penalisation factor, and C_e denotes the coupling matrix mapping the temperature delta to a normal force representing thermal expansion.

Finally, a volume fraction constraint is imposed to restrict the amount of material that can be used for a design. This constraint is defined by:

$$\int_\Omega x \leq V$$

where Ω denotes the design domain, and V is the volume fraction constraint value.

Conditions

The set of conditions ($\mathcal{C}$) defining a unique problem formulation includes: the volume fraction constraint value (V), the filter radius for the design variables $rmin$, and the corresponding locations of the fixed elements, loaded elements, and heatsink elements.

Constraints

The constraints for this problem are defined as follows:

Theoretical constraints (error)

$$\begin{aligned} \mathcal{X} &\in [0,1]^{\text{nelx}\times\text{nely}} \\ nelx &\in [1,\infty) \\ nely &\in [1,\infty) \\ V &\in [0,1] \\ rmin &\in (0,\infty) \end{aligned}$$

Theoretical constraints (warning)

$$\frac{\sum_{i=0}^{\text{nelx}} \sum_{j=0}^{\text{nely}} x_{i,j}}{\text{nelx} \times \text{nely}} \leq V$$

Implementation constraints (error)

$$rmin \in (0, min(nelx, nely))$$

Implementation constraints (warning)

$$\begin{aligned} nelx &\in [10, 1000] \\ nely &\in [10, 1000] \\ V &\in [0.1, 0.9] \\ rmin &\in [1.0, 10.0] \end{aligned}$$

Simulator

The simulation code is based on a Python adaptation[8] of the popular 88-line topology optimization code [2], modified to handle the thermal domain in addition to thermal-elastic coupling. Optimization is conducted by reformulating the integer optimization problem as a continuous one (leveraging a SIMP approach), where a density filtering approach is used to prevent checkerboard-like artifacts. The optimization process itself operates by calculating the sensitivities of the design variables with respect to total compliance (done efficiently using the Adjoint method), calculating the sensitivities of the design variables with respect to the constraint value, and then updating the design variables by solving a convex-linear subproblem and taking a small step (using the method of moving asymptotes). The optimization loop terminates when either an upper bound of the number of iterations has been reached or if the magnitude of the gradient update is below some threshold.

Dataset

The dataset for this problem contains a set of 1000 optimized thermoelastic designs in a 64x64 domain, where each design is optimized for a unique set of conditions. Each datapoint contains: the conditions under which the design was optimized (fixed elements, loaded elements, heatsink elements, volume fraction constraint, ...), the optimized design, and the objective values for the optimized design. Each datapoint's conditions are randomly generated by arbitrarily placing: a single loaded element along the bottom boundary, two fixed elements (fixed in both the x and y direction) along the left and top boundary, and heatsink elements along the right boundary. Furthermore, values for the volume fraction constraint are randomly selected in the range $[0.2, 0.5]$. The full dataset is available at: `https://huggingface.co/datasets/IDEALLab/thermoelastic_2d_v0`.

C.4 Beams2D

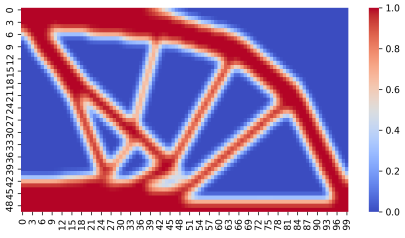


FIGURE 9: An optimized beam, representative of Beams2D.

[8] GitHub: TopOpt-MMA-Python

https://doi.org/10.52202/085713-0003

Motivation

The optimization of beam cross-sections is one of a fundamental problem in engineering, aiming to maximize the structural stiffness under some applied force. This objective is usually formulated as minimizing the compliance, which is the inverse of stiffness. In particular, TO frames the problem as one of optimal material distribution, defining a grid of elements for which the material densities must be determined on a scale from 0 to 1, where 1 represents the presence of material. After applying the beam loads and other boundary conditions, designs are typically optimized using a gradient-based approach with the help of the finite element method (FEM) [5, 2]. While this is one of the simplest TO applications, it is still a computationally expensive process requiring many iterations, opening the door for faster approximation methods such as generative inverse design.

One of the most common beam types in TO is the Messerschmitt-Bölkow-Blohm (MBB) beam, which is supported at the bottom-right and bottom-left corners, with a downward force applied on the top-center. Given this symmetric configuration, one half of the design may be optimized while representing the entire structure. We implement the MBB beam in ENGIBENCH for the most accessible comparison to previous works in this domain.

Design Space

This problem simulates the right half-section of a MBB beam under bending. This half-beam is subjected to a force at its top-left corner (corresponding to the top-center of the entire design) which may also be shifted to the right to simulate different loading conditions. A roller support at the bottom-right corner prevents vertical movement, and a symmetric boundary condition is enforced on the left edge. The design space consists of a 2D array representing solid densities, parameterized by *nelx* and *nely*, i.e., $\mathcal{X} = [0, 1]^{\text{nelx} \times \text{nely}}$. By default, a 100×50 space is used.

Objectives

We aim to minimize compliance $c(\mathbf{x})$, which may be thought of as the inverse of structural stiffness. This objective is calculated as the sum of strain energy over the structure:

$$\underset{\mathbf{x} \in [0,1]^{\text{nelx} \times \text{nely}}}{\text{minimize}} \; c(\mathbf{x}) = \sum_{e=1}^{N} E_e(x_e)\, \mathbf{u}_e^{\mathrm{T}} \mathbf{k}_0 \mathbf{u}_e \quad \text{subject to} \quad \begin{aligned} &\frac{V(\mathbf{x})}{V_0} - volfrac = 0, \\ &\mathbf{KU} - \mathbf{F} = 0. \end{aligned}$$

Where $N = \text{nelx} \times \text{nely}$ is the total number of array elements, $E_e(x_e)$ is the penalized Young's Modulus based on the Solid Isotropic Material with Penalization method (SIMP) formulation [63], $\mathbf{u}_e$ is the element displacement vector, $\mathbf{k}_0$ is the reference stiffness matrix for a solid element $x_e = 1$, $V(\mathbf{x})$ and V_0 are respectively the material volume and design domain volume, $volfrac$ is the desired volume fraction, $\mathbf{K}$ is the global stiffness matrix, $\mathbf{U}$ is the global displacement vector, and $\mathbf{F}$ is the global force vector [2].

Conditions

The conditions ($\mathcal{C}$) consist of the desired volume fraction of solid material in the design (*volfrac*), the minimum feature length of beam members (*rmin*), the fractional distance of the downward force from the default top-left corner to the top-right corner (*forcedist*), and a boolean overhang constraint: when true, this removes unsupported structures with an overhang angle of greater than 45 degrees from the vertical axis, preserving manufacturability. In practice, this constraint often leads to convergence issues when combined with an insufficient *volfrac*, so we have excluded it from the dataset for the moment.

Constraints

Theoretical constraints (error)

$$\begin{aligned} \mathcal{X} &\in [0,1]^{\text{nelx}\times\text{nely}} \\ nelx &\in [1,\infty) \\ nely &\in [1,\infty) \\ volfrac &\in [0,1] \\ rmin &\in (0,\infty) \\ forcedist &\in [0,1] \end{aligned}$$

Theoretical constraints (warning)

$$\frac{\sum_{i=0}^{\text{nelx}}\sum_{j=0}^{\text{nely}} x_{i,j}}{\text{nelx}\times\text{nely}} \leq volfrac$$

Implementation constraints (error)

$$rmin \in (0, 0.5 \times \max\{nelx, nely\}$$

Implementation constraints (warning)

$$\begin{aligned} nelx &\in [10,1000] \\ nely &\in [10,1000] \\ volfrac &\in [0.1,0.9] \\ rmin &\in [1.0,10.0] \end{aligned}$$

Simulator

Our simulation code is based on a Python adaptation[9] of the popular 88-line topology optimization code [2]. It uses the more versatile density filtering approach in combination with a standard Optimality Criteria (OC) optimization method. Two primary sensitivity matrices, one with respect to compliance (dc) and the other with respect to volume fraction (dv), are continuously updated and used to calculate a given design's compliance value. We have also ensured that during the required Lagrange multiplier search within OC, the inner optimization loop terminates if the absolute difference upper and lower bounds diminishes to a value smaller than machine precision. This prevents the code from becoming stuck at this point, which we observed in some warm-starting instances with noisy initial designs.

Dataset

This problem offers multiple datasets for various sizes of *nelx* and *nely*. Each dataset includes columns for `optimal_design`, all conditions listed above, and the corresponding objective values. For advanced usage, we also provide a column containing the optimization history. The datasets have been generated by sampling conditions over a structured grid for various problem sizes. The full set of parameter combinations and dataset generation script are available at `https://github.com/IDEALLab/EngiBench/blob/main/engibench/problems/beams2d/README.md`.

[9] GitHub: TopOpt-MMA-Python

https://doi.org/10.52202/085713-0003

C.5 Photonics2D

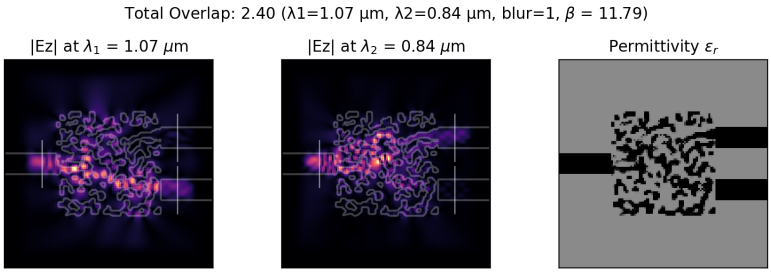


FIGURE 10: Example visualization of our Photonics2D problem. The two left plots show where different wavelengths are routed (source is on the left). The rightmost plot shows the material distribution.

Motivation

The optimization of photonic circuits in general, and multiplexers in particular, was one of the initial and most widely studied problems in the inverse design of electromagnetic/optical devices [57]. In part, this is because multiplexer devices have several interesting properties that make them more difficult to create generative models of, compared to other problems in EngiBench. This includes the fact that, due to the wave properties of the physical phenomena there are usually multiple solutions with equivalent or similar performance, which results from shifting or inverting the phase profile of the electromagnetic wave. This adds complexity to the generative model in that the solution may not have a single unique global minimum. Another motivating factor for including this problem is the complexity of the structures/designs themselves: unlike in structural or thermal compliance problems, which lead to connected structures, the photonics solutions often involve several disconnected elements whose relative position and spacing is governed by the specific wavelengths it needs to demultiplex. This is a difficult prediction and generation task, compared to, *e.g.*, generating a connected beam structure. Thus, it acts as a good counterpoint to add to the library and provides a mechanism to benchmark generative algorithms that can perform well on both connected and disconnected design topologies.

Design Space

This problem simulates a wavelength demultiplexer where the optimized device will direct an electromagnetic wave to one of two possible output ports depending on the wavelength/frequency of the incoming wave [32, 57]. Specifically, the demultiplexer targets two specific wavelengths (referred to λ_1 and λ_2 in the library), and the performance of the device is how well it can bend or direct the energy toward two specific locations in the device, as measured by how much of the electric field of each wavelength overlaps with the desired output port locations. The design space consists of a 2D array representing the presence of either a high or low permittivity, parameterized by *nelx* and *nely*, i.e., $\mathcal{X} = [0,1]^{\text{nelx}\times\text{nely}}$. By default, the library uses a 120×120 space, however this can be modified to non-square design spaces by the user.

Objectives

The main objective is to maximize the overlap of the electric field of the simulated wavelength at the target output location, with an optional penalty for the amount of material used (this penalty weight is set to a small default value ($1e^{-2}$) for consistency, but can be altered for advanced usage):

$$\underset{\mathbf{x}\in[0,1]^{\text{nelx}\times\text{nely}}}{\text{maximize}}\ c(\mathbf{x}) = \left[\int_{p_1} |m_1^T e_{\omega_1}|^2 \times \int_{p_2} |m_2^T e_{\omega_2}|^2\right] - w||\mathbf{x}||^2$$

Where $N = \text{nelx} \times \text{nely}$ is the total number of array elements, w is the penalty weight for usage of material (set by default to $1e^{-2}$), $p1$ and $p2$ are the output port locations, m_1 and m_2 are the

output mode profiles, and e_{ω_1} and e_{ω_2} are the computed fields under an input wavelength λ_1 and λ_2, respectively.

In practice, the problem uses a SIMP formulation (similarly as described in Beams2D) to gradually interpolate between two material permittivities by using a Heaviside projection operator (where the projection strength is governed by a β strength parameter) to project the designed density field onto the two target permittivities. As with other SIMP-style optimization approaches, to ensure smooth optimization toward binary designs, this projection strength undergoes a continuation scheme where it is initially set to a low value ($\beta = 1.0$) and then polynomially (quadratically) increased throughout optimization until a final maximum value ($\beta = 300$), where the projection operator is essentially outputting binary designs.

Conditions

The conditions ($\mathcal{C}$) consist of the two input wavelengths to be demultiplexed—λ_1 and λ_2, as well as a desired `blur_radius` (r_{blur}) parameter, which blurs (using a circular convolution) the pixelized design field for a chosen number of integer pixels—this blurring essentially creates a penalty on the minimum feature size of the design. The size of the device—expressed as *nelx* and *nely*—is also adjustable, and could be viewed as a possible condition for multi-resolution problems, but in practice, as with Beams2D above, this is built into the problem definition since it produces a different dataset.

Constraints

We now list the constraints for the Photonics2D problem.

Theoretical constraints (error)

$$\begin{aligned} \mathcal{X} &\in [0,1]^{\text{nelx}\times\text{nely}} \\ nelx &\in [1,\infty) \\ nely &\in [1,\infty) \\ \lambda_i &> 0 \\ r_{blur} &\geq 0 \end{aligned}$$

Implementation constraints (error)

$$\begin{aligned} \lambda_i &\in [0.5,\infty] \\ r_{blur} &\geq 0 \\ nelx &> 60 \\ nely &\geq 105 \end{aligned}$$

Implementation constraints (warning)

$$\begin{aligned} \lambda_i &\in [0.5,1.5] \\ r_{blur} &\in [0,5] \\ nelx &\in [90,200] \\ nely &\in [110,300] \end{aligned}$$

Simulator

The simulation code uses the `ceviche` library [33] and specifically, the wave demultiplexer demonstration case provided by the library authors[10] based on their related publication [32], which uses a similar formalism to an earlier demultiplexer paper by Piggott et al. [57]. The optimization method is first-order and uses the Adam optimizer. Beyond the baseline implementation already available via `ceviche`, we implemented a polynomial β continuation scheme that performed more reliably that the step-wise continuation scheme used in the original implementation, and ENGIBENCH also possesses the ability to change the starting and ending continuation values, for future research cases where one wishes to estimate or optimize the continuation schedule themselves. Other than these changes, the implementation of this problem is as consistent as possible with that of the original `ceviche` library.

[10] `https://github.com/fancompute/workshop-invdesign/blob/master/04_Invdes_wdm_scheduling.ipynb`

https://doi.org/10.52202/085713-0003

Dataset

This problem offers a single datasets of *nelx*=120 and *nely*=120, although various sizes of *nelx* and *nely* could be generated from the library if desired. The dataset includes columns for the `optimal_design`, all conditions listed above, and the corresponding objective value history as the optimizer optimized toward the optimal design provided in the dataset. The dataset was generated by sampling by sampling at random λ_1, λ_2 and r_{blur} over the conditions mentioned above. The full set of parameter combinations used for the dataset are contained in the docstring and problem documentation available at `https://github.com/IDEALLab/EngiBench/blob/main/engibench/problems/photonics2d/v0.py`.

C.6 Power Electronics

Motivation

Optimizing circuit parameters is a critical aspect of circuit design but remains challenging, particularly for power converter circuits that contain diodes and switches, which introduce significant nonlinearity and discontinuity. These characteristics make circuit simulation results and key objectives such as *DcGain* and *Voltage Ripple* highly sensitive to even small parameter variations.

Because the circuit simulator NgSpice operates as a black box and is non-differentiable, Bayesian optimization is commonly employed for parameter tuning. Surrogate models also offer a promising alternative for this task. In this work, we present experiments using a fixed circuit topology. Even under this constraint, optimizing circuit parameters to minimize the objectives remains a challenging problem for surrogate models, as we show in Section 4.

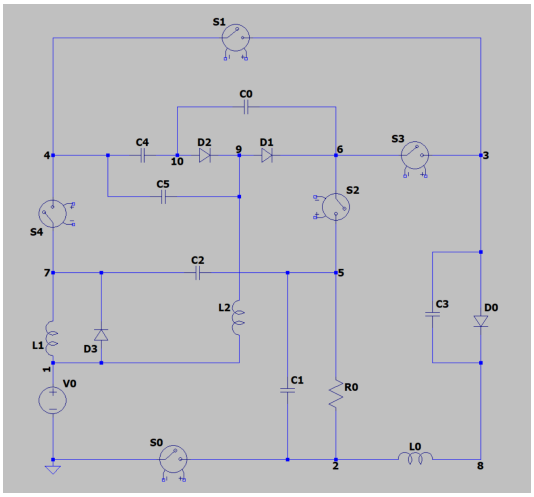


FIGURE 11: The fixed electric circuit with 5 switches, 6 capacitors, 3 inductors and 4 diodes that we optimize in PowerElectronics.

This problem models a DC-DC power converter circuit with a fixed topology. The circuit comprises 5 switches, 4 diodes, 3 inductors, and 6 capacitors. To simulate a circuit, we first encapsulate the circuit topology and parameter information into a text-based file known as a "netlist". This file is then processed by NgSpice, an open-source, cross-platform simulator widely used for analyzing analog, digital, and mixed-signal circuits [1]. For resistors "R", inductors "L", capacitors "C" (RLC) circuits, we apply transient analysis, where NgSpice formulates the system as a set of differential equations based on Kirchhoff's laws. These equations are discretized using numerical integration methods such as the Backward Euler or trapezoidal rule and are solved iteratively at each time step to compute performance metrics, as detailed in the Objectives section below.

Since NgSpice functions as a black-box simulator and does not offer differentiability with respect to input parameters, gradient-based optimization methods are not applicable. Therefore, we formulate

the problem as one with a fixed topology, ensuring that optimization is restricted to parameter tuning rather than structural modifications. Furthermore, we choose a specific on-off pattern for the 5 switches, with *Switch 1, Switch 2, Switch 3, Switch 4, Switch 5* set to *on, off, on, on, off* at the beginning and switching their states simultaneously. This setting avoids NgSpice simulation failure. Despite the fixed topology and simplification of the switch parameters, determining the optimal values for the remaining circuit parameters to minimize the objectives remains a challenging task for surrogate models, see Section 4.2 and Appendix D.

Design Space

The design space for this problem is represented as a 10-dimensional bounded box, where each dimension corresponds to a specific circuit parameter. These parameters include values for capacitors, inductors, and a shared duty cycle for all switches. Each design can be expressed as a vector x of the form:

$$x = [C_1, \ldots, C_6, L_1, L_2, L_3, T_1]^\top \in \mathcal{X} \quad \text{where} \quad \mathcal{X} = [1\text{e}{-6}, 2\text{e}{-5}]^6 \times [1\text{e}{-6}, 1\text{e}{-3}]^3 \times [0.1, 0.9]$$

Here, $C_1, \ldots, C_6$ are the capacitance values (in Farads), L_1, L_2, L_3 are the inductance values (in Henries), and T_1 is the duty cycle shared across all 5 switches. The duty cycle T_1 denotes the fraction of time during which the switches are in the "on" state and governs a periodic on-off pattern repeated at high frequency throughout the simulation.

Objectives

The simulation outputs two scalar values: *DcGain* and *Voltage Ripple*. The former represents the ratio of load to input voltage and should ideally approximate a predefined constant, such as 0.25, as closely as possible. Meanwhile, the latter quantifies the voltage fluctuation at the load. Let $t_1, \ldots t_N$ denote the time steps during the simulation, then the objectives are characterized by the following parameters:

$$\begin{aligned}\min_{\mathbf{x}\in\mathcal{X}} |\text{DcGain(x)} - 0.25| &= |\frac{\overline{V_{load}(t)}}{V_{Source}} - 0.25| \\ &= |\frac{1}{V_{Source}} \cdot \frac{1}{T} \sum_{i=1}^{N-1} \frac{V_{load}(t_{i+1}) + V_{load}(t_i)}{2} \cdot (t_{i+1} - t_i) - 0.25|\end{aligned}$$

where $\overline{V_{load}(t)}$ is the average voltage of the load calculated by the simulator during the transient analysis and V_{source} is the voltage source of 1000 volt. $T = t_N - t_1$ is the duration of the simulation.

$$\begin{aligned}\min_{\mathbf{x}\in\mathcal{X}} \text{Voltage Ripple} &= \frac{V_{pp}(t)}{\overline{V_{load}(t)}} \\ &= \frac{\max_{i\in[1,N]} V_{load}(t_i) - \min_{i\in[1,N]} V_{load}(t_i)}{\overline{V_{load}(t)}}\end{aligned}$$

where $V_{pp}(load)$ is the peak-to-peak value on the load calculated during the transient analysis.

Conditions

This problem does not include environmental or operational conditions as part of its input specification. Unlike other domains where the simulation setup may vary based on conditions (*e.g.*, load configurations or external temperatures), the PowerElectronics circuit is simulated under fixed source voltage and switching behavior. As a result, the design optimization task focuses solely on tuning internal circuit parameters, with no external conditions to vary. More complex variants of this problem—involving multiple topologies or variable source voltages—may be considered in future releases.

Constraints

We list the theoretical constraints, which represent physically infeasible conditions that always result in an error.

https://doi.org/10.52202/085713-0003

Theoretical constraints (error)

$$C_i \in [1e-6, 2e-5], \quad i = 0, 1, 2, ..., 5$$
$$L_i \in [1e-6, 1e-3], \quad i = 0, 1, 2$$
$$T_1 \in [0.1, 0.9]$$

Simulator

We use the well-established NgSpice Circuit Simulator in this problem.

Dataset

All simulation starts from 1 millisecond to 1.06 milliseconds. The dataset consists of three parts as follows:

- **Part 1:** The 6 capacitors and 3 inductors only take their minimum and maximum values to map the edge of the design space. T_1 ranges over $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. This results in:
$$2^6 \times 2^3 \times 9 = 4608 \text{ samples.}$$
- **Part 2:** Randomly sample 4608 points in the 10-dimensional space (6 capacitors, 3 inductors, and T_1). The minimum and maximum values for each dimension are excluded from the sampling.
- **Part 3:** To enhance the sampling coverage in the design space, we use Latin hypercube sampling to generate 4608 points. Each dimension is divided into 10 intervals, and the minimum and maximum values in each dimension are excluded from the sampling.

These three parts are randomly shuffled and split into training, validation, and test datasets with the ratio of $7/2/1$.

D Additional Information on Experiments

D.1 Cross-domain study

TABLE 8: Hyperparameters used for GAN2D and CGAN2D in the cross-domain study.

Hyperparameter	Value
lr_disc	0.0004
lr_gen	0.0001
n_epochs	100
hidden_layers	4
hidden_size	128
batch_size	32
b1	0.5
b2	0.999
latent_dim	32

Hyperparameters used for our experiments in Section 4.1 can be found in Tables 8 and 9. They were chosen based on our prior experience and without an extensive search, *i.e.*, reducing the number of channels and removing the transformer blocks of the diffusion model.[11] Conducting a systematic hyperparameter sweep proved impractical, since each problem and evaluation metric (beyond just MSE) would require its own tuning regime. We view the challenge of hyperparameter optimization across heterogeneous engineering problems and metrics as an open and valuable direction for future research. We ran the experiments with seeds in $\{1, \ldots, 10\}$.[12]

[11] Our implementations draw from `https://huggingface.co/learn/diffusion-course/unit1/2` and `https://github.com/togheppi/cDCGAN`.

[12] Code for GAN2D: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/gan_cnn_2d`. Code for CGAN2D: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/cgan_cnn_2d`.

TABLE 9: Hyperparameters used for CDiffusion2D in the cross-domain study.

Hyperparameter	Value
lr	0.0004
n_epochs	100
batch_size	32
b1	0.5
b2	0.999
latent_dim	100
layers_per_block	2
noise_schedule	linear

Here is an example command line used to train our CDiffusion model on the heat conduction problem. Our framework makes it possible to train this model on a different problem by only changing the `problem_id`:

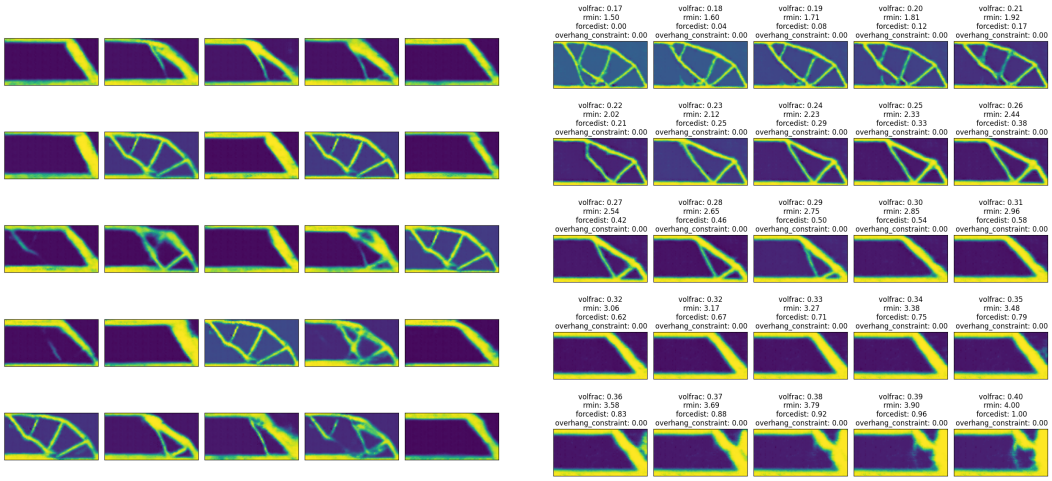
```
python engiopt/diffusion_2d_cond/diffusion_2d_cond.py
    --problem_id heatconduction2d
    --seed 10
    --save_model
    --n_epochs=100
```

For conditioning in the model evaluations, 50 conditions were sampled uniformly at random with replacement from the configurations in the test datasets. Then, these configurations were given to the conditional model generators as conditions. For the unconditional models, we used the same sampling procedure—both to ensure broader coverage of test conditions and to supply conditions for simulation and optimization experiments—but the generators themselves received only noise as input. Metrics were then computed based on the outputs relative to their associated conditions.

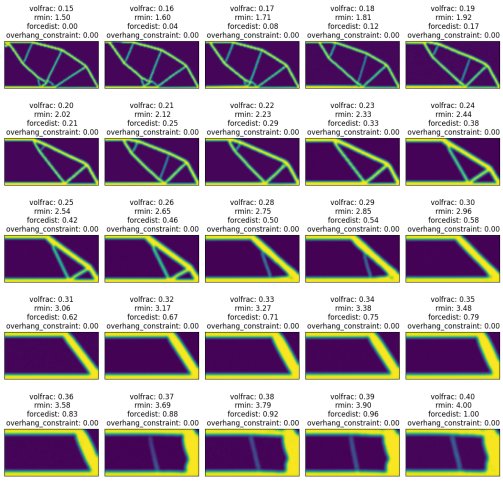
Some example outputs generated from the trained algorithms are illustrated in Figs. 12 to 14. Diffusion models tend to produce more detailed and structured outputs, whereas GANs yield blurrier designs. Interestingly, although GAN outputs may appear less refined visually, they are often easier to optimize, as reflected by lower COG values in HeatConduction2D and Beams2D (Section 4.1). The figures also highlight the contrast between unconditional and conditional models: while conditional models generate designs tailored to specific conditions, unconditional models sample freely across the entire design space. This explains the higher DPP scores observed for unconditional models, as they naturally exhibit greater diversity. Conversely, the conditioned models should better respect the constraints, as is the case for the CGAN. It is unclear why the Diffusion models violate constraints almost 100% of the time in the results. Similarly, we expected conditional models to achieve lower MMD scores, as they are evaluated against the conditional distributions aligned with the sampled test conditions. These results are surprising and should not be overinterpreted: we did not perform extensive hyperparameter tuning, and the number of seeded runs is insufficient for strong statistical claims. Rather, these findings serve as a proof of concept and highlight important directions for future investigation.

Code for CDiffusion2D: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/diffusion_2d_cond`.

https://doi.org/10.52202/085713-0003

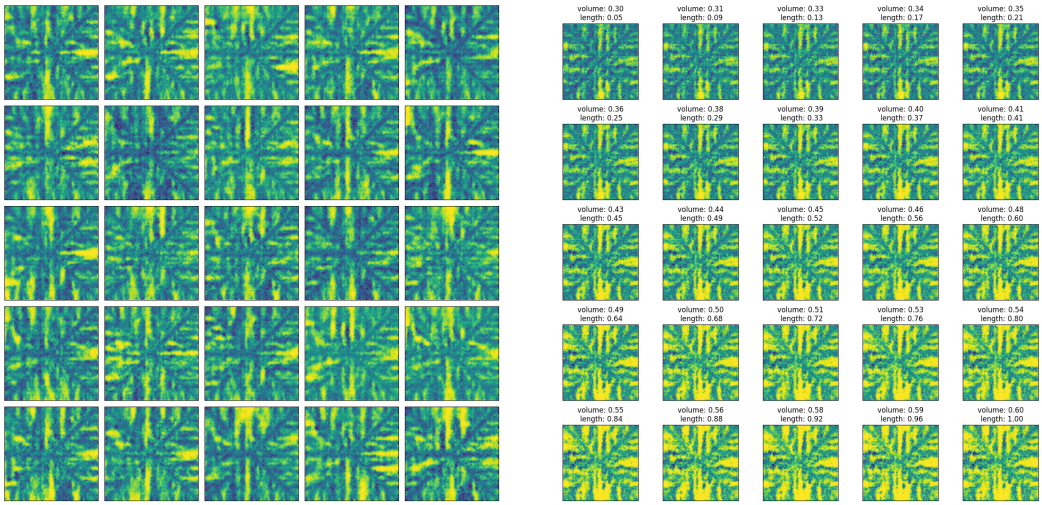


GAN2D CGAN2D

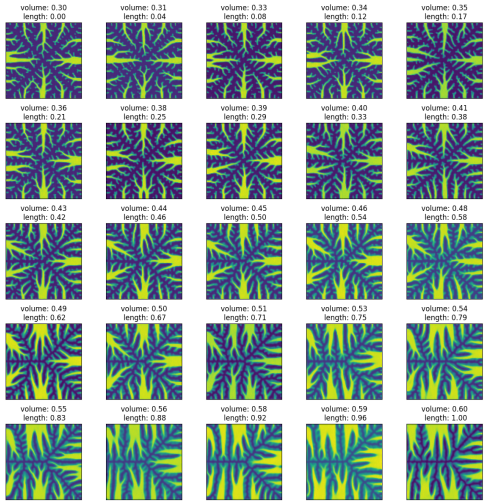


CDiffusion2D

FIGURE 12: Illustrative outputs of our generative models on the Beams2D task.



GAN2D CGAN2D



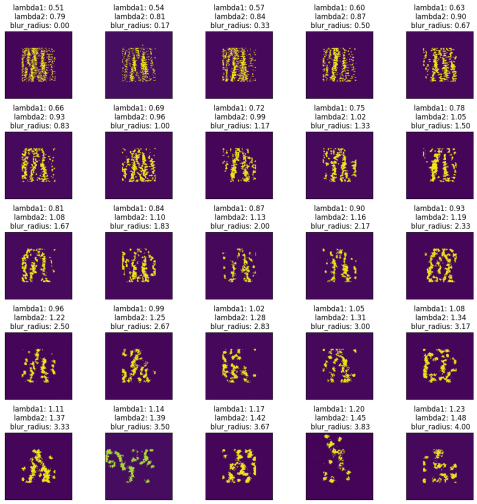
CDiffusion2D

FIGURE 13: Illustrative outputs of our generative models on the HeatConduction2D task.

https://doi.org/10.52202/085713-0003



GAN2D CGAN2D



CDiffusion2D

FIGURE 14: Illustrative outputs of our generative models on the Photonics task.

D.2 Airfoils

TABLE 10: Hyperparameters used for GAN and BézierGAN in the Airfoil experiment.

Hyperparameter	Value
lr_disc	0.0002
lr_gen	0.00005
n_epochs	5000
batch_size	32
b1	0.5
b2	0.999
latent_dim	4
noise_dim	10
bezier_control_pts	32

TABLE 11: Hyperparameters used for the Diffusion model in the Airfoil experiment.

Hyperparameter	Value
lr	0.0003
n_epochs	100
batch_size	64
b1	0.5
b2	0.999
unet_dim	32

Hyperparameters used for our experiments in Section 4.2 can be found in Tables 10 and 11. For the GANs, we used the same hyperparameter values as in Chen et al. [11]. For the diffusion model, we reused hyperparameters from `https://github.com/lucidrains/denoising-diffusion-pytorch`. We ran the experiments with seeds in $\{1, \ldots, 10\}$. [13]

Example outputs from the trained models are shown in Fig. 15. As illustrated, the standard GAN frequently produces invalid designs characterized by discontinuities or noisy coordinates along the airfoil spline, resulting in a 30% simulation failure rate. In contrast, the BézierGAN and diffusion models consistently generate designs that are smooth and valid for simulation. However, the diffusion model suffers from mode collapse, producing highly similar outputs across different samples.

[13] Code for GAN: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/gan_1d`.
Code for BézierGAN: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/gan_bezier`.
Code for Diffusion: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/diffusion_1d`.

https://doi.org/10.52202/085713-0003

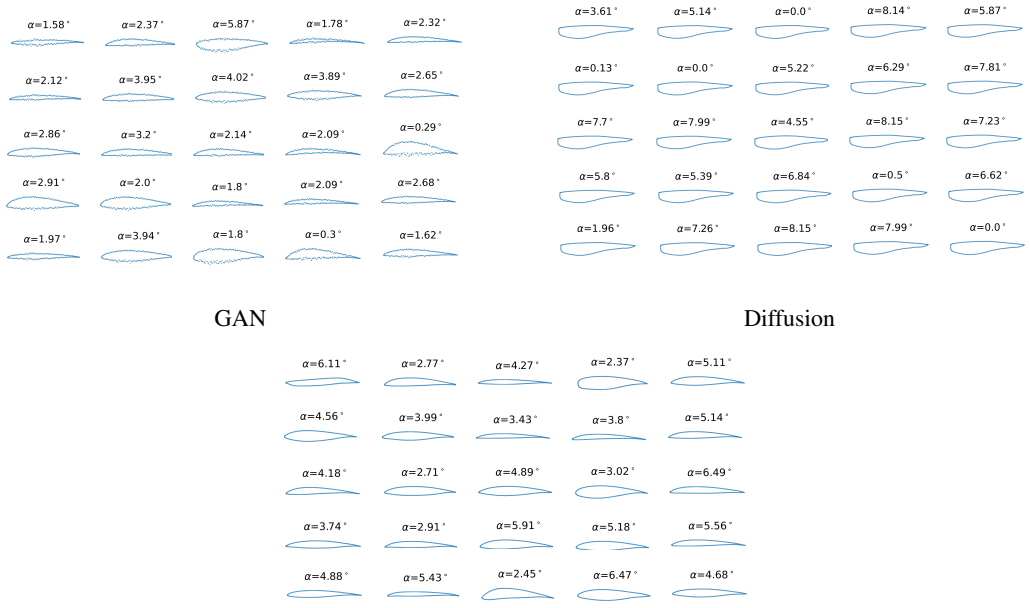


FIGURE 15: Illustrative outputs of our generative models on the Airfoil task.

https://doi.org/10.52202/085713-0003

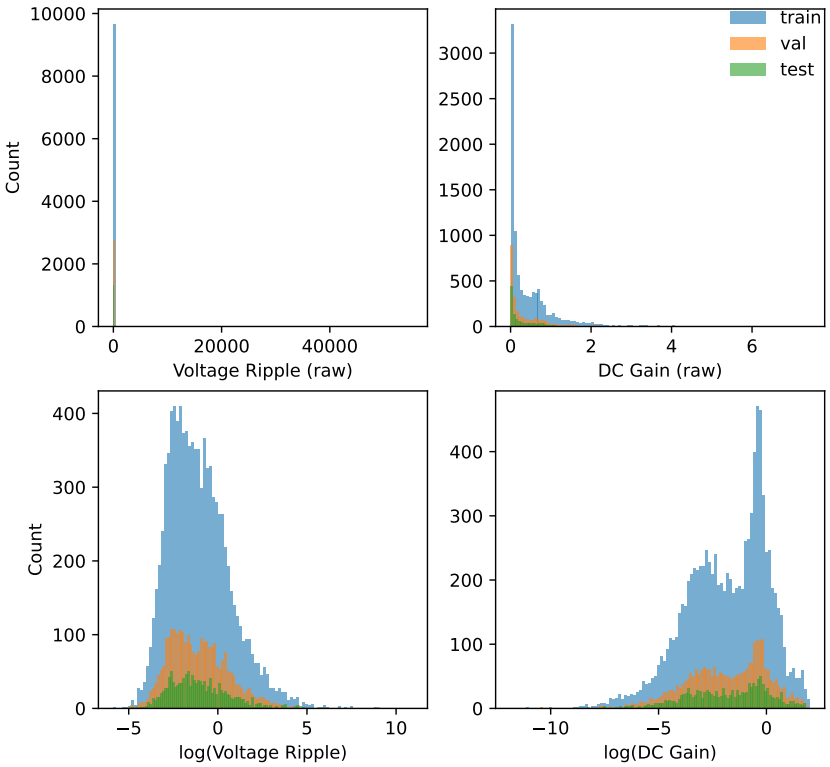


FIGURE 16: Train/val/test distributions of *Voltage Ripple* and *DcGain* in raw (top) and log (bottom) domains. Heavy skew and extreme outliers motivate the log transform and robust scaling.

D.3 Surrogate-assisted multi-objective optimization

Problem setting

For a fixed power-converter topology, the continuous design vector

$$x = [C_1, \ldots, C_6, L_1, L_2, L_3, T_1]^\top \in \mathcal{X} \quad \text{where} \quad \mathcal{X} = [1\text{e}{-6}, 2\text{e}{-5}]^6 \times [1\text{e}{-6}, 1\text{e}{-3}]^3 \times [0.1, 0.9]$$

is mapped by the NGSpice simulator to two responses: *DcGain* y_1 and *Voltage Ripple* y_2. We aim to *simultaneously* minimize

$$g_1(x) = |y_1(x) - 0.25|, \qquad g_2(x) = y_2(x).$$

Methods

We performed several steps for training our surrogate models and optimizing designs using these. [14]

Pre-processing: Every raw feature x_j has been converted to a robust-scaled value $\tilde{x}_j = (x_j - Q_{1,j})/(Q_{3,j} - Q_{1,j})$, where $Q_{1,j}$ and $Q_{3,j}$ are the 25th / 75th percentiles of feature j in the training set (inter-quartile range scaling). Each target y_k has been first stabilized by $\log(y_k)$ and then scaled with the same formula, mitigating the heavy-tailed distribution (see Fig. 16).

Surrogate model: For each target $k \in \{1{:}2\}$ (1: *DcGain*, 2: *Voltage Ripple*) we fit an MLP $f_{\boldsymbol{\theta}_k} : \mathbb{R}^{10} \to \mathbb{R}$ with parameters $\boldsymbol{\theta}_k$. Given the training set $\mathcal{D}_{\text{tr}} = \{(\tilde{x}^{(i)}, y_k^{(i)})\}_{i=1}^{N_{\text{tr}}}$, where $\tilde{x}$ is the robust-scaled design vector and $N_{\text{tr}} = 0.7N$, the parameters are obtained by

$$\boldsymbol{\theta}_k^\star \;=\; \arg\min_{\boldsymbol{\theta}} \frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \mathcal{L}_{\text{sL1}}\big(f_{\boldsymbol{\theta}}(\tilde{x}^{(i)}),\, y_k^{(i)}\big),$$

[14] Code for surrogate-based optimization: `https://github.com/IDEALLab/EngiOpt/tree/main/engiopt/surrogate_model`.

https://doi.org/10.52202/085713-0003

TABLE 12: Hyperparameter search space ($\mathcal{Z}$) and optimization budget.

Hyperparameter	Value range
`learning_rate`	$[10^{-5}, 10^{-3}]$
`hidden_layers`	$\{2, 3, 4\}$
`hidden_size`	$\{16, 32, 64, 128\}$
`batch_size`	$\{8, 16, 32, 64\}$
`l2_lambda`	$[10^{-6}, 10^{-3}]$
`activation`	{ReLU, Tanh}
Budget (trials)	50
Seeds s	$\{100, 101, 102\}$

where $\mathcal{L}_{\text{sL1}}$ denotes the Smooth-L_1 loss. Optimization uses Adam (PyTorch 2.6) with early stopping (patience 20) to prevent over-fitting.

Bayesian hyperparameter search: We used BoTorch+Ax [3, 64] to minimize the *ensemble-averaged* validation loss over the search space in Table 12. Each trial trains an *implicit ensemble* of three MLPs obtained by different initialization seeds $(s, s+1, s+2)$. Similar to other hyperparameter optimization works [14, 55, 18], we tuned hyperparameters on different seeds $s \in \{100, 101, 102\}$ for each trial, giving 3×50 trials, or $3 \times 3 \times 50 = 450$ (seed, ensemble size, trials) MLPs trained per target output; every model has been trained for 50 epochs with early stopping patience set at 15. The optimized hyperparameters are `learning_rate`, `hidden_layers`, `hidden_size`, `batch_size` penalty `l2_lambda`, and `activation`. Hyperparameters have intentionally been biased to limit the size of the networks and avoid over-fitting.

Implicit ensembles: Using the best hyperparameters (best validation loss across seeds), we trained $E = 10$ independent ensembles. Each ensemble contains $M = 7$ replicas initialized with contiguous seeds $(s, \ldots, s+6)$ for $s \in \{0, 10, \ldots, 90\}$. Note that these seeds are different from the seeds used for the hyperparameter search to avoid seed overfitting [14]. Every replica is trained for at most 300 epochs with early stopping (patience 50). Following the implicit ensemble paradigm [74, 20], predictions are averaged $\hat{y}_k(x) = \frac{1}{M} \sum_{m=1}^{M} f_{\theta_k^{(m)}}(x)$, which lowers epistemic error. Variance can be used alongside mean prediction, but it is out of the scope of this work.

Multi-objective search: For each surrogate ensemble, we launched a seed-matched NSGA-II run (population 500, generations 100) via *pymoo* [12, 8]. Seeds $s \in \{0, 10, \ldots, 90\}$ generate ten independent approximations $\hat{\mathcal{P}}^{(s)} \subset \mathcal{X}$ of the Pareto set of optimal designs.

Validation: Designs in $\hat{\mathcal{P}}$ have then been re-evaluated with the simulator, giving surrogate- and simulator-based Pareto fronts that we treat as distributions $\hat{\mathbf{g}}, \mathbf{g}$. The unbiased squared Maximum Mean Discrepancy [23], $\text{MMD}^2(\hat{\mathbf{g}}, \mathbf{g})$, with a Gaussian kernel rejects the null of identical distributions (p<0.05), exposing extrapolation bias.

Results

Bayesian optimization converged to different optima for the two targets (Table 13): the *DcGain* surface favored a small learning rate and tanh activations, whereas the harder *Voltage Ripple* surface required an order of magnitude larger learning rate and ReLU activations. The learning rate decays were empirically selected. Both networks architectures ended up with the largest number of `hidden_layers` and `hidden_size`, signaling a possible overfitting trend.

With these settings, each surrogate ensemble was able to learn a coherent Pareto front (blue curves, Fig. 17); reevaluation in NgSpice (red scatter) revealed systematic offsets for every seed. A two-sample MMD test (1000 permutations, Gaussian kernel) confirmed that the surrogate and NgSpice-generated fronts stem from different distributions: across ten runs the squared statistic ranged from 9.97×10^{-3} to 6.18×10^{-2}, while every p-value hit the permutation floor of 0.001. Hence, the null hypothesis of equal clouds is rejected for all seeds, demonstrating that the surrogate fails to generalize to the NSGA-II–proposed regions despite extensive hyperparameter tuning and ensembling.

TABLE 13: Best hyperparameter configuration selected by Bayesian optimization for each output. The learning-rate decay is the per-epoch multiplicative factor.

Hyperparameter	DC Gain	Voltage Ripple
`learning_rate`	2.67×10^{-4}	1.00×10^{-3}
`learning_rate_decay`	0.95	1.00
`hidden_layers`	4	4
`hidden_size`	128	128
`batch_size`	8	8
`l2_lambda`	1.0×10^{-6}	3.07×10^{-4}
`activation`	Tanh	ReLU

Why is this problem difficult for surrogate models?

Despite a fixed topology and well-defined design bounds, this problem remains challenging for surrogate-assisted optimization. Several intertwined factors contribute:

Stiff dynamics from mixed time constants: The circuit includes both microfarad-level capacitors and millihenry-level inductors, creating vastly different time scales in transient responses. This stiffness can lead to sharp transitions that may be difficult for approximators like MLPs to capture.

Discontinuities and threshold effects: Diodes and switches introduce strong nonlinearity and discontinuous behavior. Small parameter changes near diode conduction thresholds can result in large output shifts (e.g., clipping, ringing, or switching of current paths), making the response surface non-smooth or even piecewise.

High variance and sparse coverage: The target *Voltage Ripple* exhibits heavy-tailed behavior, with a small subset of designs leading to large, high-frequency oscillations. These regions are sparsely represented in the dataset, making them hard to learn accurately even with ensembling.

Simulation-induced numerical noise: NgSpice's numerical integration (*e.g.*, trapezoidal or backward Euler) and tolerances can introduce small but structured numerical artifacts. These manifest as "noise" from the surrogate's perspective, even though they arise from deterministic simulations.

Hidden constraints and failure modes: Although obvious simulation failures are avoided by fixing the switch pattern, near-failure designs can behave erratically or saturate, leading to sharp distortions in the response space that are hard to interpolate.

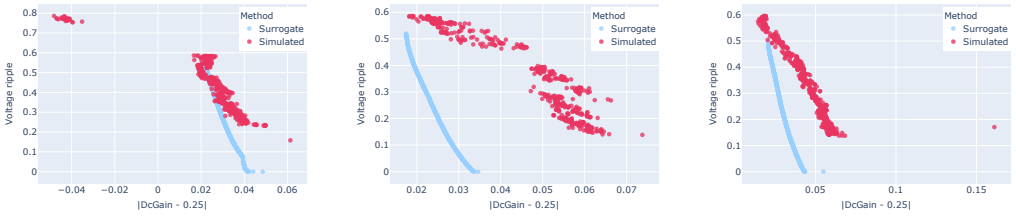
In contrast to smooth benchmark functions, this domain exhibits the highly sensitive and constrained design manifold characteristic of real-world physical systems. It demonstrates the modeling challenges that traditional surrogate models can encounter, calling for physics-informed features or structure-aware priors to perform well.

Take-aways

1. *Voltage Ripple* is intrinsically harder to approximate than *DcGain* due to stiffness (high frequency transients arise from stiff systems dynamics, a regime where standard MLPs struggle to generalize accurately), discontinuities, and skewed distributions driven by switching and nonlinear components;
2. Ensembling reduces variance but does not resolve extrapolation bias in undersampled or near-discontinuous regions;
3. Sparse representation of unstable behaviors (*e.g.*, large ripple) and overfitting in smooth regions can mislead optimization, resulting in systematic surrogate error near the predicted Pareto front;
4. Domain-aware enhancements—such as physics-informed architectures, adaptive sampling near switching thresholds, or hybrid surrogate-simulation loops—may be needed to reliably optimize high-fidelity circuit models.

Future work could explore hybrid strategies that combine adaptive space-filling design, physics-informed normalization, outlier-aware modeling (*e.g.*, classification of unstable regimes), and autoregressive architectures to couple dependent objectives.

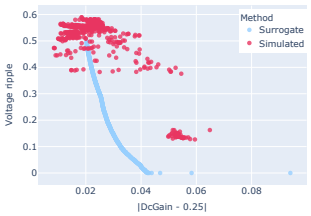
https://doi.org/10.52202/085713-0003



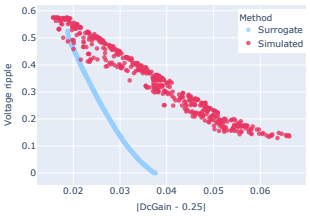
Seed 0

Seed 10

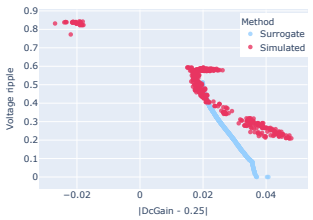
Seed 20



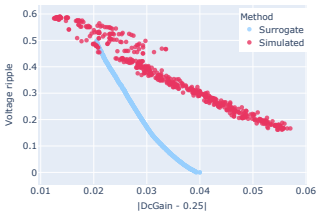
Seed 30



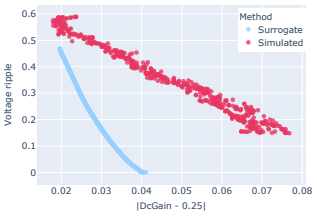
Seed 40



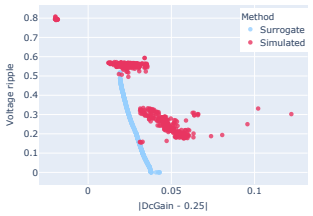
Seed 50



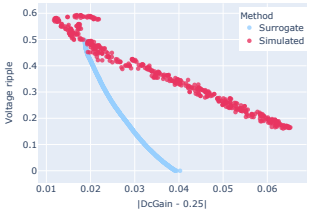
Seed 60



Seed 70



Seed 80



Seed 90

FIGURE 17: NSGA-II Pareto fronts for the ten seed-matched surrogate ensembles. Blue scatter: surrogate prediction; red scatter: NGSpice evaluation.