SEARCH HARDNESS-AWARE LLM-BASED PROBLEM FORMULATION FOR EXPENSIVE SIMULATION-DRIVEN DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Expensive simulation-driven design is widely used in engineering to identify requirement-satisfying designs with as few high-fidelity simulations as possible. Most existing efforts address this challenge by improving optimization algorithms under fixed formulations, yet the formulation itself shapes the search landscape by defining the objectives and constraints optimized by the solver. Recent LLM-based automatic problem formulation methods generate formulations from natural-language requirements, but they mainly focus on design-intent alignment and overlook whether the formulation induces an efficient search process. To address this limitation, we propose SHA-PF, a search hardness-aware LLM-based problem formulation framework. We find that a formulation is more likely to guide efficient search when it prioritizes rare samples with greater progress potential. Based on this finding, SHA-PF defines a formulation search objective guided by search hardness, scoring each candidate formulation according to the priority. SHA-PF then searches the formulation space under this objective through LLM-based generation, repair, and evolutionary refinement. Experiments on the real-world multi-objective benchmark and five expensive antenna design benchmarks show that the formulations discovered by SHA-PF require significantly fewer evaluations to reach the design requirements than other baselines. Code and data are available at the Anonymous Link.

1 INTRODUCTION

Expensive simulation-driven design has been widely used to improve the performance of complex engineering systems, such as airfoil design (Wissink et al., 2025; Li et al., 2022), and microelectronic design (Lai et al., 2024). In such problems, a time-consuming high-fidelity simulator is usually required to evaluate the performance response of a candidate design parameter. The goal is to find design parameters whose responses satisfy natural-language requirements within a limited evaluation budget. In practice, the design process usually involves two stages: problem formulation and optimization. The former converts natural-language requirements into a solvable optimization problem, while the latter searches for design parameters under the formulated optimization problem.

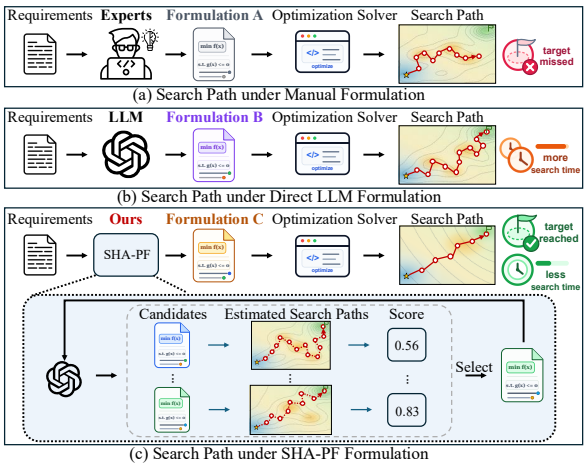


Figure 1: Comparison of search paths under different problem formulation methods.

Most existing studies focus on developing surrogate-assisted optimization algorithms to reduce high-fidelity simulations due to the high cost of simulation evaluations (Jin et al., 2019), where surrogate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

models predict promising candidates for expensive evaluations. However, they typically rely on a strong assumption that the given problem formulation is correct and effective. The formulation defines the evaluation metrics of the solutions, such as objective functions and constraints, thereby shaping the search-space landscape and determining the search path from starting solutions to target solutions, as illustrated in Figure 1. In practice, such formulations are usually manually designed by domain experts based on their knowledge and experience, making the process time-consuming and labor-intensive.

These limitations have motivated research on automated problem formulation (Zhang et al., 2025). Recent advances in large language models (LLMs), such as GPT (OpenAI et al., 2024), Gemini (Team et al., 2025), and DeepSeek (DeepSeek-AI et al., 2025), have further advanced this direction. Existing methods mainly focus on converting vague natural-language requirements into precise mathematical formulations, and can generally be divided into prompting-based and learning-based methods. Prompting-based methods, such as Chain-of-Experts (Xiao et al., 2023) and OptiMUS (Ahmaditeshnizi et al., 2024), rely on carefully designed prompting strategies or multi-agent collaboration and can be quickly deployed without additional training. Learning-based methods, such as LLaMoCo (Ma et al., 2024), ORLM (Huang et al., 2025), LLMOPT (Jiang et al., 2024), SIRL (Chen et al., 2025), and APF (Li et al., 2026), usually improve reliability in complex scenarios through data synthesis and fine-tuning. Although these methods improve the formal accuracy of converting natural-language requirements into mathematical formulations, they mainly ensure that the shaped search-space landscape defines the correct target region.

In expensive simulation-driven design, even when the target region is correctly defined, there may exist many search paths from the starting solutions to that region. Due to the different landscape hardness along these paths, they require different numbers of expensive simulation evaluations and directly affect search efficiency. Given the direct impact of search efficiency on design cycle and cost, automatically discovering formulations that induce more efficient search paths becomes essential for expensive simulation-driven design.

To address this challenge, we propose SHA-PF, a search hardness-aware LLM-based problem formulation framework that uses only the initially evaluated designs and their simulation responses to discover formulations that enable higher search efficiency. We find that the initially evaluated data contain early partial paths toward the target solution, where rarely observed paths indicate harder-to-reach and more informative search directions. If a formulation assigns higher priority to these rarely observed but promising paths, the optimizer is more likely to continue exploring along them and reach the target solution with fewer expensive evaluations than it would by following commonly observed paths. Based on this finding, SHA-PF constructs an anchor-state-based evaluation criterion to score candidate formulations and discover efficient problem formulations.

The main contributions of this paper are as follows: (1) We reveal that rarely observed but promising search paths in the initial simulation data are more valuable for continued search than commonly observed paths. (2) We propose an anchor-state-based formulation evaluation criterion that favors formulations assigning higher priority to rarely observed but promising paths, thereby encouraging the optimizer to continue exploring along these paths. (3) We propose SHA-PF, which integrates the proposed evaluation criterion with LLM-based generation, validity repair, and iterative refinement to discover efficient formulations. We validate the effectiveness of SHA-PF on a real-world multi-objective benchmark and antenna design tasks. The experimental results show that SHA-PF can significantly reduce the number of expensive simulation evaluations.

2 Related Work

2.1 Expensive Simulation-Driven Optimization

In simulation-driven optimization, evaluating each candidate design requires costly simulations, which makes the affordable number of evaluations very limited. This bottleneck has motivated surrogate-assisted optimization methods, which use inexpensive surrogate models to reduce costly true evaluations (Jin et al., 2019; Wang et al., 2023).

Surrogate-assisted evolutionary algorithms (SAEAs) tackle expensive optimization from a population-based perspective, using surrogate models to guide evolutionary search and reduce exact

evaluations (Liu et al., 2023b). Representative methods include GPEME for medium-scale single-objective expensive optimization, which uses Gaussian-process surrogates to prescreen promising offspring (Liu et al., 2014), and K-RVEA for expensive many-objective optimization, which couples Kriging models with reference-vector-guided selection (Chugh et al., 2018). Bayesian optimization (BO) is another major paradigm, using probabilistic surrogates and acquisition functions to decide which candidates to evaluate. TuRBO scales BO to high-dimensional black-box problems through local trust regions (Eriksson et al., 2019). SCBO extends trust-region BO to constrained optimization by balancing improvement and feasibility (Eriksson & Poloczek, 2021). qEHVI addresses parallel multi-objective BO with differentiable hypervolume-based acquisitions (Daulton et al., 2020). qNEHVI extends this framework to noisy multi-objective evaluations (Daulton et al., 2021).

Despite their differences, both SAEAs and BO usually assume that a well-defined optimization problem is already available before the search begins. However, problem formulation itself is an indispensable step in real simulation-driven design. For example, the unmanned underwater vehicle design task formalizes vehicle-dimension search as a constrained multi-objective optimization problem evaluated with low and high-fidelity computational fluid dynamics models (Alam et al., 2017), while SGSO formalizes filtering patch antenna design as an expensive constrained optimization problem under electromagnetic simulations (Xu et al., 2026). These methods rely on a sufficiently accurate and effective formulation, but they do not address how such a formulation should be automatically constructed from natural-language design requirements. This leaves automated problem formulation as a necessary problem for simulation-driven optimization.

2.2 LLMs for Problem Formulation

The NL4Opt benchmark and competition provided an early standardized testbed by decomposing natural-language optimization modeling into semantic-entity recognition and formulation generation tasks (Ramamonjison et al., 2022; 2023). Subsequent prompt-based and agentic methods improve formulation by decomposing the modeling process, coordinating specialized agents, or supporting interactive refinement. Chain-of-Experts and OptiMUS use multiple LLM agents to coordinate modeling, programming, and verification (Xiao et al., 2023; Ahmaditeshnizi et al., 2024). OptLLM supports multi-round dialogues for gradually refining the modeling and solving process, while MeetMate studies interactive decision support in meeting scheduling by combining LLM-based preference construction with constraint-programming-based preference incorporation (Zhang et al., 2024; Lawless et al., 2024).

Recent work has further shifted from prompt engineering to learning-based formulation. ORLM synthesizes large-scale triples of natural-language problems, formal models, and solver code to fine-tune open-source LLMs (Huang et al., 2025). OptMATH and ReSocratic improve synthetic data generation through bidirectional or inverse modeling pipelines (Lu et al., 2025; Yang et al., 2025). LLMOPT trains LLMs with optimization-oriented instructions to improve both problem definition and solver-code generation across optimization problem types (Jiang et al., 2024), while SIRL introduces solver-verifiable reinforcement learning to improve formulation reliability (Chen et al., 2025). APF extends this line of work to high-cost simulation-driven design, where formulation learning must proceed without solver-based verification because each evaluation requires expensive simulation (Li et al., 2026). However, these studies mainly evaluate whether the generated formulation is correct, executable, or solver-verifiable, rather than whether it induces an efficient optimization process. This limitation is critical in expensive simulation-driven design, where different valid formulations may encode the same design intent but lead to search paths with substantially different simulation costs.

3 Method

3.1 Search-Efficient Formulation

In expensive simulation-driven design, each design candidate $\mathbf{x} \in \mathcal{X}$ is evaluated by a costly simulator $\mathcal{S} : \mathcal{X} \to \mathcal{Y}$, yielding a simulated response $\mathbf{y} = \mathcal{S}(\mathbf{x})$. Since each simulator call can be computationally expensive, the number of evaluations is limited by a budget B.

The design intent is specified by a natural-language requirement $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$, where each r_k denotes a design criterion. For a response $\mathbf{y} \in \mathcal{Y}$, let $s_k(\mathbf{y}) \in \{0, 1\}$ indicate whether $\mathbf{y}$ satisfies

criterion r_k, and define its satisfaction state as

$$\mathbf{s}(\mathbf{y}) = (s_1(\mathbf{y}), \ldots, s_K(\mathbf{y})) \in \{0,1\}^K. \tag{1}$$

The requirement-satisfying response set is $\mathcal{Y}_{\mathcal{R}}^{+} = \{\mathbf{y} \in \mathcal{Y} \mid s_k(\mathbf{y}) = 1,\ \forall k = 1, \ldots, K\}$. The design goal is to find $\mathbf{x} \in \mathcal{X}$ such that $\mathcal{S}(\mathbf{x}) \in \mathcal{Y}_{\mathcal{R}}^{+}$.

Since $\mathcal{R}$ is specified in natural language, it must be instantiated as an executable formulation $\mathcal{F} \in \Phi$, where Φ denotes the candidate formulation space. Let $\mathcal{X}_{\mathcal{F}}^{*}$ denote the ideal solution set induced by $\mathcal{F}$. The set of requirement-aligned formulations is defined as

$$\Phi_{\mathcal{R}} = \left\{\mathcal{F} \in \Phi \mid \mathcal{S}(\mathbf{x}) \in \mathcal{Y}_{\mathcal{R}}^{+},\ \forall \mathbf{x} \in \mathcal{X}_{\mathcal{F}}^{*}\right\}. \tag{2}$$

Given a requirement-aligned formulation $\mathcal{F} \in \Phi_{\mathcal{R}}$, an optimizer $\mathcal{A}$ sequentially queries the simulator under budget B, forming a trajectory from lower-satisfaction states toward higher-satisfaction states:

$$\mathcal{T}_{\mathcal{F}}^{\mathcal{A}} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{T}, \quad \mathbf{y}_t = \mathcal{S}(\mathbf{x}_t). \tag{3}$$

The trajectory ends when it first reaches $\mathcal{Y}_{\mathcal{R}}^{+}$ or when the budget is exhausted. The search cost is defined as $\tau_{\mathcal{A}}(\mathcal{F}) = T$ if $\mathbf{y}_T \in \mathcal{Y}_{\mathcal{R}}^{+}$, and $\tau_{\mathcal{A}}(\mathcal{F}) = B + 1$ otherwise. Thus, we aim to find a formulation $\mathcal{F} \in \Phi_{\mathcal{R}}$ that minimizes $\tau_{\mathcal{A}}(\mathcal{F})$.

3.2 Finding: Initial Data Reveal Search Hardness

Expensive simulation-driven optimization algorithms usually start from a globally distributed initial set $\mathcal{D}_{\text{init}} = \{(\mathbf{x}_i^{\text{init}}, \mathbf{y}_i^{\text{init}})\}_{i=1}^{N}$, with $\mathbf{y}_i^{\text{init}} = \mathcal{S}(\mathbf{x}_i^{\text{init}})$. These space-filling evaluations provide sparse observations of the search landscape over the design space, and therefore offer accessible evidence for estimating whether a candidate formulation may induce a lower search cost.

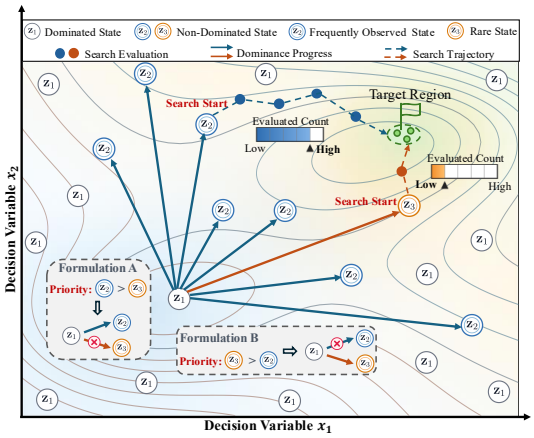


Figure 2: Illustration of search hardness revealed by the initial evaluated data in the decision space.

The satisfaction states of the initial samples in $\mathcal{D}_{\text{init}}$ provide partial search trajectories toward $\mathcal{Y}_{\mathcal{R}}^{+}$, as illustrated in Figure 2. Specifically, frequently observed states indicate regions that are easily covered by sampling and can be reached by the optimizer without additional guidance, making further prioritization of them less beneficial to the search process. In contrast, rare states correspond to hard yet promising regions that are difficult for the optimizer to reach naturally, and should be prioritized by the formulation to guide the optimizer into such regions and continue the search. These observed partial trajectories help estimate which hard trajectories are more likely to reach $\mathcal{Y}_{\mathcal{R}}^{+}$ with fewer evaluations. **This motivates discovering a formulation that prioritizes such hard but promising trajectories, thereby reducing the search cost.**

3.3 Formulation Search

3.3.1 Search Hardness Metric and Anchor Selection

Based on this finding, we define a search hardness metric for observed satisfaction states and use it to select an anchor state that is hard to reach but promising for further search. Specifically, given the initial dataset $\mathcal{D}_{\text{init}}$, we use an LLM-based judge to evaluate each response $\mathbf{y}_i$ against the requirement $\mathcal{R}$ and obtain its satisfaction state $\mathbf{z}_i = \mathbf{s}(\mathbf{y}_i)$. We denote the observed state set as $\mathcal{Z}_{\text{init}} = \{\mathbf{z}_i\}_{i=1}^{N}$. The detailed judging prompt is provided in Appendix B.1.

We define the search hardness of each observed state by jointly considering its rarity and progress potential. For each $\mathbf{z} \in \mathcal{Z}_{\text{init}}$, the search hardness score is defined as

$$h(\mathbf{z}) = \frac{|\{i \mid \mathbf{z}_i = \mathbf{z}\}|}{|\{i \mid \mathbf{z} \succ \mathbf{z}_i\}| + \epsilon}, \tag{4}$$

where the numerator is the occurrence count of $\mathbf{z}$ in the initial data, with a smaller value indicating higher rarity. The denominator measures the progress potential of $\mathbf{z}$ by counting the initial observations whose states are dominated by $\mathbf{z}$. Here, $\succ$ follows the Pareto dominance relation (Yu, 1974) over satisfaction states, where a state dominates another if it satisfies all criteria satisfied by the other state and at least one additional criterion. The constant ϵ avoids division by zero. A lower $h(\mathbf{z})$ therefore indicates a less frequently observed state with stronger dominance support, corresponding to a harder yet more promising state.

Let $\mathcal{Z}_{\mathrm{nd}}$ denote the non-dominated observed state set, defined as $\mathcal{Z}_{\mathrm{nd}} = \{\mathbf{z} \in \mathcal{Z}_{\mathrm{init}} \mid \nexists\, \mathbf{z}' \in \mathcal{Z}_{\mathrm{init}} : \mathbf{z}' \succ \mathbf{z}\}$. The anchor state is then selected from $\mathcal{Z}_{\mathrm{nd}}$ by minimizing $h(\mathbf{z})$:

$$\mathbf{z}_{\mathrm{anc}} = \arg\min_{\mathbf{z} \in \mathcal{Z}_{\mathrm{nd}}} h(\mathbf{z}). \tag{5}$$

3.3.2 Formulation Search Objective

For each candidate formulation $\mathcal{F}$, its objectives and constraints induce a feasibility-based ranking over the initial responses. We write $\mathbf{y}_i \triangleright_{\mathcal{F}} \mathbf{y}_j$ if $\mathbf{y}_i$ has higher priority than $\mathbf{y}_j$ under this formulation-induced ranking. To prioritize the selected anchor state, we construct anchor and other comparison pairs:

$$\mathcal{P}_{\mathrm{anc}} = \{(i, j) \mid \mathbf{z}_i = \mathbf{z}_{\mathrm{anc}},\ \mathbf{z}_j \neq \mathbf{z}_{\mathrm{anc}}\}. \tag{6}$$

The formulation search objective is then defined as

$$\mathcal{F}^* = \arg\max_{\mathcal{F} \in \widehat{\Phi}_{\mathcal{R}}} \frac{1}{|\mathcal{P}_{\mathrm{anc}}|} \sum_{(i,j) \in \mathcal{P}_{\mathrm{anc}}} \mathbb{I}\left[\mathbf{y}_i \triangleright_{\mathcal{F}} \mathbf{y}_j\right], \tag{7}$$

where $\widehat{\Phi}_{\mathcal{R}}$ denotes the executable and requirement-aligned candidate formulations. This objective favors formulations that rank anchor state responses above other responses, thereby guiding the optimizer toward the selected hard but promising search trajectory.

3.3.3 Formulation Generation and Repair

In the evolutionary formulation search, each individual is a candidate formulation $\mathcal{F}$ represented as an executable program $\mathcal{F}(\mathbf{y}) = (f_{\mathcal{F}}(\mathbf{y}), \mathbf{g}_{\mathcal{F}}(\mathbf{y}))$, which maps a simulated response $\mathbf{y}$ to one scalar objective and zero or more constraints. The objective is minimized, and a response is feasible if $g_{\mathcal{F},m}(\mathbf{y}) \leq 0$ for all $m = 1, \ldots, M$.

Candidate formulations are constructed through generation and repair. A generation LLM $\mathcal{M}_g$ first produces a draft formulation $\tilde{\mathcal{F}} = \mathcal{M}_g(\mathcal{R})$. Since $\tilde{\mathcal{F}}$ may be syntactically executable but semantically misaligned with the design intent, we further employ a repair LLM $\mathcal{M}_r$ to check and revise it according to $\mathcal{R}$, yielding $\mathcal{F} = \mathcal{M}_r(\tilde{\mathcal{F}}, \mathcal{R})$. This repair step aims to move the draft formulation toward the ideal requirement-aligned set $\Phi_{\mathcal{R}}$ and produces the executable candidate set $\widehat{\Phi}_{\mathcal{R}}$ used in formulation search. The detailed generation and repair prompts are provided in Appendices B.2 and B.3, respectively.

3.3.4 Evolutionary Formulation Refinement

We search the formulation space by maximizing the formulation search objective defined above. Starting from a population of repaired formulations, each generation evaluates all candidates on $\mathcal{D}_{\mathrm{init}}$, selects the best-performing formulations as parents, and samples operator prompts to generate offspring through LLM-based variation.

Offspring formulations are generated by applying operator prompts to selected parents. Rather than relying on a single mutation instruction, we define a set of formulation-space variation operators that modify candidates at different levels of abstraction. These operators cover component-level recombination, constraint-structure modification, optimization-signal reparameterization, and broader semantic exploration. The operator set contains four complementary strategies:

- **Component recombination** merges useful objective and constraint components from two best-fitness parents.
- **Constraint restructuring** modifies the number, grouping, or regional organization of constraints.

- **Signal reparameterization** changes how objective values or constraint violations are measured and aggregated.
- **Semantic exploration** introduces larger formulation-level changes to explore alternative formulation families while preserving executability and requirement alignment.

The detailed operator prompts are provided in Appendix B.4. Each offspring is repaired by $\mathcal{M}_r$ and assigned an objective value using the formulation search objective. Parents and offspring are then merged and ranked by fitness, and the top candidates are retained as the next population. The process repeats until the evaluation budget or maximum number of generations is reached, and the formulation with the highest objective value is returned for optimization.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmark Tasks. We evaluate SHA-PF on a real-world multi-objective calibration problem and five expensive antenna design problems. We focus on real-world tasks because their domain-specific requirements can often be represented by multiple valid objective and constraint formulations, allowing us to study how formulation choices affect optimization behavior. Detailed descriptions of the HBV and antenna benchmarks are provided in Appendices C.1 and C.2, respectively.

1) HBV rainfall-runoff calibration. We consider the HBV rainfall-runoff calibration problem (Reed et al., 2013), a real-world multi-objective task with 14 continuous calibration variables. The model is calibrated on the Williams River, West Virginia, using precipitation and streamflow data from the MOPEX dataset. The four objectives measure high-flow accuracy using Nash-Sutcliffe efficiency (NSE), low-flow accuracy using Box-Cox transformed RMSE (TRMSE), long-term water balance using runoff coefficient error (ROCE), and flow-regime variability using flow-duration-curve slope error (SFDCE).

2) Antenna design benchmarks. We further evaluate SHA-PF on five expensive antenna design tasks: HAE (Guo et al., 2024), CSL (Hu et al., 2020), HSL (Liu et al., 2023a), WHG (Yuan et al., 2022), and HSE (Liang et al., 2022). These tasks span diverse antenna structures and design requirements, resulting in different optimization landscapes. Each candidate design requires an electromagnetic simulation, making these tasks suitable for evaluating whether improved formulations can reduce the number of expensive evaluations needed to obtain requirement-satisfying designs.

Baselines. To the best of our knowledge, no existing method directly addresses search-efficient problem formulation for expensive simulation-driven design. We therefore construct baselines representing two common formulation strategies: manual formulation by human experts and direct formulation generation by LLMs. Specifically, we compare SHA-PF with four formulation baselines, denoted as $\mathcal{F}_{\text{Expert-A}}$, $\mathcal{F}_{\text{Expert-B}}$, $\mathcal{F}_{\text{LLM-A}}$, and $\mathcal{F}_{\text{LLM-B}}$. Since the same requirement may admit multiple valid formulations, we use two mathematically different formulations for each baseline type to avoid relying on a single baseline formulation. The expert-based formulations are manually designed and checked by human experts to ensure consistency with the design requirements, while the direct LLM-based formulations are generated using the same requirements and generation prompt as SHA-PF. The code for the baseline formulation for the HBV and antenna benchmarks are provided in Appendices D.1 and D.2, respectively.

Implementation Details. For all benchmark tasks, we use 100 initial evaluated samples to construct $\mathcal{D}_{\text{init}}$. To ensure a fair comparison, all LLM-based components use GPT-5-mini, and all formulations are represented as executable Python functions. We use SCBO (Eriksson et al., 2019) as the optimization algorithm for all formulations, so that performance differences can be attributed to the formulations rather than the optimizer. All methods are evaluated under the same budget B. For HBV, the budget is 250 evaluations. For each antenna design task, the budget is 300 HFSS simulations, where HFSS (High Frequency Structure Simulator) is run in ANSYS Electronics Desktop 2018.2.0. For SHA-PF, each formulation generation strategy produces one candidate formulation. The population size is set to $P = 5$, and the maximum number of evolutionary generations is set to $G_{\max} = 25$. A sensitivity analysis of these parameters is provided in Appendix A.

Table 1: Search cost of different formulation methods to the first requirement-satisfying antenna design, excluding the initial evaluations. Entries are evaluations / hours; best and second-best successful results for each task are shown in bold and underlined, respectively.

Method / Task		HAE	HSL	HSE	CSL	WHG
Formulation baselines with SCBO	$\mathcal{F}_{\text{Expert-A}}$ (SCBO)	–/13.8	–/8.6	–/15.1	–/31.8	–/17.4
	$\mathcal{F}_{\text{Expert-B}}$ (SCBO)	–/7.8	–/14.2	–/11.1	–/31.2	–/19.1
	$\mathcal{F}_{\text{LLM-A}}$ (SCBO)	<u>71/3.1</u>	–/11.1	–/13.5	36/6.0	–/16.9
	$\mathcal{F}_{\text{LLM-B}}$ (SCBO)	–/11.7	–/13.6	–/12.4	–/28.4	–/15.9
SHA-PF with different optimizers	SHA-PF (SCBO)	**66/2.7**	**85/7.1**	**63/3.9**	<u>10/2.1</u>	**90/8.6**
	SHA-PF (DSI)	85/7.0	–/17.3	–/19.2	**9/1.4**	103/14.8
	SHA-PF (CEBO)	76/6.4	<u>175/16.9</u>	–/22.6	15/5.2	<u>94/10.6</u>

Note: "–" means that no requirement-satisfying design is found within the search budget. Parentheses indicate the optimizer used for each method.

4.2 Main Results

4.2.1 HBV Problem

For the HBV task, the design requirement consists of both feasibility criteria and post-feasibility preferences. A solution is considered feasible if it satisfies NSE ≥ 0.50, TRMSE ≤ 1.60, ROCE ≤ 0.50, and SFDCE ≤ 0.20. Among feasible solutions, higher NSE and lower ROCE are further preferred.

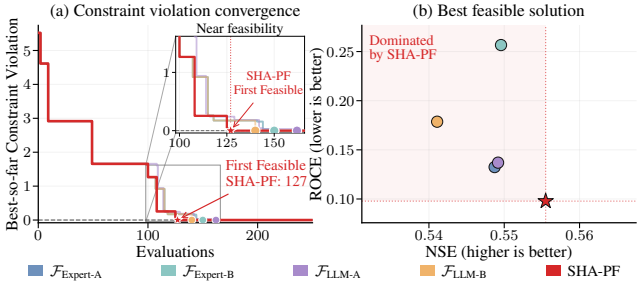


Figure 3: Comparison of feasibility convergence and best feasible solutions on the HBV problem.

Figure 3(a) shows that SHA-PF reaches feasibility at evaluation 127, earlier than all baseline formulations, indicating substantial differences in search efficiency induced by alternative formulations of the same requirements. Figure 3(b) further shows that the best feasible solution obtained by SHA-PF dominates those of all baselines in terms of NSE and ROCE. Overall, under the same optimizer, SHA-PF discovers a formulation that enables both faster feasibility attainment and better post-feasibility performance.

4.2.2 Antenna Design Benchmarks

For the antenna design tasks, the design requirements are defined over simulated frequency-response curves, including passband, stopband, and null regions. The passband and stopband regions impose task-specific response thresholds, while the null regions require radiation nulls to occur within specified frequency ranges. A candidate design is considered requirement-satisfying only if its simulated response curve satisfies all of these requirements. Detailed task-specific requirements are provided in Appendix C.3.

Table 1 reports the number of HFSS simulations and wall-clock time required to find the first requirement-satisfying design. SHA-PF succeeds on all five tasks, whereas the baseline formulations succeed only on HAE and CSL. On these two tasks, SHA-PF reaches satisfying designs with fewer evaluations and lower time cost than the successful baseline $\mathcal{F}_{\text{LLM-A}}$. On HSL, HSE, and WHG, all baselines exhaust the search budget without success, while SHA-PF finds satisfying designs using 85, 63, and 90 evaluations, respectively, saving 110–137 HFSS simulations relative to the budget limit. These results demonstrate substantial differences in search efficiency induced by alternative formulations under the same optimizer.

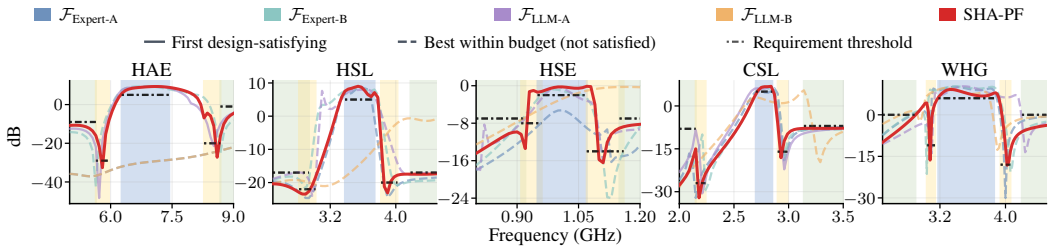


Figure 4: Comparison of formulation methods across five antenna design tasks.

Figure 4 further compares the simulated responses obtained by different formulations. The baseline formulations often improve individual frequency regions while still violating at least one requirement, whereas SHA-PF obtains response curves satisfying all specified regions on all five tasks. Together, the results show that SHA-PF more consistently identifies formulations that lead to requirement-satisfying antenna designs within a limited simulation budget.

4.3 Cross-Optimizer Evaluation

To examine whether the formulation discovered by SHA-PF is tied to a specific optimizer, we evaluate it with different expensive constrained optimization algorithms across the five antenna design tasks. Besides SCBO, we include CEBO (Liu et al., 2025) and DSI (Song et al., 2023). CEBO is a constrained Bayesian optimization method that models constraints with probabilistic surrogates and incorporates feasibility into the acquisition function. DSI is a surrogate-assisted evolutionary algorithm for expensive constrained optimization that improves feasibility prediction by sampling near the boundary of the feasible region. All optimizers use the same SHA-PF formulation and evaluation budget.

Table 1 further reports the cross-optimizer results of SHA-PF. With the same SHA-PF formulation, SCBO succeeds on all five tasks and achieves the best overall efficiency on most of them. DSI obtains the lowest cost on CSL but fails on HSL and HSE, while CEBO succeeds on four tasks but fails on HSE and generally requires more evaluations than SCBO. Nevertheless, SHA-PF with DSI or CEBO still outperforms the formulation baselines in terms of successful tasks, suggesting that optimizer choice is important but not the main reason for the performance gain. Overall, these results show that SHA-PF is not tied to a specific optimizer. The discovered formulation provides a more effective search direction across different optimizers, while stronger optimizers such as SCBO can further improve the efficiency of reaching requirement-satisfying antenna designs.

4.4 Robustness to LLM Backbones

We further examine the sensitivity of SHA-PF to the choice of LLM backbone on the HSE task. Figure 5 (a) shows the formulation-search convergence under five different backbones. Although the initial objective values and convergence speeds vary across models, all backbones eventually reach maximum formulation objective values, suggesting that SHA-PF can refine formulations even when the initial generation quality differs.

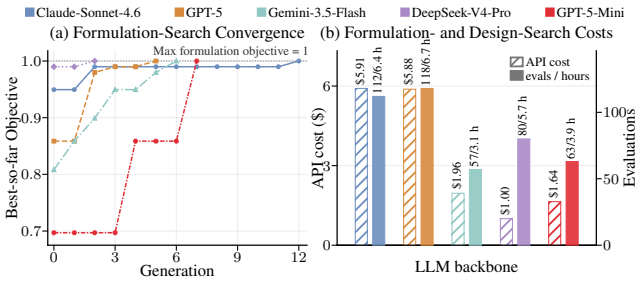


Figure 5: Impact of LLM backbone on formulation search and antenna optimization for the HSE problem.

Figure 5 (b) further compares the cost performance trade-offs of the resulting formulations. Although all backbones can produce effective formulations after refinement, their design-search costs differ. GPT-5 and Claude-Sonnet-4.6 have higher API costs, but their discovered formulations require more than 100 evaluations in the antenna design search. DeepSeek-V4-Pro con-

Table 2: Search cost of different ablation variants to the first requirement-satisfying antenna design, excluding the initial evaluations. Best results are in bold.

Method / Task	HAE	HSL	HSE	CSL	WHG
w/o Anchor Selection	102 / 5.8	92 / 7.8	– / 11.3	12 / 3.2	– / 17.4
w/o Formulation Repair	– / 7.5	– / 13.1	– / 12.1	– / 33.0	– / 17.7
SHA-PF	**66 / 2.7**	**85 / 7.1**	**63 / 3.9**	**10 / 2.1**	**90 / 8.6**

Note: Entries report evaluations / hours during the search. "–" indicates no requirement-satisfying design within the search budget, with hours reported for the full budget.

verges fastest in formulation search and has the lowest API cost, yet its design-search cost remains higher than those of Gemini-3.5-Flash and GPT-5-Mini. By contrast, Gemini-3.5-Flash and GPT-5-Mini achieve lower design-search costs with relatively low API costs, suggesting that lightweight backbones can provide better overall cost efficiency.

Overall, these results suggest that the LLM backbone affects the convergence speed and cost efficiency of SHA-PF, but it is not the primary factor determining whether SHA-PF can produce an effective formulation. SHA-PF remains effective across different LLM backbones and does not require the strongest or most expensive model.

4.5 Ablation Study

We conduct ablation studies on the antenna design benchmarks to evaluate the contribution of two key components in SHA-PF: anchor selection and formulation repair. The first variant, w/o Anchor Selection, replaces the proposed anchor selection based on search hardness with random anchor selection. The second variant, w/o Formulation Repair, removes the repair step after formulation generation. All variants use the same optimizer, initial data, and evaluation budget as SHA-PF.

As shown in Table 2, removing anchor selection consistently reduces search efficiency. The variant succeeds on only three of the five tasks and requires more evaluations than SHA-PF on all successful tasks. Removing formulation repair causes a larger degradation, with no requirement-satisfying design found on any task within the search budget. These results indicate that anchor selection helps identify more informative search states for formulation optimization, while formulation repair is critical for maintaining the reliability of generated formulations. Overall, both components contribute to the robustness and search efficiency of SHA-PF.

5 Conclusion

This work introduces SHA-PF, a search hardness-aware problem formulation framework for expensive simulation-driven design. Unlike existing LLM-based formulation methods that mainly focus on design-intent alignment, SHA-PF further considers how a formulation affects search efficiency. It uses initial evaluated data to identify hard yet promising anchor states and searches for executable formulations that prioritize these states through LLM-based generation, repair, and evolutionary refinement. Experiments on the real-world multi-objective problem and five antenna design benchmarks show that SHA-PF reaches requirement-satisfying designs with fewer evaluations and less search time than expert-based and direct LLM-generated formulations. Cross-optimizer and ablation studies further show that the discovered formulations remain effective across different optimizers and that both anchor selection and formulation repair contribute to reliable performance.

Despite these results, SHA-PF still depends on the initial evaluated data to estimate search hardness, so anchor selection may be less reliable when these samples are sparse or miss hard yet promising regions. Future work may explore iterative anchor discovery by using a small number of additional evaluations to update the satisfaction-state structure during formulation search. The current binary satisfaction state also ignores how close a response is to satisfying each criterion, and incorporating graded satisfaction or violation magnitudes may further improve search hardness estimation.

AI USE STATEMENT

Large language models were used as components of the proposed method to assess requirement satisfaction and to generate, repair, and refine candidate problem formulations. All research ideas, methodological design, experimental design, analysis, and coding were developed by the authors.

REPRODUCIBILITY STATEMENT

The SHA-PF implementation code and data are available at the Anonymous Link. The Experimental Setup specifies the LLM backbone, optimizer, evaluation budgets, and simulator version. The appendix provides the prompt templates, benchmark specifications, HFSS settings, and executable formulations for HBV and the representative antenna task.

REFERENCES

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 577–596. PMLR, July 2024.

Khairul Alam, Tapabrata Ray, and Sreenatha G. Anavatti. Design Optimization of an Unmanned Underwater Vehicle Using Low- and High-Fidelity Models. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(11):2794–2808, November 2017. ISSN 2168-2232. doi: 10.1109/TSMC.2015.2390592.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-Informed RL: Grounding Large Language Models for Authentic Optimization Modeling, May 2025.

Tinkle Chugh, Yaochu Jin, Kaisa Miettinen, Jussi Hakanen, and Karthik Sindhya. A Surrogate-Assisted Reference Vector Guided Evolutionary Algorithm for Computationally Expensive Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation*, 22(1):129–142, February 2018. ISSN 1941-0026. doi: 10.1109/TEVC.2016.2622301.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pp. 9851–9864, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, pp. 2187–2200, Red Hook, NY, USA, December 2021. Curran Associates Inc. ISBN 978-1-7138-4539-3.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. DeepSeek-V3 Technical Report, February 2025.

David Eriksson and Matthias Poloczek. Scalable Constrained Bayesian Optimization. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 730–738. PMLR, March 2021.

David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 5496–5507, Red Hook, NY, USA, December 2019. Curran Associates Inc.

Junlei Guo, Yongpin Chen, Deqiang Yang, Kai Sun, Jin Pan, and Sihao Liu. Design of Wideband Filtering Patch Antenna Array With High Aperture Efficiency and Good Filtering Performance. *IEEE Transactions on Antennas and Propagation*, 72(1):974–979, January 2024. ISSN 1558-2221. doi: 10.1109/TAP.2023.3331253.

Kun-Zhi Hu, Ming-Chun Tang, Dajiang Li, Yang Wang, and Mei Li. Design of Compact, Single-Layered Substrate Integrated Waveguide Filtenna With Parasitic Patch. *IEEE Transactions on Antennas and Propagation*, 68(2):1134–1139, February 2020. ISSN 1558-2221. doi: 10.1109/TAP.2019.2938574.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling. *Operations Research*, pp. opre.2024.1233, May 2025. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.2024.1233.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch. In *The Thirteenth International Conference on Learning Representations*, October 2024.

Yaochu Jin, Handing Wang, Tinkle Chugh, Dan Guo, and Kaisa Miettinen. Data-Driven Evolutionary Optimization: An Overview and Case Studies. *IEEE Transactions on Evolutionary Computation*, 23(3):442–458, June 2019. ISSN 1941-0026. doi: 10.1109/TEVC.2018.2869001.

Yao Lai, Jinxin Liu, David Z. Pan, and Ping Luo. Scalable and effective arithmetic tree generation for adder and multiplier designs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 139151–139178. Curran Associates, Inc., 2024. doi: 10.52202/079017-4416.

Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "I Want It That Way": Enabling Interactive Decision Support Using Large Language Models and Constraint Programming. *ACM Transactions on Interactive Intelligent Systems*, 14(3):22:1–22:33, September 2024. ISSN 2160-6455. doi: 10.1145/3685053.

Jichao Li, Xiaosong Du, and Joaquim R.R.A. Martins. Machine learning in aerodynamic shape optimization. *Progress in Aerospace Sciences*, 134:100849, October 2022. ISSN 03760421. doi: 10.1016/j.paerosci.2022.100849.

Yuchen Li, Handing Wang, Bing Xue, Mengjie Zhang, and Yaochu Jin. Solver-independent automated problem formulation via llms for high-cost simulation-driven design. *arXiv preprint arXiv:2512.18682*, 2026.

Gen-Zhu Liang, Fu-Chang Chen, Hang Yuan, Kai-Ran Xiang, and Qing-Xin Chu. A High Selectivity and High Efficiency Filtering Antenna With Controllable Radiation Nulls Based on Stacked Patches. *IEEE Transactions on Antennas and Propagation*, 70(1):708–713, January 2022. ISSN 1558-2221. doi: 10.1109/TAP.2021.3098563.

Bo Liu, Qingfu Zhang, and Georges G. E. Gielen. A Gaussian Process Surrogate Model Assisted Evolutionary Algorithm for Medium Scale Expensive Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 18(2):180–192, April 2014. ISSN 1941-0026. doi: 10.1109/TEVC.2013.2248012.

Jiao Liu, Yong Wang, Guangyong Sun, and Tong Pang. Constrained Evolutionary Bayesian Optimization for Expensive Constrained Optimization Problems With Inequality Constraints. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 55(3):2009–2021, March 2025. ISSN 2168-2232. doi: 10.1109/TSMC.2024.3504728.

Shuxuan Liu, Zhan Wang, and Yuandan Dong. A Compact Filtering Patch Antenna With High Suppression Level and Its CP Application. *IEEE Antennas and Wireless Propagation Letters*, 22 (4):769–773, April 2023a. ISSN 1548-5757. doi: 10.1109/LAWP.2022.3224845.

Songbai Liu, Qiuzhen Lin, Jianqiang Li, and Kay Chen Tan. A Survey on Learnable Evolutionary Algorithms for Scalable Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 27(6):1941–1961, December 2023b. ISSN 1941-0026. doi: 10.1109/TEVC.2023.3250350.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A Scalable Bidirectional Data Synthesis Framework for Optimization Modeling. In *Forty-Second International Conference on Machine Learning*, June 2025.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao Gong. LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation, March 2024.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, et al. GPT-4 Technical Report, March 2024.

Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-dehkordi, Zirui Zhou, and Yong Zhang. Augmenting Operations Research with Auto-Formulation of Optimization Models From Problem Descriptions. In Yunyao Li and Angeliki Lazaridou (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 29–62, Abu Dhabi, UAE, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-industry.4.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, pp. 189–203. PMLR, August 2023.

P. M. Reed, D. Hadka, J. D. Herman, J. R. Kasprzyk, and J. B. Kollat. Evolutionary multiobjective optimization in water resources: The past, present, and future. *Advances in Water Resources*, 51: 438–456, January 2013. ISSN 0309-1708. doi: 10.1016/j.advwatres.2012.01.005.

Zhenshou Song, Handing Wang, and Yaochu Jin. A Surrogate-Assisted Evolutionary Framework With Regions of Interests-Based Data Selection for Expensive Constrained Optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(10):6268–6280, October 2023. ISSN 2168-2232. doi: 10.1109/TSMC.2023.3281822.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, et al. Gemini: A Family of Highly Capable Multimodal Models, May 2025.

Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. Recent Advances in Bayesian Optimization. *ACM Computing Surveys*, 55(13s):287:1–287:36, July 2023. ISSN 0360-0300. doi: 10.1145/3582078.

Andrew M Wissink, Matthew Liu, Alec M House, Andrew Kaminsky, Andrew Simin, and Rakesh Ranjan. Multi-fidelity surrogate modeling for aerodynamics using sage. In *AIAA SCITECH 2025 Forum*, pp. 0039, 2025.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *The Twelfth International Conference on Learning Representations*, October 2023.

Yiteng Xu, Qingbin Guo, Haoran Gu, Handing Wang, and Junchi Yan. Self-Guided Simulation-Driven Optimization for Expensive Filtering Patch Antenna Design and a Benchmark. *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2026. ISSN 1941-0026. doi: 10.1109/TEVC.2026.3662703.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling, June 2025.

P. L. Yu. Cone convexity, cone extreme points, and nondominated solutions in decision problems with multiobjectives. *Journal of Optimization Theory and Applications*, 14(3):319–377, September 1974. ISSN 1573-2878. doi: 10.1007/BF00932614.

Hang Yuan, Fu-Chang Chen, and Qing-Xin Chu. A Wideband and High Gain Dual-Polarized Filtering Antenna Based on Multiple Patches. *IEEE Transactions on Antennas and Propagation*, 70 (10):9843–9848, October 2022. ISSN 1558-2221. doi: 10.1109/TAP.2022.3177494.

Jihai Zhang, Wei Wang, Siyan Guo, Li Wang, Fangquan Lin, Cheng Yang, and Wotao Yin. Solving General Natural-Language-Description Optimization Problems with Large Language Models. In Yi Yang, Aida Davani, Avi Sil, and Anoop Kumar (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pp. 483–490, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-industry.42.

Yisong Zhang, Ran Cheng, Guoxing Yi, and Kay Chen Tan. A Systematic Survey on Large Language Models for Evolutionary Optimization: From Modeling to Solving, September 2025.

Table 3: Parameter sensitivity of SHA-PF on the HSE antenna design task. Best successful result in each group is shown in bold.

Parameter	Value	Search Cost(Evaluations / Hours)
Population size sensitivity ($G_{\max} = 25$)		
P	3	–/11.1
P	5	63/3.9
P	7	**62/3.8**
P	10	–/13.2
Maximum generation sensitivity ($P = 5$)		
$G_{\max}$	3	–/12.4
$G_{\max}$	10	**46/3.3**
$G_{\max}$	25	63/3.9
$G_{\max}$	40	87/6.3

Note: P and $G_{\max}$ denote the population size and the maximum number of evolutionary generations, respectively. Entries report evaluations / hours during the search, excluding the initial evaluations. "–" indicates that no requirement-satisfying design was found within the budget of 200 evaluations, with hours reported for the full budget.

A Parameter Sensitivity Analysis

We investigate the sensitivity of SHA-PF to the population size P and the maximum number of evolutionary generations $G_{\max}$ on the HSE antenna design task. When varying P, we fix $G_{\max} = 25$; when varying $G_{\max}$, we fix $P = 5$.

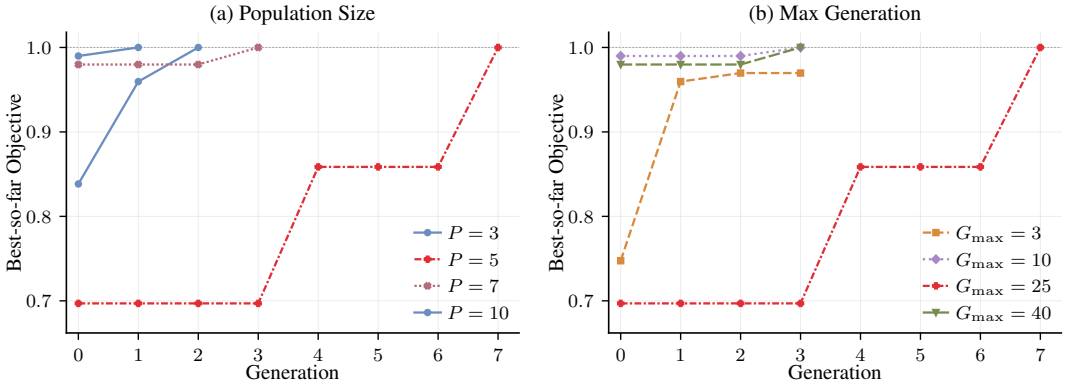


Figure 6: Convergence of the best-so-far formulation search objective under different population sizes and maximum numbers of evolutionary generations on the HSE antenna design task.

Figure 6 shows the convergence of the best-so-far objective during formulation search. As shown in Figure 6(a), all tested population sizes successfully attain the maximum objective value, while their convergence rates differ because the randomly initialized populations contain candidate formulations of different quality. Figure 6(b) further shows that a small $G_{\max}$ can terminate formulation search before the maximum objective value is reached when the initial population is of relatively low quality. Allowing more generations provides additional opportunities to improve such populations and reach the maximum objective value.

Table 3 evaluates the formulation obtained under each parameter setting in the antenna design search. The settings $P = 5$ and $P = 7$ succeed with similar costs, whereas $P = 3$ and $P = 10$ fail within the simulation budget. This suggests that a moderate population provides a better balance between formulation diversity and focused refinement. A small population may limit diversity, whereas a large population may make selection less focused because of the increased number of competing candidates. For the generation limit, $G_{\max} = 10$ achieves the lowest cost, $G_{\max} = 25$ and 40 also

succeed, and $G_{\max} = 3$ fails. Thus, too few generations may terminate refinement prematurely, while additional generations provide no consistent benefit once a strong formulation is found. We use $P = 5$ and $G_{\max} = 25$ as the default setting in the main experiments.

B PROMPT TEMPLATE

This appendix gives the prompt templates used in SHA-PF. The templates cover satisfaction-state judging, formulation generation, formulation repair, and evolutionary operator prompting. Placeholders such as {*Design_Requirements*} and {*Curve_Data*} denote task-specific content supplied at run time.

B.1 STATE JUDGE PROMPT TEMPLATE

The state judge maps a simulated response to a binary satisfaction-state representation. Given the sampled response curve and the natural-language design requirements, the judge assigns one compliance label to each requirement region. These labels form the satisfaction states used for anchor-state selection and formulation evaluation.

Prompt 1: Response-State Judge Prompt Template

Input

Curve Data; Design Requirements; Background Context.

Output

JSON Object: Compliance Vector.

Prompt Text

Role

You are an expert evaluator for simulation-derived performance curves.

Task

{*Background_Context*}. You will be given one performance curve as a list of sampled (x, y) data points in ascending order of x. For each numbered region in the design requirements, judge whether the curve meets the stated requirement. Use binary labels: 1 means that the requirement is met, and 0 means that the requirement is not met.

Rules

Use only the provided sampled points when making the judgment. Do not interpolate, extrapolate, smooth the curve, or assume unseen values between sampled points. Evaluate each region independently. For regions that require a local extremum or notch-like behavior, a boundary point may be accepted only if the sampled points support a genuine local feature. If the boundary value is merely part of a monotonic trend continuing into a neighboring region, it should not be treated as a valid local feature.

Design Requirements

{*Design_Requirements*}

Curve Data

{*Curve_Data*}

Output Format

Return a single valid JSON object with exactly the following field:
`{"compliance":[r1,r2,...,rN]}`
Each r_i must be either 0 or 1. The array order must match the region numbering. Do not output any text outside the JSON object.

B.2 Formulation Generation Prompt Template

The formulation generation prompt produces an executable optimization formulation from the design requirements. The generated formulation maps a simulated response curve to one minimization objective and zero or more constraints.

Prompt 2: Formulation Generation Prompt Template

Input

Background Context; Design Requirements.

Output

JSON Object: Objective Function; Constraint Functions.

Prompt Text

Role

You are an expert in simulation-driven engineering optimization and problem formulation.

Task

Generate one executable optimization formulation for an expensive black-box simulation-driven design problem. Each candidate design is evaluated as a sampled performance curve. The formulation should help an optimizer search efficiently for curves satisfying the design requirements under a limited evaluation budget, rather than merely restating the requirements as direct threshold checks.

For each evaluated curve, the formulation must produce one objective value and zero or more constraint values. Candidate solutions will be ranked by a feasibility-based rule: feasible solutions are preferred over infeasible ones; if both are infeasible, the one with smaller overall constraint violation is preferred; if both are feasible, the one with the better objective value is preferred.

Guidance

The design requirements define final success, but the formulation should provide smoother and more informative search guidance. Prefer graded quality or violation signals that distinguish partially improved curves. Avoid relying only on hard binary checks or trivial one-to-one restatements of the acceptance rules when continuous signals are possible.

Formulation Rules

Use a unified minimization convention for objectives. Express every constraint in the standard form $g(\cdot) \leq 0$, where a constraint is satisfied when its returned value is less than or equal to zero. Use exactly one objective function named `obj1`. The number of constraint functions may be zero, one, or more. A formulation may use composite constraints that jointly represent multiple design intentions, as long as it remains consistent with the requirements and provides useful optimization guidance.

Design Context

{*Background_Context*}

Design Requirements

{*Design_Requirements*}

Output Format

Return exactly one valid JSON object with keys `"objectives"` and `"constraints"`. The `"objectives"` field must be an array containing one executable Python function string. The objective function must use the signature `def obj1(curve): ...`, take `curve` as a 2D NumPy array whose first column is the independent variable and second column is the performance value, and return a scalar float to be minimized. The `"constraints"` field must be an array of executable Python function strings. Each constraint function must be named `cK`, use the signature `def cK(curve): ...`, take the same `curve` input, and return a scalar float such

that the constraint is satisfied when the value is less than or equal to zero. Use NumPy only, and do not output any text outside the JSON object.

B.3 Formulation Repair Prompt Template

The repair prompt checks a generated formulation before evaluation. It targets clear logical errors and mismatches with the design requirements. When such a problem is found, the LLM makes a minimal local edit while preserving the original formulation structure whenever possible.

Prompt 3: Formulation Repair Prompt Template

Input

Design Requirements; Candidate Formulation.

Output

JSON Object: Repair Summary; Repaired Formulation.

Prompt Text

Role

You are a checker and repairer for optimization formulations used in expensive black-box simulation optimization.

Task

Inspect the given formulation and repair it only when there is a clear logical error or a clear mismatch with the design intent. This is a repair task, not a reformulation or performance-improvement task. Do not rewrite, simplify, or redesign the formulation merely because another version may look cleaner, simpler, or potentially better.

The formulation contains exactly one objective function and zero or more constraint functions. Its purpose is to provide meaningful search guidance during optimization while remaining aligned with the design requirements. The repaired formulation should remain an optimization-guiding formulation rather than a direct restatement of the acceptance rules.

Checks

Only check for clear sign-direction errors, clear mismatch between a term and the stated design intent, clear logical conflict between the objective and constraints, duplicated encouragement or duplicated penalty that creates logical imbalance, missing handling of important empty-mask, boundary, or abnormal cases, or formulation logic that would clearly mislead the optimizer away from the intended behavior.

Repair Rules

When repair is necessary, preserve the original structure as much as possible. Keep exactly one objective function, and keep the same number of constraints unless one is clearly invalid and cannot be minimally repaired. Prefer minimal local edits, such as adjusting signs, thresholds, local terms, local aggregations, or boundary handling, rather than rewriting full functions. Do not modify unaffected parts. If no clear logical problem is found, return the original formulation unchanged. When in doubt, keep the original formulation unchanged.

Formulation Conventions

The objective function is defined for minimization, so lower objective values must indicate better solutions. Each constraint must follow the convention $g(\texttt{curve}) \leq 0$ means satisfied and $g(\texttt{curve}) > 0$ means violated. The repaired formulation must preserve these conventions and must not reverse the optimization direction.

Design Requirements

{*Design_Requirements*}

Formulation to Inspect

{*Candidate_Formulation*}

Output Format

Return exactly one valid JSON object with keys **repair summary** and **repaired formulation**. The repair summary must briefly state whether any clear logical problem was found and, if so, what was minimally repaired. If no clear problem was found, explicitly state that the original formulation was kept unchanged.

The repaired formulation must be a JSON object with keys **objectives** and **constraints**. The objectives field must contain exactly one executable Python function string, and the constraints field must contain an array of executable Python function strings. Do not output any text outside the JSON object.

B.4 Operator Prompts

The operator prompts below guide evolutionary formulation refinement. Each operator is inserted into the offspring-generation prompt together with selected parent formulations. We list the operators separately because they only specify how a new candidate should differ from its parent formulation(s); the shared context and output format follow the formulation generation template.

Operator 1: Component Recombination

Create one new formulation by recombining the most useful parts of the two parent formulations. Keep the components that already provide effective optimization guidance, and combine them into a stronger formulation rather than making only superficial changes.

Operator 2: Constraint Restructuring

Create one structurally different new formulation based on the parent formulation. Actively explore a different constraint organization, such as using fewer constraints, more constraints, or composite constraints across multiple regions. Do not limit yourself to the parent's current constraint structure.

Operator 3: Signal Reparameterization

Create one new formulation by exploring a different way to define the objective signal and/or constraint-violation signal from the parent formulation. You may change how objective values or constraint violations are measured, aggregated, or combined, as long as the formulation remains aligned with the design intent and provides useful optimization guidance.

Operator 4: Semantic Exploration

Create one new formulation that is intentionally more different from the parent formulation. You may substantially change the objective definition, the number of constraints, or the way regional design intentions are organized, as long as the resulting formulation remains executable and aligned with the design requirements. The goal is to explore a new formulation family rather than making only small local edits.

C Benchmark Problems

We evaluate SHA-PF on the real-world multi-objective benchmark and five antenna design tasks.

C.1 HBV Rainfall–Runoff Benchmark

HBV is a lumped conceptual rainfall–runoff model used here as a real-world multi-objective calibration benchmark (Reed et al., 2013). The benchmark calibrates the model for the Williams River, West Virginia, United States (USGS Gage 03186500), using precipitation and streamflow records from the MOPEX dataset. The native benchmark simulates an 11-year period from October 1, 1961 to September 30, 1972, where the first year is used as warmup and the remaining 10 years are used for objective evaluation. Each candidate solution is a 14-dimensional real-valued parameter vector, and evaluating it runs the HBV simulator to produce modeled daily streamflow.

Let $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{14}$ denote an HBV calibration vector. The feasible parameter domain is $\mathcal{X} = [\ell_1, u_1] \times \cdots \times [\ell_{14}, u_{14}]$ shown in Table 5. For each $\mathbf{x}$, the simulator returns a modeled daily

Table 4: Summary of the HBV benchmark setting.

Item	Description
Task	HBV rainfall–runoff model calibration
Site and data	Williams River, West Virginia; MOPEX precipitation and streamflow records
Calibration window	10 post-warmup years within the 1961–1972 simulation period
Problem size	14 continuous variables and 4 objectives
Evaluation budget	250 simulator evaluations

Table 5: Decision variables and bounds in the HBV benchmark.

Variable	Range	Role
L (mm)	$[0, 100]$	Response-routine storage threshold
K_0 (d)	$[0.5, 20]$	Fast reservoir recession time scale
K_1 (d)	$[1, 100]$	Intermediate reservoir recession time scale
K_2 (d)	$[10, 20000]$	Slow reservoir recession time scale
Perc (mm/d)	$[0, 100]$	Percolation rate
LP	$[0.3, 1]$	Evapotranspiration reduction limit
Fcap (mm)	$[0, 2000]$	Soil moisture storage capacity
B	$[0, 7]$	Soil moisture shape parameter
MaxBas (d)	$[24, 120]$	Routing/base length parameter
TT (°C)	$[-3, 3]$	Temperature threshold for snow/rain partitioning
DDF	$[0, 20]$	Degree-day snowmelt factor
CFR	$[0, 1]$	Refreezing coefficient
CWH	$[0, 0.8]$	Liquid-water holding capacity in snow
TTI (°C)	$[0, 7]$	Temperature transition interval

streamflow sequence $\hat{q}_{1:N}(\mathbf{x})$, which is compared with the observed streamflow sequence $q_{1:N}$ and precipitation sequence $p_{1:N}$ after the warmup period.

Table 6 summarizes the four criteria used in the original benchmark. More explicitly, using $\hat{q}_t$ for simulated flow, q_t for observed flow, and Y for the number of post-warmup calibration years, the implementation computes TRMSE by applying the Box–Cox map $b(z) = (z^{0.3} - 1)/0.3$ to daily flows:

$$\mathrm{TRMSE}(\mathbf{x}) = \left(\frac{1}{N} \sum_{t=1}^{N} \left[b(\hat{q}_t(\mathbf{x})) - b(q_t) \right]^2 \right)^{1/2}. \tag{8}$$

The runoff coefficient error is computed from annual mean simulated flow, observed flow, and precipitation:

$$\mathrm{ROCE}(\mathbf{x}) = \left| \frac{1}{Y} \sum_{y=1}^{Y} \frac{\bar{\hat{q}}_y(\mathbf{x}) - \bar{q}_y}{\bar{p}_y} \right|. \tag{9}$$

The flow-duration-curve slope error compares the middle-slope segment of the simulated and observed sorted flow curves:

$$\mathrm{SFDCE}(\mathbf{x}) = \left| \frac{(\hat{q}_{0.67} - \hat{q}_{0.33}) - (q_{0.67} - q_{0.33})}{q_{0.67} - q_{0.33}} \right|, \tag{10}$$

where q_α and $\hat{q}_\alpha$ denote the corresponding empirical flow-duration quantiles. The NSE objective follows the standard efficiency form:

$$\mathrm{NSE}(\mathbf{x}) = 1 - \frac{\sum_{t=1}^{N} (\hat{q}_t(\mathbf{x}) - q_t)^2}{\sum_{t=1}^{N} (q_t - \bar{q})^2}. \tag{11}$$

The design requirement for this benchmark are given in Section C.3.

Table 6: Objectives in the HBV calibration benchmark.

Objective	Direction	Hydrological meaning
NSE	Maximize	Nash–Sutcliffe efficiency; emphasizes high-flow accuracy
TRMSE	Minimize	RMSE after a Box–Cox transformation with exponent 0.3; emphasizes low-flow accuracy
ROCE	Minimize	Absolute runoff coefficient error averaged over calibration years; measures long-term water balance
SFDCE	Minimize	Absolute percent error in the middle-slope segment of the flow-duration curve; measures flow variability

C.2 Antenna Design Benchmarks

The antenna suite contains five filtering patch antenna design tasks: HAE (Guo et al., 2024), CSL (Hu et al., 2020), HSL (Liu et al., 2023a), WHG (Yuan et al., 2022), and HSE (Liang et al., 2022). For each task, a candidate solution is a vector of continuous geometric parameters in millimeters. Instantiating this vector in the antenna model defines one HFSS simulation, and the simulator returns a sampled frequency-domain response over the task-specific sweep range.

All antenna evaluations are performed in ANSYS Electronics Desktop HFSS 2018.2 using a common adaptive solution configuration. As shown in Table 7, the adaptive solver is limited to 15 passes with a maximum ΔS tolerance of 0.02. The frequency-sweep sample count specifies the number of response points returned by the simulator for each antenna; these sampled responses are the curves used by the requirement state stage.

Table 7: HFSS solution settings for the antenna simulations.

Adaptive solution		Frequency sweep samples				
Max passes	Max ΔS	HAE	CSL	HSL	WHG	HSE
15	0.02	51	51	51	101	41

Table 8: Summary of the five antenna design benchmarks.

Antenna	Structure	Full frequency range (GHz)	Dim.	Center freq. (GHz)	Bandwidth (%)	Avg. eval. time (s)
HAE	1 layer; $0.68 \times 1.14\,\lambda_0^2$	5.00–9.00	14	6.85	17.0	532
CSL	1 layer; $0.74 \times 0.45\,\lambda_0^2$	2.00–3.50	13	2.77	5.1	963
HSL	2 layers; $0.38 \times 0.38\,\lambda_0^2$	2.50–4.50	19	3.50	9.7	713
WHG	3 layers; $0.86 \times 0.86\,\lambda_0^2$	2.50–4.50	15	3.48	18.8	853
HSE	3 layers; $1.00 \times 1.00\,\lambda_0^2$	0.80–1.20	23	1.00	16.0	511

Table 8 lists the structure, dimensionality, sweep range, and average HFSS evaluation time for each antenna. The tasks contain 13–23 design variables, with average simulation times ranging from 511 to 963 seconds.

Each run uses a budget of 300 HFSS evaluations. The feasible design domain is the Cartesian product of the per-parameter intervals in Table 9. All candidate geometries submitted to HFSS must lie within these bounds.

Table 9: Search-space settings for the antenna design variables. All values are in millimeters.

HAE	l_g	w_g	w_s	l_s	d_1	d	d_2	d_3	w_{s1}	l_{s1}	w_t	l_t	l_{t1}	h
Min	49	20	0.2	4	2.0	2	4.0	3.00	0.2	2	0.3	0.8	4.5	1.5
Max	60	40	1.0	8	5.0	5	11.0	10.00	1.0	6	0.7	1.2	6.5	2.5

CSL	g	a	f_a	f	m_w	m_l	d	d_1	W_p	L_p	w_w	l_l	L_s
Min	0.6	0.5	3	8	1	5.0	4	13	30	30	6	1.5	1.8
Max	2.0	1.5	15	15	4	12.0	12	16	40	36	10	2.5	2.2

HSL	w_g	l_g	w	l	pw	pl	w_s	l_{s1}	w_{s2}	l_{s2}	w_1	l_1	w_2	l_2	l_{i1}	l_{i2}	d	h	hh
Min	60	60	40	40	25.0	25.0	2.2	20.0	2.2	20.0	1.0	5.0	0.3	5.0	2.0	2.0	8	2.5	7.0
Max	70	70	50	50	35.0	35.0	4.0	40.0	4.0	40.0	2.0	10.0	1.0	10.0	4.0	4.0	15	3.5	8.0

WHG	h	h_a	h_b	l_g	w_p	l_{p1}	l_{p2}	l_{p3}	l_f	w_f	d_f	l_s	w_s	d	g
Min	0.6	2	2.0	90	7.5	26	44	20	10.0	1.0	5	12	1.5	1.8	0.9
Max	1.4	4	3.5	110	8.5	30	46	32	30.0	3.0	15	20	2.5	2.2	1.1

HSE	l_g	d	d_p	h_f	w_{fp}	l_{fp}	d_{po}	t_{fp}	l_{p1}	g_f	t	l_{p2}	t_m	g_{p1}	g_{p2}	l_{p3}	l_{s1}	w_{s1}	l_{s2}	w_{s2}	l_{s3}	w_{s3}	k
Min	280	80	0.5	4.0	4	20	1.5	0.2	80	1.0	0.3	80	0.05	4	4.0	80	10	1	30	1	30	1.0	0.10
Max	320	120	1.5	8.0	12	32	3.0	0.8	120	2.0	0.8	120	1.00	12	12.0	120	50	5	80	5	80	5.0	0.30

C.3 Design Requirements

We use the following design requirements to judge whether a simulated response satisfies the target behavior. The HBV requirements specify hydrological thresholds and post-feasibility preferences. The antenna requirements specify the target filtering response through frequency regions and threshold values.

Requirement 1: HBV Design Requirement Specification

Design Intent

The calibrated HBV parameter setting must satisfy all four hydrological thresholds. NSE is a maximization metric and should be at or above its threshold, whereas TRMSE, ROCE, and SFDCE are minimization metrics and should each be at or below their thresholds. After all thresholds are satisfied, the formulation should prefer samples with larger NSE and smaller ROCE. TRMSE and SFDCE remain secondary tie-breaking signals once they are within their acceptable ranges.

Final Acceptance Criteria

Criterion 1. *NSE*: m_0 must satisfy $m_0 \geq 0.50$.

Criterion 2. *TRMSE*: m_1 must satisfy $m_1 \leq 1.60$.

Criterion 3. *ROCE*: m_2 must satisfy $m_2 \leq 0.50$.

Criterion 4. *SFDCE*: m_3 must satisfy $m_3 \leq 0.20$.

The five antenna tasks use the same requirement template, with task-specific frequency intervals and dB thresholds. Requirement 2 gives the common template, and Table 10 reports the task-specific thresholds.

Requirement 2: Antenna Design Requirement Template

Design Intent

The engineer expects the final optimized curve to exhibit a band-pass-like response with two notch regions and suppressed responses on both sides. Specifically, the response should remain low in the low-frequency stopband, form a clear notch in the low-frequency null region, stay high across the central passband, form a deeper notch in the high-frequency null region, and remain suppressed again in the high-frequency stopband. Overall, the desired curve should progressively approach a stopband-null-passband-null-stopband shape.

Final Acceptance Criteria

Region 1. *Low-frequency stopband*: All sampled values in $[\ell_{\mathrm{LS}}, u_{\mathrm{LS}}]$ must be no higher than τ_{LS}.

Table 10: Task-specific antenna requirement intervals and thresholds. Intervals are in GHz and thresholds are in dB.

Antenna	LS: $[\ell_{\mathrm{LS}}, u_{\mathrm{LS}}]$; τ_{LS}	LN: $[\ell_{\mathrm{LN}}, u_{\mathrm{LN}}]$; τ_{LN}	PB: $[\ell_{\mathrm{PB}}, u_{\mathrm{PB}}]$; τ_{PB}	HN: $[\ell_{\mathrm{HN}}, u_{\mathrm{HN}}]$; τ_{HN}	HS: $[\ell_{\mathrm{HS}}, u_{\mathrm{HS}}]$; τ_{HS}
HAE	$[5.00, 5.64]; -9$	$[5.64, 5.96]; -29$	$[6.27, 7.43]; 5$	$[8.28, 8.68]; -20$	$[8.68, 9.00]; -1$
CSL	$[2.00, 2.15]; -8$	$[2.15, 2.24]; -27$	$[2.70, 2.85]; 5$	$[2.90, 2.99]; -16$	$[3.14, 3.50]; -7$
HSL	$[2.50, 2.94]; -17$	$[2.82, 3.02]; -22$	$[3.38, 3.74]; 5$	$[3.82, 4.02]; -20$	$[4.18, 4.50]; -17$
WHG	$[2.50, 2.90]; 0$	$[3.04, 3.14]; -11$	$[3.18, 3.86]; 6$	$[3.94, 4.06]; -18$	$[4.20, 4.50]; 0$
HSE	$[0.80, 0.92]; -7$	$[0.91, 0.96]; -8$	$[0.95, 1.07]; -2$	$[1.07, 1.16]; -14$	$[1.15, 1.20]; -7$

Region 2. *Low-frequency null region*: The sampled curve in $[\ell_{\mathrm{LN}}, u_{\mathrm{LN}}]$ must contain a distinct local minimum whose value is no higher than τ_{LN}.

Region 3. *Passband*: All sampled values in $[\ell_{\mathrm{PB}}, u_{\mathrm{PB}}]$ must be at least τ_{PB}.

Region 4. *High-frequency null region*: The sampled curve in $[\ell_{\mathrm{HN}}, u_{\mathrm{HN}}]$ must contain a distinct local minimum whose value is no higher than τ_{HN}.

Region 5. *High-frequency stopband*: All sampled values in $[\ell_{\mathrm{HS}}, u_{\mathrm{HS}}]$ must be no higher than τ_{HS}.

D FORMULATIONS

We provide the executable formulation functions used by SHA-PF and all formulation baselines. For readability, comments are omitted; each block contains only the code supplied to the optimizer. All formulations follow the same interface and optimization convention: each function maps the simulator response to one scalar objective and zero or more scalar constraints, objectives are minimized, and a constraint is satisfied when its returned value is no greater than zero.

We construct four formulation baselines. Expert-A and Expert-B are independently developed by two domain experts from the same design requirements. For the LLM baselines, GPT-5-mini is invoked 20 times using the same task information and prompt provided in Section B.2. All generated formulations are manually inspected to exclude invalid or requirement-inconsistent formulations, after which the two experts independently select the formulation they consider most promising, yielding LLM-A and LLM-B. For all methods, the simulator, initial evaluations, SCBO implementation, evaluation budget, and success test are fixed, such that only the formulation supplied to the optimizer differs.

D.1 HBV FORMULATIONS

The HBV formulations operate on the four metrics ordered as NSE, TRMSE, ROCE, and SFDCE.

SHA-PF

Formulation 1: HBV SHA-PF

```python
def obj1(metrics):
    NSE = float(metrics[0])
    TRMSE = float(metrics[1])
    ROCE = float(metrics[2])
    SFDCE = float(metrics[3])
    t_NSE = 0.5
    t_TRMSE = 1.6
    t_ROCE = 0.5
    t_SFDCE = 0.2
    short_NSE = max(0.0, t_NSE - NSE)
    excess_TRMSE = max(0.0, TRMSE - t_TRMSE)
    excess_ROCE = max(0.0, ROCE - t_ROCE)
    excess_SFDCE = max(0.0, SFDCE - t_SFDCE)
    scale_NSE = 1.0
    scale_TRMSE = 2.0
    scale_ROCE = 1.0
    scale_SFDCE = 1.0
    pen_NSE = (short_NSE / scale_NSE) ** 2
    pen_TRMSE = (excess_TRMSE / scale_TRMSE) ** 2
    pen_ROCE = (excess_ROCE / scale_ROCE) ** 2
```

```
    pen_SFDCE = (excess_SFDCE / scale_SFDCE) ** 2
    infeasibility_penalty = 100.0 * (pen_NSE + pen_TRMSE + pen_ROCE + pen_SFDCE)
    feasible = short_NSE <= 0.0 and excess_TRMSE <= 0.0 and (excess_ROCE <= 0.0) and (
     excess_SFDCE <= 0.0)
    if not feasible:
        tie_break = -5.0 * max(0.0, min(1.0, NSE)) + 1.0 * TRMSE + 1.0 * ROCE + 0.5 * SFDCE
        return infeasibility_penalty + tie_break
    else:
        effective_NSE = min(NSE, 0.7)
        if NSE < 0.7:
            w1 = 20.0
            w2 = 10.0
        else:
            w1 = 10.0
            w2 = 20.0
        w3 = 2.0
        w4 = 2.0
        obj_value = -w1 * effective_NSE + w2 * ROCE + w3 * TRMSE + w4 * SFDCE
        obj_value += 1e-06 * (1.0 - NSE)
        return obj_value

def c1(metrics):
    NSE = float(metrics[0])
    return max(0.0, 0.5 - NSE)

def c2(metrics):
    TRMSE = float(metrics[1])
    return max(0.0, TRMSE - 1.6)

def c3(metrics):
    ROCE = float(metrics[2])
    return max(0.0, ROCE - 0.5)

def c4(metrics):
    SFDCE = float(metrics[3])
    return max(0.0, SFDCE - 0.2)
```

Baselines

The baselines include two expert-designed formulations and two direct LLM-generated formulations.

Formulation 2: HBV $\mathcal{F}_{\text{Expert-A}}$

```
def obj1(metrics):
    try:
        import numpy as np
        arr = np.asarray(metrics, dtype=float)
        if arr.ndim == 1:
            return -float(arr[0]) + float(arr[2])
        if arr.ndim == 2 and arr.shape[0] >= 1:
            return -float(arr[0, 0]) + float(arr[0, 2])
        raise ValueError('metrics must be a 1D row or a 2D matrix with at least one row.')
    except Exception:
        row = [float(value) for value in metrics]
        return -float(row[0]) + float(row[2])

def c1(metrics):
    try:
        import numpy as np
        arr = np.asarray(metrics, dtype=float)
        if arr.ndim == 1:
            value = float(arr[0])
        elif arr.ndim == 2 and arr.shape[0] >= 1:
            value = float(arr[0, 0])
        else:
            raise ValueError('metrics must be a 1D row or a 2D matrix with at least one row.')
    except Exception:
        row = [float(value) for value in metrics]
        value = float(row[0])
    return 0.5 - value

def c2(metrics):
    try:
        import numpy as np
        arr = np.asarray(metrics, dtype=float)
        if arr.ndim == 1:
```

```
            value = float(arr[1])
        elif arr.ndim == 2 and arr.shape[0] >= 1:
            value = float(arr[0, 1])
        else:
            raise ValueError('metrics must be a 1D row or a 2D matrix with at least one row.')
    except Exception:
        row = [float(value) for value in metrics]
        value = float(row[1])
    return value - 1.6

def c3(metrics):
    try:
        import numpy as np
        arr = np.asarray(metrics, dtype=float)
        if arr.ndim == 1:
            value = float(arr[2])
        elif arr.ndim == 2 and arr.shape[0] >= 1:
            value = float(arr[0, 2])
        else:
            raise ValueError('metrics must be a 1D row or a 2D matrix with at least one row.')
    except Exception:
        row = [float(value) for value in metrics]
        value = float(row[2])
    return value - 0.5

def c4(metrics):
    try:
        import numpy as np
        arr = np.asarray(metrics, dtype=float)
        if arr.ndim == 1:
            value = float(arr[3])
        elif arr.ndim == 2 and arr.shape[0] >= 1:
            value = float(arr[0, 3])
        else:
            raise ValueError('metrics must be a 1D row or a 2D matrix with at least one row.')
    except Exception:
        row = [float(value) for value in metrics]
        value = float(row[3])
    return value - 0.2
```

Formulation 3: HBV $\mathcal{F}_{\text{Expert-B}}$

```
def obj1(metrics):
    try:
        nse = float(metrics[0])
        roce = float(metrics[2])
    except Exception:
        nse = float(metrics[0][0])
        roce = float(metrics[0][2])
    if nse <= 0.0:
        return 1000000.0 + roce
    return -nse / (1.0 + roce)

def c1(metrics):
    try:
        value = float(metrics[0])
    except Exception:
        value = float(metrics[0][0])
    return 0.5 - value

def c2(metrics):
    try:
        value = float(metrics[1])
    except Exception:
        value = float(metrics[0][1])
    return value - 1.6

def c3(metrics):
    try:
        value = float(metrics[2])
    except Exception:
        value = float(metrics[0][2])
    return value - 0.5

def c4(metrics):
    try:
        value = float(metrics[3])
    except Exception:
```

```
        value = float(metrics[0][3])
    return value - 0.2
```

Formulation 4: HBV $\mathcal{F}_{\text{LLM-A}}$

```
def obj1(metrics):
    NSE = metrics[0]
    TRMSE = metrics[1]
    ROCE = metrics[2]
    SFDCE = metrics[3]
    w_nse = 5.0
    w_roce = 3.0
    w_trmse = 1.0
    w_sfdce = 1.0
    t_nse = 0.5
    t_trmse = 1.6
    t_roce = 0.5
    t_sfdce = 0.2
    eps = 1e-08
    deficit_nse = max(0.0, t_nse - NSE)
    bonus_nse = 0.0
    if NSE >= t_nse:
        bonus_nse = -0.2 * (NSE - t_nse)
    cost_nse = deficit_nse + bonus_nse
    excess_trmse = max(0.0, TRMSE - t_trmse) / (t_trmse + eps)
    excess_roce = max(0.0, ROCE - t_roce) / (t_roce + eps)
    excess_sfdce = max(0.0, SFDCE - t_sfdce) / (t_sfdce + eps)
    base_trmse = TRMSE / (t_trmse + eps)
    base_roce = ROCE / (t_roce + eps)
    base_sfdce = SFDCE / (t_sfdce + eps)
    obj = w_nse * cost_nse + w_roce * (base_roce + excess_roce) + w_trmse * (base_trmse + 0.5
     * excess_trmse) + w_sfdce * (base_sfdce + 0.5 * excess_sfdce)
    if deficit_nse == 0.0 and excess_trmse == 0.0 and (excess_roce == 0.0) and (excess_sfdce
     == 0.0):
        obj = obj - 2.0 * (NSE - t_nse) + 1.5 * (ROCE - t_roce)
    return float(obj)

def c1(metrics):
    NSE = metrics[0]
    TRMSE = metrics[1]
    ROCE = metrics[2]
    SFDCE = metrics[3]
    t_nse = 0.5
    t_trmse = 1.6
    t_roce = 0.5
    t_sfdce = 0.2
    v_nse = max(0.0, t_nse - NSE) / (abs(t_nse) + 1e-08)
    v_trmse = max(0.0, TRMSE - t_trmse) / (t_trmse + 1e-08)
    v_roce = max(0.0, ROCE - t_roce) / (t_roce + 1e-08)
    v_sfdce = max(0.0, SFDCE - t_sfdce) / (t_sfdce + 1e-08)
    max_violation = max(v_nse, v_trmse, v_roce, v_sfdce)
    return float(max_violation)
```

Formulation 5: HBV $\mathcal{F}_{\text{LLM-B}}$

```
def obj1(metrics):
    nse = metrics[0]
    trmse = metrics[1]
    roce = metrics[2]
    sfdce = metrics[3]
    w_nse = 1.0
    w_roce = 1.0
    w_trmse = 0.3
    w_sfdce = 0.3
    target_nse_shift = 1.0
    val = w_nse * -(nse - -target_nse_shift) + w_roce * roce + w_trmse * trmse + w_sfdce *
     sfdce
    return float(val)

def c1(metrics):
    return float(0.5 - metrics[0])

def c2(metrics):
    return float(metrics[1] - 1.6)

def c3(metrics):
```

```
    return float(metrics[2] - 0.5)

def c4(metrics):
    return float(metrics[3] - 0.2)
```

D.2 HSE Antenna Formulations

For the antenna benchmark, we show the HSE task as the representative case. The formulation input is a sampled frequency-response curve represented as a two-column array, with frequency in the first column and response value in the second column.

SHA-PF

Formulation 6: HSE SHA-PF

```
def obj1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    r1 = (f >= 0.8) & (f <= 0.92)
    r2 = (f >= 0.91) & (f <= 0.96)
    r3 = (f >= 0.95) & (f <= 1.08)
    r4 = (f >= 1.07) & (f <= 1.16)
    r5 = (f >= 1.15) & (f <= 1.2)

    def seg(mask):
        return y[mask]
    pb = seg(r3)
    if pb.size == 0:
        return 1000.0
    floor10 = np.quantile(pb, 0.1)
    floor25 = np.quantile(pb, 0.25)
    top75 = np.quantile(pb, 0.75)
    ripple = max(0.0, top75 - floor25)
    left_edge = y[(f >= 0.95) & (f < 0.97)]
    right_edge = y[(f > 1.06) & (f <= 1.08)]
    le = np.mean(left_edge) if left_edge.size > 0 else np.mean(pb)
    re = np.mean(right_edge) if right_edge.size > 0 else np.mean(pb)
    edge_def = max(0.0, -2.0 - min(le, re))
    curv = 0.0
    if pb.size >= 3:
        s2 = pb[:-2] - 2 * pb[1:-1] + pb[2:]
        curv = np.mean(np.maximum(np.abs(s2) - 0.2, 0.0))
    obj = -(0.7 * floor10 + 0.3 * floor25) + 0.6 * ripple + 0.4 * curv + 0.5 * edge_def

    def soft_over(x, thr, q=0.9, p=2.0):
        if x.size == 0:
            return 0.0
        t = np.quantile(x, q)
        v = max(0.0, t - thr)
        return v ** 1.0
    sb_pen = soft_over(seg(r1), -7.0, q=0.9) + soft_over(seg(r5), -7.0, q=0.9)

    def notch_shape(x):
        if x.size < 3:
            return 1.0
        i = int(np.argmin(x))
        m = x[i]
        vscore = 0.0
        if 1 <= i <= x.size - 2:
            sL = x[i] - x[i - 1]
            sR = x[i + 1] - x[i]
            vscore = max(0.0, sL) + max(0.0, -sR)
        else:
            vscore = 0.5
        left = x[:i] if i > 0 else np.array([])
        right = x[i + 1:] if i < x.size - 1 else np.array([])
        lc = 0.0 if left.size == 0 else max(0.0, m - np.max(left))
        rc = 0.0 if right.size == 0 else max(0.0, m - np.max(right))
        return 0.3 * vscore + 0.7 * (lc + rc)
    notch_guidance = 0.5 * notch_shape(seg(r2)) + 1.0 * notch_shape(seg(r4))
    return float(obj + 0.8 * sb_pen + 0.5 * notch_guidance)

def c1(curve):
    import numpy as np
```

```
    f = curve[:, 0]
    y = curve[:, 1]
    r1 = (f >= 0.8) & (f <= 0.92)
    r3 = (f >= 0.95) & (f <= 1.08)
    r4 = (f >= 1.07) & (f <= 1.16)
    r5 = (f >= 1.15) & (f <= 1.2)

    def q(x, qv, defv):
        return np.quantile(x, qv) if x.size > 0 else defv
    v_sbL = 5 * (q(y[r1], 0.95, -7.0) - -7.0)
    v_sbH = q(y[r5], 0.95, -7.0) - -7.0
    v_pb = -2.0 - q(y[r3], 0.05, -2.0)
    segH = y[r4]

    def notch_violation(seg, thr):
        if seg.size < 3:
            return 1.0
        m = np.min(seg)
        depth_v = m - thr
        dy = np.diff(seg)
        eps = 0.0001
        has_min = np.any((dy[:-1] < -eps) & (dy[1:] > eps))
        mono_v = 0.5 if not has_min else 0.0
        return max(depth_v, mono_v)
    v_nH = notch_violation(segH, -14.0)
    return float(max(v_sbL, v_sbH, v_pb, v_nH))

def c2(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    r2 = (f >= 0.91) & (f <= 0.96)
    seg = y[r2]
    if seg.size < 3:
        return 1.0
    m = np.min(seg)
    dy = np.diff(seg)
    eps = 0.0001
    has_min = np.any((dy[:-1] < -eps) & (dy[1:] > eps))
    mono_v = 7 if not has_min else 0.0
    depth_v = m - -8.0 if m > -8.0 else 0.0
    return float(max(depth_v, mono_v))
```

Baselines

The baselines again include two expert-designed formulations and two direct LLM-generated formulations.

Formulation 7: HSE $\mathcal{F}_{\text{Expert-A}}$

```
def obj1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 0.95) & (f <= 1.07)
    if np.any(mask):
        return float(-np.min(y[mask]))
    return float(0.0)

def c1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 0.91) & (f <= 0.96)
    if np.any(mask):
        return float(np.min(y[mask]) - -8.0)
    return float(1000.0)

def c2(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 1.07) & (f <= 1.16)
    if np.any(mask):
        return float(np.min(y[mask]) - -14.0)
    return float(1000.0)

```

```python
def c3(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 0.8) & (f <= 0.92)
    if np.any(mask):
        return float(np.max(y[mask]) - -7.0)
    return float(1000.0)

def c4(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 1.15) & (f <= 1.2)
    if np.any(mask):
        return float(np.max(y[mask]) - -7.0)
    return float(1000.0)
```

Formulation 8: HSE $\mathcal{F}_{\text{Expert-B}}$

```python
def obj1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    pass_mask = (f >= 0.95) & (f <= 1.07)
    low_stop_mask = (f >= 0.8) & (f <= 0.92)
    high_stop_mask = (f >= 1.15) & (f <= 1.2)
    if not np.any(pass_mask):
        return float(0.0)
    value = -np.min(y[pass_mask])
    if np.any(low_stop_mask):
        value += 0.5 * np.max(y[low_stop_mask])
    if np.any(high_stop_mask):
        value += 0.5 * np.max(y[high_stop_mask])
    return float(value)

def c1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 0.95) & (f <= 1.07)
    if np.any(mask):
        return float(-2.0 - np.min(y[mask]))
    return float(1000.0)

def c2(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 0.91) & (f <= 0.96)
    if np.any(mask):
        return float(np.min(y[mask]) - -8.0)
    return float(1000.0)

def c3(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    mask = (f >= 1.07) & (f <= 1.16)
    if np.any(mask):
        return float(np.min(y[mask]) - -14.0)
    return float(1000.0)
```

Formulation 9: HSE $\mathcal{F}_{\text{LLM-A}}$

```python
def obj1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx_pb = (f >= 0.95) & (f <= 1.07)
    if not np.any(idx_pb):
        return 1000000.0
    mean_pb = np.mean(y[idx_pb])
    idx_ls = (f >= 0.8) & (f <= 0.92)
    idx_hs = (f >= 1.15) & (f <= 1.2)
    reg_ls = np.mean(np.maximum(y[idx_ls], -100)) if np.any(idx_ls) else 0.0
```

```
    reg_hs = np.mean(np.maximum(y[idx_hs], -100)) if np.any(idx_hs) else 0.0
    return -mean_pb + 0.02 * (reg_ls + reg_hs)

def c1(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx = (f >= 0.8) & (f <= 0.92)
    if not np.any(idx):
        return 0.0
    return np.maximum(0.0, np.max(y[idx]) - -7)

def c2(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx = (f >= 0.91) & (f <= 0.96)
    if not np.any(idx) or len(y[idx]) < 3:
        return 5.0
    ys = y[idx]
    fs = f[idx]
    local_min_exists = False
    for i in range(1, len(ys) - 1):
        if ys[i] <= ys[i - 1] - 0.5 and ys[i] <= ys[i + 1] - 0.5:
            local_min_exists = True
            min_val = ys[i]
            break
    if not local_min_exists:
        diffs = np.diff(ys)
        score = max(np.max(np.maximum(diffs, 0.0)), -np.min(np.minimum(diffs, 0.0))) if len(diffs) > 0 else 5.0
        return 1.0 + score
    return np.maximum(0.0, min_val - -8)

def c3(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx = (f >= 0.95) & (f <= 1.07)
    if not np.any(idx):
        return 5.0
    min_pb = np.min(y[idx])
    return np.maximum(0.0, -2 - min_pb)

def c4(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx = (f >= 1.07) & (f <= 1.16)
    if not np.any(idx) or len(y[idx]) < 3:
        return 3.0
    ys = y[idx]
    fs = f[idx]
    local_min_exists = False
    for i in range(1, len(ys) - 1):
        if ys[i] <= ys[i - 1] - 0.3 and ys[i] <= ys[i + 1] - 0.3:
            local_min_exists = True
            min_val = ys[i]
            break
    if not local_min_exists:
        diffs = np.diff(ys)
        score = max(np.max(np.maximum(diffs, 0.0)), -np.min(np.minimum(diffs, 0.0))) if len(diffs) > 0 else 3.0
        return 0.5 + score
    return np.maximum(0.0, min_val - -14)

def c5(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx = (f >= 1.15) & (f <= 1.2)
    if not np.any(idx):
        return 0.0
    return np.maximum(0.0, np.max(y[idx]) - -7)
```

Formulation 10: HSE $\mathcal{F}_{\text{LLM-B}}$

```
def obj1(curve):
```

```
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx_pb = (f >= 0.95) & (f <= 1.07)
    if not np.any(idx_pb):
        return 1000000.0
    mean_pb = np.mean(y[idx_pb])
    idx_n1 = (f >= 0.91) & (f <= 0.96)
    idx_n2 = (f >= 1.07) & (f <= 1.16)
    min_n1 = np.min(y[idx_n1]) if np.any(idx_n1) else 0.0
    min_n2 = np.min(y[idx_n2]) if np.any(idx_n2) else 0.0
    idx_s1 = (f >= 0.8) & (f <= 0.92)
    idx_s2 = (f >= 1.15) & (f <= 1.2)
    mean_s1 = np.mean(y[idx_s1]) if np.any(idx_s1) else -100.0
    mean_s2 = np.mean(y[idx_s2]) if np.any(idx_s2) else -100.0
    return -mean_pb + 0.05 * (min_n1 - -8 + (min_n2 - -14)) + 0.01 * (mean_s1 + mean_s2)

def c1(curve):
    import numpy as np
    f,y = curve[:, 0], curve[:, 1]
    ys = y
    fs = f
    if len(ys) < 5:
        return 0.0
    d1 = np.diff(ys) / np.diff(fs)
    d2 = np.diff(d1) / (fs[2:] - fs[1:-1])
    max_curv = np.max(np.abs(d2))
    return np.maximum(0.0, max_curv - 50.0)

def c2(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]
    idx_pb = (f >= 0.95) & (f <= 1.07)
    if not np.any(idx_pb):
        return 5.0
    min_pb = np.min(y[idx_pb])
    return np.maximum(0.0, 0.0 - min_pb)

def c3(curve):
    import numpy as np
    f = curve[:, 0]
    y = curve[:, 1]

    def has_notch(idx_band, thresh_depth):
        ys = y[idx_band]
        fs = f[idx_band]
        if len(ys) < 3:
            return (False, 10.0)
        for i in range(1, len(ys) - 1):
            if ys[i] <= ys[i - 1] - 0.3 and ys[i] <= ys[i + 1] - 0.3:
                return (True, ys[i])
        return (False, np.min(ys))
    idx1 = (f >= 0.91) & (f <= 0.96)
    idx2 = (f >= 1.07) & (f <= 1.16)
    (ok1, min1) = has_notch(idx1, -8)
    (ok2, min2) = has_notch(idx2, -14)
    pen1 = 0.0 if ok1 else 2.0
    pen1 += np.maximum(0.0, min1 - -8)
    pen2 = 0.0 if ok2 else 2.0
    pen2 += np.maximum(0.0, min2 - -14)
    return pen1 + pen2
```