RITAM ISH PUBLICATIONS

Journal of Progress in Artificial Intelligence

https://www.jpairitamish.org/

ISSN: 3139-4957 (Online)

Vol. 2, Iss. 4, October 2026 – December 2026

Machine learning-based inverse design framework for mechanical springs

Mehmet Safa BİNGÖL[1]

[1]Department of Mechatronics Engineering, Faculty of Engineering, Nigde Omer Halisdemir University, 51240 Nigde, Turkey

Received: 15-Oct-2026, Manuscript No. JPAI-2026-0025; **Editor assigned:** 19-Oct-2026, PreQC No. JPAI-2026-0025 (PQ); **Reviewed:** 28-Oct-2026, QC No. JPAI-2026-0025; **Revised:** 04-Nov-2026, Manuscript No. JPAI-2026-0025 (R); **Published:** 26-Nov-2026

Citation: Bingöl, M. S. (2026). Machine learning-based inverse design framework for mechanical springs. J Prog Artif Intell.

Corresponding: Mehmet Safa Bingöl, Email: msbingol@ohu.edu.tr

ABSTRACT

Designing a helical spring means finding a wire diameter, a coil diameter and a number of coils that give the required loads at the required lengths while keeping the stresses, the safety factors and the geometry within limits. Metaheuristic optimizers can solve this task, but they must be run again for every new requirement, and purely data-driven inverse models rarely guarantee feasible designs. This study proposes a Machine Learning (ML)-based inverse design framework for compression, extension and torsion springs that combines a learned proposal with closed-form spring physics and verification by the calculation engine of the open-source Open Design Optimization Platform (ODOP). A dataset of 150,000 designs covering 17 materials was generated with Latin hypercube sampling and evaluated with ODOP. An Artificial Neural Network (ANN) and a K-Nearest Neighbors (kNN) search propose the wire diameter and the spring index. A physics-based correction then computes the number of coils and the free length or the initial tension so that the required spring rate and load points are met exactly, and 14 candidates per requirement are checked against all default ODOP constraints. On 2,713 unseen requirements, the framework returned a design that met the requirement within 1% and satisfied all default constraints for 89.8%, 90.3% and 100% of the compression, extension and torsion cases, compared with at most 5.8% for a direct inverse ANN. It needed 14 design evaluations and about 1–2 ms per requirement. Over five independent runs, the standard deviation of its

OPEN ACCESS | RESEARCH ARTICLE

success rate was at most 2.1 percentage points, and the success rate stayed above 80% in 36 of 41 tests with a material that was excluded from training. A genetic algorithm with the same physics-based correction solved all requirements and found 21–32% lighter designs, but it needed about 1,900 evaluations per requirement; when it was started from the framework candidates, 182 evaluations were enough for 98–100% success. When fatigue, buckling and hook checks were added to the selection rule, the success rate fell to 34–82%, which shows that these checks must be part of the requirement in cyclic applications. The framework is therefore best used as a fast generator of verified starting designs that can be refined when weight or additional checks are critical.

KEYWORDS: Inverse design, helical springs, machine learning, artificial neural network, k-nearest neighbors, physics-based correction, genetic algorithm, ODOP

1. INTRODUCTION

Mechanical springs are found in almost every machine. They store energy, absorb shocks and keep parts in contact. Helical compression, extension and torsion springs are the most common types. Their design is based on well-known equations for the spring rate, the stresses and the fatigue life [1]. These equations are simple. However, the design task is not simple. The designer must select the wire diameter, the coil diameter and the number of coils. The design must give the required forces at given lengths. It must also keep the stresses below the material limits and satisfy several geometric rules. In practice, this is done by trial and error or by numerical optimization. Open-source tools such as the Open Design Optimization Platform (ODOP) help the designer with this task [2]. ODOP evaluates a design and searches for a feasible one with a constrained optimization algorithm.

Machine Learning (ML), a branch of Artificial Intelligence (AI), has been used to predict the behavior of springs. Ibrikçi et al. trained Artificial Neural Networks (ANNs) on dimensionless data from a numerical buckling model. They found that the networks predicted the critical buckling loads of helical compression springs with high accuracy [3]. Sabuncuoğlu and Demirtaş used finite element analyses and a parametric correlation study to select the most important dimensions of machined torsional springs. They then trained an ANN, and its stiffness predictions agreed with the finite element results [4]. Yavuz and Özkan built parametric finite element models of leaf springs with one to ten leaves. Their ANN estimated the spring constant and the stresses with high precision [5]. Chin et al. extracted wavelet features from road vibration signals and used neuro-fuzzy models to predict the fatigue life of suspension coil springs. Their best model reached a correlation coefficient of 0.955 [6]. Beyond springs, Zhang et al. used fatigue data of ten metallic materials to compare random forest and gradient boosting models. They tested the models on a bearing steel and found that gradient boosting was more accurate, while random forest was more stable [7]. All these studies solve a forward problem. They predict the behavior of a known design. They do not tell the designer which design to choose. A forward model can be placed inside an optimization loop, but then the cost of a search is paid again for every new requirement.

Spring design has also been treated as an optimization problem. Rodriguez Cabal et al. applied the sine-cosine algorithm to minimize the volume of a closed coil helical spring. They reported smaller volumes

than earlier genetic algorithm and particle swarm results [8]. In a later study, Rodriguez Cabal et al. compared four metaheuristic methods. The vortex search algorithm gave the smallest volume in the shortest time, and a finite element check showed only a 0.11% difference in deflection [9]. The tension/compression spring problem is also a standard benchmark for new optimizers. Abualigah et al. proposed the arithmetic optimization algorithm and showed that it outperformed the compared algorithms on this spring problem [10]. Zhao et al. tested the artificial hummingbird algorithm on ten engineering design problems and reported good accuracy at a low computational cost [11]. Hashim et al. introduced the honey badger algorithm and solved four engineering design problems with it [12]. Yu et al. used pretrained transformer models inside constrained Bayesian optimization. Their method was about ten times faster than Gaussian process based methods on benchmarks that include the compression spring [13]. These optimizers work well. However, they must be run again for every new requirement, and each run needs many function evaluations. Three limitations are common to these studies. First, most of them solve a single benchmark instance with fixed loads, so their results cannot be transferred to a new requirement. Second, the constraints are usually handled with penalty functions whose weights and algorithm parameters must be tuned. Third, the computational cost is reported per run, and a complete new run is needed whenever the loads, the lengths or the material change.

Inverse design offers a different route. A model learns the map from the required performance to the design. It can then answer a new requirement almost instantly. The main difficulty is that this map is not unique. Many different designs can give the same performance. Liu et al. showed that this one-to-many nature confuses a plain inverse network during training. They proposed a tandem network that couples an inverse network with a pretrained forward network [14]. Ren et al. compared several deep inverse models and proposed the neural adjoint method. They showed a clear trade-off between accuracy and computation time [15]. Spell et al. trained several backward models with one shared forward model. The forward model then selected the best of the proposed solutions [16]. In mechanics, Bastek et al. inverted the structure property map of truss metamaterials with a combination of inverse and forward networks [17]. Deng et al. combined a neural network with an evolution strategy to design metamaterials with target nonlinear responses [18]. Zheng et al. built a continuous latent design space for truss metamaterials with a graph based variational autoencoder [19]. Li et al. trained ANNs on simulation data and used them for the gradient based design of nonlinear metamaterials [20]. Guo et al. used deep learning to predict the properties of spring based metamaterials. They also generated designs for a target stiffness and Poisson's ratio [21]. Mohammadi et al. proposed a physics enhanced ANN for the inverse design of flexible structures with nonlinear and buckling responses [22]. Regenwetter et al. reviewed deep generative models in engineering design. They noted that meeting design constraints and performance targets is still an open challenge [23]. Karniadakis et al. reviewed physics informed machine learning. They showed that adding physical knowledge can make learning more reliable with less data [24]. However, these methods were developed for problems in which the physics is available only as an expensive simulation or a learned surrogate. Their designs are usually checked with a learned forward model and not with the full set of engineering

constraints, and none of the studies above treats conventional helical springs with code-based limits on the stress, the solid length or the initial tension.

Table 1 summarizes these approaches from the viewpoint of spring design. Forward models do not propose a design. Optimizers propose a design but need a new search for each requirement. Learned inverse models answer quickly but rely on surrogates to judge feasibility. A framework that answers quickly and still checks every design against the engineering constraints is missing for helical springs.

Table 1. Comparison of the proposed framework with related approaches

Approach	**Output for a new requirement**	**Handling of design constraints**	**Main limitation for spring design**
Forward ML models for springs and spring materials[3–7]	Prediction of the behavior of a given design (buckling load, stiffness, stress, fatigue life)	Not addressed; the designer still has to search for a design	No design is proposed
Metaheuristic optimization[8–12]	A new optimization run, typically with thousands of model evaluations	Penalty functions or feasibility rules inside the optimizer	Full cost for every requirement; usually shown on one benchmark instance; parameter tuning
Surrogate-assisted Bayesian optimization[13]	A new optimization run with fewer evaluations	Constraint models inside the search	Still one search per requirement
Learned inverse models (tandem, neural adjoint, mixture networks)[14–16]	A design from a trained inverse network	Indirectly, through a learned forward model	No explicit engineering constraints
Inverse design of mechanical metamaterials and flexible structures[17–22]	Designs from inverse or generative networks, often refined by gradient-based or evolutionary search	Mostly through surrogate predictions of the target response	Not developed for code-based spring limits
Present framework	14 candidates from an ANN, local variants and kNN, corrected with closed-form spring equations	Rate and load points exact by construction; every candidate verified with the full ODOP constraint set (optionally fatigue, buckling and hook checks)	Candidate selection, not optimization; accuracy bounded by the analytical ODOP models

The comparison in Table 1 shows two gaps. First, ML-based inverse design has rarely been applied to conventional helical springs, although spring design is a routine engineering task with established design equations. Second, most inverse models do not guarantee that their designs satisfy the engineering constraints. For springs, this is a missed opportunity, because the main physics is known in closed form and an open-source calculation engine is available. The objective of this study is therefore to develop and critically evaluate a simple inverse design framework that answers a new spring requirement with a verified feasible design in milliseconds, and to clarify when

such a framework is preferable to a conventional optimizer. The framework is shown in Figure 1. The main contributions are as follows. (i) A dataset of 150,000 compression, extension and torsion spring designs was generated with Latin hypercube sampling25 and evaluated with the ODOP calculation engine. (ii) Linear regression, random forest and ANN forward models were compared. (iii) A framework was proposed in which a learned model proposes the wire diameter and the spring index, closed-form spring equations correct the remaining variables so that the required spring rate and load points are met exactly, and ODOP verifies every candidate. (iv) The framework was compared with three simpler inverse methods and with a genetic algorithm on the same requirements. (v) Practical aspects were assessed: fatigue, buckling and hook checks, standard wire diameters, unseen materials, repeated runs, hyperparameters and sensitivity to geometric, material and load variations. Section 2 describes the models and methods, Section 3 presents the results, Section 4 discusses them, and Section 5 gives the conclusions.

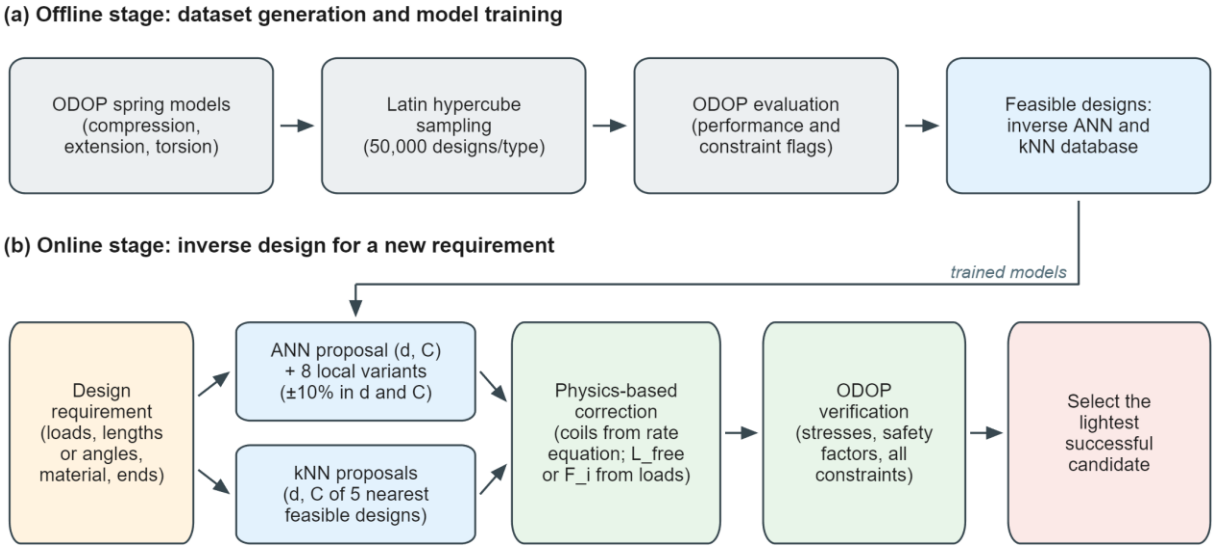

Figure 1. The proposed inverse design framework. (a) Offline stage: Dataset generation with the ODOP engine and model training. (b) Online stage: Candidate generation, physics-based correction, ODOP verification and selection of the lightest successful design

2. MATERIAL AND METHOD

2.1 Spring models and calculation engine

Three helical spring types were studied: Compression, extension and torsion springs. All calculations were made with the calculation engine of ODOP [2]. ODOP is an open-source spring design program released under the MIT license. Its equation sets for the three spring types were used (ODOP version 6.3.0dev, commit 0033fc2, 27 September 2026). The metric material table of ODOP was used. It contains 17 spring materials, from music wire and stainless steels to bronzes and titanium. The equations below are the

standard relations for helical springs [1], as implemented in ODOP [2]. The main design variables are the outside diameter *OD*, the wire diameter *d* and

$$D = OD - d, \quad C = \frac{D}{d} \tag{1}$$

For compression and extension springs, the spring rate is

$$k = \frac{K_h G d^4}{8 D^3 N_a} \tag{2}$$

where G is the shear modulus, N_a is the number of active coils and K_h is an empirical correction factor for hot-wound springs used in ODOP [2] ($K_h = 1$ for cold-wound springs). The shear stress τ under a force F includes the Wahl correction factor K_w[26]:

$$\tau = K_w \frac{8FD}{\pi d^3}, \quad K_w = \frac{4C-1}{4C-4} + \frac{0.615}{C} \tag{3}$$

For torsion springs, the rate per turn k_t and the bending stress under a moment M are given below. ODOP divides k_t by 360 to express the rate per degree.

$$k_t = \frac{K_h E d^4}{10.8 D N_a}, \quad \sigma = K_b \frac{32M}{\pi d^3}, \quad K_b = \frac{4C-1}{4C-4} \tag{4}$$

where E is the elastic modulus. The value of K_b in Equation 4 is the form used in ODOP for stress-relieved springs [2]. Budynas and Nisbett give the bending stress correction factor for the inner fiber of torsion springs as $K_i = (4C^2 - C - 1)/[4C(C - 1)]$[1]. For $C \geq 4$, the two factors differ by less than 2%, and the ODOP value is slightly larger. For springs without stress relief, $K_b = 1$ is used in ODOP [2]. The static safety factor FS_2 (called the static factor of safety in ODOP) is the ratio of the allowable stress to the working stress at the second load point. Here, τ_2 and σ_2 are the stresses under F_2 and M_2, and τ_{allow} and σ_{allow} are the allowable static stresses, which ODOP obtains from the size-dependent tensile strength of the material. The first form below is used for compression and extension springs, and the second form for torsion springs:

the number of coils. The mean coil diameter D and the spring index C are

$$FS_2 = \frac{\tau_{allow}}{\tau_2}, \quad FS_2 = \frac{\sigma_{allow}}{\sigma_2} \tag{5}$$

ODOP also computes the solid height, the fatigue safety factor, the hook stresses of extension springs, the weight and many other outputs [1,2]. Table 2 lists the independent variables and the default ODOP constraints for each spring type. Throughout this paper, the spring rate k is the load (or moment) per unit deflection, the spring index is $C = D/d$, N_a is the number of active coils and N_t is the total number of coils. A design is called *valid* when all outputs are within their physical ranges. A design is called *feasible* when it is valid and satisfies all default constraints. Three extra checks were also recorded. These were the fatigue safety factor ($FS_{fatigue} \geq 1.02$ for compression and extension springs and ≥ 1 for torsion springs, applied only to cyclic life categories), the buckling tendency of compression springs with fixed ends and the hook safety factor ($FS_{hook} \geq 1$) of extension springs. A feasible design that also passes these checks is called feasible-extended. The extra checks are not part of the default ODOP constraint set because they depend on information that is often decided later in the design process: the required life and the surface treatment (fatigue), the presence of a guide rod or a sleeve (buckling) and the bend radius of the hook (hook stress) [1,2]. In this study, they were therefore treated as an optional extension of the success criterion (Section 2.5).

Table 2. Independent variables and default ODOP constraints of the three spring types

Spring type	Independent variables	Default ODOP constraints
Compression	*OD*, *d*, free length L_{free}, total coils N_t, forces F_1 and F_2	$1 \le N_a \le 40$; deflection at $F_1 \ge 0$; $4 \le C \le 25$; $1.02 \le FS_2 \le 1.5$; $FS_{solid} \ge 1$; used deflection ≤ 90% of available deflection
Extension	*OD*, *d*, N_t, initial tension F_i, end extension, F_1, F_2	$F_1 \ge F_i$; $1 \le N_a \le 90$; $4 \le C \le 25$; initial stress within the recommended range; $1.1 \le FS_2 \le 2.0$; deflection ≤ 90% of safe deflection
Torsion	*OD*, *d*, N_t, moments M_1 and M_2, coil spacing	$1 \le N_a \le 100$; deflection at $M_1 \ge 0$; $4 \le C \le 25$; $1.05 \le FS_2 \le 1.8$; deflection ≤ 95% of safe deflection

2.2 Dataset generation

Designs were sampled with Latin hypercube sampling [25]. A two-step scheme was used to avoid a large share of meaningless designs. In the first step, the material, the end type, the life category, the wire diameter, the spring index and the number of coils were sampled. ODOP was run once to obtain the spring rate and the stress per unit load. In the second step, the loads were chosen to reach a sampled target safety factor between 0.9 and 2.0. The free length of compression springs was then set from a sampled ratio of used to available deflection. Table 3 lists the sampling ranges. Values outside the ODOP limits, such as a spring index of 3, were kept on purpose. In this way, the dataset also covers infeasible regions.

For each spring type, 50,000 designs were generated with a fixed random seed of 42. Each record contains the categorical inputs, all independent and dependent ODOP variables, the material properties and a pass/fail flag for every constraint. The whole dataset can be regenerated exactly with the same code and seed. Four further datasets with the seeds 43 to 46 were generated in the same way for the repeated runs described in Section 2.7.

Table 3. Sampling ranges used in the Latin hypercube design

Variable	Compression	Extension	Torsion
Materials	17	9 (cold-wound steels)	15 (cold-wound)
Wire diameter *d* (mm), log-uniform	0.2–12 (cold-wound); 10–50 (hot-wound)	0.2–12	0.2–12

Spring index C	3–20	3–20	3–20
Number of coils	N_a: 2–25	N_t: 3–40	N_t: 2–25
Target FS_2, log-uniform	0.9–2.0	0.9–2.0	0.9–2.0
Load ratio	F_1/F_2: 0–0.8	$(F_1-F_i)/(F_2-F_i)$: 0–0.8	M_1/M_2: 0–0.8
Other variables	Used/available deflection: 0.4–1.0	Initial stress: −25% to 125% of the recommended range; end extension: 0 (70%) or 0–1.5D	Leg length: 0.5–3D; coil spacing: 0 (80%) or 0–0.5d

2.3 Forward models

Forward models predict the performance of a given design. They were trained on the valid designs. The inputs were the logarithms of the main dimensions and loads and the load ratios. One-hot codes were added for the material, the end type (heat treatment for torsion springs) and the life category. Four outputs were predicted for each spring type: the spring rate, FS_2, one type specific output and the weight. The type specific output was FS_{solid} for compression springs, the hook safety factor FS_{hook} for extension springs and the deflection angle at M_2 for torsion springs. All outputs were log-transformed and standardized. The data were split randomly into 80% training and 20% test sets. Because every design was sampled independently, the dataset contains no duplicate designs. Whether the random split nevertheless overestimates the generalization was examined separately (Section 2.7).

Three models were compared. The first model was linear regression on the log-transformed inputs. The second model was a random forest with 50 regression trees and a minimum leaf size of 2. All predictors were tried at each split, and one forest was trained for each output. The third model was an ANN with two fully connected hidden layers of 128 neurons and ReLU activations, implemented with the fitrnet function of MATLAB. One network was trained for each output. All inputs and outputs were standardized with the mean and the standard deviation of the training set. The weights were initialized with the Glorot scheme and the biases with zeros. Training minimized the mean squared error plus an L2 penalty ($\lambda = 10^{-4}$) with the Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) quasi-Newton algorithm. L-BFGS uses a line search, so no learning rate has to be chosen. A random 10% of the training data was held out for early stopping. Training stopped when the validation loss had not improved for 20 consecutive iterations, when the default gradient, loss or step tolerances of 10−6 were reached, or after at most 1,000 iterations. The architecture and the penalty were fixed before the experiments and were not tuned on the test data. Their influence was examined afterwards (Section 2.7). Accuracy was measured with the coefficient of determination (R2), R2 on the logarithmic scale and the Mean Absolute Percentage Error (MAPE).

2.4 Inverse design problem

For compression and extension springs, a design requirement consists of two forces, F_1 and F_2, and the spring lengths at these forces, L_1 and L_2. The material and the end type are also given. For torsion springs, the requirement consists of two moments, M_1 and M_2, the deflection angle θ_2 at M_2, the leg lengths and the material. The target spring rate k_{req} follows directly from the requirement. The first form below is used for compression springs, and the second form for torsion springs:

$$k_{req} = \frac{F_2 - F_1}{L_1 - L_2}, \quad k_{req} = \frac{M_2}{\theta_2} \tag{6}$$

Extension springs become longer under load. Therefore, the denominator is $L_2 - L_1$ for this type.

The inverse problem is to find d, OD and the number of coils, together with L_{free} for compression springs or F_i for extension springs. The spring must meet the requirement and all constraints. This problem has many solutions. According to Equation 2, a thicker wire can be balanced by a larger coil diameter or by more coils. This one-to-many nature is illustrated in Section 3.2.

2.5. Inverse design methods

Four methods were compared. M1, ANN (direct): An inverse ANN predicts all design variables from the requirement. It has the same structure as the forward ANN. Its inputs are the logarithms of the loads, the target rate and the lengths (or the moments and the angle). The load ratio and the one-hot codes of the categorical inputs are also used. It was trained only on feasible designs. M2, kNN + correction: The nearest feasible training design is found in the standardized requirement space. Its wire diameter and spring index are kept, and the other variables are corrected. M3, ANN + correction: The inverse ANN proposes only d and C, and the other variables are corrected. M4, proposed framework: The ANN proposal is combined with eight local variants and five kNN proposals. The local variants multiply d and C by 0.9, 1.0 or 1.1. This gives 14 candidates per requirement. All candidates are corrected and verified with ODOP. The lightest successful candidate is selected. If no candidate is successful, the requirement is counted as unsuccessful, and the corrected ANN proposal is returned as a fallback design.

Two variants of the framework were also evaluated. In M4-X, the three extra checks (fatigue, buckling and hook safety) are added to the success criterion, and the lightest candidate that also passes them is selected from the same 14 candidates. In M4-S, the wire diameter is restricted to the 76 standard metric wire sizes of ODOP (0.025–65 mm)[2]. The ANN proposal is rounded to the nearest standard size, and this size and its two neighbors in the list are combined with the factors 0.9, 1.0 and 1.1 for C. The wire diameters of the five kNN proposals are also rounded to the nearest standard size. This again gives 14 candidates, all with standard wire diameters. Because the correction is applied after the rounding, the rate and the load points are still met exactly.

The physics-based correction solves the rate equation for the number of coils. For compression and extension springs, Equation 2 gives

$$N_a = \frac{K_h G d^4}{8 D^3 k_{req}} \tag{7}$$

For compression springs, the free length is then set so that the spring has the length L_1 at the force F_1:

$$L_{free} = L_1 + \frac{F_1}{k_{req}} \quad (8)$$

For extension springs, the free length L_{free} follows from d, N_t and the end geometry. The initial tension is then computed so that the spring has the length L_1 at F_1:

$$F_i = F_1 - k_{req}(L_1 - L_{free}) \quad (9)$$

For torsion springs, Equation 4 is solved for the active coils. The contribution of the legs, ($L_{end,1}$ + $L_{end,2}$)/(3πD), is then subtracted to obtain the body coils. These steps make the rate and the load points exact by construction. The only exception is when a corrected value must be clipped to its allowed range. This mainly concerns the initial tension of extension springs, which must lie between zero and F_1. However, these steps do not ensure that the stresses and the other constraints are satisfied. Therefore, every candidate was verified with the ODOP engine.

A design was counted as *successful* when three conditions were met. First, it was feasible. Second, its spring rate error was below 1%. Third, its length (or angle) errors at the two load points were below 1% of the stroke. Test requirements were taken from the feasible designs in the test set, and the original design was used as a reference design. For compression springs, the test set contained only 713 feasible designs, so all of them were used. Together with 1,000 requirements each for extension and torsion springs, 2,713 requirements were used in total. The same success criterion and the same ODOP evaluation were used for all methods, including the genetic algorithm described in Section 2.6.

2.6 Genetic algorithm baseline

To compare the framework with a conventional optimization approach, a real-coded genetic algorithm (GA) was applied to the same requirements. Four variants were run. In *GA-direct*, the GA searches all design variables: d, C, N_a and L_{free}/L_1 for compression springs, d, C, N_t and F_i/F_1 for extension springs, and d, C and N_t for torsion springs. The required spring rate and load points are then equality constraints that the search must meet. In *GA + correction*, the GA searches only d and C, and the remaining variables are computed with the same physics-based correction as in the framework. *GA + correction (extended)* uses the extended success criterion of M4-X. Finally, in the *framework + GA refinement*, the initial population of a short GA run consists of the 14 framework candidates and six random designs.

The variables were searched on a logarithmic scale within the sampling ranges of the dataset (d = 0.2–12 mm for cold-wound and 10–50 mm for hot-wound materials) and within the ODOP limits $4 \le C \le 25$. The additional variables of GA-direct were bounded by $1 \le N_a \le 40$ and $1 \le L_{free}/L_1 \le 3$ (compression), $1 \le N_t \le 90$ and $0 \le F_i/F_1 \le 1$ (extension), and $1 \le N_t \le 100$ (torsion). Constraints were handled with the feasibility rules of Deb[27]: a successful design is always preferred to an unsuccessful one, successful designs are ranked by their weight, and unsuccessful designs are ranked by the sum of the ODOP constraint violation and the rate and position errors. Binary tournament selection, blend crossover (BLX-0.5)[28] with a probability of 0.9, Gaussian mutation with a probability of $1/n_v$ per variable (n_v is the number of variables) and a standard deviation of 10% of the variable range, and elitism of the two best designs were used. With a population of 40 and 50 generations, each GA run used 1,902 ODOP evaluations per requirement. The refinement run used a population of 20 and 10 generations, which gives 182 evaluations including the 14 framework candidates. The GA was applied to 300 randomly selected test requirements per spring type, and the framework

results for the same requirements were used for the comparison. In each generation, the populations of all requirements were evaluated with ODOP in one batch, as in the framework.

2.7 Robustness, generalization and sensitivity tests

Repeated runs. The whole procedure was repeated with four further random seeds (43–46). For each seed, a new dataset was generated, a new 80/20 split was drawn and new networks were trained, so that the five runs are fully independent. The random forest was omitted from the repeated runs because it was the weakest forward model.

Similarity between training and test data. To check whether the test requirements were unusually close to training requirements, the Euclidean distance from each test requirement to its nearest training requirement was computed in the standardized input space of the inverse model. It was compared with the nearest-neighbor distances within the training set (leave-one-out).

Unseen materials. A leave-one-material-out (LOMO) test was performed. In the inverse ANN and the kNN search, the one-hot code of the material was replaced by physical material descriptors: the logarithms of the shear (or elastic) modulus and the density, the hot-winding factor K_h, the two coefficients a and b of the size-dependent tensile strength $S_{ut} = a + b \log_{10} d$ used in ODOP, and the ratios of the allowable stresses to the tensile strength. For each material in turn, all designs of that material were removed from the training data, the models were retrained, and up to 200 test requirements of the removed material were solved. The results were compared with the same descriptor-based model trained on all materials.

Hyperparameters. The inverse ANN was retrained with hidden layers of [64 64], [128 128], [256 256] and [128 128 128] neurons and with L2 penalties of 10^{-5}, 10^{-4} and 10^{-3}.

Sensitivity analysis. The successful designs of M4 were perturbed one parameter at a time: the wire diameter d (with D fixed), the mean coil diameter D (with d fixed), the number of active coils N_a, the maximum load F_2 (or M_2), the shear (or elastic) modulus and the tensile strength. Each perturbed design was evaluated again with ODOP. The material properties were changed by scaling the tabulated values in the ODOP driver; the equations themselves were not changed. Local elasticities, $e = (\Delta Y/Y)/(\Delta x/x)$, were computed from perturbations of ± 1%.

2.8 Implementation

All models were implemented in MATLAB R2025b with the Statistics and Machine Learning Toolbox. The ODOP engine was bundled into a single JavaScript file and called from MATLAB through Node.js (v24.21). The candidate designs were written to a file, verified by ODOP in one batch per spring type and read back. The reported time per requirement is the wall-clock time of the online stage divided by the number of requirements. It includes the ANN prediction, the kNN search, the physics-based correction and the ODOP verification of all 14 candidates, including the start of the Node.js process and the file input and output. It does not include the offline stage (dataset generation and training). All computations were run on a workstation with an Intel Xeon E5-1650 processor (3.2 GHz, 6 cores, 12 threads), 16 GB of RAM and Windows 10 Pro, without a GPU. A fixed random seed makes all results reproducible, and the code is available from the corresponding author upon reasonable request.

3. SIMULATION RESULTS

3.1 Dataset

Table 4 summarizes the dataset. Almost all designs were valid. However, the share of feasible designs differed strongly between the spring types. Only 7.2% of the compression springs were feasible. In contrast, 46.8% of the extension springs and 63.3% of the torsion springs were feasible. Figure 2 shows the distributions of d, C and FS2. Feasible designs cover the whole wire diameter range. The feasible spring index starts at 4 because of the ODOP limit. Figure 3 explains the low feasible share of compression springs. Three constraints act together: FS2 ≤ 1.5, FSsolid ≥ 1 and a used deflection of at most 90%. Only about 40% of the compression designs satisfied the solid safety factor. For extension springs, the hook safety factor was the most limiting extra check, and only about 20% of the designs passed it.

Table 4. Summary of the generated dataset

Spring type	Samples	Valid (%)	Feasible (%)	Feasible-extended (%)	Materials	End types	Default constraints
Compression	50,000	100.0	7.2	3.5	17	10	9
Extension	50,000	99.9	46.8	12.7	9	5	10
Torsion	50,000	100.0	63.3	37.0	15	1	8

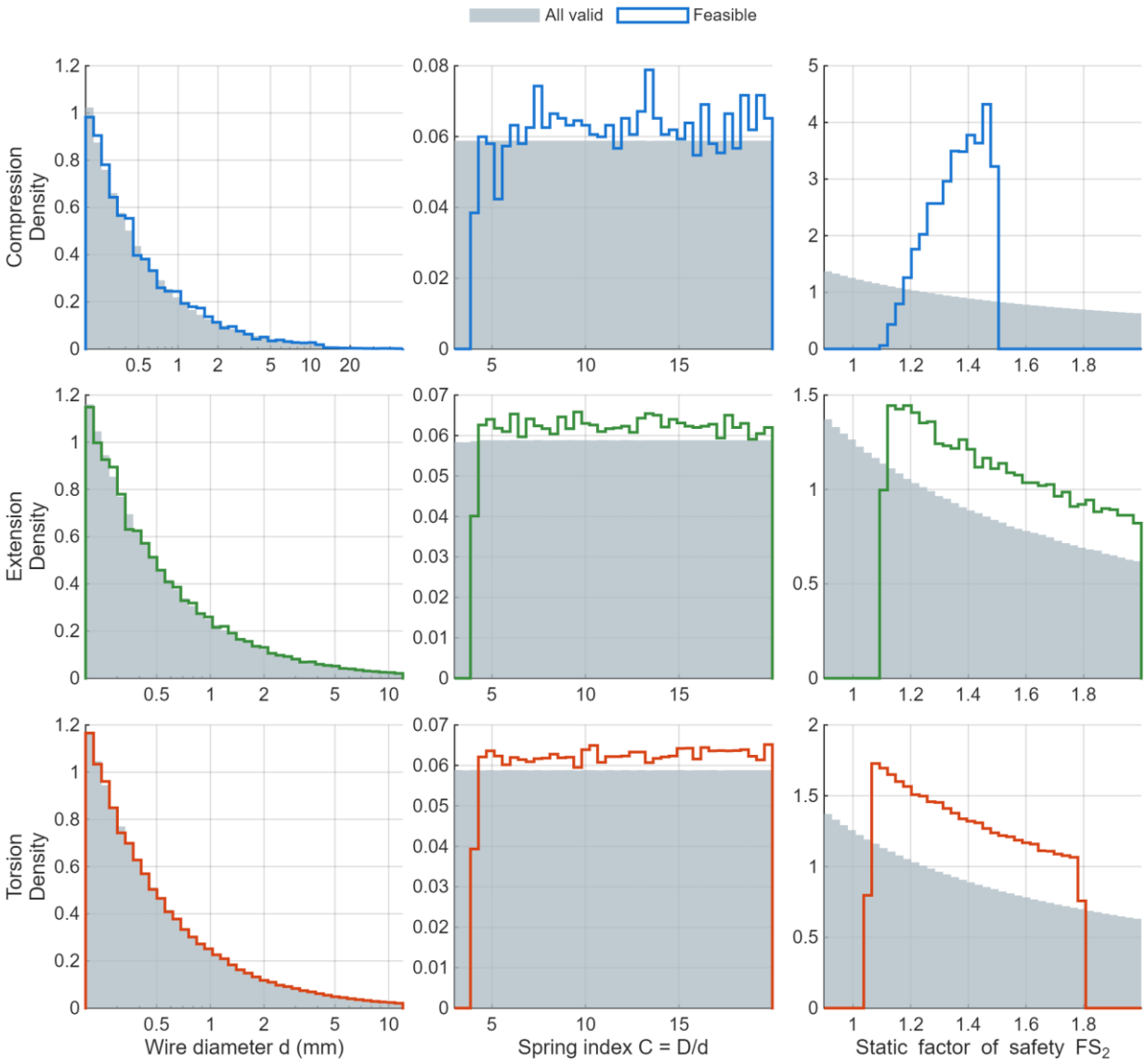

Figure 2. Distributions of the wire diameter, the spring index and the static safety factor for all valid designs (grey) and for feasible designs (colored line)

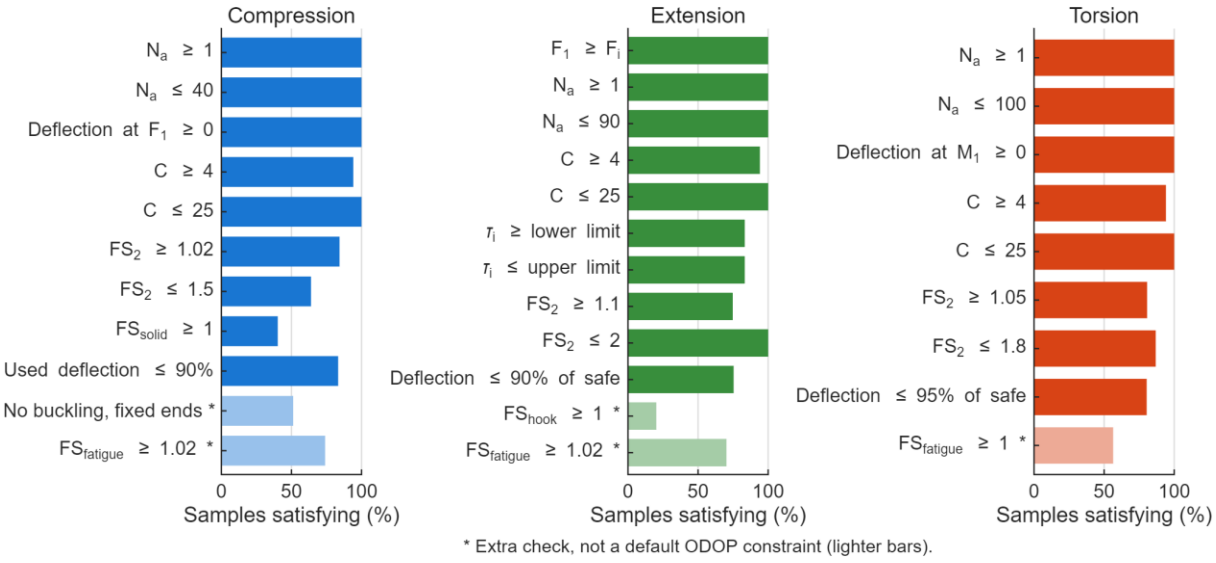

Figure 3. Share of the samples that satisfy each constraint. Lighter bars marked with an asterisk are extra checks that are not part of the default ODOP constraints

3.2 One-to-many nature of the inverse problem

Figure 4 shows all valid compression springs whose rate is within ± 15% of 0.188 N/mm and whose second force is within ± 15% of 11.5 N. There were 144 such designs, and 23 of them were feasible. The feasible designs had wire diameters from about 0.7 to 1.3 mm and mean diameters from about 9 to 23 mm. Their weights differed by a factor of almost ten. Hence, the same requirement can be met by very different springs. A direct inverse model trained with a squared error loss tends to average these solutions. The averaged design may not be feasible at all.

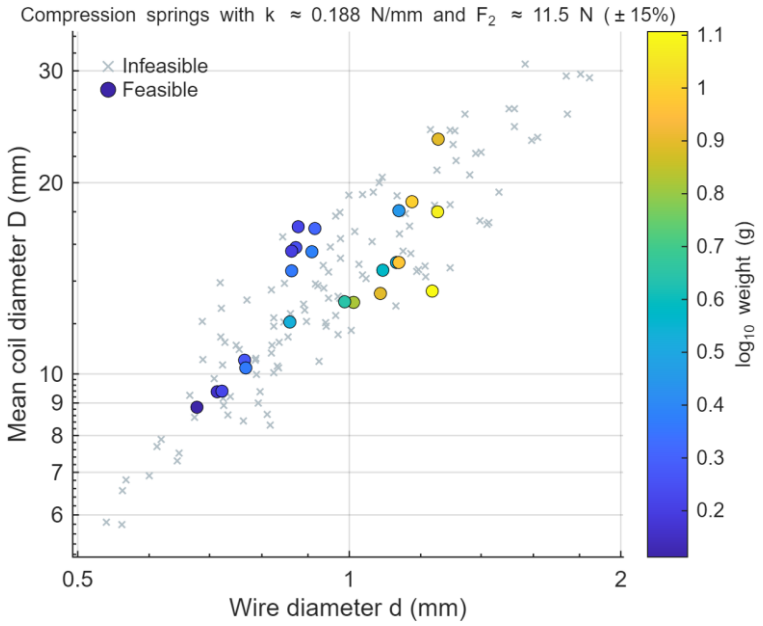

Figure 4. Valid compression springs with nearly the same rate (k ≈ 0.188 N/mm) and the same second force (F2 ≈ 11.5 N), both within ±15%. Colors show the weight of the feasible designs

3.3 Forward models

Table 5 compares the forward models. The ANN gave the best results for the safety factors. Its R2 values on the logarithmic scale were at least 0.996 for FS2, FSsolid and FShook. Its MAPE values for these outputs were between 0.49% and 1.24%. Linear regression was much weaker for the safety factors, with MAPE values between 4.5% and 13.5%. On the other hand, linear regression was very accurate for the spring rate, the weight and the deflection angle. Its R2 values on the logarithmic scale were at least 0.998, and its MAPE for the weight was below 3%. This is expected. These outputs are power laws of the inputs, so they become linear after a logarithmic transform. The random forest was the weakest model for all outputs. Over the five independent runs (Section 2.7), the standard deviation of R2 on the logarithmic scale was below 0.001 for all ANN outputs (Appendix Table A1).

Table 5. Accuracy of the forward models on the test set

Spring type	Output	R^2 (log scale)			MAPE (%)		
		LR	RF	ANN	LR	RF	ANN
Compression	Spring rate	0.9984	0.9858	0.9993	6.15	21.30	4.55
Compression	FS2	0.9051	0.5293	0.9976	5.33	13.12	0.86
Compression	FS_solid	0.7395	0.5664	0.9961	13.53	18.63	1.24
Compression	Weight	0.9999	0.9988	0.9995	2.98	11.37	7.86
Extension	Spring rate	0.9996	0.9882	0.9994	3.35	16.46	4.03

Extension	FS2	0.9341	0.7490	0.9990	4.49	9.22	0.54
Extension	FS_hook	0.9312	0.7762	0.9993	5.92	11.55	0.63
Extension	Weight	0.9999	0.9997	0.9993	2.36	5.10	7.68
Torsion	Spring rate	1.0000	0.9969	0.9995	1.38	15.94	6.67
Torsion	FS2	0.8739	0.8455	0.9992	6.46	7.29	0.49
Torsion	Deflection angle θ2	0.9996	0.9257	0.9994	1.38	20.32	1.78
Torsion	Weight	0.9999	0.9991	0.9992	2.40	7.89	8.05

Figure 5 shows the parity plots of the ANN. The points lie close to the diagonal over five orders of magnitude for the spring rate. Figure 6 shows the training curves of the networks for the spring rate (forward model) and the wire diameter (inverse model). All of these networks were stopped by the early stopping rule after 71 to 296 iterations.

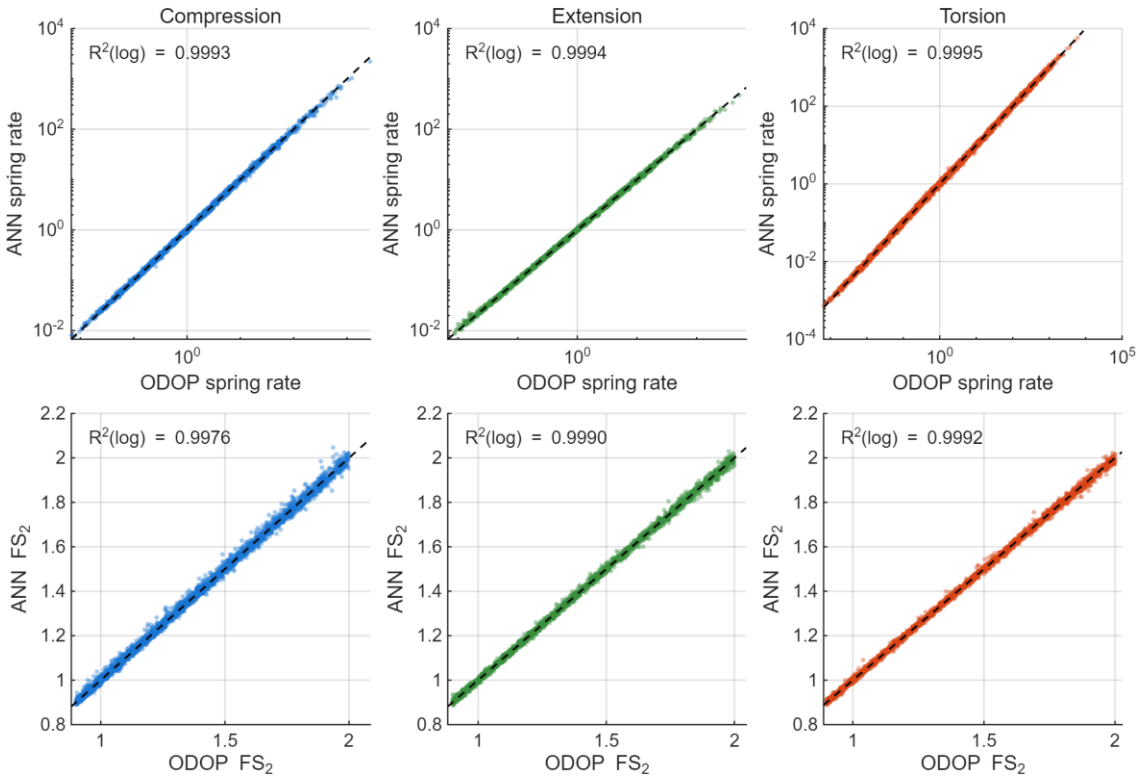

Figure 5. Parity plots of the forward ANN on the test set for the spring rate (top) and the static safety factor FS_2 (bottom)

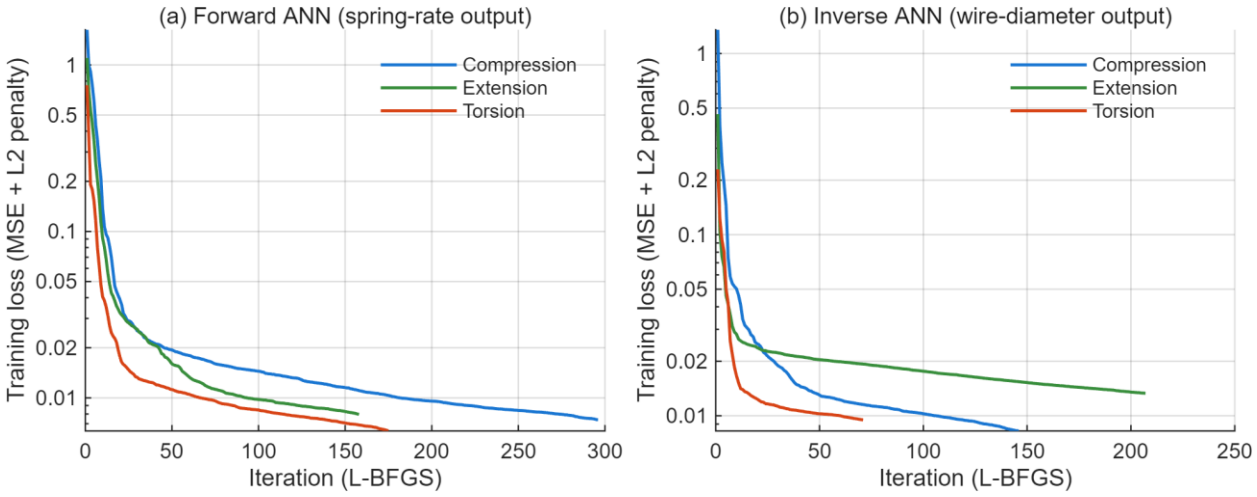

Figure 6. Training loss of (a) the forward ANN for the spring rate output and (b) the inverse ANN for the wire diameter output. The plotted loss is the L-BFGS objective function, which includes the L2 penalty in addition to the MSE

3.4 Inverse design

Table 6 and Figure 7 compare the four inverse design methods. The direct ANN (M1) almost never produced a successful design. Its success rate was 0% for compression, 0.3% for extension and 5.8% for torsion springs. Many of its designs were feasible, especially for torsion springs (98.8%). However, they did not meet the requirement. The median spring rate error of M1 was 14.3%, 11.7% and 6.7% for the three types, and its median position error was between 11.5% and 28.6% of the stroke.

Table 6. Results of the inverse design methods on the test requirements. All values were obtained with the ODOP engine

Spring type	Method	Success (%)	Feasible (%)	Feasible -ext. (%)	Median rate error (%)	Median position error (%)	Median weight ratio	Time (ms)
Compression	M1 ANN (direct)	0.0	22.2	8.6	14.3	27.3	–	0.12
	M2 kNN + correction	2.8	2.8	1.0	$<10^{-8}$	0	0.989	0.62
	M3 ANN + correction	44.7	44.7	17.1	$<10^{-8}$	0	1.007	0.13
	M4 Proposed framework	**89.8**	**89.8**	**33.7**	**$<10^{-8}$**	**0**	**0.912**	**2.25**

	M4-S Framework, standard wire	89.3	89.3	33.9	$<10^{-8}$	0	0.926	2.22
Extension	M1 ANN (direct)	0.3	88.0	24.8	11.7	28.6	0.759	0.09
	M2 kNN + correction	17.0	20.0	6.4	$<10^{-8}$	0	0.937	0.18
	M3 ANN + correction	50.5	52.4	15.5	$<10^{-8}$	0	1.053	0.09
	M4 Proposed framework	**90.3**	**90.5**	**17.1**	**$<10^{-8}$**	**0**	**0.880**	**1.26**
	M4-S Framework, standard wire	88.1	88.7	16.6	$<10^{-8}$	0	0.886	1.24
Torsion	M1 ANN (direct)	5.8	98.8	61.0	6.7	11.5	1.045	0.07
	M2 kNN + correction	31.9	31.9	19.0	$<10^{-8}$	$<10^{-8}$	0.968	0.16
	M3 ANN + correction	98.8	98.8	61.0	$<10^{-8}$	$<10^{-8}$	0.999	0.07
	M4 Proposed framework	**100.0**	**100.0**	**40.2**	**$<10^{-8}$**	**$<10^{-8}$**	**0.739**	**1.07**
	M4-S Framework, standard wire	100.0	100.0	46.7	$<10^{-8}$	$<10^{-8}$	0.772	1.06

M1–M4: Inverse design methods (Section 2.5); M4-S: Framework restricted to standard wire diameters. Success: Feasible design whose spring rate error and load-point position errors are both below 1%. Feasible-ext.: Feasible design that also passes the fatigue, buckling and hook checks. Position error: Larger length (or angle) error at the two load points, as a percentage of the stroke. Weight ratio: Weight of the successful design divided by the weight of the reference design. Time: Wall-clock time of the online stage per requirement, including the ODOP verification of all candidates (Section 2.8). For requirements without a successful candidate, the feasibility values refer to the fallback design.

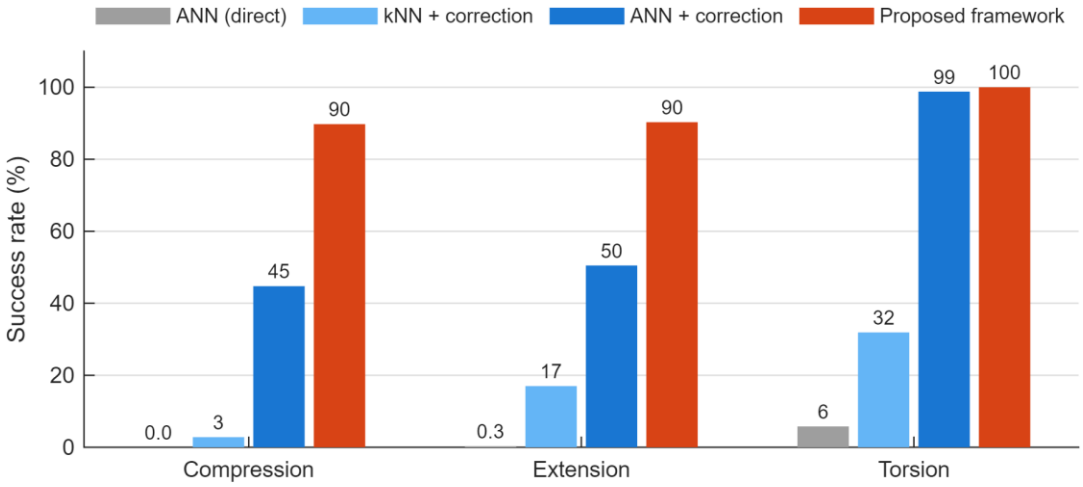

Figure 7. Success rates of the four inverse design methods

The physics-based correction removed the rate and position errors in most cases. For M2, M3 and M4, the median rate error was below 10^{-8}%, which is the level of numerical precision. Figure 8 shows this result clearly. For compression and torsion springs, the curves of the corrected methods reach 100% at very small errors. For extension springs, the initial tension must stay between zero and F_1. When the corrected initial tension falls outside this range, it is clipped, and the load points are no longer met exactly. The practical consequences of this clipping are analyzed in Section 3.5.

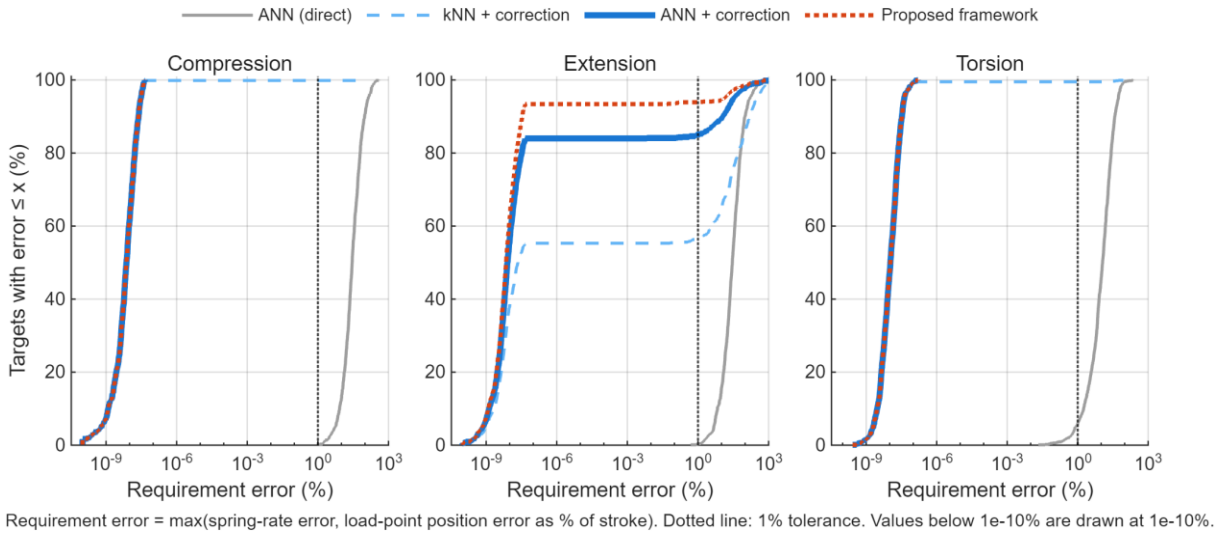

Figure 8. Cumulative distribution of the requirement error (the larger of the spring rate error and the load-point position error). The M3 curve is drawn with a thicker line because it coincides with the M4 curve for compression and torsion springs

In Figure 8, the curves of M3 and M4 coincide for compression and torsion springs. After the correction, feasibility became the main issue. The kNN method (M2) gave few successful designs, with 2.8%, 17.0% and 31.9%. The wire diameter and the spring index of the nearest neighbor often led to high stresses for the new requirement. The ANN with correction (M3) performed much better and reached 44.7%, 50.5% and 98.8%. The proposed framework (M4) gave the best results in all cases. It found a successful design for 89.8% of the compression, 90.3% of the extension and 100% of the torsion spring requirements. Most of the selected designs came from the local variants (67.9% to 75.3%). The ANN proposal itself was selected in 0% to 22.0% of the cases, and the kNN proposals in 2.7% to 32.1% of the cases.

Figure 9 compares the weights of the framework designs with the weights of the reference designs. The framework designs were lighter in 71.6% of the compression, 70.2% of the extension and 78.9% of the torsion cases. The median weight ratio was 0.912, 0.880 and 0.739, respectively. These ratios refer to random feasible reference designs and not to optimal designs; the distance of the framework designs from optimized designs is quantified in Section 3.7. The time per requirement was between 1.1 and 2.3 ms, including the ODOP verification of all 14 candidates (Section 3.10).

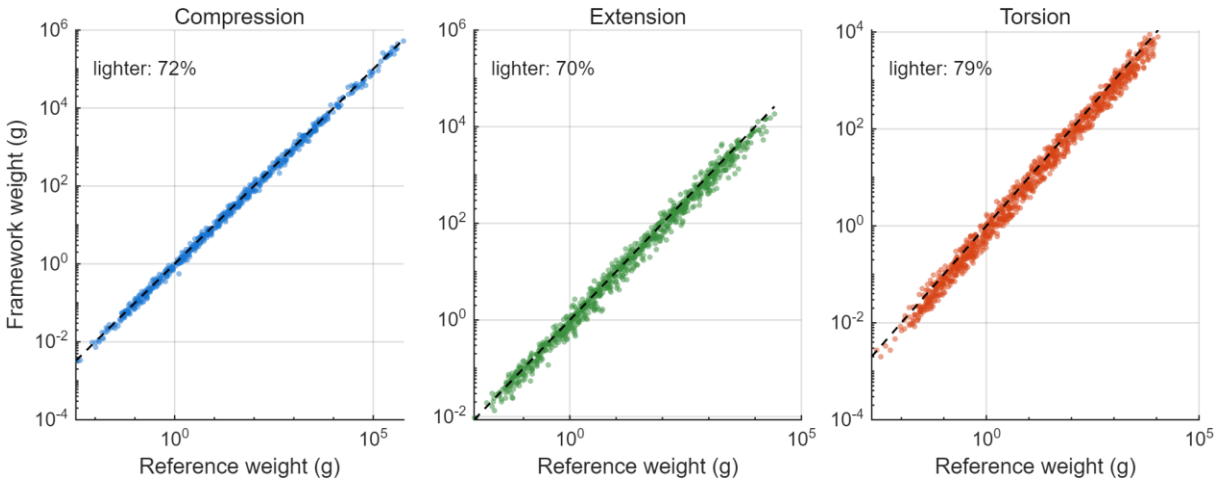

Figure 9. Weight of the designs selected by the framework versus the weight of the reference designs (successful cases only)

When the wire diameter was restricted to standard sizes (M4-S), the success rate changed only slightly, to 89.3%, 88.1% and 100% (Table 6). For the requirements solved by both variants, the standard-wire designs were heavier by a median of 1.0% for compression, 1.1% for extension and 5.7% for torsion springs. Discrete wire sizes can therefore be handled simply by rounding the proposals before the physics-based correction, because the correction then adapts the number of coils and the free length or the initial tension to the rounded diameter.

Table 7 shows the constraint violation rates of M3, M4 and M4-X for selected checks. The framework

reduced the violations of the default ODOP constraints sharply. For compression springs, the violations of the upper FS2 limit dropped from 20.2% to 4.6%. The violations of the solid safety factor dropped from 27.9% to 3.6%. For extension springs, the violations of the lower initial stress limit dropped from 34.8% to 6.9%. However, the extra checks, which are not part of the default ODOP constraints and were not used in the selection rule of M4, did not improve. The fatigue check failed for 16.3% of the compression, 20.2% of the extension and 59.8% of the torsion designs. About half of the compression designs showed a buckling tendency with fixed ends. These results and the effect of adding the extra checks to the selection rule (M4-X) are analyzed in Section 3.6.

Table 7. Constraint violation rates (%) of M3, M4 and M4-X for selected checks

Check	Compression			Extension			Torsion		
	M3	**M4**	**M4-X**	**M3**	**M4**	**M4-X**	**M3**	**M4**	**M4-X**
FS_2 lower limit	2.1	1.0	1.7	1.5	0.5	1.4	0.5	0.0	0.1
FS_2 upper limit	20.2	4.6	11.8	2.7	1.5	2.3	0.7	0.0	0.1
$FS_{solid} \geq 1$	27.9	3.6	16.0	–	–	–	–	–	–
Used deflection ≤ 90% of available	17.4	5.9	9.1	–	–	–	–	–	–
Deflection ≤ safe deflection limit	–	–	–	1.3	0.4	1.2	0.5	0.0	0.1
Initial stress lower limit	–	–	–	34.8	6.9	23.8	–	–	–
Initial stress upper limit	–	–	–	10.7	1.9	9.8	–	–	–
Hook safety factor ≥ 1*	–	–	–	71.3	78.8	60.7	–	–	–
Fatigue safety factor, all requirements*	15.3	16.3	10.1	9.0	20.2	9.0	38.3	59.8	18.2
Fatigue safety factor, cyclic life only*	20.3	21.6	13.4	12.0	26.9	12.0	52.9	82.6	25.1
No buckling, fixed ends*	47.3	52.2	41.7	–	–	–	–	–	–

* Extra check that is not part of the default ODOP constraints; – not applicable. The fatigue check applies only to cyclic life categories; the row "cyclic life only" gives the violation rate among these requirements. For requirements without a successful candidate, the values refer to the fallback design (the corrected ANN proposal), which explains why M4-X shows more violations of the default constraints than M4.

Table 8 shows one example for each spring type. In the compression example, the direct ANN gave a feasible

design, but its position error was 5.9%. The kNN design met the load points but violated at least one constraint. M3 and M4 selected the same design. It met all requirements and was 16% lighter than the reference design. In the extension example, the M3 design was light but infeasible. The framework selected a slightly heavier design that satisfied all constraints. In the torsion example, the framework design was 22% lighter than the reference design.

Table 8. Example designs for one requirement of each spring type

Design	d (mm)	OD (mm)	N_t	Rate*	FS_2	Weight (g)	Position error (%)	Feasible
Compression: F_1 = 38.07 N at L_1 = 162.2 mm; F_2 = 102.1 N at L_2 = 62.04 mm; k_{req} = 0.6398 N/mm; MUSIC WIRE; PigtailClosed&Ground								
Reference	2.631	44.15	12.38	0.6398	1.400	62.16	0	Yes
M1	2.422	37.67	14.54	0.6208	1.295	53.48	5.9	Yes
M2	2.016	24.53	24.41	0.6398	1.173	41.29	0	No
M3	2.422	37.67	14.17	0.6398	1.295	52.01	0	Yes
M4	2.422	37.67	14.17	0.6398	1.295	52.01	0	Yes
Extension: F_1 = 10.05 N at L_1 = 49.89 mm; F_2 = 20.69 N at L_2 = 72.36 mm; k_{req} = 0.4733 N/mm; CHROME SILICON; 75% Loop								
Reference	0.910	8.92	27.68	0.4733	1.588	3.82	0	Yes
M1	0.982	11.76	15.69	0.4632	1.522	3.57	14.0	Yes
M2	1.079	11.81	22.82	0.4733	1.989	6.02	0	No
M3	0.982	11.76	15.35	0.4733	1.522	3.50	0	No
M4	0.982	10.69	21.13	0.4733	1.668	4.20	0	Yes
Torsion: M_1 = 12.73 N-mm; M_2 = 210.7 N-mm; θ_2 = 160.6 deg; k_{req} = 1.312 N-mm/deg; INCONEL X-750								
Reference	1.481	14.75	14.81	1.3118	1.526	9.49	0	Yes

M1	1.417	14.98	12.09	1.3120	1.341	7.37	0.021	Yes
M2	1.008	10.54	3.99	1.3118	0.492	1.12	$<10^{-8}$	No
M3	1.417	14.98	12.09	1.3118	1.341	7.37	$<10^{-8}$	Yes
M4	1.417	16.34	10.99	1.3118	1.341	7.37	$<10^{-8}$	Yes

* Rate unit: N/mm for compression and extension springs, N·mm/deg for torsion springs

3.5 Analysis of unsuccessful cases

The framework failed for 73 compression (10.2%) and 97 extension (9.7%) requirements and for no torsion requirement. In all of these cases, at least one of the 14 candidates met the spring rate and both load points within 1%. The failures were therefore caused only by constraint violations and not by requirement errors. Moreover, the GA with the same correction (Section 3.7) found a successful design for every requirement of its 300-requirement subsets, including the 31 compression and 30 extension requirements of these subsets for which the framework failed. Feasible designs thus existed, and the failures are a limitation of the small candidate set.

For compression springs, the failures came from the interaction of three constraints that depend on the solid length. The ANN proposals of the failed requirements violated the limit on the used deflection (57.5%), the upper FS2 limit (45.2%) and the solid safety factor (35.6%). The ANN proposal was not worse for these requirements than for the successful ones: its median deviation from the reference design in d or C was 11.8% in both groups, and the reference design lay outside the ±10% box of the local variants in about 54% of both groups. Rather, these requirements leave only a narrow band of feasible designs between springs that become solid too early and springs whose stress at F2 is so low that the upper FS2 limit is exceeded. The failure rate differed between materials (from 2.4% for SAE 9250 to 22.6% for Monel) and end types (from 4.5% to 14.5%), but not systematically between life categories.

For extension springs, the dominant cause was the lower limit of the initial stress (71.1% of the failed ANN proposals), followed by its upper limit (19.6%) and the upper FS2 limit (15.5%). Here, the ANN proposals of the failed requirements were further from the reference design (median deviation of 23.2% compared with 14.1% for the successful requirements), and the reference design lay outside the local box in 81.4% of the failed and 62.8% of the successful cases. Since the allowed initial stress range depends on the spring index, a proposal with an unsuitable index leads directly to an infeasible initial tension. A wider set of local variants or a short refinement (Section 3.7) removes most of these failures.

The clipping of the initial tension mainly affects the single ANN proposal (M3). It was needed for 16.0% of the M3 proposals, and in 71.9% of these cases the corrected initial tension would have been negative. Only 1.3% of the clipped M3 proposals were successful, compared with 59.9% of the unclipped ones, and their median position error was 18.3% of the

stroke. In the framework, however, clipping occurred in only 0.3% of the successful selected designs, and in none of the failed requirements were all candidates clipped, because the other candidates provide d–C combinations for which the required initial tension lies between zero and F1. Physically, a negative initial tension means that the free length of the proposed spring is too short: even without initial tension, the spring would carry more than F1 at the length L1. Besides switching to another candidate, as the framework does, a designer can lengthen the ends (end extension or hook length), which increases the free length without changing the spring rate. In this study, the end extension was part of the requirement and was therefore kept fixed.

3.6 Fatigue, buckling and hook checks

The extra checks are demanding even for the reference designs: only 47.3% of the compression, 26.5% of the extension and 59.8% of the torsion test requirements had a feasible-extended reference design. Table 9 shows the success rates when the extra checks are included in the success criterion. The default selection rule of M4 picks the lightest feasible candidate. A lighter spring works at a higher stress, so this rule moves the designs towards the lower limit of FS2 and away from the fatigue limit. This is why M4 passed the extra checks less often than M3 for torsion springs (40.2% compared with 61.0%).

The fatigue failures depend clearly on the life category (Appendix Table A2). No fatigue failure occurred for static loading, for which the check does not apply. For the torsion springs selected by M4, the violation rate grew from 74.7% at 105 cycles to 85.0% at 106 and 88.4% at 107 cycles. For compression springs at 107 cycles, shot peening reduced the violation rate from 50.6% to 21.9%. The reason is that the static allowable stress, which controls FS2, is much higher than the endurance strength of the material. A design that is selected only with the static constraints, with FS2 close to its lower limit, cannot be expected to survive millions of cycles.

Table 9. Success rates when the fatigue, buckling and hook checks are part of the success criterion

Spring type (requirements with feasible-extended reference)	Method	Success, all requirements (%)	Success, requirements with feasible-extended reference (%)	Median weight ratio
Compression (337 of 713)	M3 ANN + correction	17.1	30.3	1.000
	M4 Proposed framework	33.7	63.8	0.935
	M4-X Framework, extra checks	**46.6**	**78.6**	**0.977**
Extension (265 of 1000)	M3 ANN + correction	14.7	41.1	1.111
	M4 Proposed framework	17.1	55.1	0.929
	M4-X Framework, extra checks	**33.7**	**80.8**	**1.046**

Torsion (598 of 1000)	M3 ANN + correction	61.0	86.3	1.033
	M4 Proposed framework	40.2	62.7	0.750
	M4-X Framework, extra checks	**81.7**	**96.3**	**0.954**

Success: Feasible-extended design whose spring rate and load-point position errors are below 1%. M4-X uses this criterion in the selection; M3 and M4 are evaluated with it after the selection. Weight ratio: median over the successful requirements.

When the extra checks were added to the selection rule (M4-X), the success rate with extra checks rose to 46.6%, 33.7% and 81.7% for all requirements and to 78.6%, 80.8% and 96.3% for the requirements with a feasible-extended reference design (Table 9). The fatigue violation rate of torsion springs in cyclic life categories fell from 82.6% to 25.1% (Table 7). The price is weight: The median weight ratio increased from 0.912 to 0.977, from 0.880 to 1.046 and from 0.739 to 0.954. The GA with the extended criterion (Section 3.7) reached 85.0%, 49.3% and 93.0% on its subsets, compared with 46.3%, 34.7% and 80.0% for M4-X on the same requirements. For many requirements, a feasible-extended design therefore exists but is not among the 14 candidates. When fatigue or buckling is critical, the extra checks should be part of the selection rule, and the candidate set should be enlarged or refined.

The three checks also differ in their practical meaning. The buckling of a compression spring with fixed ends depends on its slenderness, L_{free}/D, and on its relative deflection [1]. A spring with a buckling tendency is usually acceptable when it works in a bore or over a guide rod, which is common in machines, and ODOP therefore treats this check as optional [2]. The hook safety factor depends on the bend radius of the hook, which was not a design variable in this study; the default hook geometry of ODOP was used for all designs. In practice, a larger bend radius or another end type can raise the hook safety factor without changing the spring body [1]. The fatigue check, on the other hand, cannot be ignored when a spring is loaded cyclically. For such applications, the extended criterion should be used, or the lower FS_2 limit should be raised.

3.7 Comparison with a genetic algorithm

Table 10 and Figure 10 compare the framework with the four GA variants on 300 requirements per spring type. Three findings stand out. First, the physics-based correction is as important for the GA as it is for the framework. Without it (GA-direct), the GA had to meet the spring rate and both load points through the penalty function. After 1,902 evaluations per requirement, it succeeded for only 23.7% of the compression and 37.0% of the extension requirements. Only for torsion springs, where the load points follow from the rate alone because the deflection is measured from the free position, did it reach 98.7%. With the correction, the GA succeeded for all requirements.

Second, the GA with correction found much lighter designs. The median weight of its designs was 71.4%, 57.3% and 58.6% of the reference weight, compared with 89.3%, 89.7% and 76.6% for the framework on the same requirements. In other words, the framework designs were a median of 27%, 47% and 26% heavier than the GA designs. The framework therefore does

not find optimal designs; it selects the lightest of 14 candidates. Third, the GA needed 1,902 evaluations and 218–244 ms per requirement, whereas the framework needed 14 evaluations and 1.1–2.3 ms, that is, about 100 to 200 times less time.

The two approaches are complementary. When the 14 framework candidates were used as the initial population of a short GA run with 182 evaluations, the success rate rose to 98.3%, 99.0% and 100%, and the median weight came within 12%, 11% and 2% of the full GA result (Table 10). With the same budget, the GA with correction started from a random population reached only 79.0%, 92.7% and 100% success, with heavier designs (Figure 10). The framework plus refinement took 24–28 ms per requirement. A designer can thus obtain a feasible design almost instantly and refine it only when the weight matters.

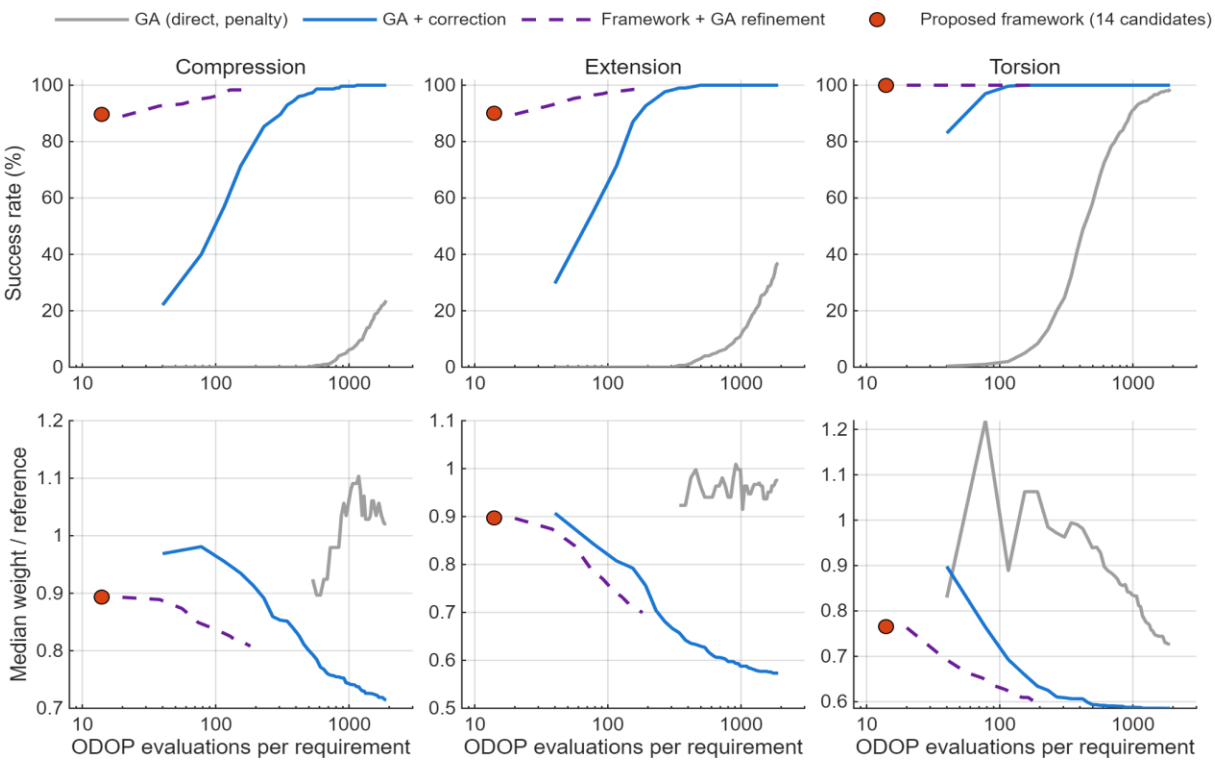

Figure 10. Success rate (top) and median weight relative to the reference design (bottom) versus the number of ODOP evaluations per requirement for the GA variants and the proposed framework (300 requirements per spring type). The weight ratio is computed over the successful requirements; for GA-direct it is based on few successful requirements at small budgets

Table 10. Comparison of the framework with a genetic algorithm on 300 requirements per spring type

Spring type	Method	Evaluations per requirement	Success (%)	Median weight ratio (*vs* reference)	Median weight ratio (*vs* GA + correction)	Time (ms)
Compression	M4 Proposed framework	14	89.7	0.893	1.266	2.3
	M4-X Framework, extra checks	14	46.3	0.941	1.198	2.3
	GA-direct	1,902	23.7	1.020	1.409	245.1
	GA + correction	1,902	100.0	0.714	1.000	243.5
	GA + correction (extended)	1,902	85.0	0.881	1.000	240.3
	Framework + GA refinement	**182**	**98.3**	**0.808**	**1.120**	**27.7**
Extension	M4 Proposed framework	14	90.0	0.897	1.470	1.3
	M4-X Framework, extra checks	14	34.7	1.077	1.131	1.3
	GA-direct	1,902	37.0	0.975	1.659	232.6
	GA + correction	1,902	100.0	0.573	1.000	231.9
	GA + correction (extended)	1,902	49.3	0.998	1.000	233.5
	Framework + GA refinement	**182**	**99.0**	**0.699**	**1.112**	**28.4**
Torsion	M4 Proposed framework	14	100.0	0.766	1.261	1.1
	M4-X Framework, extra checks	14	80.0	1.001	1.229	1.1
	GA-direct	1,902	98.7	0.727	1.188	213.4
	GA + correction	1,902	100.0	0.586	1.000	218.0
	GA + correction (extended)	1,902	93.0	0.817	1.000	226.1
	Framework + GA refinement	**182**	**100.0**	**0.600**	**1.024**	**23.8**

Weight ratio vs reference: Median over the successful requirements. Weight ratio vs GA + correction: Median over the requirements solved by both methods; methods with the extended criterion are compared with GA + correction (extended). Time: Wall-clock time per requirement (Section 2.8).

3.8 Repeated runs and generalization

Table 11 summarizes the five independent runs. The success rate of the framework was 87.3 ± 2.1%, 89.4 ± 0.9% and 100 ± 0% (mean ± standard deviation) for compression, extension and torsion springs, and the standard deviation of its median weight ratio was at most 0.011. The ranking M1 < M2 < M3 < M4 was the same in all runs and for all spring types. The largest variation occurred for compression springs, which have the smallest number of feasible training designs (about 2,900) and test requirements (667 to 730).

The random split did not place near-copies of training designs in the test set. No test requirement had a training requirement at a distance below 10−6. The median distance from a test requirement to its nearest training requirement (0.877, 0.415 and 0.527 in standardized units) was about the same as the median nearest-neighbor distance within the training set (0.904, 0.438 and 0.529). Hence, the test requirements were not closer to the training data than the training designs were to each other. The success of the framework decreased only moderately with this distance: from 95.5% in the nearest quartile to 83.1% in the farthest quartile for compression springs and from 92.4% to 86.8% for extension springs, and it was 100% in all quartiles for torsion springs.

A stricter test is a material that was not seen in training (Table 12). With material descriptors instead of one-hot codes, the framework reached 91.2%, 90.1% and 100% for seen materials, which is close to the one-hot model. For unseen materials, the success rate fell to 81.3%, 86.7% and 99.5% on average, and it stayed above 80% for 36 of the 41 combinations of material and spring type (Appendix Table A3). The lowest values were found for titanium (30.2% for compression springs) and for the hot-wound 5160H steel (32.4%). In both cases, a key property lies outside the range of the remaining materials: Titanium has the lowest density and the highest ratio of tensile strength to shear modulus, and 5160H has the lowest hot-winding factor (Kh = 0.91). When 5160H-CG (Kh = 0.96, inside the range) was left out instead, the success rate was 86.5%. A lower value was also found for 17-7 stainless steel in extension springs (60.0%), for which only eight other materials were available. The physics-based correction uses the true properties of the new material, so the rate and the load points remain exact; the drop comes from the ANN and kNN proposals, which must extrapolate. The framework can therefore be applied to a new material whose properties lie within the range of the training materials. Otherwise, designs of the new material should be added to the training data, which takes only a few seconds with ODOP. The results were also insensitive to the ANN hyperparameters (Appendix Table A4). For networks with 64 to 256 neurons per layer, two or three hidden layers and penalties of 10−5 or 10−4, the success rate of the framework varied between 87.4% and 89.8% for compression and between 89.6% and 90.5% for extension springs, and it was 100% for torsion springs. Only the strongest penalty (10−3) reduced the success of M3 noticeably, for example from 37.0–45.0% to 29.5% for compression springs, and that of M4 to 85.0% and 86.3%. The local variants and the kNN proposals compensate for most of the differences between the networks. Larger networks mainly increased the training time, up to 102 s for extension springs.

Table 11. Results of five independent runs (seeds 42–46): mean ± standard deviation (minimum–maximum)

Quantity	Compression	Extension	Torsion
M1 ANN (direct) – success (%)	0.2 ± 0.1 (0.0–0.3)	0.3 ± 0.2 (0.1–0.5)	5.4 ± 0.7 (4.7–6.3)
M2 kNN + correction – success (%)	3.0 ± 0.4 (2.5–3.5)	16.4 ± 0.8 (15.1–17.0)	32.8 ± 1.8 (30.5–35.0)
M3 ANN + correction – success (%)	43.5 ± 1.5 (41.3–44.9)	51.0 ± 2.2 (47.9–53.8)	99.0 ± 0.3 (98.8–99.4)
M4 Proposed framework – success (%)	**87.3 ± 2.1 (85.2–89.8)**	**89.4 ± 0.9 (88.0–90.3)**	**100.0 ± 0.0 (100.0–100.0)**
M4-S Framework, standard wire – success (%)	87.2 ± 1.8 (85.3–89.3)	88.8 ± 0.7 (88.1–89.8)	100.0 ± 0.0 (100.0–100.0)
M4-X Framework, extra checks – success (%)	47.6 ± 1.6 (45.7–49.9)	33.7 ± 0.9 (32.2–34.5)	80.5 ± 0.8 (79.7–81.7)
M4 – median weight ratio	0.917 ± 0.005 (0.912–0.925)	0.892 ± 0.009 (0.880–0.903)	0.753 ± 0.011 (0.739–0.765)
Forward ANN – R^2 (log) of FS_2	0.9975 ± 0.0002	0.9990 ± 0.0002	0.9992 ± 0.0000

For each run, a new dataset, a new train/test split and new networks were used. M4-X success is the success with the extra checks.

Table 12. Success rates (%) for seen and unseen materials (leave-one-material-out test)

Spring type	Materials	M3 seen	M4 seen	M3 unseen	M4 unseen	Materials with M4 unseen ≥ 80%	Lowest M4 unseen (material)
Compression	17	53.3	91.2	41.2	81.3	13 of 17	30.2 (Titanium)
Extension	9	54.7	90.1	42.3	86.7	8 of 9	60.0 (17-7 stainless)
Torsion	15	99.1	100.0	93.4	99.5	15 of 15	92.0 (Titanium)

Seen: descriptor-based model trained on all materials. Unseen: the material was removed from the training data. Rates are weighted by the number of requirements per material (up to 200). Details per material are given in Appendix Table A3.

3.9 Sensitivity to geometric, material and load variations

Table 13 shows how many of the successful framework designs remained feasible after a single parameter change, and Table 14 gives the local elasticities. The spring rate was most sensitive to the wire diameter (elasticity 4.0), followed by the mean

coil diameter (−3.0 for compression and extension, −0.97 for torsion springs), the number of active coils (−1.0) and the modulus (+1.0). These values reproduce the exponents of Equations 2 and 4. A 1% error in the wire diameter thus changes the spring rate by about 4%. The static safety factor was most sensitive to the wire diameter (2.7–2.9), the load (−1.0) and the tensile strength (+1.0).

Because the framework selects the lightest successful candidate, many of its designs lie close to a constraint boundary. A load 10% above the specified F2 (or M2) made 56.4% of the compression, 28.6% of the extension and 47.6% of the torsion designs infeasible, and a 10% lower tensile strength had an even larger effect on compression springs (only 19.1% remained feasible). Compression springs were also sensitive to the geometry: A 2% change in d or D made 23–43% of them infeasible, mainly through the constraints related to the solid length. Extension and torsion springs were much less sensitive to the coil diameter and the number of coils, and their feasibility did not depend on the modulus. In practice, these results call for a design margin. The lower FS2 limit can be raised in the selection rule, or the selected design can be checked against the expected tolerances and load scatter. In addition, spring makers usually compensate small deviations of the wire diameter by adjusting the number of coils or the free length during production, so the rate changes in Table 14 overstate the final deviation of a manufactured spring.

Table 13. Share (%) of the successful framework designs that remain feasible after a single parameter change

Parameter change	**Compression**	**Extension**	**Torsion**
Wire diameter d, −2%	57.3	77.1	67.4
Wire diameter d, −1%	81.9	89.7	81.5
Wire diameter d, +1%	88.8	91.1	100.0
Wire diameter d, +2%	77.2	83.7	100.0
Mean coil diameter D, −2%	59.4	90.8	99.3
Mean coil diameter D, +2%	66.4	92.8	100.0
Active coils N_a, −2%	77.0	100.0	100.0
Active coils N_a, +2%	87.8	99.9	100.0
Maximum load F_2 or M_2, +5%	74.7	84.9	71.3
Maximum load F_2 or M_2, +10%	43.6	71.4	52.4
Shear or elastic modulus, −5%	73.0	100.0	100.0
Shear or elastic modulus, +5%	54.5	100.0	100.0
Tensile strength, −5%	52.8	83.9	70.4
Tensile strength, −10%	19.1	68.9	48.3

Wire diameter changed with D fixed; mean coil diameter changed with d fixed. Designs: 640 (compression), 903 (extension) and 1,000 (torsion)

Table 14. Median local elasticities of the spring rate, FS_2 and weight of the framework designs

Parameter	Spring rate			FS_2			Weight		
	C	E	T	C	E	T	C	E	T
Wire diameter *d*	4.00	4.00	4.00	2.79	2.74	2.86	2.00	2.00	2.00
Mean coil diameter *D*	−3.00	−3.00	−0.97	−0.89	−0.89	0.00	0.98	1.00	0.90
Active coils N_a	−1.00	−1.00	−1.00	0.00	0.00	0.00	0.92	0.93	0.94
Maximum load F_2 or M_2	0.00	0.00	0.00	−1.00	−1.00	−1.00	0.00	0.00	0.00
Shear or elastic modulus	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Tensile strength	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00

Elasticity $e = (\Delta Y/Y)/(\Delta x/x)$ from perturbations of ±1%. C: compression, E: extension, T: torsion

3.10 Computational cost

All timings refer to the workstation described in Section 2.8. Per requirement, the ANN prediction took 0.004–0.012 ms, the kNN search 0.09–0.49 ms and the physics-based correction of all candidates 0.02–0.13 ms. The ODOP verification took 0.07–0.12 ms per candidate when all candidates of a spring type were verified in one batch, including the start of Node.js and the file input and output. In total, the online stage took 2.25, 1.26 and 1.07 ms per requirement for compression, extension and torsion springs (mean over the five runs: 1.39, 1.19 and 1.02 ms). When a single requirement was solved alone, with its own Node.js call, the latency was 175–183 ms (median of 20 requirements), almost all of which was the start of the Node.js process and the file transfer. A tighter integration, for example a persistent ODOP process or a direct implementation of the equations in the host program, would remove this overhead. The offline stage is a one-time cost: generating the 150,000 designs took about 11 s, and training the inverse ANN took 7, 49 and 36 s. The trained inverse networks needed 3.7–15.6 MB of memory and the kNN databases 0.8–5.0 MB. The complete analysis, including all datasets, used about 3.8 GB of memory in MATLAB.

4. DISCUSSION

4.1. Main findings

The results lead to three main points. First, a purely data-driven inverse ANN is not enough for spring design. Even when it was trained on all feasible designs of the training set (about 2,900 for compression, 18,700 for extension and 25,400 for torsion springs), its median rate errors of 7% to 14% are too large for practical use. The main reason is the one-to-many nature of the problem. This agrees with earlier findings for other inverse problems. Second, simple physics can fix this problem at almost no cost. Once d and C are fixed, the number of coils and the free length (or the initial tension) follow directly from the rate equation. The rate and the load points are then

met exactly, except when the initial tension of an extension spring has to be clipped. This supports the view that physical knowledge makes learning based models more reliable. Third, generating a few candidates and verifying them is a cheap way to handle feasibility. The local variants were the most useful candidates. This shows that the ANN usually points to the right region of the design space, but not always to its feasible part.

The forward model results are also worth noting. Linear regression on logarithmic features was as good as the ANN for the spring rate and the weight. These outputs follow power laws, so a logarithmic transform makes them linear. The safety factors depend on the Wahl factor, the material strength and the load level in a nonlinear way. Here, the ANN was clearly better. Hence, simple models should not be dismissed. Good features that reflect the physics can be as valuable as a complex model.

4.2 Comparison with optimization-based approaches

The GA comparison clarifies the role of the framework. Metaheuristic studies on springs, such as those using the sine-cosine, vortex search or arithmetic optimization algorithms [8–10], report the best volume for one benchmark requirement after thousands of evaluations. In this study, a GA with the same physics-based correction found designs that were 21–32% lighter than those of the framework, but it needed about 1,900 evaluations, and this cost is paid again for every new requirement. The framework moves the cost to an offline stage and answers a new requirement with 14 evaluations. Its designs are feasible and meet the requirement exactly, but they are not optimal, and their weight savings with respect to the random reference designs (9–26%) should not be read as optimization results.

The comparison also shows a less obvious point: the treatment of the equality constraints matters more than the choice of the search algorithm. A GA that had to meet the spring rate and the load points through a penalty function failed for most compression and extension requirements, even with 1,900 evaluations, whereas the same closed-form correction made both the GA and the framework reliable. Finally, the two approaches can be combined. Seeding a short GA with the framework candidates gave 98–100% success with designs within 2–12% of the full GA result at about one tenth of its cost. A similar effect can be expected for other optimizers, including the local search in ODOP. The proper role of the framework is therefore that of a fast generator of feasible starting designs, not a replacement for optimization when the lightest design is required.

4.3. Validity of the analytical models

All data in this study come from the analytical equations of ODOP, and no experiments or Finite Element Analyses (FEA) were performed. This has two consequences. First, the framework is exactly as accurate as the design equations it uses. These equations are the standard relations of spring design [1,26] and the basis of common design practice and spring design software. However, they contain known model errors. The Wahl factor is conservative compared with more exact solutions and experiments [29], and classical rate formulas can significantly overestimate the stiffness of springs with few active coils, because the end coils are not modeled in detail [30]. In the latter study, the FE model agreed with tests on 54 springs within 2.7% on average [30]. For metaheuristic spring designs, an FE check found a

deflection difference of only 0.11%9. Hence, the 1% tolerance used here refers to the ODOP model, and the rate of a manufactured spring may differ by a few percent, especially for springs with few active coils. Second, the dataset contains no measurement noise or scatter. The success criterion therefore measures how well the framework inverts the design model, not how well the designs perform physically. The sensitivity analysis in Section 3.9 partly addresses this point by quantifying the effect of geometric, material and load variations.

Importantly, the framework does not depend on the analytical model as such. The ML part only proposes d and C, and the verification step can use any model. If a more accurate model is available, such as an FE model or a model calibrated with test data, it can replace ODOP in the verification and in the dataset generation without changing the framework. Because only 14 evaluations are needed per requirement, even an FE-based verification would remain affordable. A natural next step is to manufacture and test selected designs, which would show whether the gap between the analytical model and real springs affects the success of the framework.

4.4 Integration into design workflows

The framework fits naturally into the early stage of spring design. A typical workflow has four steps. (1) The designer enters the loads, the lengths or angles, the material and the end type, as in any spring calculator. (2) The framework returns within milliseconds the lightest successful candidate and, if required, the other successful candidates as alternatives. Standard wire diameters can be enforced as in M4-S, and the extra checks can be switched on as in M4-X. (3) The designer refines the selected design, for example with a short optimization run or in ODOP, adds margins for tolerances and load scatter (Section 3.9), and checks application-specific requirements such as the installation space, the natural frequency or the relaxation. (4) The design is passed to the spring maker, who may adjust the number of coils or the free length to meet the specified loads. Because no new training is needed for a new requirement, the framework can be embedded in a web calculator, a CAD plug-in or a configurator for standard springs. It can also generate many feasible variants quickly, for example for a design study or a spring catalog. The offline stage must be repeated only when the material table, the constraint set or the range of requirements changes.

4.5 Limitations and future work

Several limitations remain. (i) The data come from analytical models and were not validated with experiments or FEA (Section 4.3). (ii) The framework selects the lightest of 14 candidates and does not guarantee an optimal design; a short refinement is needed when weight is critical (Section 3.7). (iii) With the default selection rule, many designs fail the fatigue check in cyclic applications. The extended rule reduces this problem but lowers the success rate, and the hook geometry and buckling guides were not modeled (Section 3.6). (iv) The designs have little margin against overload and strength scatter (Section 3.9). (v) The generalization to materials outside the property range of the training materials is limited (Section 3.8). (vi) The requirements are limited to the sampling ranges of Table 3 and to the inputs of ODOP; the installation space, the natural frequency and the cost were not considered. Future work should validate selected designs experimentally, include the extra checks and design margins in the training data and the selection rule, enlarge the candidate set adaptively for

difficult requirements, and extend the framework to other spring types and design codes.

5. CONCLUSIONS

This study proposed and evaluated an ML-based inverse design framework for helical compression, extension and torsion springs. An ANN and a kNN search propose the wire diameter and the spring index, closed-form spring equations make the required spring rate and load points exact, and the open-source ODOP engine verifies every candidate. The main conclusions are as follows:

• A purely data-driven inverse ANN is not sufficient for spring design because of the one-to-many nature of the problem. The closed-form correction removes the requirement errors and is the most important element, both for the ML framework and for a genetic algorithm.

• A small set of corrected and verified candidates gives feasible designs for about 90% of the compression and extension requirements and for all torsion requirements within a few milliseconds. The results are stable across independent runs and network settings, and they carry over to unseen materials whose properties lie within the range of the training materials.

• The framework is a fast generator of feasible designs, not an optimizer. A genetic algorithm finds clearly lighter designs at roughly one hundred times the cost, and a short refinement that starts from the framework candidates combines most of the benefits of both approaches.

• Standard wire diameters can be handled with almost no loss of success, whereas fatigue is the main open issue in cyclic applications. When fatigue, buckling and hook checks are required, they must be part of the selection rule, which lowers the success rate and calls for a larger or refined candidate set.

For engineering practice, the framework can serve as the front end of spring calculators and configurators: it gives an immediate, verified starting design that the designer can refine. Its accuracy is bounded by that of the analytical design model. Experimental validation of selected designs, explicit design margins and an adaptive candidate set are therefore the main directions for future work.

DECLARATIONS

DATA AND CODE AVAILABILITY: The datasets and the MATLAB code used in this study are available from the corresponding author upon reasonable request. The ODOP calculation engine is open source (MIT license) and publicly available. The sampling ranges, the random seeds and all model and algorithm settings needed to reproduce the results are given in Sections 2.2–2.8.

FUNDING: This research received no external funding.

CONFLICT OF INTEREST: The author declares that they have no conflict of interest.

6. REFERENCES

[1] Budynas RG, Nisbett JK. Shigley's mechanical engineering design. 11th ed. New York: McGraw-Hill Education; 2020.

[2] ODOP: Open Design Optimization Platform. Commit 0033fc2 [Internet]. GitHub; 2026 [cited 2026 Oct 5]. Available from: https://github.com/thegrumpys/odop.

[3] Ibrikçi T, Saçma S, Yıldırım V, Koca T. Application of artificial neural networks in the prediction of critical buckling loads of helical

compression springs. Stroj Vestn J Mech Eng. 2010;56(6):409-417.

[4] Sabuncuoğlu B, Demirtaş O. Development of an artificial neural network using parametric correlation technique for the determination of machined torsional spring stiffness. J Fac Eng Archit Gazi Univ. 2021;36(1):105-18. doi:10.17341/gazimmfd.641491.

[5] Yavuz S, Özkan MT. Analyze of leaf springs with parametric finite element analysis and artificial neural network. J Polytech. 2022;25(2):827-42. doi:10.2339/politeknik.1074516.

[6] Chin CH, Abdullah S, Singh SSK, Ariffin AK, Schramm D. Fatigue life modelling of steel suspension coil springs based on wavelet vibration features using neuro-fuzzy methods. Materials. 2023;16(6):2494. doi:10.3390/ma16062494.

[7] Zhang X, Liu F, Shen M, Han D, Wang Z, Yan N. Ultra-high-cycle fatigue life prediction of metallic materials based on machine learning. Appl Sci. 2023;13(4):2524. doi:10.3390/app13042524.

[8] Rodríguez Cabal MA, Grisales Noreña LF, Ramírez Vanegas CA, Arias Londoño A. Application of the sine-cosine algorithm to the optimal design of a closed coil helical spring. Trans Energy Syst Eng Appl. 2022;2(2):33-8. doi:10.32397/tesea.vol2.n2.5.

[9] Rodriguez Cabal M, Grisales Norena L, Montoya OD, Hincapie D, Cortes Caicedo B. Optimal design of helical springs using metaheuristic techniques. Math Model Eng Probl. 2024;11(10):2595-605. doi:10.18280/mmep.111001.

[10] Abualigah L, Diabat A, Mirjalili S, Abd Elaziz M, Gandomi AH. The arithmetic optimization algorithm. Comput Methods Appl Mech Eng. 2021;376:113609. doi:10.1016/j.cma.2020.113609.

[11] Zhao W, Wang L, Mirjalili S. Artificial hummingbird algorithm: a new bio-inspired optimizer with its engineering applications. Comput Methods Appl Mech Eng. 2022;388:114194. doi:10.1016/j.cma.2021.114194.

[12] Hashim FA, Houssein EH, Hussain K, Mabrouk MS, Al-Atabany W. Honey badger algorithm: new metaheuristic algorithm for solving optimization problems. Math Comput Simul. 2022;192:84-110. doi:10.1016/j.matcom.2021.08.013.

[13] Yu RT, Picard C, Ahmed F. Fast and accurate Bayesian optimization with pre-trained transformers for constrained engineering problems. Struct Multidiscip Optim. 2025;68:66. doi:10.1007/s00158-025-03987-z.

[14] Liu D, Tan Y, Khoram E, Yu Z. Training deep neural networks for the inverse design of nanophotonic structures. ACS Photonics. 2018;5(4):1365-9. doi:10.1021/acsphotonics.7b01377.

[15] Ren S, Padilla W, Malof J. Benchmarking deep inverse models over time, and the neural-adjoint method. Adv Neural Inf Process Syst. 2020;33:38-48.

[16] Spell GP, Ren S, Collins LM, Malof JM. Mixture manifold networks: a computationally efficient baseline for inverse modeling. Proc AAAI Conf Artif Intell. 2023;37(8):9874-81. doi:10.1609/aaai.v37i8.26178.

[17] Bastek JH, Kumar S, Telgen B, Glaesener RN, Kochmann DM. Inverting the structure-property map of truss metamaterials by deep learning. Proc

Natl Acad Sci U S A. 2022;119(1):e2111505119. doi:10.1073/pnas.2111505119.

[18] Deng B, Zareei A, Ding X, Weaver JC, Rycroft CH, Bertoldi K. Inverse design of mechanical metamaterials with target nonlinear response *via* a neural accelerated evolution strategy. Adv Mater. 2022;34(41):2206238. doi:10.1002/adma.202206238.

[19] Zheng L, Karapiperis K, Kumar S, Kochmann DM. Unifying the design space and optimizing linear and nonlinear truss metamaterials by generative modeling. Nat Commun. 2023;14(1):7563. doi:10.1038/s41467-023-42068-x.

[20] Li Y, Coros S, Thomaszewski B. Neural metamaterial networks for nonlinear material design. ACM Trans Graph. 2023;42(6):186. doi:10.1145/3618325.

[21] Guo X, Zheng X, Zhou J, Yamada T, Yi Y, Watanabe I. Spring-based mechanical metamaterials with deep-learning-accelerated design. Mater Des. 2025;252:113800. doi:10.1016/j.matdes.2025.113800.

[22] Mohammadi M, Kouzani AZ, Bodaghi M, Zolfagharian A. Inverse design of adaptive flexible structures using physical-enhanced neural network. Virtual Phys Prototyp. 2025;20(1):e2530732. doi:10.1080/17452759.2025.2530732.

[23] Regenwetter L, Nobari AH, Ahmed F. Deep generative models in engineering design: A review. J Mech Des. 2022;144(7):071704. doi:10.1115/1.4053859.

[24] Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nat Rev Phys. 2021;3(6):422-40. doi:10.1038/s42254-021-00314-5.

[25] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics. 1979;21(2):239-45. doi:10.2307/1268522.

[26] Wahl AM. Mechanical springs. 2nd ed. New York: McGraw-Hill; 1963.

[27] Deb K. An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng. 2000;186(2-4):311-38. doi:10.1016/S0045-7825(99)00389-8.

[28] Eshelman LJ, Schaffer JD. Real-coded genetic algorithms and interval-schemata. In: Whitley LD, editor. Foundations of genetic algorithms 2. San Mateo (CA): Morgan Kaufmann; 1993. p. 187-202.

[29] Čakmak D, Tomičević Z, Senjanović I, Wolf H, Božić Ž, Semenski D. A study on cylindrical coil spring deflection and stress done using analytical and numerical methods. Eng Model. 2022;35(1):57-81. doi:10.31534/engmod.2022.1.ri.04m

[30] Baran R, Michalczyk K, Warzecha M. Effect of the end coil shape of the helical compression spring on its stiffness and distribution of transverse reactions during axial loading. Acta Mech Autom. 2025;19(3):471-84. doi:10.2478/ama-2025-0055.

APPENDIX

This appendix gives detailed results of the additional analyses summarized in Section 3.

Table A1. Forward models over five independent runs (mean ± standard deviation)

Spring type	Output	R^2 (log scale)		MAPE (%)	
		LR	ANN	LR	ANN
Compression	Spring rate	0.9983 ± 0.0000	0.9992 ± 0.0001	6.21 ± 0.04	4.83 ± 0.28
	FS_2	0.9085 ± 0.0023	0.9975 ± 0.0002	5.30 ± 0.03	0.88 ± 0.04
	FS_{solid}	0.7442 ± 0.0056	0.9963 ± 0.0008	13.61 ± 0.12	1.27 ± 0.16
	Weight	0.9999 ± 0.0000	0.9993 ± 0.0001	3.00 ± 0.03	8.73 ± 0.74
Extension	Spring rate	0.9996 ± 0.0000	0.9994 ± 0.0000	3.35 ± 0.00	3.97 ± 0.16
	FS_2	0.9335 ± 0.0016	0.9990 ± 0.0002	4.52 ± 0.04	0.54 ± 0.05
	FS_{hook}	0.9341 ± 0.0024	0.9992 ± 0.0001	5.87 ± 0.06	0.67 ± 0.03
	Weight	0.9999 ± 0.0000	0.9993 ± 0.0001	2.36 ± 0.01	7.39 ± 0.36
Torsion	Spring rate	1.0000 ± 0.0000	0.9994 ± 0.0001	1.38 ± 0.01	6.74 ± 0.24
	FS_2	0.8739 ± 0.0017	0.9992 ± 0.0000	6.51 ± 0.06	0.49 ± 0.01
	Deflection angle θ_2	0.9996 ± 0.0000	0.9993 ± 0.0000	1.38 ± 0.01	1.80 ± 0.07
	Weight	0.9999 ± 0.0000	0.9993 ± 0.0001	2.37 ± 0.02	7.48 ± 0.55

Table A2. Fatigue check violation rates (%) by life category

Spring type	Life category	Requirements	M3	M4	M4-X
Compression	Static, not peened	91	0.0	0.0	0.0
	10^5 cycles, not peened	99	16.2	17.2	8.1
	10^6 cycles, not peened	79	26.6	29.1	17.7
	10^7 cycles, not peened	87	41.4	50.6	31.0
	Static, shot peened	86	0.0	0.0	0.0
	10^5 cycles, shot peened	92	7.6	4.3	4.3
	10^6 cycles, shot peened	83	8.4	8.4	4.8
	10^7 cycles, shot peened	96	22.9	21.9	15.6
Extension	Static, not peened	250	0.0	0.0	0.0

	10^5 cycles, not peened	246	5.3	16.7	5.3
	10^6 cycles, not peened	251	12.7	28.7	12.7
	10^7 cycles, not peened	253	17.8	35.2	17.8
Torsion	Static, not peened	276	0.0	0.0	0.0
	10^5 cycles, not peened	249	43.0	74.7	15.7
	10^6 cycles, not peened	233	54.5	85.0	24.5
	10^7 cycles, not peened	242	61.6	88.4	35.5

Table A3. Leave-one-material-out results per material: success rates (%)

Spring type	Material	Requirements	M3 seen	M4 seen	M3 unseen	M4 unseen
Compression	Hard-drawn wire	40	67.5	95.0	42.5	87.5
	Music wire	50	62.0	96.0	58.0	96.0
	Oil-tempered MB	61	52.5	91.8	63.9	86.9
	Chrome vanadium	58	62.1	96.6	67.2	94.8
	Chrome silicon	37	64.9	91.9	51.4	86.5
	SAE 9250	41	70.7	97.6	58.5	92.7
	302 stainless	42	42.9	90.5	40.5	88.1
	316 stainless	34	47.1	88.2	23.5	73.5
	17-7 stainless	34	61.8	94.1	50.0	91.2
	Spring brass	41	39.0	82.9	31.7	75.6
	Phosphor bronze	40	37.5	90.0	45.0	92.5
	Monel	31	35.5	83.9	29.0	80.6
	Inconel X-750	43	46.5	90.7	27.9	81.4
	Beryllium copper	44	45.5	90.9	38.6	93.2
	Titanium	43	55.8	93.0	4.7	30.2
	5160H	37	45.9	83.8	2.7	32.4
	5160H-CG	37	62.2	86.5	35.1	86.5
Extension	Hard-drawn wire	200	52.0	90.5	46.5	90.5

	Music wire	200	49.0	88.5	39.5	89.0
	Oil-tempered MB	200	50.5	89.5	54.0	89.0
	Chrome vanadium	200	56.5	90.0	48.0	92.0
	Chrome silicon	200	50.5	92.5	50.5	94.0
	SAE 9250	200	51.0	87.0	48.5	86.5
	302 stainless	200	63.0	91.5	48.5	92.0
	316 stainless	200	62.0	89.5	40.5	87.0
	17-7 stainless	200	58.0	92.0	4.5	60.0
Torsion	Hard-drawn wire	200	99.5	100.0	100.0	100.0
	Music wire	200	99.0	100.0	99.0	100.0
	Oil-tempered MB	200	99.5	100.0	99.5	100.0
	Chrome vanadium	200	100.0	100.0	99.5	100.0
	Chrome silicon	200	100.0	100.0	99.0	100.0
	SAE 9250	200	100.0	100.0	99.0	100.0
	302 stainless	200	98.5	100.0	99.0	100.0
	316 stainless	200	100.0	100.0	99.5	100.0
	17-7 stainless	200	100.0	100.0	88.0	100.0
	Spring brass	200	98.5	100.0	98.5	100.0
	Phosphor bronze	200	97.5	100.0	96.5	100.0
	Monel	200	100.0	100.0	97.0	100.0
	Inconel X-750	200	99.0	100.0	99.5	100.0
	Beryllium copper	200	96.5	100.0	98.5	100.0
	Titanium	200	99.0	100.0	29.0	92.0

Table A4. Sensitivity of the inverse design results to the hyperparameters of the inverse ANN

Spring type	Hidden layers	λ	Iterations	Training time (s)	Validation loss	M3 success (%)	M4 success (%)	M4 median weight ratio

Compression	[64 64]	10^{-4}	214	4.6	0.097	39.8	88.5	0.923
	[128 128]	10^{-4}	146	6.7	0.096	44.7	89.8	0.912
	[256 256]	10^{-4}	112	16.0	0.100	45.0	89.1	0.917
	[128 128 128]	10^{-4}	144	11.6	0.099	37.0	87.4	0.922
	[128 128]	10^{-5}	146	7.4	0.094	41.9	87.9	0.923
	[128 128]	10^{-3}	98	4.8	0.093	29.5	85.0	0.919
Extension	[64 64]	10^{-4}	248	18.4	0.263	50.8	89.6	0.884
	[128 128]	10^{-4}	207	39.1	0.265	50.5	90.3	0.880
	[256 256]	10^{-4}	189	102.3	0.266	48.7	89.8	0.877
	[128 128 128]	10^{-4}	326	78.2	0.266	50.3	89.7	0.882
	[128 128]	10^{-5}	236	38.6	0.267	51.2	90.5	0.883
	[128 128]	10^{-3}	69	29.0	0.260	41.7	86.3	0.890
Torsion	[64 64]	10^{-4}	141	17.7	0.151	99.7	100.0	0.746
	[128 128]	10^{-4}	71	31.8	0.155	98.8	100.0	0.739
	[256 256]	10^{-4}	71	81.5	0.154	99.1	100.0	0.738
	[128 128 128]	10^{-4}	81	55.9	0.155	98.7	100.0	0.741
	[128 128]	10^{-5}	224	46.5	0.155	100.0	100.0	0.759
	[128 128]	10^{-3}	37	38.7	0.148	84.4	99.8	0.736

The default configuration is [128 128] with $\lambda = 10^{-4}$. Validation loss: smallest mean squared error on the standardized outputs of the early-stopping set, averaged over the outputs.