cGenEDA: Code-Generation for Electronic Design Optimization

PHILIPP GÉRARD TRÉMUEL, Informatics, University of Amsterdam Faculty of Science, Amsterdam, Netherlands and Fachhochschule Ostschweiz, Buchs SG, Switzerland
FERNANDO BENITES, Institute for Computational Engineering, Fachhochschule Ostschweiz, Buchs SG, Switzerland
CHRISTOPH WÜRSCH, Institute for Computational Engineering, Fachhochschule Ostschweiz, Buchs SG, Switzerland
ULF HÜGEL, Huber+Suhner AG, Herisau, Switzerland
FLORIAN WOLF, Huber+Suhner AG, Herisau, Switzerland
ANDREAS BÜCHEL, Huber+Suhner AG, Herisau, Switzerland
EFSTRATIOS GAVVES, Informatics, University of Amsterdam Faculty of Science, Amsterdam, Netherlands

Addressing inverse design problems in engineering generally requires managing complex challenges. Traditionally, such systems are optimized by manually setting up a model in a numerical simulation software and applying parameter optimization. Topology changes and variable parameter sets result in challenging mixed-integer optimization problems. Such problems are hard to solve automatically; that's why they normally need to be initialized manually with high effort by the design engineer. Generative probabilistic models, like Large Language Models (LLMs), can efficiently produce diverse, valid design concepts that are fully parameterizable, simulatable, and optimizable. We introduce cGenEDA, a *Co-Pilot*–a system based on multiple LLM agents using in-context prompting to coordinate these tasks, providing a seamless, adaptable toolchain for automated design generation, optimization problem definition, and design automation. Our approach demonstrates that long, interacting simulation model code sequences can be generated accurately and directly used as model input for domain-specific simulators for precise forward simulations. Simultaneously, cGenEDA will set up the optimization problem, combine it with well-known, black-box optimization algorithms, and efficiently solve inverse design problems.

CCS Concepts: • **Hardware → Physical design (EDA)**; • **Computing methodologies → Modeling and simulation**; **Machine learning;**

Additional Key Words and Phrases: Code generation, large language model, computer-aided electronic design, electronic design automation, numerical simulation automation, analog circuit design, bayesian optimization

Authors' Contact Information: Philipp Gérard Trémuel (corresponding author), Informatics, University of Amsterdam Faculty of Science, Amsterdam, Netherlands and Fachhochschule Ostschweiz, Buchs SG, Switzerland; email: p.g.tremuel@uva.nl; Fernando Benites, Institute for Computational Engineering, Fachhochschule Ostschweiz, Buchs SG, SG, Switzerland; e-mail: fernando.benites@ost.ch; Christoph Würsch, Institute for Computational Engineering, Fachhochschule Ostschweiz, Buchs SG, SG, Switzerland; e-mail: chistoph.wuersch@ost.ch; Ulf Hügel, Huber+Suhner AG, Herisau, Switzerland; e-mail: ulf.huegel@hubersuhner.com; Florian Wolf, Huber+Suhner AG, Herisau, Switzerland; e-mail: florian.wolf@hubersuhner.com; Andreas Büchel, Huber+Suhner AG, Herisau, Switzerland; e-mail: andreas.buechel@hubersuhner.com; Efstratios Gavves, Informatics, University of Amsterdam Faculty of Science, Amsterdam, Netherlands; e-mail: e.gavves@uva.nl.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).
ACM 1084-4309/2026/06-ART140
https://doi.org/10.1145/3800953

ACM Reference Format:
Philipp Gérard Trémuel, Fernando Benites, Christoph Würsch, Ulf Hügel, Florian Wolf, Andreas Büchel, and Efstratios Gavves. 2026. cGenEDA: Code-Generation for Electronic Design Optimization. *ACM Trans. Des. Autom. Electron. Syst.* 31, 6, Article 140 (June 2026), 32 pages. https://doi.org/10.1145/3800953

1 Introduction

The design and optimization of complex, broadband passive **radio-frequency** (**RF**) PCB boards used for high-end applications such as millimeter-wave antenna arrays, high-speed interconnects, and RF front-ends for 5G/6G systems remains, up to this day, mainly manual work by expert high-frequency engineers. They use sophisticated design, simulation, and optimization software suites to iteratively solve the inverse problem. This design process is costly and time-consuming since the basis of RF engineering is wave propagation with all its effects: interference, reflection, refraction, diffraction, radiation, and attenuation. In RF design, electric fields propagate. They are therefore non-local and coupled. For specific applications, fast surrogate solvers or even lumped models may be used, but in order to achieve greater generalizability and applicability, it is essential to model the 3D geometry. Due to the geometric complexity of the structures, generative neural networks often fail to produce valid three-dimensional geometries. Similarly, physics-informed neural surrogate models, such as PINNs and PINOs, struggle to efficiently and accurately predict the electric field distribution for unseen RF designs [45], while also lacking the flexibility and generalizability characteristic of numerical solvers. In high-frequency electromagetics, a 5% accuracy is not sufficient. Accuracies in the range of -60dB (10^{-6}) are needed. However, generative neural models such as LLMs are good at *coding*. Using the generative power of LLMs for *simulation code generation* for 3D simulation models in RF design takes the best of the two worlds: the generative power of the LLMs helps to bridge designs of a wide design space, and the accuracy of the exact numerical forward solver delivers the necessary reliability and accuracy.

Coding, as in computer programming, has become a universal language in engineering. It plays an essential role in automating design, evaluation, and optimization processes across domains. In electronic design, coding is integral to streamlining workflows and enabling sophisticated automation. Many implementations and tools attest to its pivotal role, from **hardware description languages** (**HDLs**) such as Verilog [17, 52] to circuit simulation tools like SPICE [28, 31, 32, 57, 61], often interfaced through Python bindings such as PySpice. These tools demonstrate how coding allows engineers to *describe, simulate, and optimize* complex systems efficiently. Advanced simulation software for 3D RF designs is needed to assure signal quality before production testing. These applications offer extensive **application programming interfaces** (**APIs**), typically accessed through Python, facilitating precise and complex interactions with these simulation tools. Examples include Ansys PyAnsys for Electronics Desktop [3] and pycst for CST Studio Suite [47], which underscore the growing importance of code-based automation in modern electronic design. Figure 1 illustrates the interaction between LLMs and EDA stages across data sources, representations, and applications, emphasizing RF design and 3D geometry.

1.1 Related Work and the Frontier in RF/mmWave EDA

Electronic design automation (**EDA**) encompasses diverse domains with distinct workflows, design abstractions, and computational challenges. Table 1 outlines the differences between digital/-logic, **analog mixed-signal** (**AMS**), and radio-frequency (RF/mmWave) design domains across key workflow steps. While digital and AMS workflows have already benefited from significant automation using LLM coding agents, RF and **electromagnetic** (**EM**)-centric design remains one of the most demanding areas due to its reliance on full 3D geometry representations and computationally

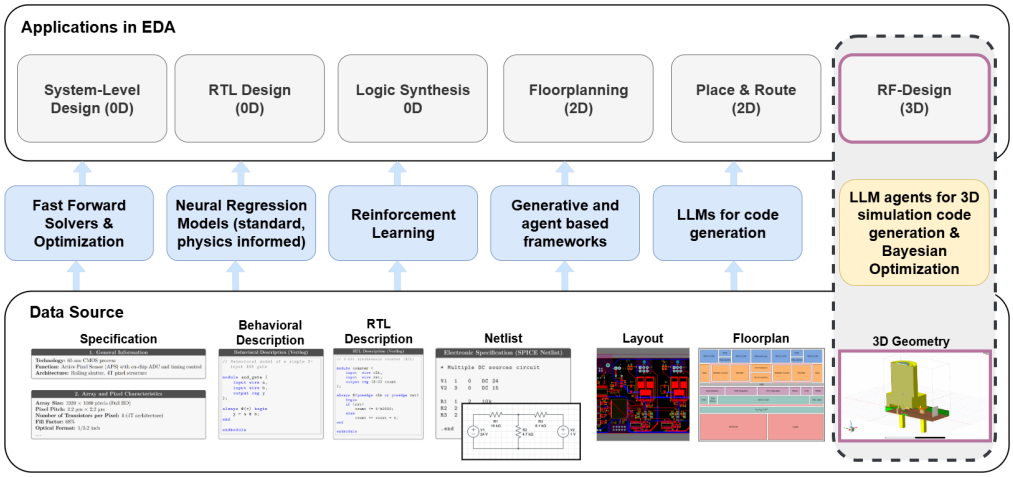

Fig. 1. Overview of Large Language Models (LLMs) in Electronic Design Automation (EDA) Applications. This diagram illustrates the integration of LLMs across EDA workflows, from high-level system design to specialized RF design. Data sources vary by domain–from abstract textual behavioral descriptions and schematic RTL (0D) to 2D layouts and floorplans–while complex RF-design demands mandatory 3D geometries, introducing additional process complexity. Representations (e.g., graphs, images, voxels) bridge inputs to LLM-driven outputs for enhanced automation.

Table 1. Overview of Electronic Design Fields: Differences in Design Process and Representation

Aspect	Logic/Digital	Analog Low Frequency	Analog High Frequency
Design Representation	HDLs (e.g., Verilog/VHDL); gate-level netlists	Circuit schematics and SPICE netlists; lumped transistor/resistor/capacitor models	3D geometric models (e.g., CAD files); EM field distributions and material properties
Design Process	Synthesis, placement, routing; automated with well-defined flows	Transistor sizing and parameter tuning; iterative simulation and manual adjustments	Topology generation, 3D modeling, EM simulation; requires manual topology initialization and meshing
Optimization Challenges	Discrete/combinatorial (e.g., logic minimization); scalable with heuristics	Continuous parameters (e.g., widths/lengths); multi-objective but limited dimensionality	Mixed-integer (topologies + continuous params); high-dimensional, non-convex due to 3D EM effects; computationally intensive
Typical Tools/ Simulators	Synopsys VCS, Cadence Xcelium; logic simulators	HSPICE, Cadence Spectre; circuit-level simulators	HFSS, CST Microwave Studio, Cadence Clarity 3D Solver, Keysight EMPro; full-wave EM solvers
Complexity Drivers	Scalability with gate count, timing closure	Tradeoffs in power/performance; sensitivity to process variations	EM wave propagation, parasitics, and 3D spatial effects; mesh accuracy and simulation time
LLM Applicability in Existing Methods	Code generation (e.g., Verilog completion) [52]; less emphasis on physics	Parameter optimization (e.g., ADO-LLM [61]); in-context learning on netlists	Limited; our cGenEDA enables 3D concept generation and inverse design automation

intensive simulations. In digital design, deep learning and reinforcement learning approaches have achieved advances in logic synthesis [22, 23], placement [38, 50], and routing [37, 51]. In analog design, generative models and agent-based frameworks have been applied to topology generation, device placement, and parameter sizing [8, 28], particularly when coupled with feedback-driven optimization and autonomous agent systems [31, 61]. These successes demonstrate that generative AI can capture complex design heuristics, accelerate exploration of large design spaces, and reduce manual engineering effort.

Despite these remarkable advances, particularly in chip design [59, 60], the application of generative models to complex, high-dimensional RF design problems remains largely unexplored. Specifically, challenges persist in integrating 3D geometry generation, physics-based simulation, and automated performance evaluation within a unified framework. This gap motivates our work to extend current state-of-the-art multi-agent EDA [21, 56, 60] methodologies to 3D geometry optimization for RF design. Our approach incorporates these capabilities into an iterative improvement framework that scales effectively as the design database expands.

A key bottleneck is representation: typical mesh- or voxel-based generative models lose CAD-kernel semantics (B-reps, NURBS, CSG trees, and feature histories) that are essential for parameterization, constraint satisfaction, and robust meshing. Text-only LLMs also lack grounding to EM solvers, manufacturing rules, and material libraries, leading to shapes that are syntactically plausible but physically invalid (e.g., non-manifold edges, sliver faces, unreachable tapers, illegal stackups). Moreover, most datasets pair geometry with sparse or single-frequency labels, whereas high-frequency RF demands broadband S-parameters, radiation patterns, and tolerance sweeps to capture sensitivity.

This gap presents a major opportunity for advancing automated, high-fidelity RF design. While some efforts focus on high-accuracy modeling of individual components such as striplines and via transitions [48, 49], these are often constrained by narrow design spaces or frequency bands. Moreover, design optimization is more effective when conducted directly on geometric representations rather than simplified idealized models, as shown in Reference [61]. The integration of LLM agents for end-to-end code synthesis of complex geometries, constraints, and boundary conditions could thus play a transformative role in next-generation EDA systems, given the right context and tools.

The work presented in Reference [61] integrates LLMs with **Bayesian Optimization** (**BO**) for analog circuit design optimization. Several key differences distinguish their approach from ours. Yin et al. [61] address frequency-independent objectives by optimizing a scalar **Figure of Merit** (**FOM**), whereas our work tackles broadband frequency-domain optimization. In our setting, a major challenge is that complex designs exhibit resonance frequencies that induce parasitic effects and degrade signal quality; these resonances are difficult to predict without simulation. Therefore, our work extends their paradigm from 0D netlist-based representations to full 3D geometric models, introducing substantial complexity through spatial constraints that influence both the initial board topology generation and subsequent optimization. Moreover, the LLM in our framework can autonomously formulate the complete optimization problem, including topology-dependent constraints, rather than operating within a predefined design space.

A comprehensive survey of LLM applications across EDA domains is provided in Reference [44].

RF and mmWave design workflows exemplify these challenges. Unlike logic and analog circuits, RF systems require full-wave electromagnetic solvers that operate directly on detailed 3D geometries. Industrial-grade tools such as Ansys HFSS [4], CST Studio Suite [11], Keysight EMPro [27], Cadence Clarity 3D Solver [7], and COMSOL Multiphysics [10] are indispensable for designing and validating critical interconnects in PCBs, IC packaging, and **system-on-chip** (**SoIC**) integration. These solvers enable the accurate modeling of antennas, filters, couplers, and other RF components, while

capturing **electromagnetic interference** (**EMI**), **electromagnetic compatibility** (**EMC**), and cross-domain coupling effects that cannot be abstracted away.

The industrial relevance of this domain is substantial. RF design underpins disruptive technologies including 5G, **high-performance computing** (**HPC**), automotive and UWB radar, satellite communications, and the **Internet of Things** (**IoT**). Electronic systems in these contexts are often safety- or mission-critical, where performance degradation, interference, or failure may have severe consequences. As operating frequencies increase and system integration becomes more complex, electromagnetic interactions become harder to predict, and the need for robust simulation-driven design intensifies. Simulation reduces the reliance on costly and time-consuming physical prototyping, allowing EMI/EMC issues to be mitigated early while enabling the delivery of high-performance, compliant, and reliable designs.

1.2 A Learning-Aided Design (LAD) Approach for 3D RF Optimization

Taken together, these considerations highlight RF design as not only a critical industrial application area but also a frontier for advancing generative AI in EDA. Addressing the challenges of high-dimensional geometry, physics-based constraints, and human-in-the-loop optimization will be central to realizing the potential of generative AI in this domain.

From this point, it seems obvious that the next step from traditional **computer-aided design** (**CAD**) and the next evolution of design tools is **learning-aided design** (**LAD**), powered by **generative AI** (**GenAI**). The generated designs incorporate feedback from simulation-driven performance analysis, moving beyond simple design generation to a more robust, human-supervised, AI-driven design process. Such a LAD framework is visualized in Figure 2.

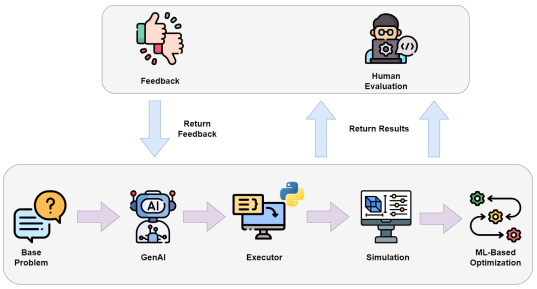

Fig. 2. Learning-Aided Design Framework (LAD). GenAI generates solutions, which are evaluated through code execution, simulation, and optimization.

This work targets the automated extraction of design parameters and the generation of code modules for the optimization of complex **three-dimensional** (**3D**) structures in connector-to-**printed circuit board** (**PCB**) transitions. These transitions are critical in high-frequency applications, particularly in ultra-wideband systems operating up to 67 GHz, which corresponds to the current state-of-the-art in high-end **radio frequency** (**RF**) design. At such frequencies, the electromagnetic behavior of interconnect structures becomes highly sensitive to geometric and material variations. Consequently, even minor changes in design parameters can lead to significant performance degradation. This imposes stringent demands on design accuracy, modeling fidelity, and the automation of the simulation workflow to enable robust optimization. The code for the simulation model (geometry, parameters) and the optimization (boundary conditions, sampling) is generated in a single shot using context information from previous simulations. Furthermore, the parameters are tuned to reach optimal performance. As the next step toward a LAD system, such a system would be continuously improved through feedback to the generative model.

We introduce and evaluate an information extraction and code generation toolchain designed to customize and adapt code for a Python API that interfaces with a numerical simulation software within a design optimization framework. Our approach leverages in-context learning to incorporate domain-specific knowledge, highlighting both the potential and limitations of this method for automated electronic design. In addition, we outline future directions to improve the adaptability

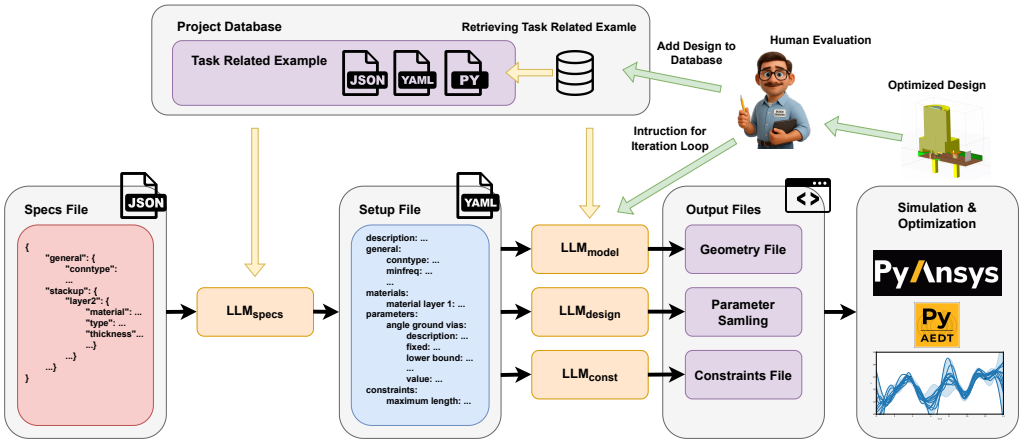

Fig. 3. *Overview of the cGenEDA Toolchain*: The heart of the system is an expert team of four LLMs. Each of them has a specific task. (1) $\text{LLM}_{\text{specs}}$: it takes the instruction and specification files, as well as a task-related example to generate the problem definition file F_{def}. The setup-file F_{setup} is a structured YAML file containing all the geometrical information and performance specification. (2) $\text{LLM}_{\text{model}}$: takes the setup-file, as well as context information from the task-related example, to generate the Python code to set up the geometry F_{geom}. (3) $\text{LLM}_{\text{design}}$: takes the setup-file and information from the task-related example to generate the Python code for the sampler F_{sample} (brute-force optimization). (4) $\text{LLM}_{\text{const}}$: takes the setup-file and information from the task-related example to generate the Python code which defines the constraint geometrical constraints F_{const}.

and performance of the toolchain. By integrating code generation directly into the optimization loop, we further streamline the design process with an additional code generation agent that generates the constraints for the optimization algorithm. The generative capabilities of LLMs enable flexible adaptation to diverse problems, topology changes, and varying parameters, while the rapid convergence of constraint-driven Gaussian Process Bayesian Optimization ($\mathcal{GP}$-BO) enhances efficiency. Additionally, the LLM-generated constraint functions improve interpretability by describing the constraints of the optimization as analytical formulas and ensuring valid design solutions.

The in-context learning approach allows the toolchain to adapt to specific problems without requiring model fine-tuning, relying instead on a relevant task-related example. Unlike other approaches, this method operates directly on the 3D geometry rather than an idealized lumped representation, presenting additional challenges for optimization due to complex geometric constraints.

Our approach addresses the entire workflow, from defining design parameters and setting up simulation code to managing geometric constraints and executing black-box optimization for board parameter optimization. Multiple LLM agents, each using a contextual prompt, can focus on parts of the overall task, creating a streamlined and adaptable toolchain for automated development. Additionally, the contained and focused prompt for each agent restrains the problem of context length degradation.

2 Method

The code generation toolchain is embedded within an end-to-end optimization framework (see Figure 3). At its core lies a **multi-agent LLM system** (**MAS**). We deliberately adopt a multi-agent approach rather than a single-agent one, since singular LLM-based agents often encounter limitations when tackling real-world problems that span multiple domains and require

ALGORITHM 1: Code Generation and GP-Optimization

Input : specification S, N_{init}, N_{iter}
Context : context E
$X_f \leftarrow$ extract fixed parameters from S and E
$X_v \leftarrow$ extract variables parameters from S and E
$C \leftarrow$ extract fixed parameters from S and E
Generate code files with $\text{LLM}_{\text{model}}$, $\text{LLM}_{\text{design}}$, $\text{LLM}_{\text{const}}$
$(X_{v_\text{init}}, Y_{v_\text{init}}) \leftarrow$ Randomly sample N_{init} points form
    Brute-Force sampler/optimizer
Fit an initial surrogate model $\mathcal{GP}$ on $(X_{v_\text{init}}, Y_{v_\text{init}})$
for $t = 1$ to N_{iter} **do**
    Select a candidate point X_v^t by maximizing the
        acquisition function l : $X_v^t \leftarrow \arg\min_X l(X, \mathcal{GP})$
    $Y_t \leftarrow$ Simulate system with the candidate point
        parameters X_v^t
    Update surrogate model $\mathcal{GP}$ with (X_v^t, Y^t)
end for
return The best selected point (X_v^*, Y^*)

expertise across diverse fields. By contrast, MAS systems leverage the complementary strengths of specialized agents [53], each with distinct skills and responsibilities, working collaboratively toward a common objective. Such collaborative mechanisms–e.g., structured debate or distributed task execution–have been shown to foster divergent thinking, reduce context length, and ultimately improve solution quality [30].

In our implementation, the toolchain consists of one *specification agent* and three specialized *coding agents*, each aligned with a major design stage: (i) modeling and setting up the geometry (*modeler*: LLMmodel), (ii) defining the design space (*designer*: LLMdesign), and (iii) handling design constraints (*constraint handler*: $\text{LLM}_{\text{const}}$). By distributing responsibilities in this manner, we mitigate the challenges of overextended context windows and improve the accuracy and consistency of code generation. The proposed agent architecture is inspired by the workflow of a human designer and leverages task decomposition to ensure that each agent receives sufficient, well-defined information. The *specification agent* functions as a planning agent, orchestrating the overall design process. Meanwhile, the three coding agents independently generate the essential modules required for the complete simulation code. This modular structure is flexible and can be readily adapted to other RF and electronic design applications.

In a first step, the specification agent ($\text{LLM}_{\text{specs}}$) transforms the *specification* file into a *problem description* file F_{desc} in YAML format that defines the entire electronic board in a well-defined, structured way, understandable for the following LLM coding agents. This problem description file is then provided to the coding agents, who translate the specific information into runnable code. The modeler agent ($\text{LLM}_{\text{model}}$) generates the code to set up the board geometry F_{geom}, the designer agent ($\text{LLM}_{\text{design}}$) generates the code to sample parameters F_{sample} under the constraints defined by the input specifications and the board geometry constraints, and the constraint handler agent ($\text{LLM}_{\text{const}}$) generates the code to formulate the board constraints F_{const}. By this multi-agent structure, the problem is split into well defined, into well-defined subtasks that can be solved independently. The code can then be used to run the simulation and optimization, either with the generated brute-force optimizer or another black-box optimizer like Bayesian optimization [19], using the defined constraints. The whole toolchain with a Bayesian optimization using Gaussian-Process ($\mathcal{GP}$) is shown in Algorithm 1.

In each optimization cycle, the candidate design is evaluated by a human engineer. This human-in-the-loop step provides the final supervisory layer, ensuring physical plausibility and correctness. When additional iterations or refinements are required, the engineer may instruct the language models to modify the design topology, as demonstrated in Prompt 10. Designs that pass this evaluation are incorporated into the database, thereby contributing to the continuous improvement of the generative process.

2.1 Generate Problem Definition File

The problem setup is defined by a structured input file containing geometrical and material-specific parameters. This file can vary in terms of the number and type of parameters, depending on some application-specific requirements. Additionally, some text instructions for the design can be given as a context E. This file is called the *specification* S and can be represented as

$$S = \{s_1, s_2, \dots, s_m\}, s_i \in \mathcal{T} \ \forall \ i \in \{1, 2, \dots, m\}, \tag{1}$$

where $\mathcal{T}$ is the space of possible textual descriptions, and s_i are individual elements (sentences, words, or clauses) from the specification. The LLM to extract the information from the specification S and generate the problem definition is denoted as a function $\text{LLM}_{\text{specs}}$ (see Appendix A: Figures 11, 12, for the exact system and user prompts used). The problem definition contains a tuple of fixed parameters X_f, variable parameters X_v, and the set of constraints C:

$$(X_f, X_v, C) = \text{LLM}_{\text{specs}}(S, E). \tag{2}$$

As it is a very complex problem, the parameters and constraints are formulated in a structured text file. This also makes it flexible to different problem setups. The fixed parameters are those that do not vary during optimization and within the problem constraints. Let X_f be a set of fixed parameters:

$$X_f = \left\{x_f^1, x_f^2, \dots, x_f^p\right\}, \tag{3}$$

where $x_f^i \in \mathbb{R}$ or $\mathbb{Z}$, i.e., each fixed parameter is a real or integer-valued variable that is extracted by the LLM. Variable parameters are those that are allowed to change within the problem's constraints. They form the *design space* and are optimized in the later optimization step. Let X_v be the set of variable parameters:

$$X_v = \left\{x_v^1, x_v^2, \dots, x_f^q\right\}, \tag{4}$$

where $x_v^j \in \mathbb{R}$ or $\mathbb{Z}$, subject to constraints defined in the set C. The set of constraints $C = \{c_1, c_2, \dots, c_n\}$ represents the restrictions or relationships between the fixed and variable parameters. Each constraint c_i is a function of both fixed and variable parameters:

$$\begin{aligned} &c_i(X_f, X_v) = 0 \quad \text{or} \\ &c_i(X_f, X_v) \leq 0 \quad \forall i \in \{1, 2, \dots, n\}. \end{aligned} \tag{5}$$

These constraints can be equality or inequality constraints. More formally, we can represent:

$$C = \{c_i \ : \ \mathbb{R}^{p+q} \to \mathbb{R}\}, \quad \forall i \in \{1, 2, \dots, n\}. \tag{6}$$

2.1.1 In-Context Information. Incorporating in-context information E into the generation process for parameters and constraints enables the integration of domain knowledge through single-shot or few-shot learning paradigms [6]. This approach can significantly improve generation accuracy while reducing the need for fine-tuning or reliance on **retrieval-augmented generation** (**RAG**) systems [29], which are commonly used to enhance the performance of large language models (LLMs). Wang et al. [58] introduced the CodeRAG-Bench benchmark to evaluate the impact of RAG systems on code generation tasks, showing that RAG can improve output quality for smaller

language models. However, these gains are not always consistent, particularly for complex tasks involving entire code repositories or highly specialized domains.

For such cases, more structured retrieval strategies, including Graph-RAG approaches [35], have been proposed to better capture the relationships inherent in repository-level code. Nevertheless, standard embedding-based retrieval remains challenging in EDA and numerical simulation workflows, where subtle but physically meaningful differences in parameter configurations can dominate performance. Generic code embeddings often fail to reflect these distinctions, complicating effective context selection.

To address this limitation, we adopt an alternative strategy that selects in-context examples using a similarity metric computed directly over physically grounded parameter spaces rather than relying on conventional text-based RAG pipelines. The selected contextual information is formalized through an extraction function applied to the most relevant examples. In-context data is represented as a set of paired examples $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X}$ denotes an input setup file and $\mathcal{Y}$ the corresponding output code. Prior work has shown that such paired input–output examples are more effective for parameter inference and code generation than providing output templates or isolated code snippets alone. In our setup, each pair consists of a configuration file for a related electromagnetic design problem and the corresponding Python code used to generate its geometry, providing concrete guidance on correct API usage during generation.

Similarity-based Retrieval for Cross-Configuration Learning. To enable cross-configuration learning, we select context examples based on similarity in a physically meaningful design space. This directly addresses the question of how relevant context is identified and how sensitive the system is to context choice. Rather than performing approximate nearest-neighbor search over parameter vectors x defined by the fixed and variable parameters (x_f, x_v), similarity is computed over normalized stackup and design parameters that are known to govern electromagnetic behavior. This ensures that retrieved examples differ minimally along dimensions that matter for simulation fidelity, making the system less brittle to superficial code or formatting differences.

All stackup-dependent features from our database are first transformed into a configuration-agnostic representation by re-indexing layer properties relative to the signal layer. For a board with N layers and a signal trace located at layer index i_{sig}, each layer i is assigned an offset

$$\delta = i - i_{\text{sig}}, \qquad \delta \in \{-i_{\text{sig}}, \dots, -1, 0, +1, \dots, N - i_{\text{sig}} - 1\}.$$

Layer-specific attributes such as thickness and relative permittivity are then encoded as $\{t_\delta, \epsilon_{r,\delta}, \dots\}$, so that a given offset (e.g., $\delta = +1$) always represents the same physical relationship to the signal trace, independent of the absolute layer count. This normalization allows meaningful comparison across heterogeneous stackups while preserving locality effects that dominate field confinement and impedance behavior.

Configuration and Design Similarity Metrics. Configuration similarity between two boards $C_1 = (N_1, p_1)$ and $C_2 = (N_2, p_2)$, with normalized signal position $p = i_{\text{sig}}/N$, is quantified as

$$S_{\text{config}}(C_1, C_2) = \alpha e^{-\lambda |N_1 - N_2|} + (1 - \alpha)\,(1 - |p_1 - p_2|)\,,$$

where the exponential term penalizes differences in total layer count and the linear term enforces alignment in relative signal placement. The weighting parameter α reflects the physical insight that impedance and field confinement are generally more sensitive to the signal's relative position than to the absolute board thickness.

Beyond this structural comparison, design-level similarity is evaluated in the re-indexed feature space using a weighted Euclidean distance. Given two design parameter vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ and

feature importance weights $\mathbf{w}$, we define

$$S_{\text{design}}(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{1 + \sqrt{\sum_{j=1}^{d} w_j \left(x_{1j} - x_{2j}\right)^2}}.$$

The weights $\mathbf{w}$, which are shown in Figure 4 are extracted from Random Forest models trained to predict the same scalar performance metric used as the optimization objective (e.g., impedance error). As a result, similarity is emphasized along dimensions that most strongly influence the task objective. From our database, we extract 115 unique features. New features are initialized with mean weight value for the most similar configuration.

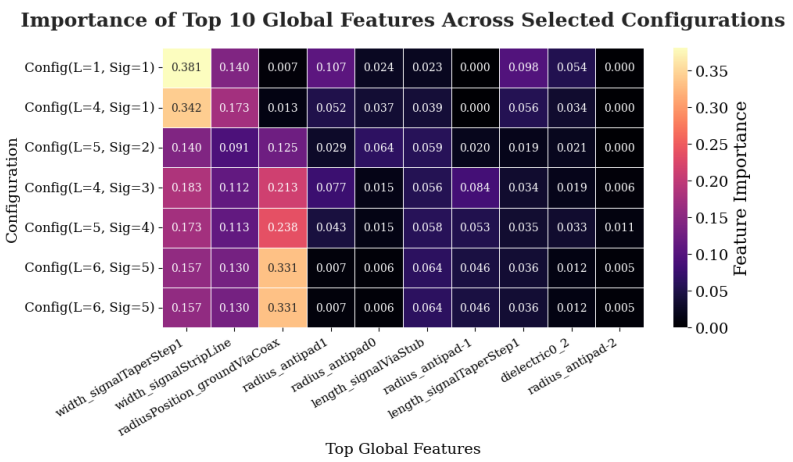

Fig. 4. *Feature Importance Plot*: Feature importance of the top ten global features for the different board configurations to show how the relevance changes with the configuration of the board.

In practice, the system is moderately sensitive to the choice of context example: performance degrades gracefully as similarity decreases rather than failing catastrophically. When no sufficiently similar example exists in the database, the method naturally falls back to lower similarity scores, effectively reverting to zero- or weak-shot generation. In these cases, the LLM relies more heavily on its pretrained prior and the explicit constraints in the prompt, avoiding misleading guidance from poorly matched examples. This behavior contrasts with text-based RAG systems, where semantically similar but physically irrelevant examples can introduce systematic errors.

2.2 Code Generation

The problem description file F_{desc} translates the problem into a language that an LLM can interpret, as direct manipulation of the 3D geometry is currently a challenging task. Some approaches, like in Reference [1, 62], use a visual transformer to generate code for CAD models and interact, manipulate, and optimize them. In addition to setting up the simulation, an intermediary step is required for simulation access. This necessitates communication with the simulation tool, which is achieved through an API. Many simulation software tools offer API access for this purpose. In this work, we employ the Python library pyAEDT [3] to interface with the Ansys simulation software. AEDT was selected due to its comprehensive Python API and the availability of extensive online documentation and community resources, which enable pretrained language models to effectively leverage prior knowledge of its syntax and usage. Alternative interfaces, such as `pycst` for CST Studio Suite [47], could be integrated with minimal effort. Switching between simulation backends is straightforward and can be achieved by providing an appropriate in-context example and re-prompting the model.

The process of code generation is divided into three key stages: (1) geometry setup, (2) constraint validation, and (3) variable initialization for a brute-force optimizer. The code files, generated by LLM agents, use both the problem definition file and a context file as inputs.

2.2.1 Geometry Setup for Simulation. The geometry setup for the electronic design simulation follows a consistent and structured format, typically comprising the following five components:

(i) **Material Definitions**—Specification of material properties used in the model.
(ii) **Parameter Definitions**—Declaration of design parameters such as dimensions, positions, and other configurable variables.
(iii) **Geometrical Object Construction**—Creation of geometrical shapes and composite structures from the defined parameters and materials.
(iv) **Boundary and Symmetry Conditions**—Specification of boundary conditions and exploitation of geometrical or physical symmetries to reduce computational complexity.
(v) **Port Definition**—Placement and configuration of input/output ports used to excite and probe the electromagnetic structure.

This structure is encoded as Python code within a single script file, which serves as input to the simulation engine. Contextual prompts are leveraged to guide the **large language model** (**LLM**) in generating code with this format. Incorporating structural context not only constrains the output space but also minimizes hallucinations, thereby enhancing the reliability of code synthesis [16]. This is particularly effective given the few-shot learning capabilities of modern LLMs [6].

The full system and user prompt used for code generation are provided in Appendix A, see Listings 13 and 14.

2.2.2 Define Constraints. The constraints file F_{const} generated by the LLM_{const} defines the geometrical restrictions for a specific design. These constraints ensure that the electronic board configuration remains physically meaningful, complies with manufacturing requirements, and satisfies all predefined design rules. They are formulated as equality or inequality conditions, as summarized in Eq. 5, and are implemented through a callable function, an example of which is shown in Listing 1.

```python
def constraint_fn(free_params, fixed_params):
    ...
    constraints_arr = np.array([
      # Constraint 1: Total stripline length
      free_params['length_stripline_section1']+free_params['length_stripline_section2'] +
          fixed_params['offset_substrate']-fixed_params['length_substrate'],
      # Constraint 2: Width consistency
      free_params['width_stripline_section2']-free_params['width_stripline_section1'],
      # Constraint 3: Via clearance
      free_params['width_stripline_section1']-fixed_params['dia_catch_sigVia'],
      # Constraint 4: Minimum gap requirement
      fixed_params['min_gap'] - free_params['gap'],
    ])
    return constraints_arr
```

Listing 1. Constraint Function Definition

Design constraints are evaluated directly through the pyAEDT API, which generates detailed error logs for invalid configurations. These logs are utilized to iteratively refine candidate designs and resolve inconsistencies in either the generated code or the underlying constraint definitions. While physical feasibility is maintained via explicit parameter bounds, structural issues–such as geometric intersections–are identified using pyAEDT's built-in validation routines. An example of a log file reporting an invalid parameter due to conflicting constraints is provided in Appendix A: Figure 22.

The constraint function is essential during both the initial variable sampling stage (Section 2.2.3) and the constrained optimization procedure (Section 2.3). The code for this function was generated using a structured prompt methodology. The relevant system and user prompts that guided the code generation are detailed in Appendix A, specifically in Listings 15 and 16.

2.2.3 Initialize Variables. This code module initializes the sampling of free parameters used either for initializing the $\mathcal{GP}$-optimizer (Section 2.3.2) or for executing a brute-force optimization routine (Section 2.3.1). Accurate specification of parameter boundaries is crucial for effective sampling, and a separate constraints file is used to ensure all geometric requirements are satisfied.

In the current implementation, the sampling procedure is guided exclusively by an in-context example, as described in the system and user prompts (Appendix A, Prompts 17–18). These prompts define both the variable boundaries and the method for sampling.

In future iterations, this initialization process may be enhanced by leveraging previously sampled or optimized designs to provide informed priors for the parameter distributions.

2.3 Optimization

We assume, that the exact physical forward model $\hat{y} = f_\theta(X)$ that has been generated by the numerical simulator accessed by the LLM tool chain delivers a quantitative response $y \in \mathbb{R}^{n_1 \times \cdots \times n_p}$, where each n_i (for $i = 1, \ldots, p$) represents a certain instance of the board design response such as e.g. scattering parameters $S(f)$ over a defined bandwidth $\mathcal{B} = [f_1, f_2]$, time domain reflectivity curves for a defined e.g., Gaussian input pulse or electric field strength $\vec{E}$, impedance $Z(f)$ or phase at given board positions. Moreover, we define an optimization objective function $\mathcal{L} : \mathcal{X} \to \mathbb{R}$ that has to be minimized, e.g., in the form of an L_2-norm. The parameter vector θ contains all immutable, fixed parameters of the physical forward model.

$$X^* = \arg\min_{X \in \mathcal{X}} \mathcal{L}(X), \tag{7}$$

$$\mathcal{L}(X) = \|y - f_\theta(X)\|^2. \tag{8}$$

We compare two optimization methods built on top of the code generation, the Brute-Force Optimizer and the Gaussian Process Bayesian Optimization. These are explained and evaluated in the following section.

2.3.1 Brute-Force Optimizer. The code generation toolchain is designed to generate code that applies uniform sampling to the free parameters, adhering to the constraints specified for a given problem. These constraints, defined in a generated constraints file, limit the range and behavior of the random variables. All variables must satisfy the specified constraints, including those involving multiple random variables. This sampler can be used to create a dataset to train a surrogate model $\hat{y} = f_\theta(X_i)$. The sampler itself can be used as a brute-force optimizer, serving as a baseline. Due to complex constraints and for a small feature space, this can be an efficient way to explore the feature and solution space. In addition to the sampling, the Brute-Force Optimizer can be used as a baseline to compare with the ($\mathcal{GP}$-BO) method, as it is guaranteed to converge.

2.3.2 Gaussian Process Bayesian Optimization. Bayesian optimization (BO) is an efficient strategy for optimizing functions that are expensive to evaluate [39]. Given an objective function l, a surrogate model $\tilde{l}_\theta$ is trained to approximate it. In our case, the surrogate model is a Gaussian Process ($\mathcal{GP}$) regressor [46] parameterized by θ. The $\mathcal{GP}$ is initially trained on a dataset $\{(X_{v_\text{init}}, Y_{v_\text{init}})\}$, where $Y_{v_\text{init}} = g(X_{v_\text{init}})$. When applied to the optimization problem defined in Equation 8, the $\mathcal{GP}$-BO results in the following formulation:

$$X_v^* = \arg\min_{X_v \in \mathcal{X}} g_\theta(X_v). \tag{9}$$

Table 2. Design Parameters for via and Stripline Transitions

No.	Parameter	Description	Type [Unit]
1	Via_Sig	Diameter of vertical signal via	Discrete [mm]
2	Pad_Sig	Catchpad diameter for signal via	Discrete [mm]
3	Via_GND	Diameter of ground via	Discrete [mm]
4	Pad_GND	Catchpad diameter for ground via	Discrete [mm]
5	R_GND	Radial distance of ground vias from signal via center	Continuous [mm]
6	Angle_GND	Angular spacing between adjacent ground vias	Continuous [deg]
7	N_GND	Number of ground vias around the signal via	Integer [–]
8	Stub_Len	Length of unconnected signal via portion (stub)	Continuous [mm]
9	W_1	Width of first stripline taper section	Continuous [mm]
10	W_2	Width of second stripline taper section	Continuous [mm]
11	L_1	Length of first stripline taper section	Continuous [mm]
12	L_2	Length of second stripline taper section	Continuous [mm]
13	Cutout_R1	Radius of ground cutout in layer 1 around signal via	Continuous [mm]

$\mathcal{GP}$-BO proceeds iteratively. At each iteration t, a new candidate point X_v^t is selected by optimizing an *acquisition function* over the surrogate model. The full optimization procedure is summarized in Algorithm 1.

Acquisition Function and Constraint Handling: We employ the *Expected Improvement* criterion [40] as the acquisition function basis. Since the problem is subject to inequality constraints C, we adopt the constrained $\mathcal{GP}$-BO method proposed by Gardner et al. [18], which incorporates constraint handling directly into the acquisition function, penalizing candidates that violate constraints. We utilize the Python library described in Reference [42], where infeasible candidates are appropriately penalized based on evaluations from our LLM-generated constraint-checking routines.

Parallel and Asynchronous Optimization Strategies: To better exploit HPC resources, batch-parallel BO methods can be employed, selecting multiple candidates per iteration using criteria such as q-EI [9, 36] or its numerically stable variant q-LogEI [2]. Beyond synchronous batching, asynchronous strategies (e.g., Thompson Sampling; [26]) offer a promising direction to reduce idle time under heterogeneous evaluation costs.

Scalability in High-Dimensional Design Spaces: As design dimensionality increases, several complementary strategies may be integrated to maintain $\mathcal{GP}$-BO performance. These include principled scaling of length-scale priors for vanilla $\mathcal{GP}$-BO [24], methods exploiting low intrinsic dimensionality such as TuRBO [13] or sparse additive models (SAASBO; [12]), and sparse $\mathcal{GP}$ approximations based on inducing points [54]. Together, these approaches provide a pathway to extend $\mathcal{GP}$-BO to increasingly complex multi-layer design problems.

Figure 6 compares the convergence behavior of the two optimization algorithms on a constrained optimization task involving a multilayer PCB layout with a minimum of 9 free parameters. A detailed description of the setup and results is provided in Section 3. As discussed therein, $\mathcal{GP}$-BO exhibits faster convergence and achieves better solutions compared with the brute-force optimizer.

Note: The number and type of parameters may vary depending on the PCB layer configuration and the number of stripline taper sections. The minimum set consists of 9 parameters for a single-layer design with no taper.

3 Experiments

The toolchain was employed to optimize the transition between a coaxial connector and a **printed circuit board** (**PCB**) —a complex task, given the significant mode conversions that occur at the

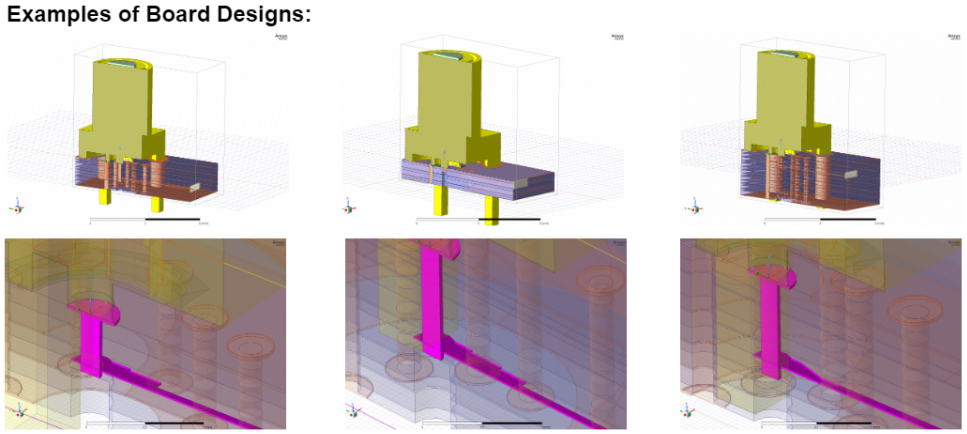

Fig. 5. Examples of Connector-PCB transitions used as examples to optimize with the proposed toolchain. The boards have different numbers of layers, materials, number of parameters, and the output can be on different layers (top). The micro stripline can be designed as a taper with different shapes (multi-step taper, linear, ...) depending on the application or requirements (bottom).

interface between the coaxial connector, the PCB, and the subsequent microstrip line. For high-frequency ultra-broadband applications with stringent performance requirements, such as those in modern information transmission systems, maximizing signal integrity and minimizing reflections is essential. Adjusting board parameters enables waveform shaping to reduce impedance mismatch and thereby smoothen transitions, leveraging interference effects to mitigate reflection issues. This process requires detailed design expertise and a thorough understanding of the electromagnetic behavior within the system. Figure 5 illustrates examples of such connector-to-PCB transitions from this study. Ground vias and ground plane cutouts were incorporated to shape the signal propagation in the vertical transition. Additionally, a tapered stripline was implemented to facilitate a smooth transition between the via catchpad and the microstrip line, effectively guiding the wave from the vertical to the horizontal plane. Due to the dynamic range of parameters, fully automating this simulation process is challenging. The clearly defined problem setup and geometry make this scenario ideal for the toolchain's capabilities. Specifications and requirements can be defined in a structured text file, with performance quantified through scattering parameters.

By leveraging BO, the toolchain significantly reduces the need for extensive training data compared with neural network surrogate models, such as those in References [25, 41].

The proposed framework was evaluated in a real-world scenario by generating simulation and parameter sampling code for connector-PCB transitions, as illustrated in Figure 5. The results demonstrate that the toolchain can generalize across varying specifications, including changes in parameters and topologies. The code generation system, powered by a LLM agent, is capable of extending existing code by introducing new elements without requiring domain-specific knowledge of the design.

One notable limitation is the model's ability to manage long-context files and accurately extract information from them. Among the models tested, Google's Gemini 1.5 Pro [14] achieved the best performance, outperforming OpenAI's GPT-4o [43]. This advantage may be attributed to Gemini's optimization for long-context applications (Gemini 1.5 Pro: 1M tokens vs. GPT-4o: 128k tokens). With the rapid development of LLMs, it is reasonable to expect incremental improvements in these capabilities over time; however, the results presented here should be interpreted as a snapshot

reflecting the state of the art at the time of evaluation rather than as definitive or long-term conclusions.

To evaluate the information extraction capabilities of large language models, we constructed a benchmark dataset comprising 50 synthetically generated specification files modeled after real-world RF-domain microchip test boards. Each specification file contained up to 20 dielectric layers with predefined parameters. The models were tasked with generating the corresponding problem definition files from these inputs. Table 3 summarizes the results of this evaluation. The proposed method offers broad applicability across diverse designs and operates independently of specific input examples or parameters, making it both efficient and highly adaptable. The primary limitation lies in the computational efficiency of the Gaussian process-based BO ($\mathcal{GP}$-BO). This constraint can be mitigated through two approaches: populating a reference database with representative examples or implementing more advanced $\mathcal{GP}$-BO algorithms.

Table 3. Percentage of Correctly Generated Parameter Files under pass@1, Including both Accurate Board Structures and All Associated Parameters

Model	Context Pair	Single Context
Gemini 1.5 Pro	84.0%	32.0%
GPT-4	40.0%	40.0%

In a single-shot setting where both the input specification and output context files were provided, Gemini successfully extracted all parameters in 84.0% of cases under pass@1 [15], with pass@2 increasing the success rate to 91.0%. The remaining 16% of failure cases reveal systematic patterns, particularly a higher error frequency in the generation of constraints. These failures appear to stem from a limited geometric understanding. While Gemini is proficient at extending or adapting known patterns from in-context information, it struggles with the spatial reasoning required for complex constraint definitions.

Within the $\text{LLM}_{\text{specs}}$ model–the core stage of our pipeline–these initial errors are critical. Although downstream agents can leverage in-context examples to correct minor structural errors, fundamental parameter inaccuracies tend to propagate through the pipeline, ultimately degrading the quality of the final code generation. Additional improvements could be achieved by partitioning the board design into smaller sections or incorporating compiler feedback for automatic refinement in agentic mode. In contrast, when only the reference output file was used as context, Gemini's accuracy dropped significantly to 32.0%. GPT-4 maintained a consistent 40.0% accuracy across both settings, though it exhibited more hallucinations, especially in files with longer content. Despite these limitations, both Gemini and GPT-4 remain among the leading models for code generation. These findings align with results reported in Reference [34], where Gemini-1.5-Flash, GPT-4o, and Anthropic's Claude-3 Haiku [5] were evaluated for long-context performance. Gemini-1.5-Flash demonstrated the smallest decline in performance, highlighting its superior robustness and consistency when handling extended context and long-text generation tasks.

The test scenario also included multiple real-world optimization examples, each described in a structured JSON specification file. The Gemini 1.5 Pro model [14] was selected for code generation based on its superior performance in prior evaluations, particularly with long-context handling. The generated code, combined with a sampling configuration file, was used to initialize a Gaussian process for optimization. A constraints code file was also generated to validate whether the proposed candidate points resided within the feasible design space.

To assess the performance, the $\mathcal{GP}$-BO algorithm was compared against BF optimization. The BF optimization achieves strong convergence through extensive parameter sampling. The free parameters are extracted from the specs file and vary depending on the board setup. An example of the optimization parameters is given in Table 2. Figure 6 illustrates that $\mathcal{GP}$-BO improves upon the initial design more rapidly, especially in the first epoch. After 100 iterations, $\mathcal{GP}$-BO identifies an

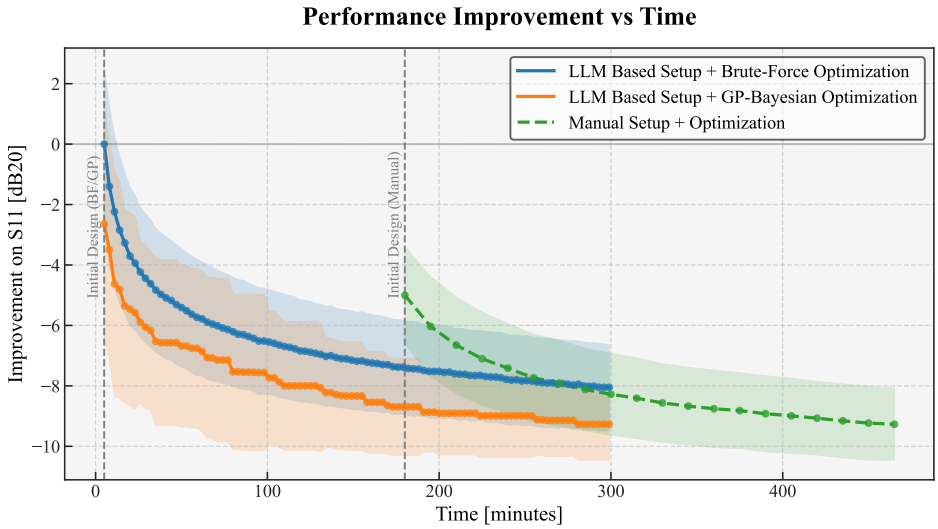

Fig. 6. Improvement of the $\mathcal{GP}$-BO algorithm and the BF-Optimizer. The $\mathcal{GP}$-BO has a faster convergence compared with the BF-Optimizer and finds better solutions over the whole optimization process.

overall superior solution compared with BF optimization. The improvement metric is based on the reduction in signal reflections within the board, as measured by the S_{21} parameter. By comparing the cGenEDA framework with the current design workflow–where designs are manually set up based on customer specifications, constraints are defined, and optimization is performed semi-automatically using Ansys integrated tools under human instruction guidance–the advantages of cGenEDA become evident, particularly in reducing the time required for design initialization and optimization.

In the conventional approach, manual setup of a multilayer board typically requires 8 hours, followed by approximately 20 optimization steps, each taking around 15 minutes and guided by domain knowledge. In total, this results in a design cycle of roughly 13 hours for complex boards. In contrast, cGenEDA requires only about 5 minutes for automatic code generation of the design setup, and optimization iterations can be executed in approximately 3 minutes each. While the framework may initially require a larger number of iterations ($\approx$ 100 in this study, due to the absence of embedded domain knowledge), the overall optimization process remains significantly more time-efficient. The information corresponds to the current costs for designing a board for the relevant areas of application.

This comparison highlights that cGenEDA not only accelerates the initial design setup but also enables a scalable optimization process, reducing the total design effort from several hours to less than one hour, thereby demonstrating its potential to substantially improve the efficiency of PCB design workflows.

$$\mathcal{L} = \int_{f_{\min}}^{f_{\max}} |S_{21}(f)|^2 \, df. \tag{10}$$

Table 4 presents the improvement of the objective function 8 between a frequency range $f_{\min} =$ 0.1 GHz and $f_{\max} = 67$ GHz values at different iteration epochs, based on the initially sampled design. The objective function is defined as the mean of the S_{21} parameters in dB over the entire frequency range from 0.1 GHz to 67 GHz. This objective can vary depending on the application and may be defined and weighted differently across frequency regions. It can include considerations of S_{11} and S_{22} reflection or S_{21} and S_{12} transmission scattering parameters.

Furthermore, Table 4 quantifies the performance of the optimization algorithms, revealing that the mean overall improvement after 100 epochs is more than −1 dB greater when using the $\mathcal{GP}$-BO algorithm compared with the BF optimizer. Additionally, the final improvement achieved by the BF optimizer (approximately −8 dB after 100 iterations) is attained after 40 iterations with the $\mathcal{GP}$-BO algorithm, highlighting its faster convergence rate.

Table 4. Improvement of the Mean Value on the S21 Parameter

Iterations	**BF-Optimizer**	**$\mathcal{GP}$-BO**
10	−4.84 dB	−6.57 dB
20	−5.94 dB	−7.12 dB
40	−6.93 dB	−8.04 dB
60	−7.48 dB	−8.74 dB
100	−8.09 dB	−9.32 dB

The optimized results presented in Figure 9 demonstrate significant improvements in the scattering parameters across multiple board designs after 60 optimization iterations. Notably, the S_{11} parameter shows a substantial reduction across the entire frequency range from 0 to 67 GHz, indicating a consistently improved impedance matching performance compared with the initial design.

In optimization runs 1, 2, and 4, the resonance peaks in the S_{21} parameter were effectively reduced, resulting in smoother transmission characteristics. This improvement is particularly advantageous for broadband signal applications in information technology, where minimizing ripples is crucial for signal integrity.

However, in optimization run 3, increased damping was observed in the S_{21} parameter beyond 50 GHz. This limitation is attributed to the technical and manufacturing constraints inherent to the design space, which restrict the extent of achievable performance enhancement. Despite these limitations, the improvements achieved within the feasible design space remain significant.

Overall, these results demonstrate that the proposed method consistently identifies superior design solutions within the given constraints, achieving substantial performance gains in only 60 optimization steps. This represents a considerable acceleration in design convergence and enhanced flexibility in addressing challenging design problems compared with current non-LLM-based methods.

The S_{11} parameter curve was consistently reduced across the entire frequency domain, aligning with its designation as the primary optimization objective.

The optimization runs presented in Figure 6 provide a statistical analysis of convergence for board designs without incorporating any prior knowledge of feature importance. These results demonstrate that, even for mixed-integer problems, when initialized with entirely new designs, the proposed method outperforms the conventional design approach and significantly reduces optimization time. A key strength of the framework lies in its ability to continuously improve with each validated design. Such designs can be reused as in-context examples for the LLMs, serve as parameter initialization, and provide initial samples for the $\mathcal{GP}$-BO. By selecting more suitable initial parameter sets, the optimization process can be accelerated, as the number of required iterations decreases due to a more localized search.

Figure 7 illustrates how the quality of the initial design scales with the number of entries in the database, compared with random initialization. Notably, even with a relatively small dataset of 50 designs, the initial performance is substantially improved, moving the framework closer to the behavior of a skilled designer with domain knowledge of critical board parameters. It is important to emphasize that a nearest-neighbor search in the parameter space alone is insufficient and corresponds to random initialization. By contrast, balancing parameter similarity with board quality yields notable improvements. To achieve this, the retrieval method first identifies the board type and subsequently performs a cosine similarity search on parameters weighted by performance, as measured through S_{11} values.

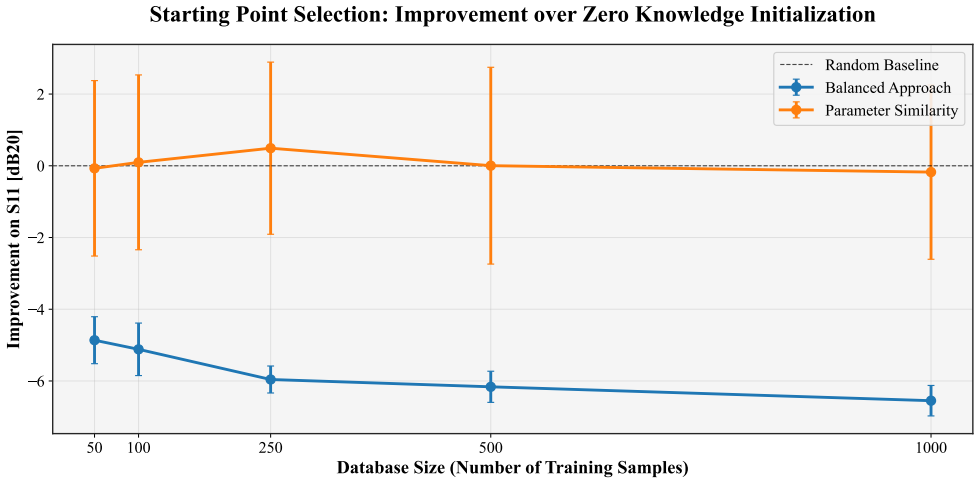

Fig. 7. Improvement of the selected starting point as a function of database size. With each optimization cycle and validated design, the system continuously enhances its ability to retrieve well-performing designs as initial candidates, thereby reducing optimization time and increasing the likelihood of correct board generation.

Another important aspect of the framework is its ability to generalize to previously unseen board designs, such as novel stackups or signal output layers. In these cases, the database retrieval system identifies structurally similar designs, which can then be used to approximate the new configuration. Compared with a human designer, the cGenEDA framework demonstrates a clear advantage immediately after design initialization. The initial performance benefits from database-driven generalization, resulting in significantly better outcomes than manual setup.

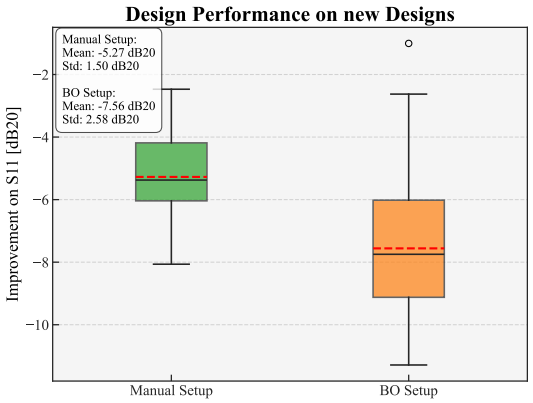

Fig. 8. Initial design performance comparison. Database-driven BO with cGenEDA initialization significantly outperforms manual setup, demonstrating effective generalization.

This improvement is quantitatively illustrated by the statistics: manual setup achieves a mean of −5.27 dB (Std: 1.50dB), whereas the BO initialization within cGenEDA achieves a mean of −7.56 dB(Std: 2.58 dB), corresponding to a performance advantage of approximately −2.28 dB. These results highlight the framework's ability to leverage prior designs to enhance initial design quality and initialize designs like a human designer.

To summarize, we can state the following advantages of the proposed methodology over the state-of-the-art engineering process in EDA:

(a) *Efficiency:* These stable results enable direct application to the exact HFSS simulations while significantly reducing the required number of computationally intensive simulations. This efficiency highlights the effectiveness of the optimization approach in managing high-fidelity simulations.

(b) *Wider Exploration and smooth integration of conceptual or topological changes:* The proposed technology enables rapid exploration–exploitation cycles between the electronic design engineer and the optimization tool, facilitating efficient design iteration. Engineers can conduct advanced design studies that go beyond simple parametric modifications. This includes topological changes such as:
— Adding or removing ground vias,
— Modifying signal layer outputs,
— Introducing backdrills to mitigate stubs,
— Changing the shape of tapered signal striplines,
— Adding via fences (via stitching to create a shielded stripline),
— Rapidly changing the signal output layer.
By instructing the LLM to analyze and propose modifications to different geometrical shapes, designs can be iteratively refined. Representative examples are provided in the Appendix (Figures 19, 20, and 21). The system accommodates these modifications by automatically adding and adapting the relevant code sections for geometry generation, updating geometric constraints, and introducing new optimization variables as required.
However, a limitation arises when a new topology differs substantially from the available contextual examples. In such cases, the system may struggle to autonomously generate the complex geometric constraints necessary to ensure that optimization parameters remain valid. At present, constraint extraction is most robust when the LLM can reference similar in-context examples.

4 Conclusion

The proposed method demonstrates the effectiveness of utilizing LLMs in electronic design optimization applications, particularly when integrating 3D representations. By formulating geometric data as structured text and employing multiple LLM domain expert agents, the approach successfully generates the complete code for the optimization loop, which can be executed to solve specific optimization problems. To the best of our knowledge, this methodology represents one of the first systems capable of generating full 3D electronic designs from input specifications and automatically optimizing them.

This method allows for rapid modifications of geometry by providing task-specific examples, thereby enhancing the exploration phase of the design process and facilitating the development of novel and innovative designs. However, challenges remain when addressing designs that involve complex constraints and dependencies, particularly when using a method that lacks intrinsic domain knowledge. Additionally, the text-based format does not inherently capture geometric dependencies critical for creating physically and manufacturably meaningful designs.

Nonetheless, this approach serves as a valuable tool for accelerating the design process, functioning as an assistive system, and marking a significant first step toward further research in EDA and optimization.

Furthermore, the results demonstrate that this method can consistently produce design solutions with significantly improved performance characteristics within a few iteration steps. By efficiently exploring the design space and automatically adapting to various design constraints, the method enables faster convergence and enhanced flexibility compared with traditional non-LLM-based optimization approaches. This reduces the time and effort required to achieve high-performance designs, all without the need for fine-tuning the LLM agents. This represents a substantial improvement in the electronic design workflow and highlights the potential of LLM-based methods to transform design automation by making advanced optimization techniques more accessible and adaptable to real-world engineering challenges.



Fig. 9. Example results for different optimization iterations. S_{11} and S_{12} scatter-parameters for the initial and optimized designs. It shows a reduction of the reflections in the S_{11} parameter, as formulated in the objective function.

5 Outlook

The integration of LLMs into electronic design is a rapidly emerging field, with applications ranging from chip design to analog circuit development. Code serves as a vital communication medium among various stakeholders, including engineers, specification documents, and software tools. As advancements in LLMs continue to evolve, significant research efforts are being directed toward improving code generation. Enhancing code quality, generating diverse code snippets, and

Example of LLM Prompt and Response

Prompt: In the PCB setup, a taper with two steps is used for the impedance adjustment. Now the code should be adjusted to three steps. Answer the following questions:

— Which sections in the parameter file and code file describe the taper?
— Which parameters have to be added or adjusted for a taper with three steps?
— Which code sections have to be added or adjusted for a taper with three steps?
— Which geometrical constraints have to be added for a taper with three steps?
— Which alternative taper geometries could be used instead of a step taper, for better impedance adjustments?

Response: You're right, modifying the existing two-step taper to a three-step taper requires careful adjustments in both the parameter file and the Python code. Here's a breakdown of the changes:

1. Identifying Taper Sections:

- **Parameter File:** The parameters `length_stripline_section1`, `length_stripline_section2`, `width_stripline_section1`, and `width_stripline_section2` define the two-step taper. The constraints `Maximum length taper sections` and `Maximum width taper section 1/2` are related to the taper geometry.
- **Code:** The code block starting with `create layer 1 signal stripline` handles the creation of the stripline sections.

2. Parameter Adjustments & Additions:

- **New Parameters:**
  - `length_stripline_section3`: Length of the third taper section (float, mm).
  - `width_stripline_section3`: Width of the third taper section (float, mm).

3. Code Modifications:

- **Add Stripline Section:** Within the `create layer 1 signal stripline` block, add:

```
strip_section3_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section3/2',
    '-length_stripline_section1-length_stripline_section2',
    '-thk_layer1'], ['width_stripline_section3',
    '-length_stripline_section3', 'thk_layer1'],
    name='layer1_stripSection3', material='copper')
```

developing functions that collaboratively contribute to comprehensive projects are central themes in this domain.

To optimize code generation further, leveraging retriever systems that access databases of code documents, API documentation, and related projects could be beneficial. Implementing strategies to automatically segment code into smaller chunks would reduce context length and enhance processing efficiency. Additionally, establishing a cohesive system to integrate different agents and align the generated files, such as ensuring compatibility among interfaces, could streamline the overall development process.

In the realm of optimization, LLMs hold the potential to enhance optimization algorithms, as suggested in Reference [33]. They can serve as optimization agents in complex mixed-integer optimization scenarios, and the concept of employing a reward model to fine-tune LLMs for generating more effective initial design guesses presents an intriguing avenue for further exploration.

A critical aspect of this research is the representation of geometry, which may explain the prevalence of studies focusing on chip design utilizing HDLs or abstract representations of analog circuits. Direct interaction with 3D geometry necessitates a representation that can seamlessly integrate with LLMs. Future work could explore *graph-based representations* of elements using primitives, as well as geometrical deep learning techniques [20], or adopt mesh- and resolution-free representations of geometry, as demonstrated in the context of tapered lines by Reference [55]. The

Example of LLM Prompt and Response

- **Modify `signal_strip_obj`:** Adjust the position and length of the final stripline segment:

```
hfss_obj['length_stripline_out'] =
    'length_substrate-length_stripline_section1-
    length_stripline_section2-length_stripline_section3-
    offset_substrate'
signal_strip_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_out/2',
     '-length_stripline_section1-length_stripline_section2-
     length_stripline_section3', '-thk_layer1'],
     ['width_stripline_out', '-length_stripline_out',
     'thk_layer1'], name='layer1_signalStrip',
     material='copper')
```

- **Unite Stripline Sections:** Include `'layer1_stripSection3'` in the `hfss_obj.modeler.unite(...)` call.

4. Geometrical Constraint Additions:

- **Maximum Length:** Update the `Maximum length taper sections` constraint:

```
# Maximum length taper section:
(length_stripline_section1 + length_stripline_section2 +
length_stripline_section3 + offset_substrate) - length_substrate' <= 0
```

- **Maximum Width:** Add a constraint for the third section's width:

```
 # Maximum width taper section 3:
 width_stripline_section3 - width_stripline_section2 <= 0
```

5. Alternative Taper Geometries:

- **Exponential Taper:** Provides a smoother transition, potentially leading to better impedance matching over a wider frequency range.
- **Klopfenstein Taper:** A more advanced taper design that aims to minimize reflections over a specified bandwidth. It often provides superior performance compared with step or exponential tapers.
- **Bezier Curve Taper:** Offers flexibility in controlling the taper shape, allowing for fine-tuning of the impedance matching characteristics.

Implementation Notes:

- **pyAEDT API:** pyAEDT might have built-in functions for creating these advanced taper types. Refer to the API documentation for specific methods.
- **Optimization:** Consider using optimization algorithms within Ansys HFSS or pyAEDT to automatically determine the optimal taper parameters (lengths, widths) for your desired impedance matching performance.

Fig. 10. Example of prompting to give information about the thoughts for changing the tapered microstip line.

textual description file lacks in describing physical dependencies on the geometry level, as further described in Section 4.

Building on the findings of this research, there remains a significant opportunity for improvement. We will focus on extending the toolchain to enhance its generalizability and robustness by coupling the LLMs to a *knowledge graph* that represents geometrical and physical relationships as well as functional, physical, and geometrical properties of each element, enabling the toolchain to accommodate larger design space changes and further expand the exploration capabilities of electronic design optimization.

References

[1] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. 2024. Generating CAD Code with Vision-Language Models for 3D Designs. arXiv:2410.05340. DOI: 10.48550/arXiv.2410.05340

[2] Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. 2025. Unexpected Improvements to Expected Improvement for Bayesian Optimization. arXiv:2310.20708. DOI: 10.48550/arXiv.2310.20708

[3] ANSYS, Inc. 2023. Ansys [pyAEDT], Release [0.10.2]. Retrieved from https://aedt.docs.pyansys.com/version/stable/. Software.

[4] ANSYS, Inc. 2025. ANSYS HFSS. Retrieved from https://www.ansys.com/products/electronics/ansys-hfss. ANSYS High Frequency Structure Simulator (HFSS).

[5] Anthropic. 2024. Introducing the next generation of Claude. Retrieved from https://www.anthropic.com/news/claude-3-family. Claude Haiku large language model.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell. 2020. Language Models are Few-shot Learners. arXiv:2005.14165. DOI: 10.48550/2005.14165

[7] Cadence Design Systems, Inc. 2025. Cadence Clarity 3D Solver. Retrieved from https://www.cadence.com/en_US/home/tools/system-analysis/em-solver/clarity-3d-solver.html. 3D Finite Element Analysis Electromagnetic Solver.

[8] Chen-Chia Chang, Yikang Shen, Shaoze Fan, Jing Li, Shun Zhang, Ningyuan Cao, Yiran Chen, and Xin Zhang. 2024. LaMAGIC: Language-Model-based Topology Generation for Analog Integrated Circuits. arXiv:2407.18269 [cs.AR] https://arxiv.org/abs/2407.18269

[9] Clément Chevalier and David Ginsbourger. 2013. Fast computation of the multi-points expected improvement with applications in batch selection. In *Revised Selected Papers of the 7th International Conference on Learning and Intelligent Optimization - Volume 7997* (Catania, Italy) *(LION 7)*. Springer-Verlag, Berlin, 59–69. DOI: https://doi.org/10.1007/978-3-642-44973-4_7

[10] COMSOL AB. 2025. COMSOL Multiphysics. Retrieved from https://www.comsol.com/rf-module. Multiphysics Simulation Software.

[11] Dassault Systèmes. 2025. CST Studio Suite. Retrieved from https://www.3ds.com/products/simulia/cst-studio-suite. Full-wave FEM simulation software.

[12] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable Expected Hypervolume Improvement for Parallel Multi-objective Bayesian Optimization. arXiv:2006.05078. DOI: 10.48550/arXiv.2006.05078

[13] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D. Turner, and Matthias Poloczek. 2019. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/6c990b7aca7bc7058f5e98ea909e924b-Paper.pdf

[14] Gemini Team et al. 2024. Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context. arXiv:2403.05530. DOI: 10.48550/arXiv.2403.05530

[15] Mark Chen et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374. DOI: 10.48550/arXiv.2107.03374

[16] James Flemings, Wanrong Zhang, Bo Jiang, Zafar Takhirov, and Murali Annavaram. 2024. Characterizing Context Influence and Hallucination in Summarization. arXiv:2410.03026. DOI: 10.48550/arXiv.2410.03026

[17] Yonggan Fu, Yongan Zhang, Zhongzhi Yu, Sixu Li, Zhifan Ye, Chaojian Li, Cheng Wan, and Yingyan Lin. 2023. GPT4AIGChip: Towards Next-generation AI Accelerator Design Automation via Large Language Models. arXiv:2309.10730. DOI: 10.48550/2309.10730

[18] Jacob R. Gardner, Matt J. Kusner, Zhixiang Eddie Xu, Kilian Q. Weinberger, and John P. Cunningham. 2014. Bayesian optimization with inequality constraints. In *ICML*, Vol. 2014. 937–945.

[19] Roman Garnett. 2023. *Bayesian Optimization*. Cambridge University Press.

[20] Negar Heidari and Alexandros Iosifidis. 2024. Geometric Deep Learning for Computer-aided Design: A Survey. arXiv:2402.17695. DOI: 10.48550/arXiv.2402.17695

[21] Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. 2025. VerilogCoder: Autonomous Verilog Coding Agents with Graph-based Planning and Abstract Syntax Tree (AST)-based Waveform Tracing Tool. arXiv:2408.08927. DOI: 10.48550/2408.08927

[22] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 581–586. DOI: https://doi.org/10.1109/ASP-DAC47756.2020.9045559

[23] Shan Huang, Jinhao Li, Zhen Yu, Jiancai Ye, Jiaming Xu, Ningyi Xu, and Guohao Dai. 2025. *LLSM: LLM-Enhanced Logic Synthesis Model with EDA-Guided CoT Prompting, Hybrid Embedding and AIG-Tailored Acceleration*. Association for Computing Machinery, New York, NY, USA, 974–980. DOI: https://doi.org/10.1145/3658617.3697618

[24] Carl Hvarfner, Erik Orm Hellsten, and Luigi Nardi. 2024. Vanilla Bayesian Optimization Performs Great in High Dimensions. arXiv:2402.02229. DOI: 10.48550/arXiv.2402.02229

[25] Jing Jin, Feng Feng, Jianan Zhang, Shuxia Yan, Weicong Na, and Qijun Zhang. 2020. A novel deep neural network topology for parametric modeling of passive microwave components. *IEEE Access* 8 (2020), 82273–82285. DOI: https://doi.org/10.1109/ACCESS.2020.2991890

[26] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. 2018. Parallelised bayesian optimisation via thompson sampling. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 84)*. Amos Storkey and Fernando Perez-Cruz (Eds.). PMLR, 133–142. Retrieved from https://proceedings.mlr.press/v84/kandasamy18a.html

[27] Keysight Technologies. 2025. Keysight EMPro. Retrieved from https://www.keysight.com/us/en/products/software/pathwave-design-software/pathwave-em-design-software.html. 3D Electromagnetic Simulation Software.

[28] Yao Lai, Sungyoung Lee, Guojin Chen, Souradip Poddar, Mengkang Hu, David Z. Pan, and Ping Luo. 2024. AnalogCoder: Analog Circuit Design via Training-free Code Generation. arXiv:2405.14918. DOI: 10.48550/2405.14918

[29] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NIPS '20* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[30] Tian Liang, Zhiwei He,Wenxiang Jiao, XingWang, YanWang, RuiWang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging Divergent Thinking in Large Language Models through Multi-agent Debate. arXiv:2305.19118. DOI: 10.48550/arXiv.2305.19118

[31] Chengjie Liu, Weiyu Chen, Anlan Peng, Yuan Du, Li Du, and Jun Yang. 2024. AmpAgent: An LLM-based Multi-agent System for Multistage Amplifier Schematic Design from Literature for Process and Performance Porting. arXiv:2409.14739. DOI: 10.48550/arXiv.2409.14739

[32] Chengjie Liu, Yijiang Liu, Yuan Du, et al. 2024. LADAC: Large Language Model-driven Auto-Designer for Analog Circuits. Retrieved from https://doi.org/10.36227/techrxiv.12345678.v1. Preprint.

[33] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. 2024. Large Language Models to Enhance Bayesian Optimization. arXiv:2402.03921. DOI: 10.48550/arXiv.2402.03921

[34] Xiang Liu, Peijie Dong, Xuming Hu, and Xiaowen Chu. 2024. LongGenBench: Long-Context Generation Benchmark. arXiv:2410.04199 [cs.CL] https://arxiv.org/abs/2410.04199

[35] Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Shieh, and Wenmeng Zhou. 2024. CodexGraph: Bridging Large Language Models and Code Repositories via Code Graph Databases. arXiv:2408.03910. DOI: 10.48550/arXiv.2408.03910

[36] Sébastien Marmin, Clément Chevalier, and David Ginsbourger. 2019. Differentiating the Multipoint Expected Improvement for Optimal Batch Design. arXiv:1503.05509. DOI: 10.48550/arXiv.1503.05509

[37] Kyungjun Min, Seongbin Kwon, Sung-Yun Lee, Dohun Kim, Sunghye Park, and Seokhyeong Kang. 2023. ClusterNet: Routing congestion prediction and optimization using netlist clustering and graph neural networks. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. DOI: 10.1109/ICCAD57390.2023.10323942

[38] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, et al. 2021. A graph placement methodology for fast chipdesign. *Nature* 594, 7862 (01 Jun 2021), 207–212. DOI: https://doi.org/10.1038/s41586-021-03544-w

[39] J. Mockus. 2012. *Bayesian Approach to Global Optimization: Theory and Applications*. Springer Netherlands. Retrieved from https://books.google.ch/books?id=VuKoCAAAQBAJ

[40] J. Mockus, V. Tiesis, and A. Zilinskas. 1978. The application of Bayesian methods for seeking the extremum. *Towards Global Optimisation* 2 (1978), 117–129.

[41] Stephen Newberry and Ata Zadehgol. 2023. A novel deep neural network methodology for fast scattering-parameter extraction of discretized structures in three-dimensional space. (May 2023). DOI: https://doi.org/10.36227/techrxiv.22814807.v1

[42] Fernando Nogueira. 2014–. Bayesian Optimization: Open source constrained global optimization tool for Python. Retrieved from https://github.com/bayesian-optimization/BayesianOptimization

[43] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774. DOI: 10.48550/arXiv.2303.08774

[44] Jingyu Pan, Guanglei Zhou, Chen-Chia Chang, Isaac Jacobson, Jiang Hu, and Yiran Chen. 2025. A survey of research in large language models for electronic design automation. *ACM Trans. Des. Autom. Electron. Syst.* 30, 3, Article 34 (Feb. 2025), 21 pages. DOI: https://doi.org/10.1145/3715324

[45] Shiyuan Piao, Hong Gu, Aina Wang, and Pan Qin. 2024. A domain-adaptive physics-informed neural network for inverse problems of maxwell's equations in heterogeneous media. *IEEE Antennas and Wireless Propagation Letters* 23, 10 (2024), 2905–2909. DOI: https://doi.org/10.1109/LAWP.2024.3413851

[46] Carl Edward Rasmussen and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. The MIT Press.

[47] renanmav. 2025. pycst: CST Studio Suite API for Python. Retrieved from https://github.com/renanmav/pycst. Accessed: 2025-10-02.

[48] Renato Rimolo-Donadio, Sebastian Müller, Xiaomin Duan, Miroslav Kotzev, Heinz-D. Brüns, and Christian Schuster. 2012. Signal integrity: Efficient, physics-based via modeling: Principles and methods. *IEEE Electromagnetic Compatibility Magazine* 1, 1 (2012), 55–61. DOI: https://doi.org/10.1109/MEMC.2012.6244946

[49] Renato Rimolo-Donadio, Giuseppe Selli, Francesco de Paulis, Xiaoxiong Gu, Young H. Kwark, James L. Drewniak, Heinz-D. Bruens, and Christian Schuster. 2012. Signal integrity: Efficient, physics-based via modeling: Integration of striplines. *IEEE Electromagnetic Compatibility Magazine* 1, 2 (2012), 74–81. DOI: https://doi.org/10.1109/MEMC.2012.6244976

[50] Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. 2022. *PrefixRL: Optimization of Parallel Prefix Circuits Using Deep Reinforcement Learning*. IEEE Press, 853–858. DOI: https://doi.org/10.1109/DAC18074.2021.9586094

[51] Krzysztof Rusek, Jose Suarez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications* 38, 10 (Oct. 2020), 2260–2270. DOI: https://doi.org/10.1109/jsac.2020.3000405

[52] Shailja Thakur, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. 2024. AutoChip: Automating HDL Generation using LLM Feedback. arXiv:2311.04887. DOI: 10.48550/2311.04887

[53] Aaron Xuxiang Tian, Ruofan Zhang, Jiayao Tang, Young Min Cho, Xueqian Li, Qiang Yi, Ji Wang, Zhunping Zhang, Danrui Qi, Zekun Li, Xingyu Xiang, Sharath Chandra Guntuku, Lyle Ungar, Tianyu Shi, and Chi Wang. 2025. Beyond the Strongest LLM: Multi-turn Multi-agent Orchestration vs. Single LLMs on Benchmarks. arXiv:2509.23537. DOI: 10.48550/arXiv.2509.23537

[54] Michalis Titsias. 2009. Variational learning of inducing variables in sparse gaussian processes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 5)*. David van Dyk and Max Welling (Eds.). PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 567–574. Retrieved from https://proceedings.mlr.press/v5/titsias09a.html

[55] Philipp Gérard Trémuel, Efstratios Gavves, Christoph Würsch, Klaus Frick, and Robin Vetsch. 2023. Parameter-free neural field-based optimal design of nonuniform transmission lines. In *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 1–4. DOI: https://doi.org/10.1109/ICECS58634.2023.10382765

[56] Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. 2024. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models. arXiv:2311.16543. DOI: 10.48550/2311.16543

[57] Deepak Vungarala, Sakila Alam, Arnob Ghosh, and Shaahin Angizi. 2024. SPICEPilot: Navigating SPICE Code Generation and Simulation with AI Guidance. arXiv:2410.20553. DOI: 10.48550/arXiv.2410.20553

[58] Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025. CodeRAG-Bench: Can Retrieval Augment Code Generation? arXiv:2406.14497. DOI: 10.48550/arXiv.2406.14497

[59] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. ChatEDA: A large language model powered autonomous agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 10 (Oct. 2024), 3184–3197. DOI: https://doi.org/10.1109/tcad.2024.3383347

[60] Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. 2025. Divergent thoughts Toward One Goal: LLM-based Multi-agent Collaboration System for Electronic Design Automation. arXiv:2502.10857. DOI: 10.48550/2502.10857

[61] Yuxuan Yin, Yu Wang, Boxun Xu, and Peng Li. 2024. ADO-LLM: Analog Design Bayesian Optimization with In-context Learning of Large Language Models. arXiv:2406.18770. DOI: 10.48550/2406.18770

[62] Xinxin Zhang, Zhuoqun Xu, Guangpu Zhu, Chien Ming Jonathan Tay, Yongdong Cui, Boo Cheong Khoo, and Lailai Zhu. 2024. Using Large Language Models for Parametric Shape Optimization. arXiv:2412.08072. DOI: 10.48550/arXiv.2412.08072

A Appendix

Setup File Generation System Prompt

You are an electronics engineer and an expert in defining Printed Circuit Board designs. Given a specification file, you should create a structured file containing all the information and parameters to design a PCB. Here is an example of how to set a PCB parameter file from a given specification file: The example specification file
\n —-- \n {specs_example_context} \n —-- \n
Results in the following parameter file: \n —-- \n
{params_example_context} \n —-- \n
The Parameter file contains all the information. Write the parameter file in the same structured form as provided in the example to initialize the design and all parameters and geometrical constraints.
Make sure to have all information provided:
1. description
2. General
3. materials
4. parameters
5. geometrical constraints.
Customize the answer for the PCB with the following information available:

Fig. 11. System Prompt to generate the structured board setup file.

Setup File Generation User Prompt

t For the additional instructions: {instructions}
Customize the parameter file for the following specification file:
\n —-- \n {specs_file} \n —-- \n
Write the complete parameter file with the same structure as the parameter file in the example.
Start with this structure:

description: ...
general: ...
materials: ...
geometrical constraints: ...

Fig. 12. User Prompt to generate the structured board setup file.

Simulation Code Generation System Prompt

You are a coding assistant with expertise in numerical simulation with Ansys and the pyAEDT API.
Here is a Python pyAEDT code example to set up a PCB geometry in Ansys HFSS:
\n —— \n {code_context} \n —— \n
Write the complete pyAEDT code to generate the geometry in Ansys HFSS for the parameter file given in the user question.
Ensure the following JSON structure:

```
{
    "imports": "...", # of external tools
    "code": """...""", # code block
}
```

Customize the code example for the given parameters, defining the PCB without skipping any section.

Fig. 13. System Prompt for the board geometry pyAEDT code generation.

Simulation Code Generation User Prompt

Write the complete Python code to generate the PCB geometry for an Ansys EDT HFSS simulation using the pyAEDT API.
The board is defined by the following set of parameters:
\n —— \n {params_input_file} \n —— \n

Insert the generated function code in the «<FUNCTION CONTENT»> section.
Return the entire code block without any additional information, including the function definition and the content, in executable Python format following PEP 8 style.

The fixed and free parameters are already defined as variables of the HFSS instance.

Code:

```
import ...  # make sure of the correct import names depending on the project_name

def generate_geometry(hfss_obj):
    """
    Initialize the HFSS simulation using pyAEDT by creating an interactive
    An instance of HFSS.

    Args:
        hfss_obj: Interactive instance of HFSS

    Returns:
        None
    """
    <<<FUNCTION CONTENT>>>
    return
```

Respond in JSON with string keys: ”imports”: ”...”, ”code”: ”””...”””

Fig. 14. User Prompt for the board geometry pyAEDT code generation.

Constraints File Generation System Prompt

You are a coding assistant with expertise in numerical Simulation with Ansys and the pyAEDT API.
Here is a Python code example of a function to check if the geometrical constraints of a given PCB are fulfilled in Ansys HFSS
\n —— \n {code_context} \n —— \n
Write the complete Python code for a function to check if the geometrical constraints are fulfilled.
Ensure the following JSON structure

```
{
    "imports": "...", # list of external tools
    " code": """...""", # code block
}
```

Customize the code example for the parameters and constraints given in the parameter file without skipping any section.

Fig. 15. System Prompt for the geometrical constraints code generation.

Constraints File Generation User Prompt

Write the complete Python code for a function to check if all the constraints are fulfilled.
The board is defined by the following set of parameters and constraints:
\n ——
textbackslash n {params_input_file} \n —— \n
Insert the generated function code in the «<FUNCTION CONTENT»> section.
Return the entire code block, including the function definition and the content, in executable Python format following PEP 8 style.
List the function arguments and the required dictionary keys in the function documentation header.

Code:

```
import ...

def constraint_function(free_params, fixed_params):
    """
    Function to check if the geometrical constraints on the defined PCB are
    fulfilled,
    given the free and fixed board parameters.
    If the constraint is fulfilled, it returns a value < 0.

    Args:
        free_params: dictionary of free parameters to sample
        fixed_params: dictionary of pre-defined fixed parameters
    Returns:
        Array: array containing the constraint values

    """

    <<<FUNCTION CONTENT>>>
    return
```

Respond in JSON with string keys: ”imports”: ”...”, ”code”: ”””...”””.

Fig. 16. User Prompt for the geometrical constraints code generation.

Brute-Force Optimizer System Prompt

You are a coding assistant with expertise in numerical Simulation with Ansys and the pyAEDT API.
Here is a Python code example to sample not defined free parameters with respect to the given constraints to set up a PCB \n —–- \n {code_context} \n —–- \n.
Write the complete Python code to sample the free parameters from the parameter file given in the user question.
Ensure the following JSON structure

```
{
    "imports": "...", # of external tools
    "code": """...""", # code block
}
```

Customize the code example for the given parameters without skipping any section.

Fig. 17. System Prompt for the free geometry parameter sampling code generation.

Simulation Code Generation User Prompt

Write the complete Python code to sample the free parameters of a PCB.
The board is defined by the following set of parameters and the given constraints:
\n —–- \n {params_input_file} \n —–- \n
Insert the generated function code in the «<FUNCTION CONTENT»> section.
Return the entire code block, including the function definition and the content, in executable Python format following PEP 8 style.
List the function arguments and the required dictionary keys in the function documentation header.
IMPORTANT: Make sure you use the correct name for the custom import file.
Depending on the project name defined in the parameters file

Code:

```
import ... # import names depending on the project_name

def sampling_under_constraints(free_params, fixed_params):
    """
    Function to sample under defined constraints.

    Args:
        free_params: dictionary of free parameters to sample
        fixed_params: dictionary of pre-defined fixed parameters

    Returns:
        dict: dictionary containing the sampled free parameters
        dict: dictionary of pre-defined fixed parameters
    """

    <<<FUNCTION CONTENT>>>
    return
```

Respond in JSON with string keys: ”imports”: ”...”, ”code”: ”””...”””.

Fig. 18. User Prompt for the free geometry parameter sampling code generation.

Taper section Code Generation: Taper with two Sections

Example of a two signal stripline on the first layer with two taper sections.
Code:

```
""" constraints layer 1 signal stripline two taper sections """
free_params['length_stripline_section1'] + \
    free_params['length_stripline_section2'] + \
        fixed_params['offset_substrate']-fixed_params['length_substrate'],
free_params['width_stripline_section2']-free_params['width_stripline_section1'],
free_params['width_stripline_section1']-fixed_params['dia_catch_sigVia'],

""" create layer 1 signal stripline """
strip_section1_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section1/2', '0mm', '-thk_layer1'],
    ['width_stripline_section1', '-length_stripline_section1', 'thk_layer1'],
    name='layer1_stripSection1', material='copper')
strip_section2_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section2/2', '-length_stripline_section1', '-thk_layer1'],
    ['width_stripline_section2', '-length_stripline_section2', 'thk_layer1'],
    name='layer1_stripSection2', material='copper')
signal_strip_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_out/2',
     '-length_stripline_section1-length_stripline_section2',
     '-thk_layer1'],
    ['width_stripline_out',
     '-length_substrate+length_stripline_section1+length_stripline_section2 + \
        offset_substrate',
     'thk_layer1'],
    name='layer1_signalStrip', material='copper')
```

Fig. 19. Example of a two signal stripline on the first layer with two taper sections.

Code Modification: Transitioning from Two to Three Taper Sections

The user requests an additional impedance matching stage. The system must update both the constraint logic (to ensure the total length fits the substrate) and the geometry generation. (Compare to Figure: 19) Added/adapted Code:

```
# Adapted and extended constraint function for three taper sections
free_params['length_stripline_section1'] +
    free_params['length_stripline_section2'] +
    free_params['length_stripline_section3'] +
    fixed_params['offset_substrate']-fixed_params['length_substrate'],
free_params['width_stripline_section3']-free_params['width_stripline_section2'],
free_params['width_stripline_section2']-free_params['width_stripline_section1'],
free_params['width_stripline_section1']-fixed_params['dia_catch_sigVia'],

# Adapted and extended geometry creation for three taper sections
""" create layer 1 signal stripline three taper sections """
strip_section1_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section1/2', '0mm', '-thk_layer1'],
    ['width_stripline_section1', '-length_stripline_section1', 'thk_layer1'],
    name='layer1_stripSection1', material='copper')
strip_section2_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section2/2', '-length_stripline_section1', '-thk_layer1'],
    ['width_stripline_section2', '-length_stripline_section2', 'thk_layer1'],
    name='layer1_stripSection2', material='copper')
strip_section3_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_section3/2',
    '-length_stripline_section1-length_stripline_section2', '-thk_layer1'],
    ['width_stripline_section3', '-length_stripline_section3', 'thk_layer1'],
    name='layer1_stripSection3', material='copper')
signal_strip_obj = hfss_obj.modeler.create_box(
    ['-width_stripline_out/2',
     '-length_stripline_section1-length_stripline_section2 - \
        length_stripline_section3',
     '-thk_layer1'],
    ['width_stripline_out',
     '-length_substrate+length_stripline_section1+length_stripline_section2 + \
        length_stripline_section3+offset_substrate',
     'thk_layer1'],
    name='layer1_signalStrip', material='copper')
```

Fig. 20. Example of the system extending a signal path topology by adding a third taper section and corresponding geometric constraints.

Conceptual Change: Backdrill Implementation

To reduce signal reflections, the user instructs the system to add backdrilling. This requires creating a new cylindrical primitive and performing a boolean subtraction from the stackup.
Added Code Section:

```
# Logic for calculating backdrill depth and performing subtraction
backdrill_height = 'board_height' + 'thk_layer3' - 'stub_length'
backdrill = ctx.hfss.modeler.create_cylinder(
    orientation="Z",
    origin=[x, y, '-board_height'],
    radius='backdrill_diameter' / 2,
    height=backdrill_height,
    name=f"backdrill"
)

# Boolean operation to modify the existing topology
hfss.modeler.subtract(stackup_parts_names + [via_obj.name],
                      backdrill.name, keep_originals=False)
```

Fig. 21. Implementation of a backdrill topology change, demonstrating the framework's ability to handle boolean operations and vertical layer modifications.

```
<TIMESTAMP> - ERROR - hfss_obj.modeler.create_cylinder('Z', ['0mm', '0mm', '-
(thk_layer1+thk_layer2+thk_layer3)'], 'radius_antipad_layer3', 'thk_layer3', name='antipad_layer3',
material='material_layer')
<TIMESTAMP> - ERROR - File "<PYTHON_ENV>/site-packages/pyaedt/modeler/cad/Primitives3D.py", line 255, in
create_cylinder
<TIMESTAMP> - ERROR - new_object_name = self.oeditor.CreateCylinder(vArg1, vArg2)
<TIMESTAMP> - ERROR - AEDT API Error on create_cylinder
<TIMESTAMP> - ERROR - Last Electronics Desktop Message - [error] script macro error: cylinderparameters:
radius_antipad_layer3 is not a valid data for radius, radius should be greater than 0! ...
```

Fig. 22. pyAEDT *Error Logs*: Example showing logs generated by an incorrect constraint, resulting in an invalid parameter.

Received 2 June 2025; revised 27 February 2026; accepted 1 March 2026