2024

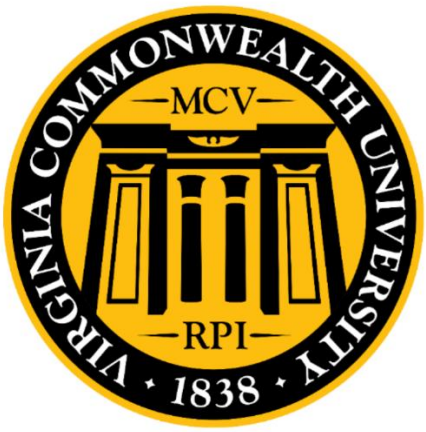
Machine Learning Assisted Optimization for Calculation and Automated Tuning of Antennas

Lauren Linkous
Virginia Commonwealth University

Machine Learning Assisted Optimization for Calculation and Automated Tuning of Antennas

A Dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy at Virginia Commonwealth University



by

Lauren Linkous

Bachelor of Science, Electrical Engineering, Virginia Commonwealth University, 2018

Bachelor of Science, Physics, Virginia Commonwealth University, 2018

Master of Science, Computer Science, Virginia Commonwealth University, 2024

Director: Erdem Topsakal, Ph.D.,

Senior Associate Dean for Strategic Initiatives and Enrollment Management,

Department of Electrical and Computer Engineering

Virginia Commonwealth University

Richmond, Virginia

September 2024

Acknowledgements

I would like to express my deep appreciation to all those who supported me throughout the completion of this work, and everything leading up to it. This would not have been possible without my friends and family, who have my heartfelt gratitude for their unwavering encouragement and enthusiasm, which kept me motivated during challenging times. Your belief in my abilities provided me with the strength to persevere, and your love has been a constant source of inspiration throughout this journey. I would also like to thank my advisor, who agreed to a dissertation with sparkly, glow-in-the-dark cat-shaped antennas. Your support was monumental to my success and will have an impact lasting much longer than all the glitter I have left in the lab.

Table of Contents

List of Abbreviations

Abbreviation	Meaning
AntennaCAT	Antenna Calculation and Autotuning Tool
API	Application Programming Interface
BW	Bandwidth
CAD	Computer-aided design
CEM	Computational Electromagnetics
EM	Electromagnetic
FDM	Finite Difference Method
FDTD	Finite Difference Time Domain
FEM	Finite Element Method
FETD	Finite Element Time Domain
FLC	Fuzzy Logic Controller
FNBW	First Null Beamwidth
GP	Gaussian Process
GUI	Graphical User Interface
HF	High Frequency
HFSS	(Ansys) High Frequency Simulation Software
HPBW	Half Power Beamwidth
KNN	K-Nearest Neighbor
ML	Machine Learning
MLAO	Machine Learning Assisted Optimization
MOM	Method of Moments
NN	Neural Network
PCB	Printed Circuit Board
PDE	Partial Differential Equation
PSO	Particle Swarm Optimization
RE	Regular Expression
RF	Radio Frequency
SDK	Software Development Kit
SO	(AntennaCAT) Simulation Object
SVM	Support Vector Machine
VBA	Visual Basic Analysis

List of Figures

List of Tables

Abstract

This document introduces the Antenna Calculation and Autotuning Tool (AntennaCAT) software suite, which automates the generation, computer-aided design (CAD) modeling, simulation, data collection, and optimization process for antenna design. AntennaCAT is the first comprehensive implementation of machine learning to automate, evaluate, and optimize the design process using several well-recognized commercially available EM simulation software. In particular, this work includes the capability to create and export structured datasets from the aforementioned EM software for iterative improvement and includes an expandable selection of optimizers.

As part of the antenna generation process, AntennaCAT has an antenna calculator for three topologies (a rectangular patch antenna, a half-wave monopole, and a quarter-wave dipole), and an expandable internal library of parameterized, customizable replication study designs included to encourage exploration. The AntennaCAT software also supports importing existing scripts and loading in project references for file modification. All calculated, replicated, imported, and loaded projects are compatible with the eleven optimizers included in the internal optimizer suite. This optimizer suite includes eight swarm-based optimizers, a sweep optimizer with random and grid search options, a Bayesian optimizer, a Python implemented MultiGLODS optimizer, and optional surrogate model kernels compatible with these optimizers. In total, there are 90 optimizer-surrogate model combinations, and additional configurations such as boundary condition handling which may cause unique optimizer behavior. With the internal design options, this creates more than 1,500 combination options for users to choose from, and then customize.

To address the breadth of designs AntennaCAT can optimize, a library of machine learning models has been implemented to suggest optimizer hyperparameters. Data collected using a suite of single-objective and multiple-objective benchmark functions was used to train a series of connected machine learning models for suggesting optimizer hyperparameters based on knowledge of the number of controllable problem variables and the number of target values being optimized. To encourage replication of research, an expandable internal library of parameterized designs is included for customization. Furthermore, this work integrates with several of the most frequently used EM simulation suites in research today, allowing for dynamic CAD model generation, optimization and tuning on most research platforms.

CHAPTER 1
Background: Computation, Automation, and Tuning

Antenna design is a complex process marked by the diverse requirements of modern communication systems. To keep pace with evolving technology, antennas come in various shapes and sizes, each designed for specific applications and operating frequencies. Antennas are essential components in wireless communication systems and take forms from easily recognizable devices such as radios, televisions, cell phones, and Wi-Fi routers, to complex networks of satellite communication systems. Each of these applications demand a careful balance of performance requirements, such as gain, bandwidth, impedance, and radiation pattern, and physical specification constraints including size, materials, durability, bendability, and weather proofing [1-6]. Achieving optimal performance requires navigating numerous challenges, including precise knowledge of wave propagation, radiation behavior, environmental factors, and electromagnetic interference, often necessitating advanced analytical tools and computational techniques [7-9]. Trade-offs between conflicting design parameters present additional challenges. For instance, increasing gain may require sacrificing antenna size or narrowing bandwidth. Fabricating antennas with precise geometries and material properties pose manufacturing challenges, particularly for high-frequency designs or complex structures, where tolerances and fabrication techniques play a critical role in performance.

Despite these challenges, effective antenna design is essential for enabling reliable and efficient communication in modern wireless systems. Computational electromagnetics (CEM) involves using numerical methods and computer simulations to analyze the behavior of electromagnetic fields. In the context of antenna design, computational electromagnetics allows for the modeling and simulation of antenna performance, predicting radiation patterns, analyzing impedance matching, and optimizing designs without the need for costly physical prototypes. However, while optimization techniques are employed to refine antenna designs and improve performance, this process is intrinsically interactive, experimental, and time-consuming [3, 7, 10].

To address these obstacles, much work has been done in automating processes for integrating optimization with electromagnetic (EM) simulation. EM simulation software suites may have built in optimizers [11-14], or researchers may have implemented their own custom solutions to meet

specific needs [15-17]. However, there exist several key issues with these methods and current resources, notably; 1) limitations with EM software-integrated optimizers, 2) lack of resources for applying new optimizers, 3) difficulty of replication of research on alternate systems or setups, and 4) a high barrier to entry into the field for students or young researchers.

To understand the computational difficulties of CEM, and the existing resources for antenna design, Chapter 1 provides a review of the current state of the art, and uses for automation, optimization, replication, and machine learning. This provides the motivation for the Antenna Calculation and Autotuning tool suite introduced in Chapter 2, the integrated optimization tools in Chapter 3, and our efforts to make antenna design simpler, replicable, and more accessible in Chapters 4-5, with example results in Chapter 6.

1.1 Computational Electromagnetics and Simulation

All computational electromagnetics begins with Maxwell's equations. Maxwell's equations are a set of fundamental partial differential equations (PDE) in classical electromagnetism named after the Scottish physicist James Clerk Maxwell who formulated them in the 19th century [18]. These equations are:

1. Gauss's law for electric fields

$$(\nabla \cdot \boldsymbol{E}) = \frac{\rho}{\epsilon_0} \qquad (eq.1)$$

Where **E** is the electric field, ρ is the charge density, and ϵ_0 is the permittivity of free space. This describes how electric fields originate from electric charges.

2. Gauss's law for magnetic fields

$$\nabla \cdot \boldsymbol{B} = 0 \qquad (eq.2)$$

Where **B** is the magnetic field. This describes how magnetic field lines in a system form closed loops.

3. Faraday's law of electromagnetic induction

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \qquad (eq.3)$$

Where **E** is the electric field and **B** is the magnetic field. This connects the relation between electric and magnetic fields, and how a changing magnetic field induces and electric field.

4. Ampère's law, with Maxwell's addition

$$\nabla \times \boldsymbol{B} = \mu_0 \boldsymbol{J} + \mu_0 \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \qquad (eq.4)$$

Where **B** is the magnetic field, **J** is the current density, μ_0 is the permeability of free space, and ϵ_0 is the permittivity of free space. This describes how magnetic fields are related to electrical currents and changing electric fields, and how material permeability and permittivity affect this relation.

Together, these equations describe the interaction of electric and magnetic fields, and how they propagate through space in the time domain. However, solving Maxwell's equations analytically is often challenging, if not impossible, for complex geometries and many boundary conditions. To address this, numerical methods are employed to approximate the solutions. One such method is the Finite Element Method (FEM), which converts the PDEs above to a system of algebraic equivalents.

Using this method, first, the set of Maxwell's equations must be transformed into the phasor domain. This simplifies the analysis of time-harmonic (steady-state) fields by representing fields as complex exponentials, and equations can be expressed in terms of sinusoidal functions [18], and the partial differential equations become algebraic equations. To simplify this step, and the following steps, the equation set can be rewritten as the following equalities:

$$\epsilon(\nabla \cdot \boldsymbol{E}) = \boldsymbol{\nabla} \cdot \mathbf{D} = \rho \qquad (eq.5)$$

Where **D** is the electric displacement. ϵ is still permittivity, though not specifically the permittivity of free space.

$$\mu(\nabla \cdot \boldsymbol{H}) = \nabla \cdot \boldsymbol{B} = 0 \qquad (eq.6)$$

Where **H** is the magnetic field strength, or the magnetic field intensity vector. μ is still permeability, though not necessarily the permeability of free space.

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} = -\mathrm{j}\omega\mu\boldsymbol{H} \qquad (eq.7)$$

Where **E** is still the electric field and **B** is the magnetic field. ω is the angular frequency of the electromagnetic wave, this represents the rate of oscillation of the electric and magnetic fields in the phasor domain. μ is the permeability of the medium, not necessarily of free space.

Through the relations in equations 5 and 6, Ampère's law in equation 4 then becomes:

$$\nabla \times \boldsymbol{B} = \boldsymbol{J} + \frac{\partial \boldsymbol{D}}{\partial t} \qquad (eq.8)$$

For solving Maxwell's equations in the frequency domain, and describing how waves propagate in a medium, the next step is to derive the Helmholtz partial differential equation. First, apply the curl to the phasor form of Faraday's Law, resulting in the following:

$$\begin{aligned} \nabla \times \boldsymbol{E} &= \text{-}\mathrm{j}\omega\mu\boldsymbol{H} \\ \nabla \times \nabla \times \boldsymbol{E} &= -\mathrm{j}\omega\mu(\nabla \times \boldsymbol{H}) \\ \nabla \times \boldsymbol{H} &= -\mathrm{j}\omega\varepsilon\boldsymbol{E} \\ \nabla \times \nabla \times \boldsymbol{E} &= -(\mathrm{j}\omega)^2\mu\varepsilon\boldsymbol{E} \end{aligned}$$

Then, simplfy the equation by grouping coefficients into γ such that:

$$\begin{aligned} \gamma &= \mathrm{j}\omega(\mu\varepsilon)^{1/2} \\ \nabla \times \nabla \times \boldsymbol{E} &= \gamma^2\boldsymbol{E} \\ \nabla(\nabla \cdot \boldsymbol{E}) &= \nabla^2\boldsymbol{E} - \gamma^2\boldsymbol{E} \\ 0 &= \nabla^2\boldsymbol{E} - \gamma^2\boldsymbol{E} \end{aligned} \qquad (eq.9)$$

This results in the Helmholtz PDE, noting that γ is related to the wave number, *k*, such that $\gamma^2 = -k^2$. The notation γ is used here to prevent confusion with the stiffness matrix, *k*, in the next steps.

To be used to solve discrete problems, eq. 9 still needs to be rewritten via the finite element method derivation. This process is, briefly:

First, rewrite Faraday's law such that $\nabla \times \boldsymbol{E} = -j\omega\mu\boldsymbol{H}$ becomes:

$$\frac{\partial \mathrm{U}(x)}{\partial \mathrm{x}} = -\mathrm{j}\omega\mu\frac{U(x)}{\eta} \qquad (eq.10)$$

Where U(x) is related to the integral of the magnetic field $\boldsymbol{H}$, and η is the wave impedance.

Following that, let the **y** and **z** directions be represented as:

$$\boldsymbol{E} = U(x)\boldsymbol{y} \qquad (eq.11)$$

$$\boldsymbol{H} = \frac{U(x)}{\eta}\mathbf{z} \qquad (eq.12)$$

The stiffness matrix, *k*, and the force (or load) vector, f, are defined as follows:

$$k_j i \coloneqq \int_0^1 \left(\frac{(dN_i)}{dx}\frac{(dN_j)}{dx} + N_i N_j\right) dx \qquad (eq.13)$$

$$f_j i \coloneqq \int_0^1 (xN_j) dx \qquad (eq.14)$$

Where N_i and N_j are the shape functions (basis functions) for the i-th and j-th elements. The stiffness matrix represents the system of linear questions that must be solved to approximate a solution to a differential equation. The load vector, *f*, represents the impact of external forces, source terms, or boundary conditions to the system. This vector becomes larger as more materials and geometries are added to the problem, and thus more boundaries are added. The inclusion of additional boundaries also increases the complexity of the problem, which emphasizes why PDEs and a numerical solution are important to electromagnetics design.

The weak form of the PDE, eq. 9, is then multiplied by a test function, $w(x)$, in eq. 15, and solved via integration by parts, and substitution, to isolate U(x), such that *k,* f, and U(x) are related as in eq. 16.

$$\int w(x)\big(U''(x) + k^2\mathrm{U}(\mathrm{x})\big)\mathrm{dx} = 0 \qquad (eq.15)$$

$$\boldsymbol{k}U(x) = \boldsymbol{f} \qquad (eq.16)$$

This derivation is extensively documented across various sources due to its significant utility in electromagnetics and multiphysics computations. The finite element method presented here becomes applicable for numerically solving discrete physical problems, which are traditionally formulated as partial differential equations, only at the form in eq. 16. By translating these problems into algebraic equations, this method enables more efficient computational resource utilization, which becomes important in designs with complex geometries, multiple materials, and with many boundary conditions. This is the basis of computational electromagnetics and simulation.

Computational electromagnetics (CEM) is a field that focuses on using numerical methods and computer simulations to solve approximations of Maxwell's equations to study electromagnetic phenomena in various systems and structures. That is, CEM is the digital modeling of the interaction between electromagnetic fields, physical objects, and the environment. CEM encompasses a wide range of applications, including antenna design, microwave and radio frequency (RF) circuits, medical imaging, material characterization, printed circuit board (PCB) design, and more. Numerical methods used in CEM are based on discretizing space and time, dividing the computational domain into small elements, and solving Maxwell's equations numerically at discrete points or intervals. In simulation, solving at smaller intervals increases the resolution of a solution. Common numerical techniques in CEM include finite difference methods (FDM), finite element methods (FEM), finite difference time domain (FDTD) method, finite element time domain (FETD) method, method of moments (MOM), and others [7, 18, 19], all of which have their own unique advantages. CEM's predictive, approximative modeling capabilities allow for the optimization of electromagnetic systems, facilitating the identification of optimal designs that meet specific performance criteria while minimizing costs and constraints.

Several prevalent electromagnetic simulation software suites are discussed in the following sections. These software suites utilize at least one numerical solution technique for simulating antenna, and other electromagnetic-based, designs. Due to the complexity of solving CEM problems, the rest of this document focuses on integrating with commercial software suites for simulation and numerical solutions rather than suggesting improvements to existing software

directly. A modular approach has been taken using a custom two-layered application programming interface (API) to automate the experimentation process for repeatable, accessible experimentations.

1.2 Existing Resources for EM Software Suite Integration

While web-based antenna calculators for common topologies are plentiful [20- 23], there are very few programs capable of integrating directly with EM simulation software. Regarding official software, AntennaMagus [13] is the primary example for automating project creation that can then be run with CST Studio Suite, which is owned by the same company. In 2022, Ansys released official Python libraries, including integration for HFSS [24], which has been under continuous development. Likewise, Altair Feko also has an official API [25-28] with other interface information documented. Several independent, and often unsupported or defunct, libraries and APIs have been created by individuals for integration with Ansys HFSS [29, 30]. While multiple studies have used MATLAB and HFSS integration with success [31-35], these solutions are often problem specific, do not modify easily, and require both a MATLAB and EM software license. A cost-efficient alternative is utilizing third-party APIs. However, a core issue with existing third-party APIs is that even when they are exceptionally well documented, they may require almost as much software scripting knowledge to use as writing the scripts for the EM software directly (i.e., IronPython for HFSS, and Lua for FEKO). More script resources have been created for COMSOL Multiphysics using their Java API [36, 37, 38, 39], but some existing APIs rely on a chain of libraries and other APIs to interface with the EM simulation software. Few resources offer native interfacing with an EM simulation software, and none offer interfacing options with multiple EM simulation software.

This integration complexity highlights the need for a streamlined and accessible interface. Streamlining the process for design, CAD, simulation, and analysis, and increasing the simplicity of the user-facing process not only facilitates the replication of experiments by providing a consistent and reproducible environment, but it also significantly lowers the barrier to entry for students and young researchers. The software suite proposed in Chapter 2 and the replication studies in Chapter 3 discuss this process. Automation is essential in CEM for ensuring reproducibility, minimizing errors, improving efficiency, and integrating simulations with the design process. By automating repetitive tasks and leveraging computational resources effectively,

researchers can focus on the design process. To further lower the barrier to entry at an institutional level, this document proposes a modular approach to interfacing with multiple EM software suites to increase accessibility. This removes software vendor as a limitation to cross-platform replication.

1.3 Optimization, Machine Learning, and Surrogate Modeling

Optimization is almost as significant to antenna design as simulation in terms of performance. The prevalent interest in design automation, simulation integration, and intelligent optimization has a basis in the existence of a broad range of analytical, semi-analytical, and non-analytical models, of which EM problems could be any. Some designs, such as rectangular patch antennas and basic monopoles, are widely explored and analytically defined. These, and similar topologies, have little variation in final physical design characteristics, and benefit much more from simulation than advanced optimization. Other topologies, such as horn antennas, are mostly or semi-analytically defined, where they can be mathematically approximated and then finalized in simulation. Other designs, especially those created with multiple materials, or complex geometries, are likely non-analytical in nature. The variation in design needs, provided the same or similar topologies, make it impossible to create a singular solution, or to apply a singular optimizer for all needs.

Current machine learning methods in microwave research trend towards variations of machine learning assisted optimization (MLAO) with the goal of blending individually controllable, but variable, problem constraints, machine learning model prediction, and analytical verification into streamlined methods for antenna design [10,40-46]. In practice, MLAO utilization is broad: incorporating both online and offline learning; combining global, local, and multi-objective optimization; implementing surrogate models jointly with optimizers to simplify the problem space; and exploring the differences between parallel and sequential optimization on surrogate (or reduced fidelity) models of given antenna designs. Understandably, EM simulations with complex geometries, multiple materials, and those that operate at high frequencies are difficult to address as the matrices needed to mesh these designs are large and time-consuming to solve. Low-complexity surrogate models, which have a lower computational cost, are popular in existing literature [40, 42] to address the issue of large mesh-geometries, but its widely recognized that this

comes at the cost of lower model fidelity. In some cases, reduced model fidelity is an acceptable trade-off for lower computational costs, even if the number of simulations is slightly increased, as the simulation time required is still lower due to not needing to calculate the large matrices of the original problem. In [46], low fidelity models have also been used to influence higher fidelity models that only use small amounts of high frequency (HF) data. However, due to the variety of features that can be used in antenna design, surrogate models do not necessarily transfer across designs. Irrelevant to the prediction confidence of the model, surrogate models (and surrogate model-optimizer pairs) are restricted to a specific state space defined by the model they are meant to represent. While all these methods are suitable for addressing certain groups of computational or topological problems, selecting a suitable optimization method for a specific EM problem, and then adjusting hyperparameters is an exercise in patience for your average researcher. Furthermore, even after effort has been expended to tune hyperparameters experimentally, not all models are a good fit for all problems and may still underperform or fail to converge.

Popular EM software suites may have optimizers integrated into the software suite, or as a companion software with additional features [11-14]. Integrated optimizers are typically a limited, set selection that rely on users to properly parameterize and define design boundaries, and offer no or limited feedback on potential design incompatibilities. In some instances, there is also no ability to recover progress from a simulation that fails to resolve or a software crash.

Following the concept of No Free Lunch [47], it is expected that no optimizer will be efficient on every problem, and not every problem can be solved by every optimizer. In the simplest case, an optimizer that performs well on several common topologies may perform unexpectedly poorly on a single topology or configuration, with no obvious explanation. In other cases, single objective optimizers may be applied to multi-objective functions and may fail to present a complete solution, or to resolve at all. Additionally, users might lack insight into how an optimizer behaves without substantial preexisting knowledge and be unable to match optimizers to a design problem.

Expanding the set of integrated optimizers in an EM simulation software suite is also not accessible to the average user or may not exist at all through the software or official APIs. These aspects become increasingly important when developing complex antenna designs that may take hundreds

of simulations, or simulations that take days to weeks to resolve. The need for efficiency can be somewhat addressed by properly tuning optimizers to the specific problem in cases where problem-optimal hyperparameters are known. Different optimization problems may require different settings for hyperparameters to achieve optimal performance [47]. Hyperparameter tuning allows the optimizer to adapt to the specific characteristics of the problem at hand, such as its dimensionality, complexity, and smoothness, improving the efficiency and effectiveness of the optimization process. Optimizers with well-tuned hyperparameters can achieve better convergence rates and solution quality, leading to more efficient optimization processes. This is particularly important in real-world applications where computational resources are limited, and improving the efficiency of the optimization process can have significant practical implications. Techniques to improve the process of both hyperparameter tuning and the application of optimizers on complex problems have included machine learning or machine learning assisted optimization and the usage of surrogate models [10, 40-42]. It is imperative to address the need for both single and multi-objective optimization for electromagnetics problems to cover optimization needs. Even as research moves towards more efficient and intelligent solutions, all methods are still bound by problem design requirements.

Adding to the problem difficulty, antenna design and optimization via CEM is a broad and constantly expanding field with many moving parts. As designs, materials, and environmental constraints become more complex, the ability to replicate and reproduce research becomes more important. These studies are crucial for validating and verifying scientific findings, ensuring the reliability and robustness of research outcomes, and expanding on innovative discoveries. By independently replicating previous studies, researchers can assess the consistency and reproducibility of results, identify errors or discrepancies, and confirm the generalizability of conclusions across different contexts. In a 2016 survey [48] across fields, respondents reported that potentially 70% of researchers had attempted and failed to reproduce research. It is proposed that the majority of factors behind this are benign (i.e., inherent variability in an experimental system, limited ability or inability to control complex variables, chance, lack of experimental rigor, etc.), but may also include pressure to publish. Introducing automation into the experimental process not only reduces the barrier to entry and decreases time spent on iterative design, allowing more researchers to explore complex practices, but it introduces some inherent standardization to

the experimental process that reduces human error by exclusion rather than introducing additional research guidelines to the process.

Increasing the availability of open-source resources for replication and experimentation can help in addressing these issues. Chapter 3 presents a series of internal, and expandable, replication studies based on a selection of literature that have been integrated into the software suite presented in Chapter 2. Many of these designs have also been used in optimization studies and are all compatible with the proposed software suite's internal optimizer library. Additionally, replication studies provide opportunities for education, training, and skill development among students and researchers. Ultimately, these studies play a vital role in building a solid foundation of evidence-based knowledge, driving scientific progress, and addressing challenges with confidence and accuracy.

There are eleven optimizers included in the software suite presented in Chapter 2. The core eleven optimizers include Particle Swarm Optimization, Cat Swarm Optimization, Chicken Swarm Optimization, a basic sweep optimizer, a Bayesian-based optimizer with a Gaussian Process (GP) kernel, and a MultiGLODS optimizer. Several variations of each optimizer are included, which are detailed in Chapter 4 of this document. Ten of the optimizers are also then compatible with the surrogate model library, expanding the potential optimization options. To accommodate a broad variety of EM problems, the optimizer function suite available at [49] was used to create a dataset to train machine learning models to suggest hyperparameters. The Hyperparameter Prediction Network model, including machine learning models and a controller, is discussed in Chapter 5, and the expansion discussed in Chapter 7.

CHAPTER 2
The AntennaCAT Software Suite

2.1 Software Specifications

The Antenna Calculation and Autotuning Tool (AntennaCAT) is an open-source software suite designed for antenna design, experimentation, and optimization. AntennaCAT [50, 51] provides a user-friendly interface with dynamic visualization of designs created using the internal calculator or replication modules. Built on Python 3.9, the AntennaCAT software suite uses no EM software specific APIs or libraries to interface with commercial CEM solver software. AntennaCAT must be run locally with the EM simulation software, and licenses for the EM simulation software are required. These licenses are not included in AntennaCAT and cannot be saved in AntennaCAT. AntennaCAT was developed using Ansys HFSS as that was the licensed software available for testing. Other EM software suites are compatible and are in the early stages of integration.

The Graphical User Interface GUI is created using wxPython (4.2.0), and all plots, 3D previews, and other visualizations are handled with Matplotlib (3.5.3). A custom graphics library has been built to work with Matplotlib to display dynamic antenna design previews that can be manipulated by users in 3D space. The internal design library uses these graphics to display parameter manipulation in real time.

Functions for .DXF generation and export utilize ezdxf (1.0.3), and the Gerber files generated by the internal calculator use pcb-tools-extension (0.9.3). Templates generated by AntennaCAT for various EM software suites are written in the required language for the EM software and are run by their respective software. For example, while AntennaCAT is written in Python, the template library for Ansys HFSS is written in IronPython, but stored as .txt files. This method streamlines a process where templates can be loaded, filled in, combined, and saved as the corresponding file type necessary to run with the specified EM software suite. This approach facilitates the integration of new features, new EM software suites, and to accommodate updates without affecting other existing program aspects.

The internal optimizers use numpy (>1.24.3) for calculations and data manipulation. When necessary, pandas (>2.0.1) is used to export debug logs as .csv files. The quantum inspired optimizers do not use quantum computing inspired libraries or software development kits (SDK)

such as Qiskit. This is intended to mimic early literature approaches and is discussed in later sections.

The most recent versions of AntennaCAT are available as open-source software from [52]. This repository also contains progress for EM simulation software integration. In cases where a specific EM simulation software is not compliant with Virginia state law, collaboration efforts have been pursued to further development.

2.2 Template-Based Multi-Software Integration

AntennaCAT is EM simulation software agnostic, providing compatibility with multiple platforms rather than a unified API. Rather than creating a command dictionary for individual EM software suites and attempting to continuously match development updates, AntennaCAT employs a modular, template-based for integration ensuring consistency across all supported EM software. This process is consistent across all integrated EM software suites. Figure 1, below, shows the general groupings of the internal template library structure. The exact process for creating scripts from templates and executing them is discussed in the Simulation Object section (2.4, Simulation Object and Simulation Integrator instances) of this chapter. Each integrated EM software has its own template library. This library is comprised of a dictionary linking to a series of .txt files containing functionally complete actions split into elements that have been reduced to the fewest steps possible. The actions (e.g., open, save, close, delete) are written in the scripting language of the corresponding EM simulation software.

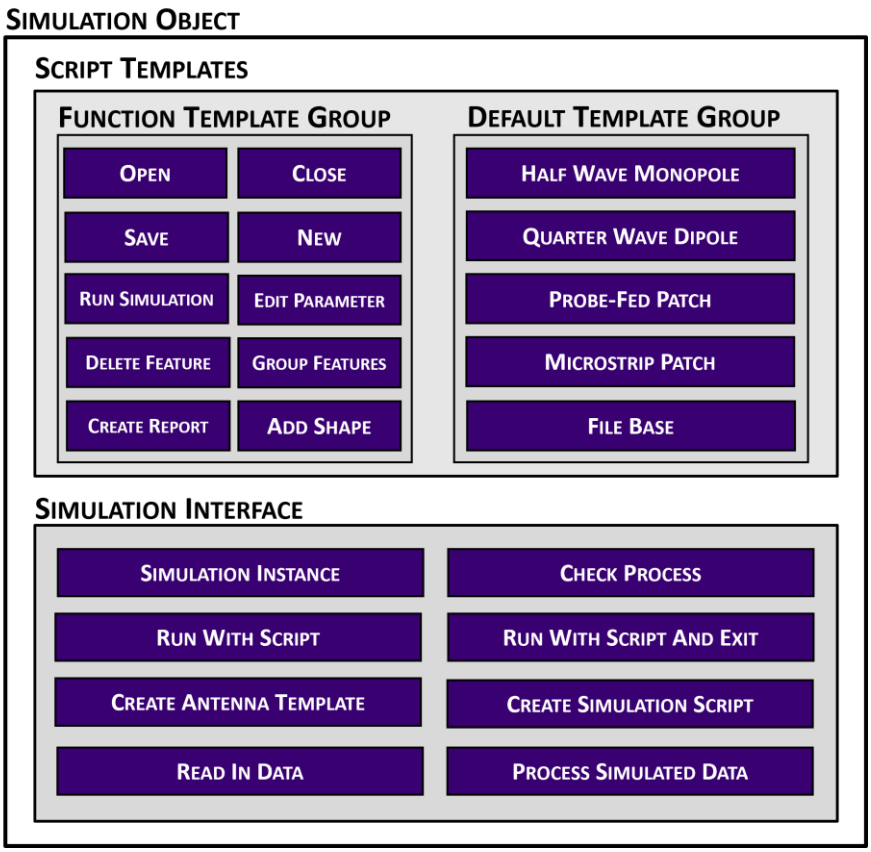


Figure 1 *Example groupings of .txt templates for EM simulation software integration in AntennaCAT's Simulation Object.*

This template-based approach takes advantage of parameterization used by EM software. This parameterization is used for CAD dimensions (i.e., length, width, height, gaps, etc.), materials, (i.e., copper, vacuum, FR4, etc.), and other mutable variables in simulation. Advantages for parameterizing designs include ease of adjusting modeling during the design process, and for replication. For templating, in terms of the replication studies and internal AntennaCAT scripts, it allows for limitless variations of the existing designs in terms of variable and material manipulation. This encourages students to experiment with these designs in a repeatable and consistent environment.

The template groupings in Figure 1 are used for example and are not exhaustive. The Default Template Group shows a sample of scripts for the default calculation and replication design. These are complete design templates that have place holders for parameter values. When the Antenna Generator tab is used to customize a selected design, the parameters from the GUI are written into memory and used to fill in the template when a script is generated. Each script in this collection is

fully self-contained and can be run in its respective EM simulation software suite when the placeholder values have been replaced. Other scripts represented by the groups in Figure 1 are not complete, executable scripts. However, they are complete actions that can be chained together by AntennaCAT into executable scripts.

Figure 2 *Examples of the AntennaCAT template library, with parameterized variables. a.) a script example declaring parameterized values, b.) a script example creating a rectangle with parameterized values, and c.) an example of a non-customized script template with placeholder "INSERT_" values.*

Figure 2 shows three examples of parameterization needed to generate scripts via AntennaCAT. In Figure 2.a and 2.c two samples have been pulled from a genetic-algorithm script to show the declaration, 2.a, of variables $seed_width and $seed_length, and their usage in the creation of a rectangle at the origin of a CAD model in Ansys HFSS, Figure 2.c. Figure 2.b. shows an example of a raw template where the placeholder "INSERT_" variables have not been replaced. These "INSERT_" variables are used as placeholders in all AntennaCAT templates. When a design has been selected, the appropriate script is loaded into memory, and regular expressions (RE) are used to replace these text strings with their user-selected, or program generated, values. This is done

using an EM software compatible Simulation Integrator, which is discussed in the Simulation Object section.

2.3 User Interface

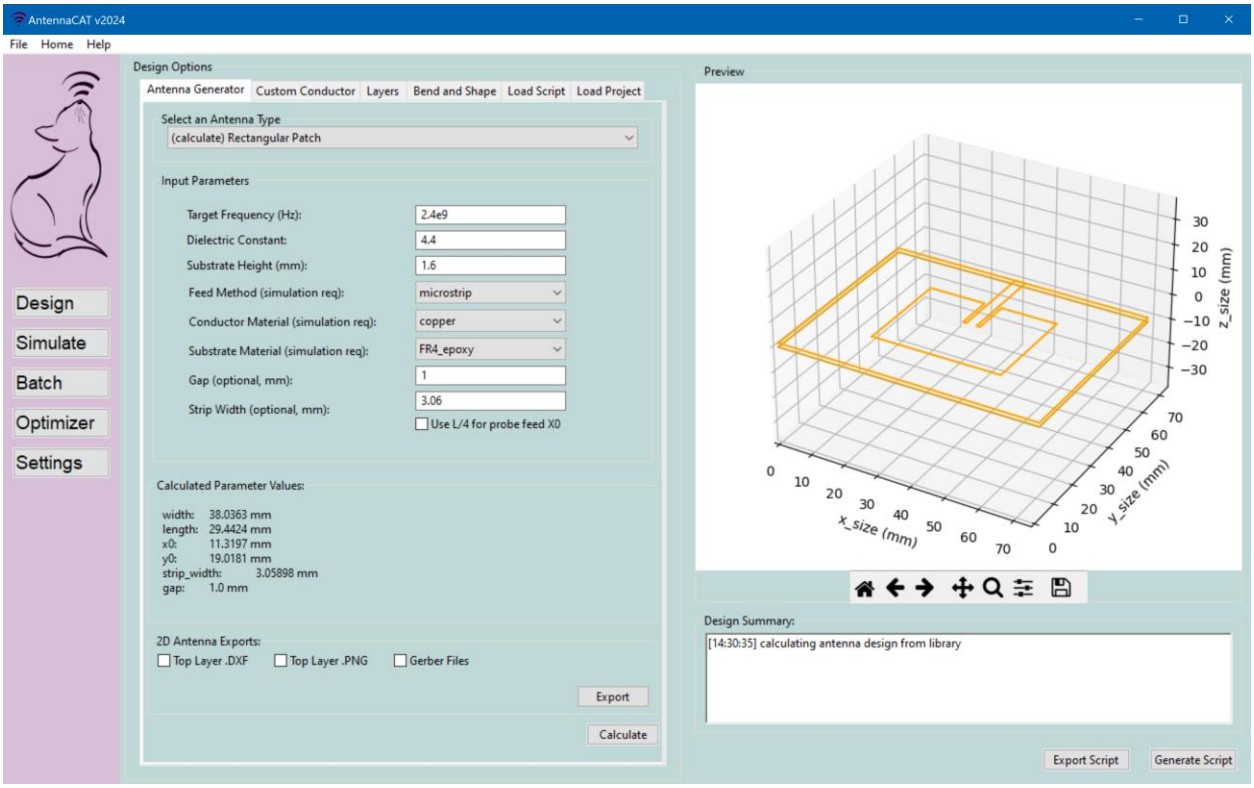


Figure 3 *The main Design page after creating a new project or loading an existing project. The Antenna Generator process has been used to create a rectangular patch antenna with the internal calculator.*

Figure 3 displays the main Design screen after creating a new project or loading an existing .ANCAT project. The GUI, built with wxPython widgets, adapts to the operating system's theme, leading to potential appearance variations on different platforms. Creating a new project and selecting a specific EM software is covered later in this chapter.

The GUI is designed to streamline the setup for design and simulation for experimentation through a top-to-bottom workflow that guides users in providing necessary information. In Figure 4, below, a generalized layout for the Design page is presented. The button menu on the left is consistent across the Design, Simulation, Batch, Optimizer, and Settings options. These can be used at any time to change the GUI's page without losing progress. The menu bar at the top uses 'File' for traditional file saving and opening options, 'Home' to return to the project creation page, and

‘Help’ for providing additional details. The following subsections summarize the available options in the AntennaCAT GUI.

The Design Options section covers several ways that antenna can be created via calculation, replication, and several advanced options including adding layers, bending, and importing. Imports can include custom conductor .DXF files, existing scripts, and existing projects for EM simulation software suites. The Simulation Options subsection briefly covers the simulation options available for Ansys HFSS. However, there are direct equivalents for other AntennaCAT compatible EM simulation software. In the Batch Options subsection, using AntennaCAT for batch data collection is covered. Settings and creating a new project are shown in the Settings and ANCAT Files section.

The Optimizer page is briefly covered in terms of GUI functionality in Optimizer Options. The optimizers and their use are discussed in the Optimizer chapter.

Design Options

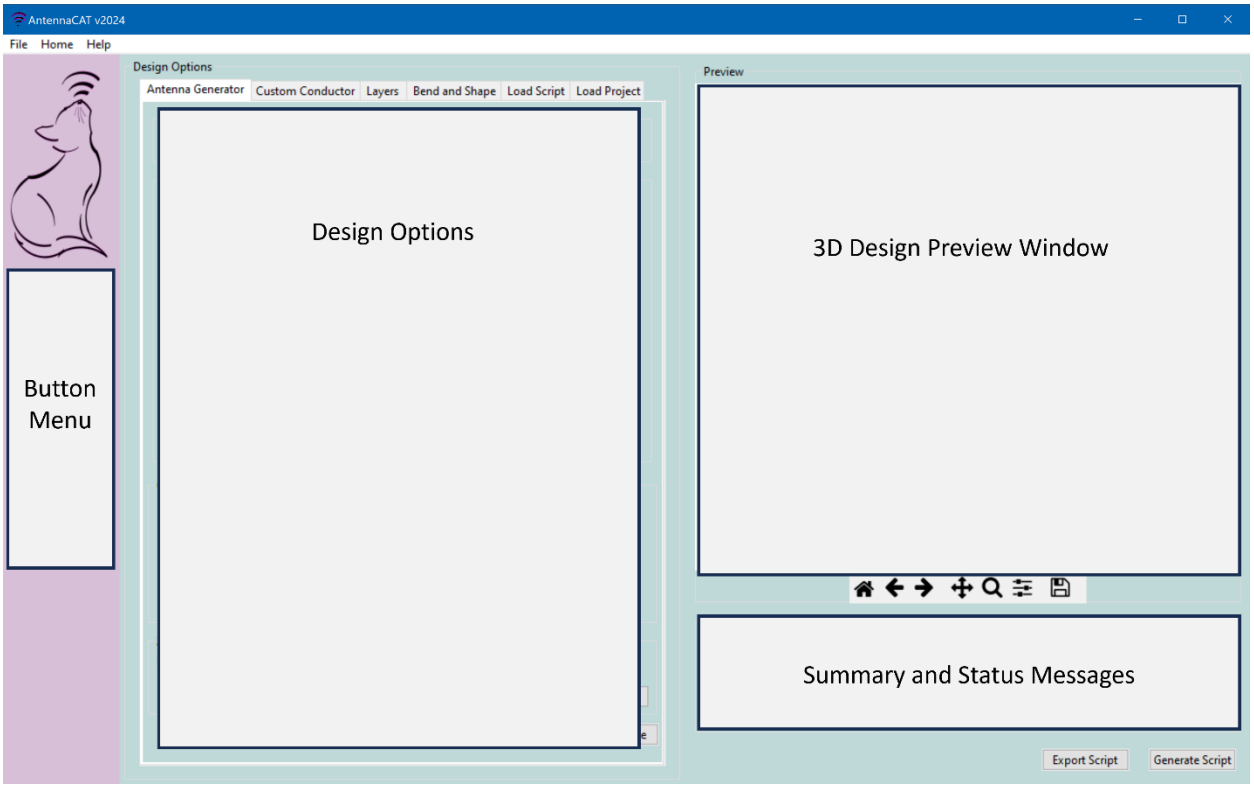


Figure 4 *The generalized layout of AntennaCAT’s Design page for using a calculated topology in the Antenna Generator.*

The Design Options field has several tabs for generating antenna designs, creating custom conductors, creating layered designs, bending layers, and loading scripts or importing existing EM simulation software project paths. The 3D design preview window and the summary and message fields are always displayed on the Design page. The design preview window is updated as design choices are confirmed, while status messages are displayed with calculation, generation, script output, and other actions.

The *Antenna Generator* tab on the Design page has two layout formats; one featured when selecting topologies that use the internal antenna calculator [53, 54], and one featured when using a replication study topology. This distinction is shown in Figure 5, with the window on the left showing a calculated rectangular patch, and the window on the right showing a replication study version of the rectangular patch. On the left image of Figure 5, users provide material and dielectric constant values that are then used to calculate the physical parameter values. On the right, these physical parameter values are set by the user.

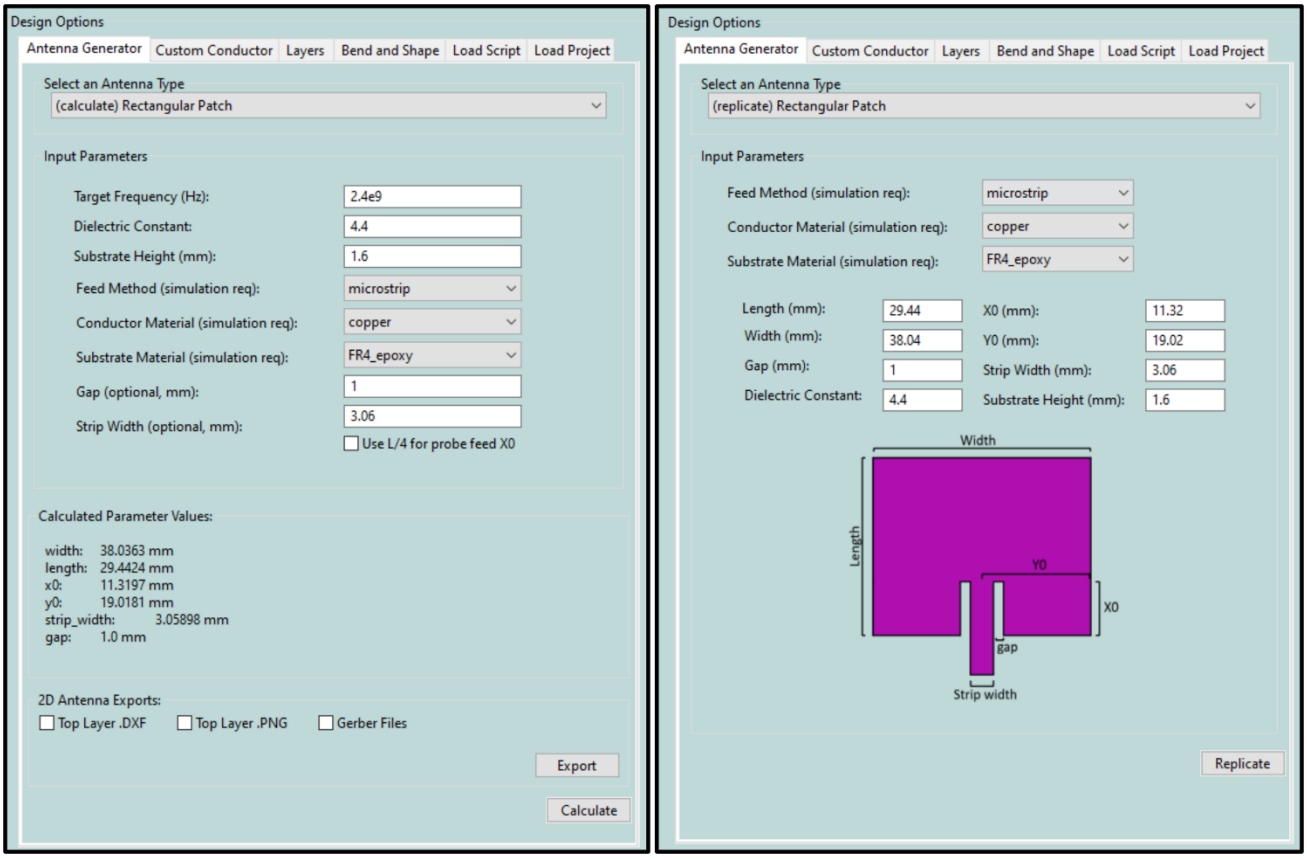


Figure 5 *A side-by-side comparison for creating planar rectangular patch antennas. Left, using the internal antenna calculator. Right, the replication study topology.*

The Antenna Generator tab in Figure 5 has three primary parts, starting at the top with the design configurations where a series of topologies with their own unique parameter sets can be selected. When using the AntennaCAT GUI for design, users control factors such as the desired target resonance, substrate dielectric constant and height, the feed method (microstrip or probe, when selection available), and conductor and substrate materials selected from an internal materials library. If a topology using the internal antenna calculator is selected, then fields for the calculator results and export options are displayed. The calculation and export options are handled by the AntennaCalculator [54], which calculates values for three topologies: a rectangular patch antenna, a half-wave monopole, and a quarter-wave dipole. Working with the user provided information, the open-source AntennaCalculator returns parameter values that are then used in template-based CAD creation. If replication study topologies are selected, images with the corresponding parameter locations are displayed. In both cases, material properties and other physical characteristics of these designs are user controllable.

The second tab in Design Options is the *Custom Conductor*, which allows users to import a .DXF file that can be converted to a custom conductor for a planar antenna. Currently, support for importing .DXF files for non-planar antennas does not exist but may be an expansion in future work. Figure 6, below, shows two example imports. On the left is a design created using the Antenna Generator, exported as a .DXF file, and then re-imported. This design has several polylines that appear in the 2D preview window at the top, each of which can be assigned in ‘Assign Shapes’ at the bottom of the left side. In cases such as the one in Figure 6 where a microstrip feed has been imported, the manual assignment of port location happens in the preview window. On the right side, the AntennaCAT logo has been imported as a .DXF as a single object. Due to the difference in .DXF file structure, the AntennaCAT logo only has one identifiable layer, and individual polylines are not identified. This difference is caused by the creation of the .DXF, and how ezdxf processes polylines. In the case of the AntennaCAT logo, due to the number of line segments (>4000), the .DXF was imported as a single object to reduce issues with scaling and processing.

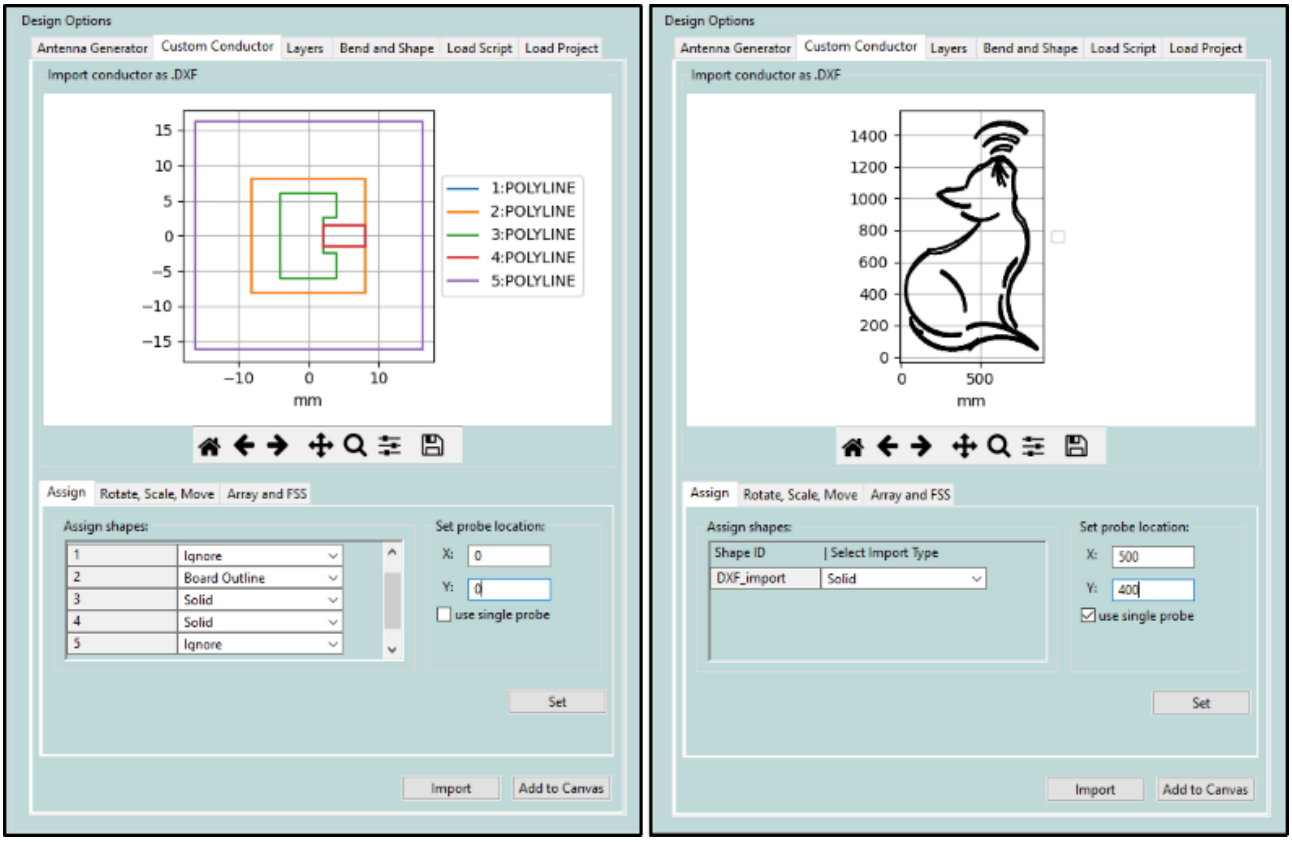


Figure 6 *The Custom Conductor tab with two import examples in Design Options. Left, a multi-polyline import of a microstrip-fed rectangular patch. Right, the AntennaCAT logo imported as a single layer.*

Material assignment for imports is not handled in the Custom Conductor tab. If no material is assigned, copper will be used for any solids as a default. Under the *Layers* tab, shown in Figure 7, the imported conductor can be used as a template for the conductor shape. The top input box for Conductor Layers can then be used to either leave the conductor as a singular layer, or to dynamically create multiple layers.

Similarly, substrate (below the conductor layer) and superstrate (above the conductor layer) options can be used to create multi-layer materials. If no conductor is selected, the substrate and superstrate boxes can be used to create designs with multi-layered materials. By default, these are rectangular boxes with length of 50 mm and width of 60 mm, with the length and width individually parameterized. To configure these layers, the *Bend and Shape* tab can be used to select an angle of deflection. This angle of deflection is calculated based on equations in [55], and can be used to create convex or concave shapes.

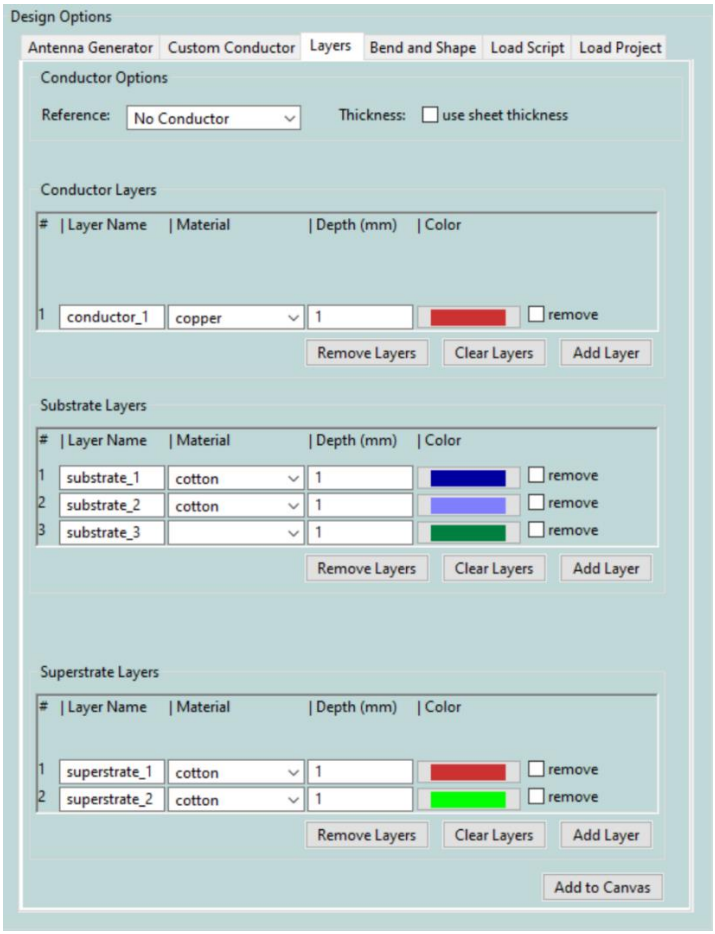


Figure 7 *The Design Option Layers tab showing the custom conductor material selection, substrate layer options, and superstrate layer options.*

Due to the complex nature of dynamically creating antennas on layered substrates, the conductor layer is not adjusted to retain a specific frequency when applied to a new substrate combination. If a calculated topology is used with the Layers tab, the calculated values become static, and recalculation must occur to change conductor size. This does not affect the thickness of the size of the layers.

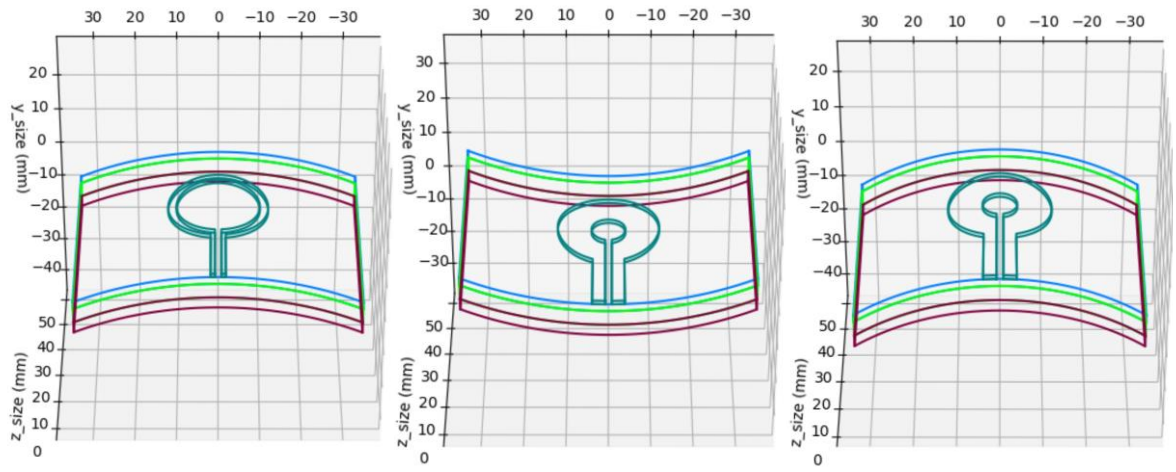


Figure 8 *Examples of a planar loop antenna on multi-layered substrates with positive and negative angles of deflection.*

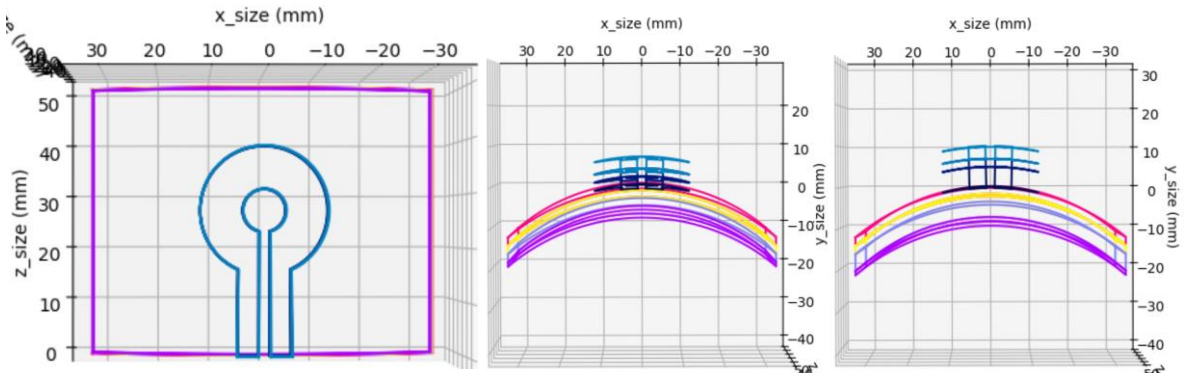


Figure 9 *Examples of a planar loop antenna on multi-layered substrates, with multi-layered conductors to show consistent angle of deflection on all layers.*

Figures 8 and 9 show examples of planar loop antennas made with a custom replication study option. These figures demonstrate the bend functionality (a dynamic angle of deflection) with multi-layered substrates. Figure 8 uses a single layer conductor and shows positive and negative angles of deflection. Figure 9 shows exaggerated thicknesses for the multiple layered loop antenna to demonstrate the consistency of the angle of deflection. In the custom Matplotlib graphics library, the arc length and thickness of each layer are preserved. Actual implementation of the bend depends on the EM simulation software suite, and if the preferred method is to use equation defined planes or to project 2D designs onto a curved surface. This difference affects if a conductor, or uniquely shaped substrate, can be drawn directly with polylines or if it must be projected.

Importing Existing EM Projects and Scripts

Importing projects and scripts exists as part of the Design Options tab collection on the Design page. Using this functionality, if a project has been created in a compatible EM simulation software suite, it can be loaded into AntennaCAT as a file path. Loaded projects are not parsed for parameters or other project information. For them to be usable in AntennaCAT, several conditions must be met. First, there must be at least one EM simulation software license related to the imported project on the computer running AntennaCAT. Secondly, the project must be parameterized as AntennaCAT cannot create parameters in the CAD model for an existing design. Finally, users must manually enter parameters by name, exactly as they appear, in the EM simulation software project. An initial value for the parameter must be entered into the Load Project tab, as shown in Figure 10.

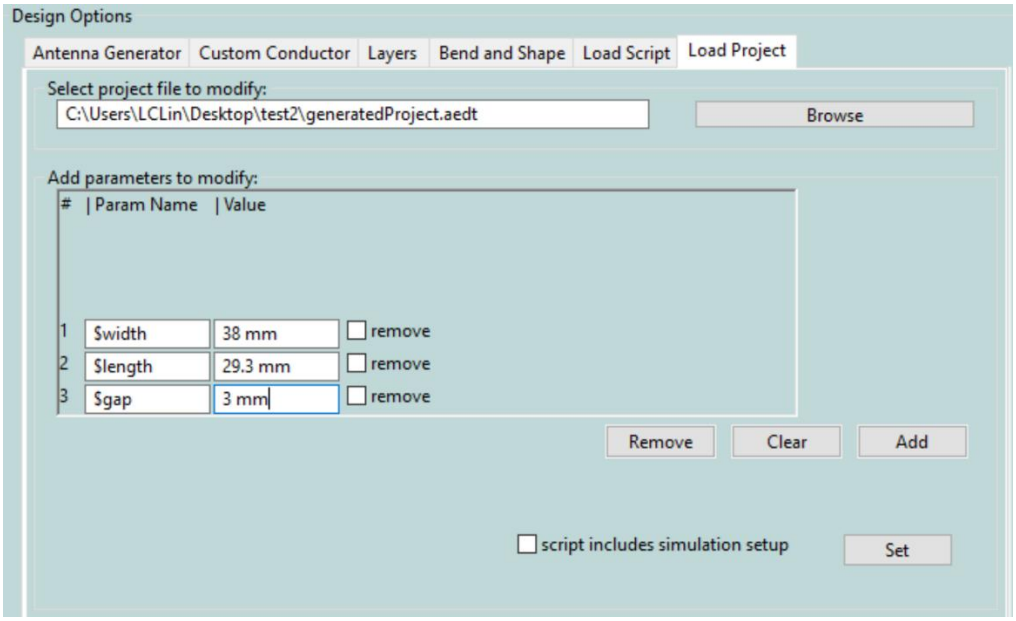


Figure 10 *The Design Options window showing the Load Project tab where an existing project path has been added into AntennaCAT and parameters have been added manually.*

Initial parameter values need not be exact and will default to millimeters if units are not specified. These values will be used in the Batch and Optimizer processes for creating ranges for parameter manipulation. If using AntennaCAT for simulation setup and exporting results, these values will not be overwritten in the project.

Parameters do not need to be manually entered into AntennaCAT if importing an existing script, which is the preferred method for incorporating existing projects. This can be done in the Load

Script tab. Figure 11 shows an Ansys HFSS script selected and imported from an existing AntennaCAT project. As Ansys HFSS is one of the compatible EM simulation software suites, AntennaCAT's file parser can extract existing parameterized values. Proper parameter naming and adherence to the software's conventions are necessary for accurate parsing. However, errors can be corrected in this window by selecting the name or value and typing the correction.

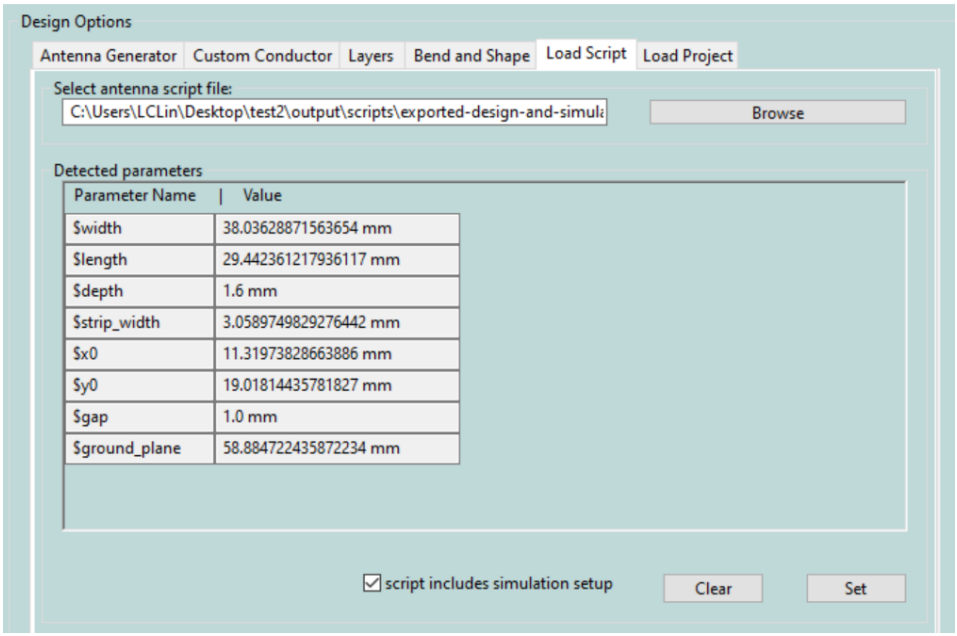


Figure 11 *The Design Options window showing the Load Script tab where an existing Ansys HFSS script has been loaded into AntennaCAT, with parameters automatically detected.*

Simulation Options

The Simulate page in AntennaCAT remains consistent for all compatible EM simulation software suites. On the left side of the window in Figure 12, there are tabs for simulation and solution setup (top), and report selection (bottom). Half of the page is designated for simulation configuration summary messages and status updates to properly display configurations for simulation settings. Figure 13 shows simulation options for Ansys HFSS. The differentiation in EM software suite is made to highlight that report settings, and output options, may differ slightly between software suites due to naming conventions or terminal/modal compatibility.

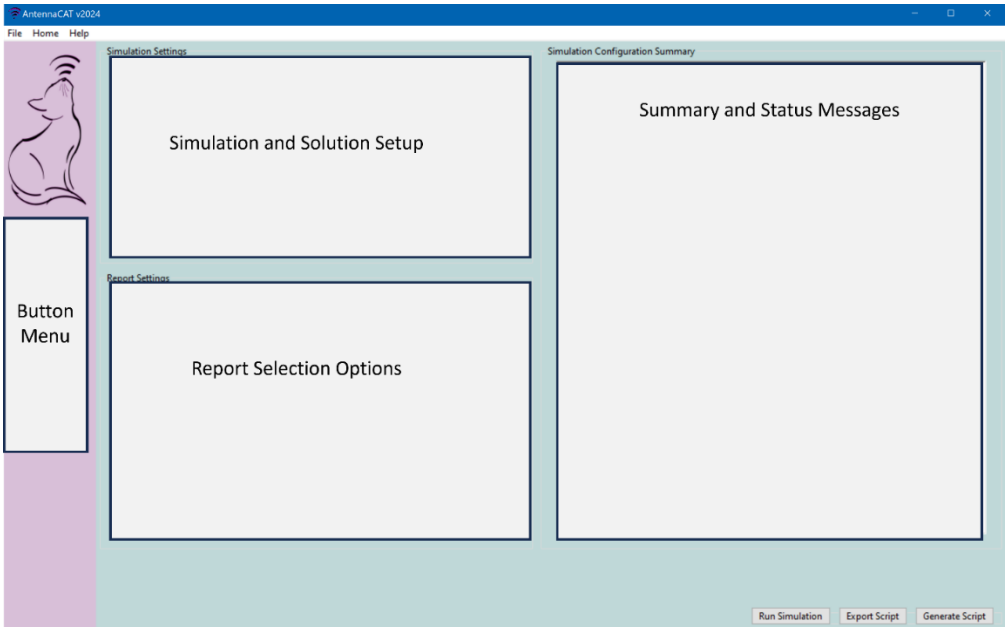


Figure 12 *The Simulate page generalized layout, with fields for report selection, simulation and solution setup, and output messages.*

In the Solution Setup tab, top left on the Simulate page, there is the option to use multiple frequencies in simulation, including multiple deltas should there be a need to change resolution. If the multiple frequency checkbox is marked, but one or fewer frequencies are entered, the user will be prompted to enter frequencies.

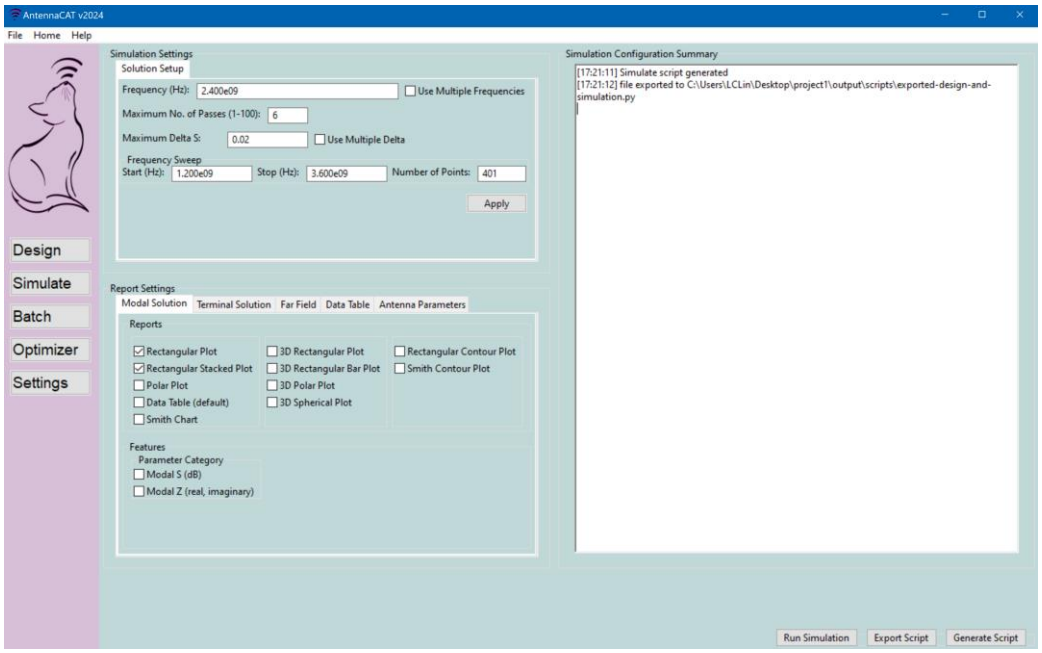


Figure 13 *The Simulate page for Ansys HFSS simulation and solution options. A script for a 2.4 GHz patch antenna has been generated, and reports for Rectangular Plot and Rectangular Stacked plot will be created after the simulation.*

Report Settings for Ansys HFSS include tabs for Modal Solutions, Terminal Solutions, Far Field reports, Data Tables, and Antenna Parameter outputs. When possible, there are options to export S, Z, or both parameters. All reports featured in the Report Settings tabs are native to their respective EM simulation software and may differ slightly in name between software suites. If a simulation software has been selected, the design and simulation created in AntennaCAT can be run directly from this window using the Run Simulation button, or a script can be saved to be manually run using the Export Script button.

Batch Options

Unlike the Design and Simulate pages, the Batch page does not change fields based on selected projects or designs. This page is designed to help users quickly select parameters and ranges to sweep in iterating simulations for data collection purposes. The simulation and parameter changes with this process are fully automated via the Tuning process. Reports for return loss, gain, and directivity are exported with numbers corresponding to the parameter combination for later references.

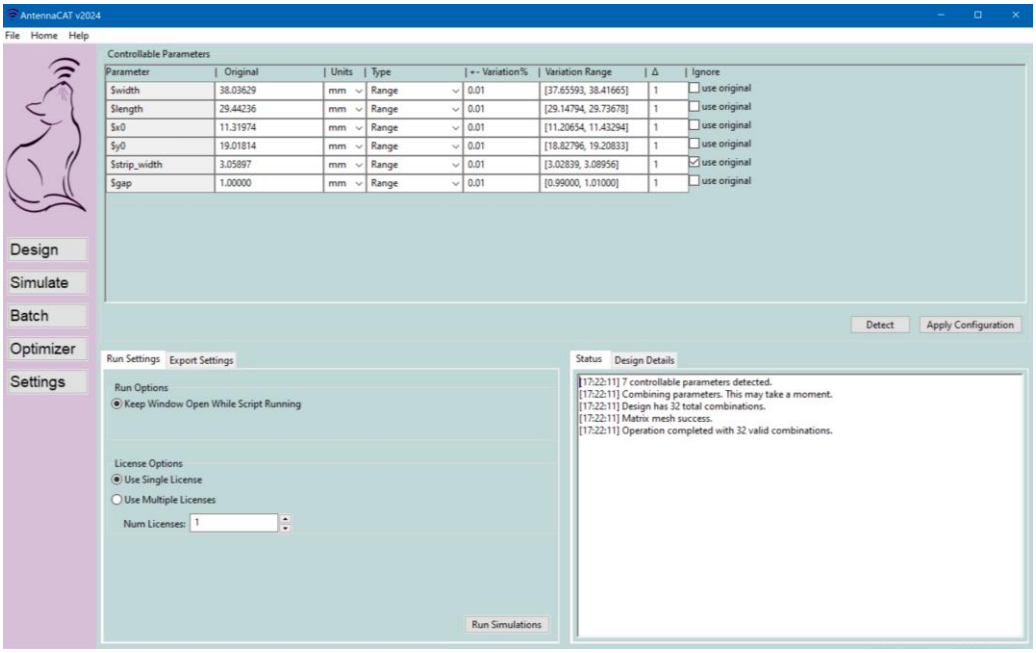


Figure 14 *The Batch page featuring detected Controllable Parameters field with values from a microstrip-fed rectangular patch antenna.*

On the Batch page, parameters from imported scripts, manually entered from loaded projects, calculated designs, or replication study designs can be selected from memory and parsed into the Controllable Parameters field at the top of the page. This process is automated using the Detect button. Parameters will be split into columns for the name of the parameter, the original value, units, type of value (range, constant, material, parameter), a variation percentage, a range, and a delta for incrementing values within the range bounds. On the far right is a checkbox for ‘use original’ to make a parameter static.

When using the variation percentage or range, there are several options. Entering a percentage will set the boundary above and below the original value. However, the boundary can be set manually by entering values directly into the Variation Range field. Changing the delta will change the step size sweeping through the variation range. By default, the delta value is set to 1.

The Run Settings tab on the lower left controls how many simulation licenses can be used during this process. AntennaCAT natively manages parallel simulation coordination, and can constrain its own license usage, but has no method to track the actual licenses assigned to a machine or allowed per user. Export settings allow users to select if parameter variation occurs in the same simulation file, or if a new simulation project is created each time. Currently, there is no option to select which reports are generated and exported, but that feature is in development.

Optimizer Options

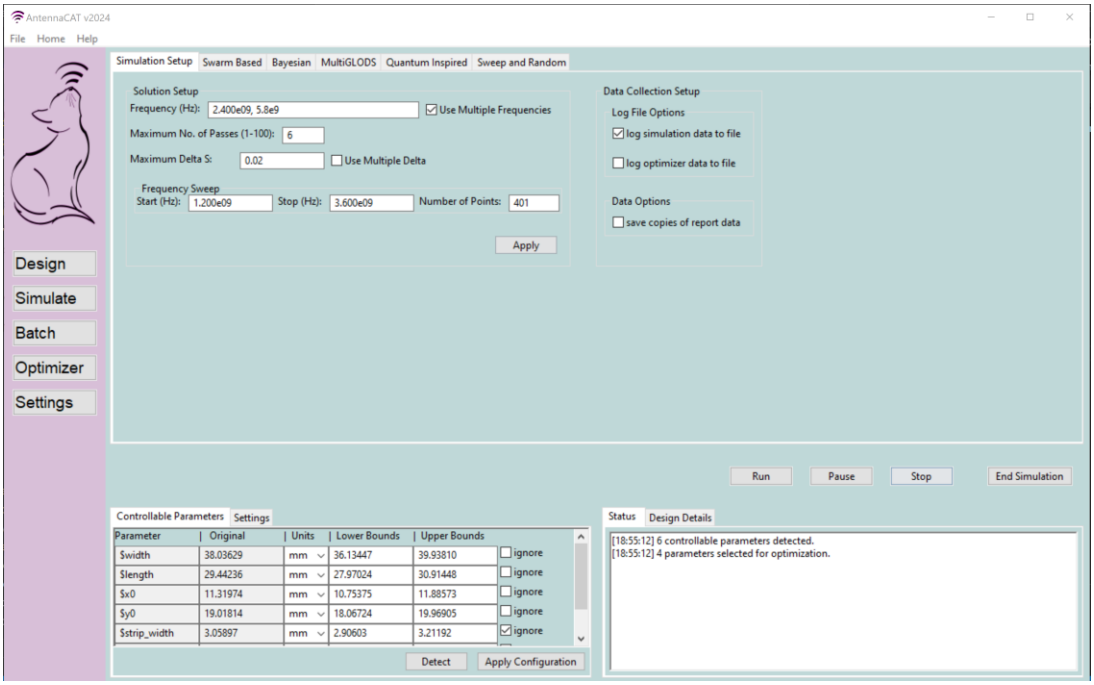


Figure 15 *The Optimizer page featuring the initial simulation setup tab, controllable parameters window, and status message window.*

AntennaCAT includes an internal optimizer suite. Features of the optimizer suite are discussed in Chapter 4 of this document, including details of specific optimizers and the hyperparameter tuning. The GUI includes simulation setup on this page that imports simulation options from the Simulate page if previously set. However, if a project was imported the process is streamlined by including the basic simulation setup as a tab on this page. There are no options for generating reports on the optimizer page; based on selected target values (i.e., return loss, gain, etc.) on the specific optimizer tab AntennaCAT will log data in a .csv compatible file for reference, but exports specific reports to simplify parsing.

Similar to the Batch page, the Controllable Parameters of the Optimizer page are parsed from memory into parameter name, original value, units, upper bounds, and lower bounds. There is less emphasis on tuning parameter bounds for the optimizers. These bounds are absolute limits and are used to control internal features such as random number generation, step size, decision space sweeping, and others. When applicable, resolution options for an optimizer replaces the Batch page's delta. Otherwise, the optimizers control how samples are taken in the feasible decision space.

Figure 16 shows an example of the swarm-based selection prior to optimizer selection. On the far left, the Parameter Summary field indicates the total number of detected controllable input parameters, the selected parameters that will be mutable for the optimizer, and the lower and upper bounds for the parameters. The center field shows the selectable Optimizer Targets, which include S_{11}, gain, bandwidth (BW), directivity, and efficiency. These five metrics were selected as they are commonly used for performance evaluation. Experimentally, S_{11} and gain were the most relevant to achieve desired performance results. However, the other metrics are extremely valuable for evaluation and are often desired beam characteristics themselves. The Parameter Summary and Optimizer Targets are consistent across all optimizers. The Optimizer Targets list may be expanded in the future, but currently focuses on several practical features for optimization.

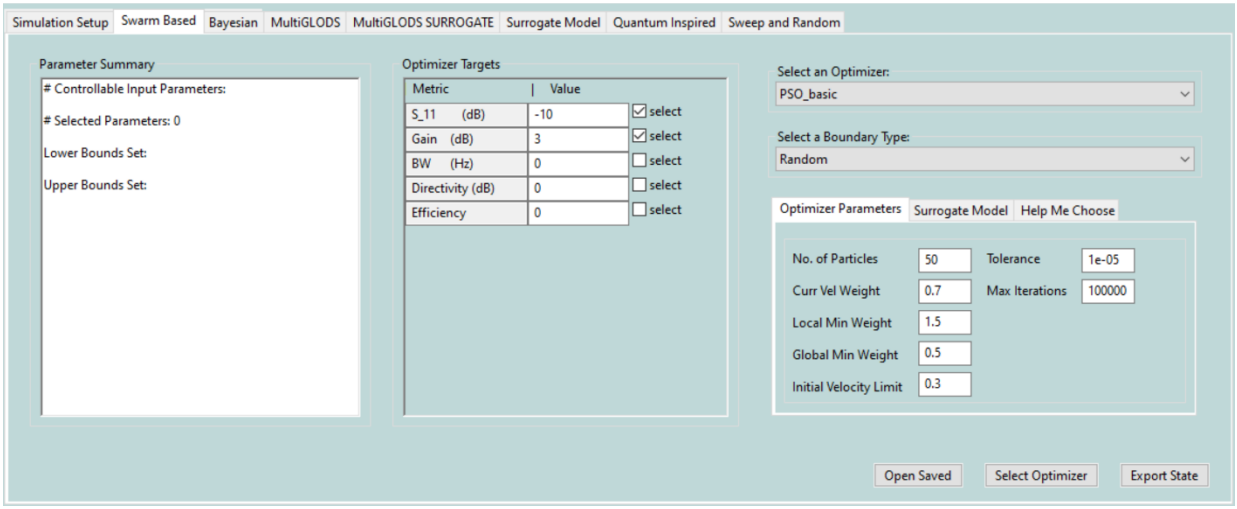


Figure 16 *The default values for the Swarm Based optimizer tab on the Optimizer page. Two optimizer targets, S_{11} and gain, have been selected as target values for PSO.*

In all optimizer tabs, the fields on the right of the GUI are used to select optimizer parameters (hyperparameters) for the chosen optimizer. Some tabs, such as the Swarm Based tab, have multiple optimizer options, while others such as the MultiGLODS and Bayesian tabs have singular optimizers. When applicable, options for compatible surrogate models are selectable.

The 'Help Me Choose' tab is discussed in Chapter 4 as part of the Hyperparameter Prediction Network model interface, with the results of utilizing machine learning on the data set collected using the objective function library. This tab is used to suggest hyperparameters based on the chosen optimizer, the number of input controllable parameters, and the number of targets being optimized for.

Settings and ANCAT Files

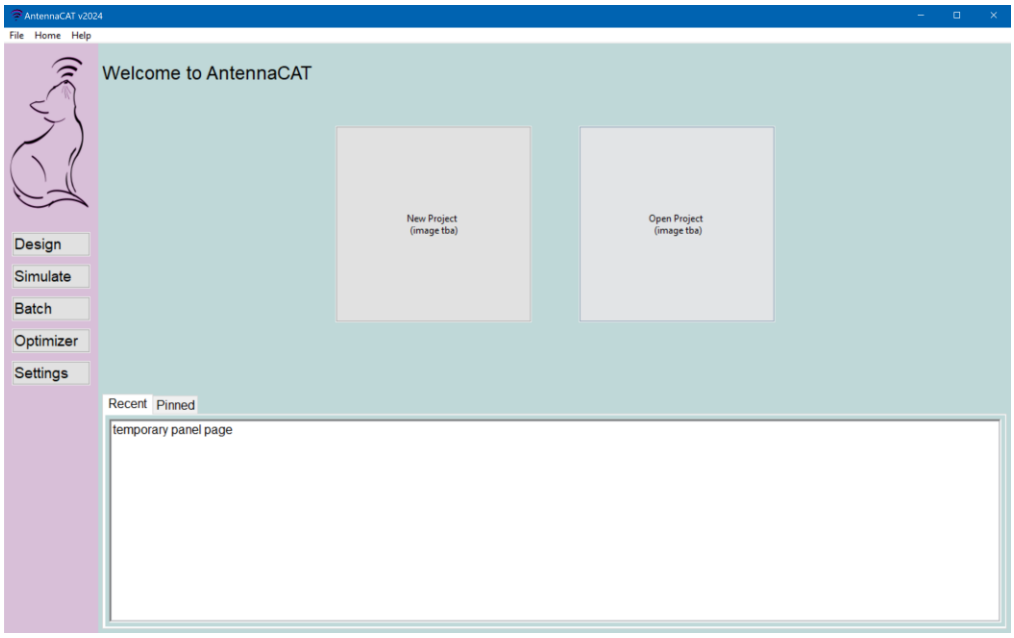


Figure 17 *The AntennaCAT home screen for project selection. This page includes new project creation and open project options, and a set of tabs for opening recent or pinned projects.*

When an instance of AntennaCAT is started for the first time, the home page in Figure 17 is shown. A new .ANCAT project can be created with the New Project button, or an existing project can be opened with Open Project. If a new project is created, the Settings page in Figure 18 will be automatically opened to configure the EM simulation software suite selection.

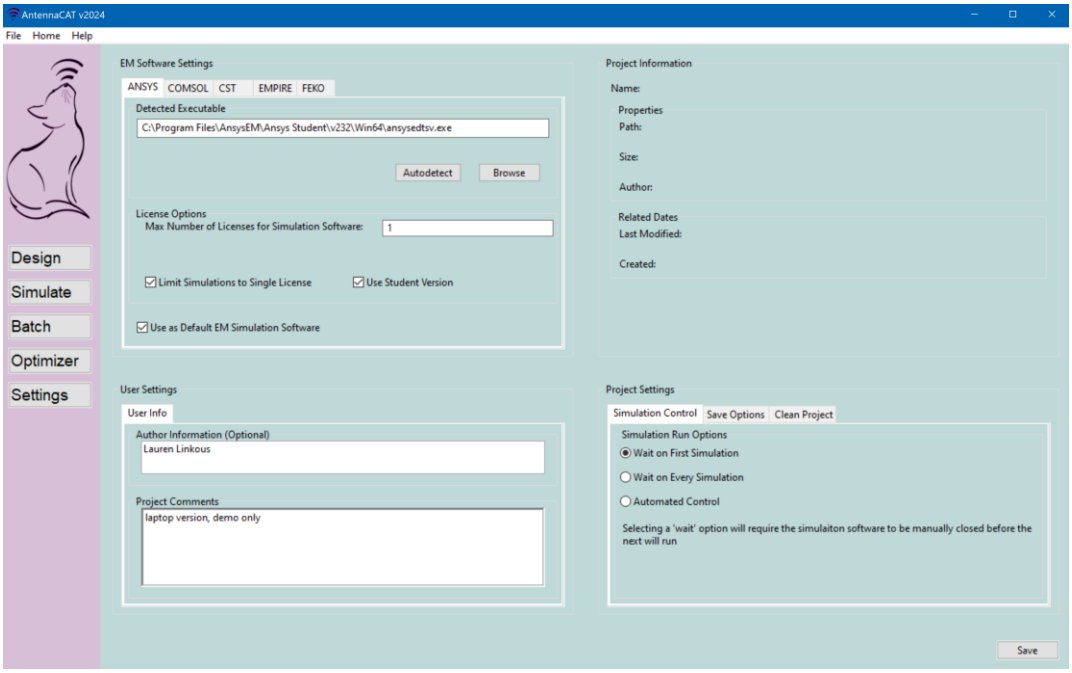


Figure 18 *The Settings page for AntennaCAT featuring the EM software selection and user information fields.*

The Settings page (Figure 18) features several compatible EM software suites for selection. For AntennaCAT to properly control the simulation process, the software executable must be selected and the path saved. The executable can be located manually using the Browse button, or AntennaCAT can search for the executable. It is recommended to use the Autodetect button as some EM software suites have multiple executables in their directories. Here, the value of the maximum number of licenses that AntennaCAT can use can be set, which will take precedence over other settings for Batch or Optimizer functions. The maximum number of licenses set must be less than or equal to the maximum number of licenses available to the local machine running AntennaCAT and the EM simulation software. AntennaCAT

Project settings for default simulation behavior, save options, and options to clean a project are available on the lower right of the window. On this page it is possible to give AntennaCAT fully automated control over all simulations, or to wait for a manual closing of the simulation window. If ‘Wait on First Simulation’ is selected, AntennaCAT will wait until a user has closed the simulation window before continuing with automated simulation control. This allows the user to inspect the design and simulation results. When ‘Wait on Every Simulation’ is selected, AntennaCAT will expect the user (or a separate process) to close the simulation software before the next simulation is started. It is not possible to edit this preference while a simulation process

is running. The Clean Project tab has options to remove scripts, projects, or other data from memory if there are recurring errors.

2.4 Modular Scripting and Automation Process

AntennaCAT dynamically generates scripts, creates CAD files, and simulates antenna topologies or user-uploaded models. The presented software comprises three main components: the GUI, the program kernel controlling optimization and data management, and the Simulation Object (SO) which interfaces with commercial EM software. The modular, template-based SO increases the accessibility for custom antenna creation, automating the optimization process from feature selection and calculation to CAD generation, simulation, and parameter tuning. In addition to creating antennas based on user-set parameters, within the program operation is the batch project simulation for data collection and automated design tuning via optimizers popular in literature and machine learning techniques. Genetic algorithm implementation has been discussed in previous AntennaCAT publications [50, 51] and is available in the derivative GeneticCAT [56], which has limited features and is not part of the core AntennaCAT software suite.

AntennaCAT and all derivatives work on the same modular, template based SO approach to create a unified system where one SO class is used to interface directly with each EM software. The SO for Ansys HFSS generates scripts written in IronPython, the Altair Feko SO uses Lua, the COMSOL Multiphysics SO uses Java, and finally the SO for Dassault Systemes CST Studio Suite generates and implements VBA Macros. AntennaCAT executes these scripts to first create the initial CAD model, and then run simulations to collect baseline data on the topology. When utilizing the tuning, batch scripting, or data collection features, AntennaCAT generates new scripts to edit the CAD models and rerun the simulation.

Simulation Object and Simulation Integrator Instances

AntennaCAT's modular design allows for easy adaptation to new EM simulation software, replication studies, and updates to existing integrated EM suites. This adaptability is achieved through a series of Simulation Objects and Simulation Integrator instances. Similarly, the

Optimizer Objects and Optimizer Integrators follow a comparable structure but are not necessary for the base functionality of AntennaCAT.

AntennaCAT utilizes template-based Simulation Object (SO) classes to create a unified interface for well recognized commercially available EM simulation software. To facilitate feature expansion, all SOs follow a design template that allows for modular class implementation. This structure enables consistent function calls for each object, but creating the Python object at run time based on user input. That is, all SO objects are interchangeable as far as the AntennaCAT front end is concerned, and if an EM software can be controlled via script, a SO for that EM software can be added without major modifications, no matter the interface language. Figure19 shows the clear division between individual SOs and their respective EM simulation software.

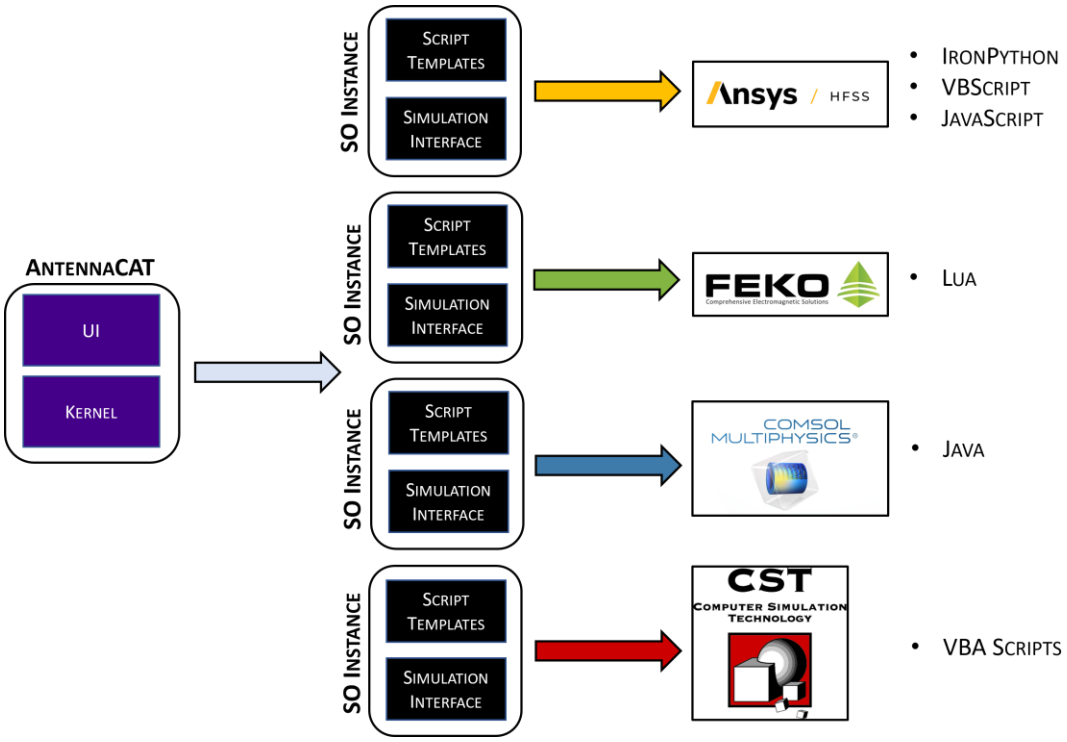


Figure 19 *A high-level visualization of the modularity of the Simulation Object instances and the relation to several compatible EM simulation software suites. The respective scripting languages for each software are listed on the right.*

Figure 20 shows the process as the AntennaCAT kernel is aware of it, while Figure 21 shows the process with the 'invisible' Simulation Integrator and template generator. This distinction is important because the modular, interchangeable Simulation Integrator layers are the basis for adding or adapting the compatible EM simulation software suites. The Simulation Integrator is the first layer of integration, and the primary point of interfacing with the EM simulation software.

The Simulation Object as the AntennaCAT kernel sees it, is the EM simulation software. However, AntennaCAT has no way of differentiating EM simulation software at this point in the simulation process flow, nor does it have a reason to. As far as the AntennaCAT kernel is concerned, there is only one type of EM simulation software.

The Simulation Object is dynamically selected on the Settings page at the .ANCAT project creation. The Simulation Integrator uses the Simulation Object to control which template generation script classes are passed back as objects. This gives the Simulation Integrator control over the process. The template generation script classes are unique to their specific EM simulation software suite, and much like the template library are organized based on the action they complete. That is, from the point of view of the AntennaCAT kernel, calling a *create_rectangular_patch()* function will look the same for every EM simulation software suite. However, from the template generation side, it will trigger a series of commands and script templates unique to the specific EM simulation software suite and scripting language.

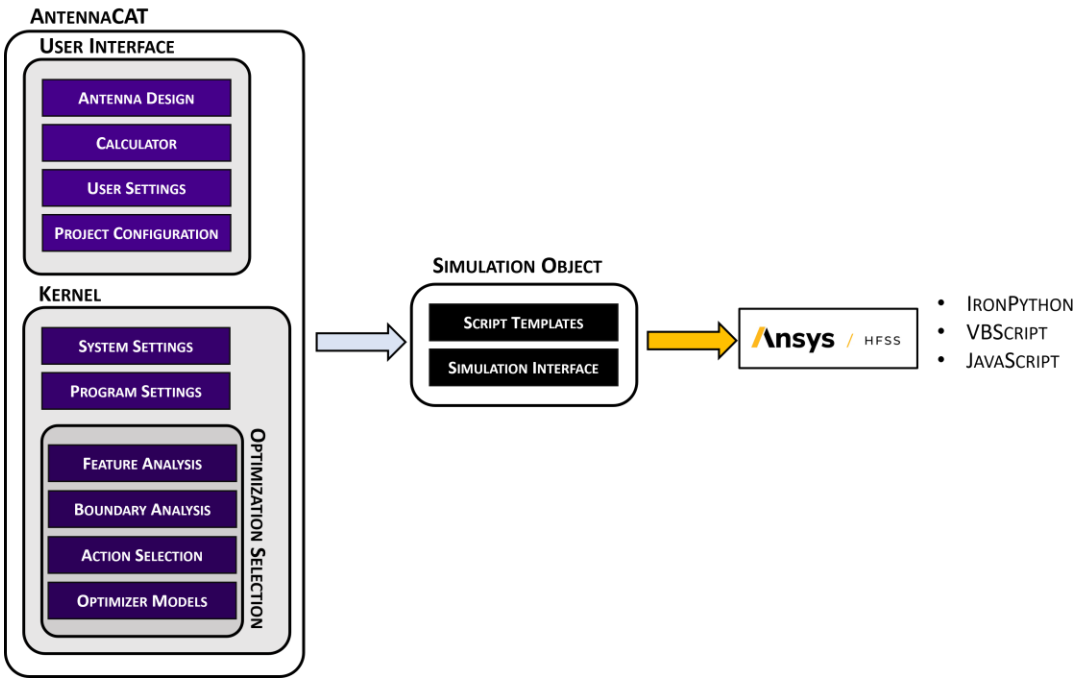


Figure 20 *The high-level simulation control process as seen by the AntennaCAT kernel. The kernel controls the SO, which controls the script execution in the EM simulation software suite.*

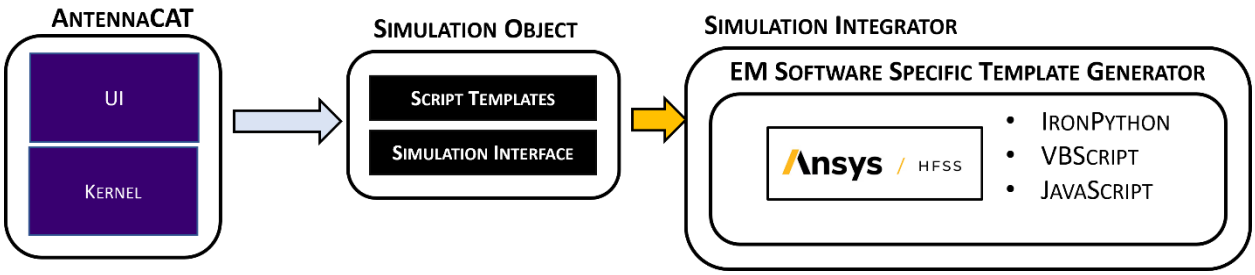


Figure 21 *The high-level simulation control process with the invisible simulation integrator and EM software suite specific template generator. The UI and kernel have been condensed visually, but the process remains the same.*

Figure 20 shows the split in the kernel between the operating system level settings, and the optimization selection. Running basic simulations or the Batch process uses only system commands to execute scripts with the respective EM simulation software suite. Optimization introduces an additional layer with Optimization Integrators and Optimization Objects, analogous to Simulation Integrators and Simulation Objects. The difference between the two sets is that the optimizer runs, returns values to the kernel, and then the kernel passes those values to the Simulation Object, which will command the Simulation Integrator to edit or create templates based on those new values.

Choosing which SO is utilized during tuning depends on the EM software selected by the user. While the Design page does not utilize the SO, the functionality is required beginning when simulation inputs are configured to generate antenna CAD and simulation scripts. In addition to the SO being designed based on a template, the SO contains text file templates for creating, editing, operating, and closing the EM software via script. All templates in the SO have been designed to operate independently, modularly, and without conflict against previous script commands.

Template Creation

The modularity of templates in the SO is driven by designing for how an antenna is created geometrically and procedurally, rather than how a human user might organically design, edit, and repeat commands. The advantage with this approach is that it naturally reduces potential conflict in script commands by ensuring that templated CAD elements have been reduced to a collection of the most minimal steps possible, and that all scripts for a feature are complete (i.e., do not rely on previous scripts to create a feature). Figure 22 shows an example block diagram for this process for creating a microstrip patch antenna on the first tuning iteration. When an antenna design is

simulated, three scripts have been created and stored in the project folder. The first script is the antenna CAD creation script. The second script configures the simulation properties, report creation, and export. Finally, the third script combines the two scripts aforementioned scripts, and is the script that AntennaCAT uses with the EM software. The CAD and basic simulation scripts can be executed by a human user outside of AntennaCAT to recreate designs and rerun simulations, but the scripts used by AntennaCAT in the tuning process contain paths to local data storage that should not be edited.

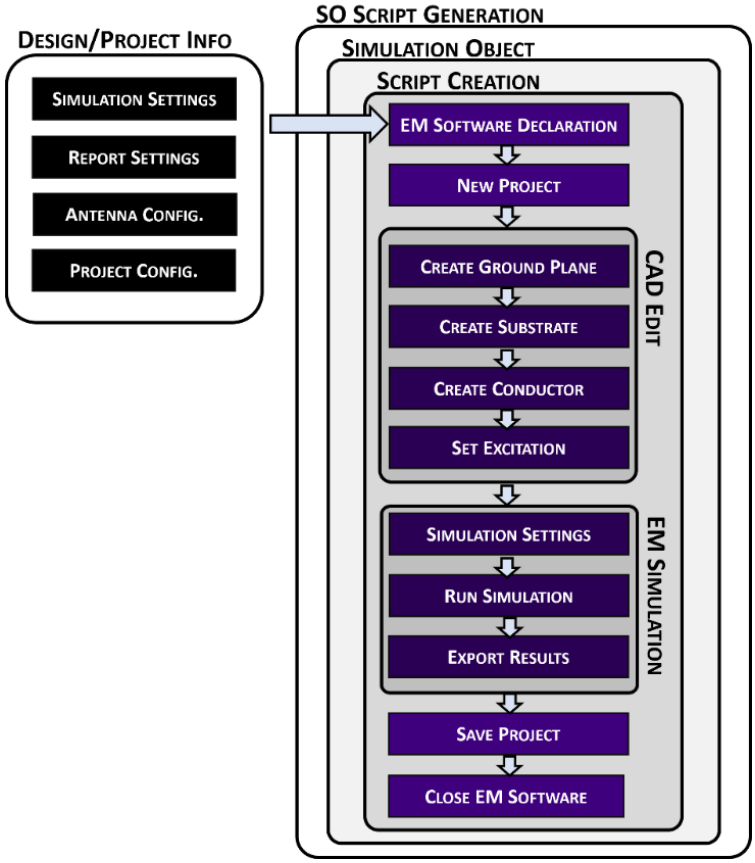


Figure 22 *A block diagram of the patch antenna creation process for the first iteration of tuning. The Simulation Object contains a collection of templates used to create a script file that can be used by the selected EM software.*

The templates in each SO are written in the language used by the EM software (i.e., IronPython for HFSS, and Lua for FEKO), but are managed by Python subprocess library when being manipulated by AntennaCAT. Figure 22 shows an example of how the SO would combine a series of templates to generate a script for simulation. Files in the template library have been reduced to the smallest possible instruction set to preserve the modular independence of each element. In cases such as the batch data generation and optimization, rather than creating a new project, after

the first simulation run, the template block for ‘New Project’ would be replaced with an ‘Open Project’ template, and the CAD Edit grouping replaced with parameter manipulation. This reduces the time processing needed for complex designs.

Antenna Tuning

The Tuning process is the catch-all term for AntennaCAT’s iterative parameter adjustment and automated simulation control capabilities. This process incorporates functionality from design, data collection, and optimization. Figure 23 shows an example of the simulation, analysis, and report export used in the iterative process.

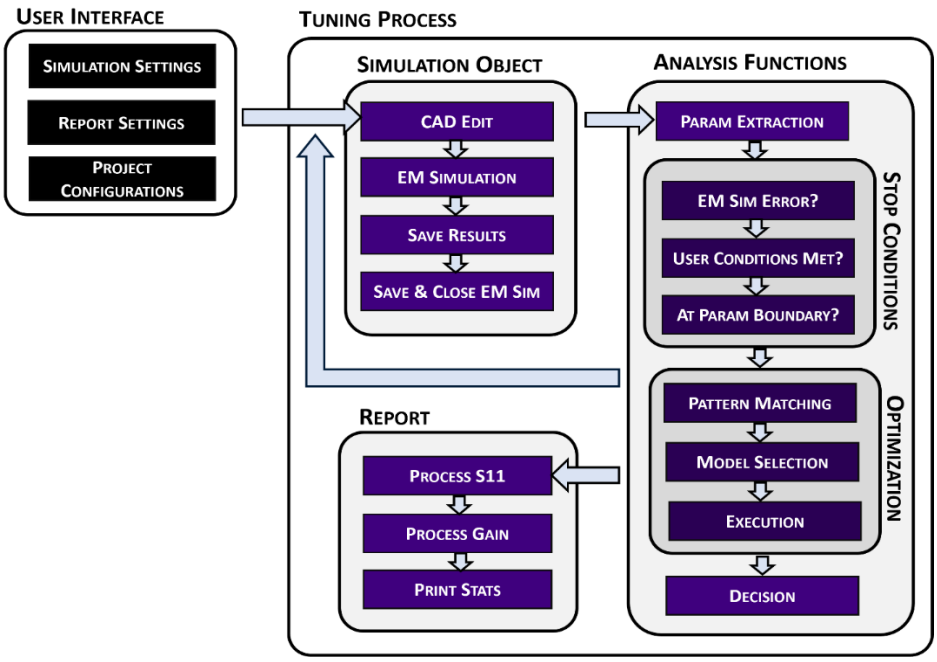


Figure 23 *Features of the Tuning process, including the Simulation Object and core features of the analysis and GUI report functions.*

Tuning uses a looping CAD-simulate-analysis process to automate the antenna design process. Whenever this process is initiated, such as when beginning a new optimization process, a new SO is created based on the set EM software choice. The first iteration of the SO creates a new project with the selected EM simulation software, executes the script to model the antenna using the CAD process, and runs the first simulation. Simulation results in the EM software project are exported to a local directory, and then the project is saved and closed. Closing the EM software does not

terminate the AntennaCAT tuning process and is used to save the results between running scripts to enable recovery from simulation errors. The exported simulation results are read into AntennaCAT to analyze several factors, such as the simulated resonant frequency, and closeness to target gain, and simulated S_{11}. If a simulation is completed without issue, and data recorded, then functions within the Analysis Functions block will pass sampled data to the Report block, which updates the GUI display. Parameter adjustment varies based on the specified optimization method and the known features of the antenna topology. The Optimization chapter goes into further details of example strategies and considerations. Decisions from the optimization method are then integrated into a new script designed to open the previously created project, then edit existing parameters, and finally rerun simulation. If a solution has been found that meets the user-set minimum constraints, then the results are returned, and tuning is ended.

2.5 Batch Data Collection

The primary purpose of the batch processing functionality is to create a dataset for a selected antenna design from a defined set of controllable parameters. Using the ranges of changeable values, all combinations of parameter manipulation are simulated, similar to a parameter sweep common in most EM simulation software.

During simulation iterations, AntennaCAT exports results to a .csv compatible format containing data from the automated batch simulation. Users can select which values (S_{11}, gain, BW, etc.) should be recorded in the dataset when generating customized sets. All identified physical parameters that can be adjusted in the design will have their values recorded at every iteration to complete the dataset. Other topologies with specific design features, such as dual-band antennas, will have the option to include multiple detected resonances, but will need to be specified by the user. The Batch functionality does not use any intelligent parameter value adjustment. Batch and Optimization processes operate independently of each other and cannot be run simultaneously within a single application instance. User-uploaded scripts must have CAD dimensions controlled by parameter variables and follow predictable naming conventions. This allows configuration, optimization, and tuning of non-analytical, or semi-analytical topologies, not created by the designer.

2.6 Open-Source and Availability

The AntennaCAT suite, its optimizers, and other utilities aside from the proprietary electromagnetics software used for simulations, are open-source online from their various authors. The complete collection of repositories, tutorials, and related publications are listed in Chapter 9. The AntennaCAT software and peripherals are written in Python, though versions in other languages may be available from their respective authors.

CHAPTER 3
Replication Studies

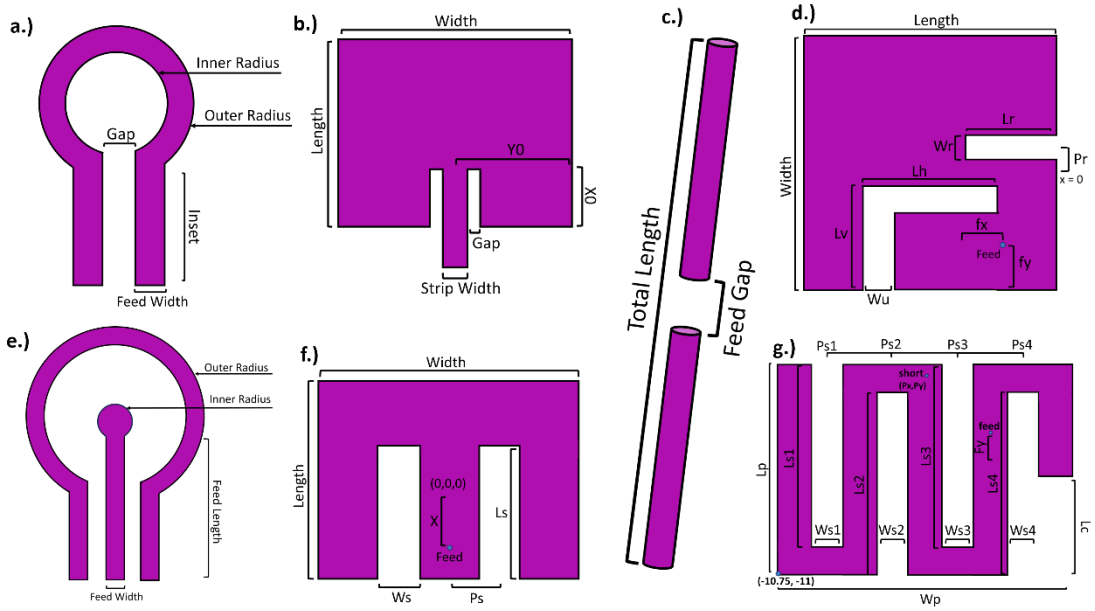


Figure 24 *A selection of reference images for the replication studies available in AntennaCAT. a) a simple loop antenna, b.) a microstrip fed rectangular patch antenna, c) a half-wave dipole, d) a slotted patch antenna [57], e) a coplanar keyhole antenna [58], f) a microstrip E patch antenna [59], and g) a dual band serpentine patch [60].*

The AntennaCAT software includes over a dozen built-in replication studies, with plans for continuous expansion. These include versions of the antennas created using the integrated AntennaCalculator (i.e., the rectangular patch antenna, the half-wave dipole, and quarter-wave monopole), and other designs popular in literature. Figure 24 shows 7 examples included in AntennaCAT either from literature [57, 58, 59, 60], the internal calculator (24.b-c), or designs from request by early users (24.a). These options are selectable from the Antenna Generator on the Design page, as shown in Figure 25.

Replication designs were selected to be incorporated either due to their prevalence in antenna design and benchmarking (e.g., rectangular patches, slotted patches, and the E patch), or due to their complexity [60, 61, 62]. The full list of current replication study designs is included in Table 1 below.

Table 1 *A summary of current designs included in the replication study set, the number of controllable parameters related to the study, and the source(s) of their designs.*

Design	**Num. Parameters**	**Source**	**Design**	**Num. Parameters**	**Source**
Coplanar Keyhole	4	[58, 63]	Coplanar Monopole	16	[68]
Square Loop (FSS)	4	[64]	Double-Sided Bowtie	16	[69]
Square Spiral (FSS)	4	[33]	Serpentine Patch	18	[60, 61]
E	6	[59,65]	Hexagonal, Ring-Shaped Fractal	21	[62]
Slotted Patch	10	[57, 66]	substrate integrated waveguide cavity-backed L-shaped slot antenna	23	[32]
Double-Sided Vivaldi	11	[67]	Others	4+	[70,71,72]

All replication studies include settings for the substrate height, the conductor material, the substrate material, and any parameters included in the literature for the specific topology. In Figure 26, a dual band serpentine antenna [60] with 20 controllable physical parameters has been selected to show the labeled controllable parameters from literature, the diagram showing the physical characteristic associated with each parameter, and the 3D preview showing how the parameterized design varies from the reference image.

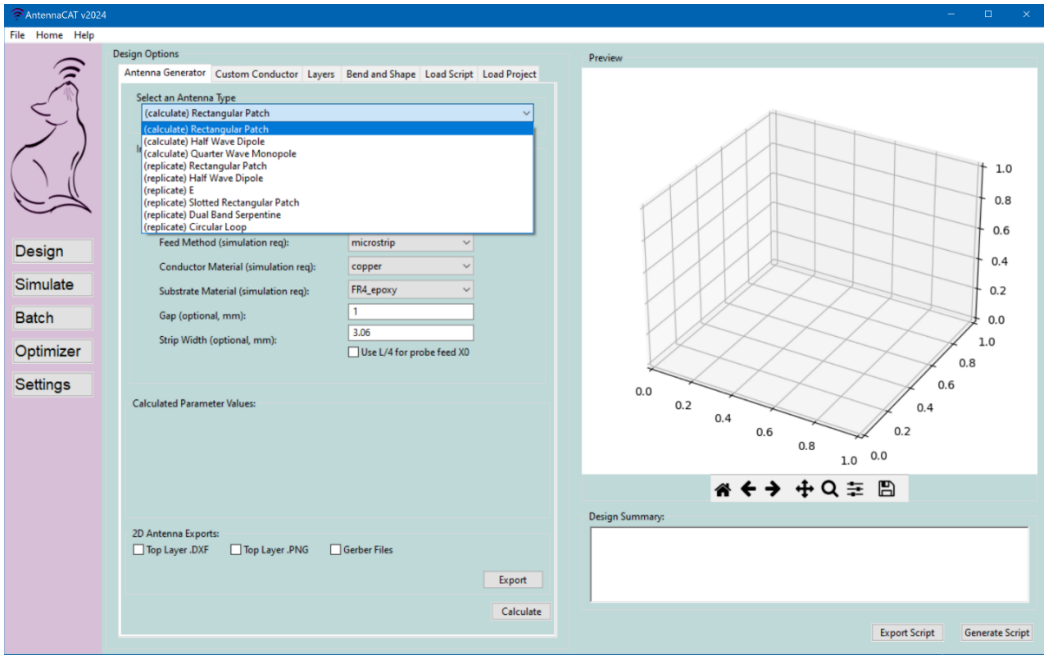


Figure 25 *The AntennaCAT Design page showing a selection of calculation and replication options for antenna design.*

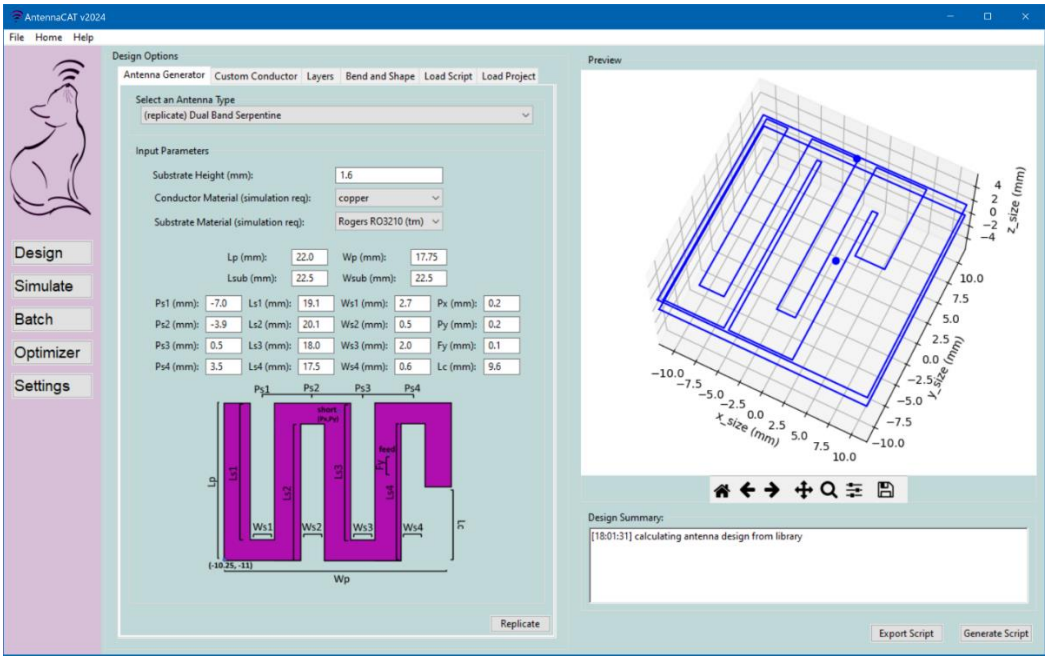


Figure 26 *The dual band serpentine replication study from [60] on the Design page. Left, the default parameters. Right, the 3D preview including the probe location.*

The 3D preview is generated with the 'Replicate' button at the bottom of the Antenna Generator tab and will accurately reflect the CAD model generated in the EM simulation software suite in terms of scale. This is exceedingly important for cases such as Figure 26, where the edge of the

substrate (represented by the outer box around the serpentine) is close to the edge of the antenna. To encourage experimentation, AntennaCAT does not place limits on parameter values, or add constraints to the parameter relations to each other in the replication studies. As an effect of this, it is possible to create antennas with invalid configurations, intersecting lines, probe feeds not located within the substrate, etc.. The generated CAD files are carefully parameterized to handle these inputs, but that does not mean such designs are valid for simulation or solution setup.

All the included replication study designs are compatible with the internal optimizer set, and some have been used to collect hyperparameter performance data for the neural network trained on the data collected from the objective function suite in Chapter 5

CHAPTER 4
AntennaCAT Optimization Suite

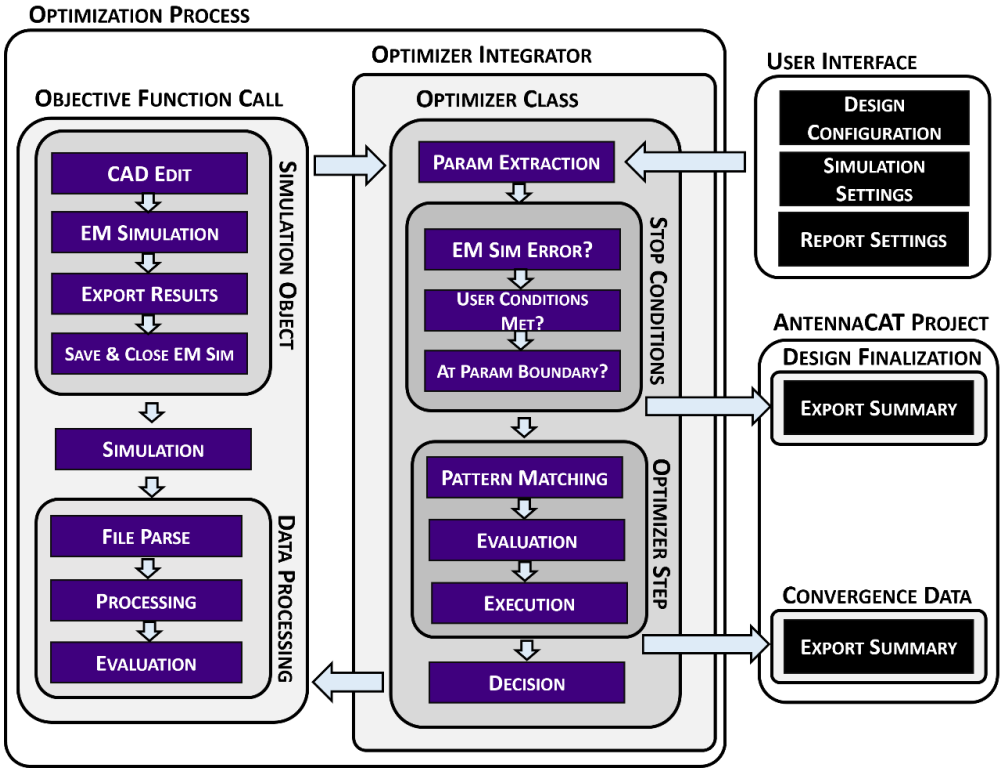


Figure 27 *An overview of the optimization process. The Optimizer Integrator is used with a dynamically declared optimizer class object so that new optimizers can be easily integrated with AntennaCAT.*

There are eleven core optimizers integrated into the AntennaCAT optimization suite. Figure 27 shows a high-level diagram of AntennaCAT's optimization process, including the Optimizer Integrator, which allows the optimizers to be used interchangeably. Figures 28 and 29 show the configuration screens for two types of optimizers: a traditional particle swarm optimizer, and the MultiGLODS optimizer. These screens provide the optimizer configuration data needed in the 'User Interface' block in Figure 27. This data is parsed by AntennaCAT and provided to the Optimization Object (similar to the Simulation Object described in Chapter 2) that manages the optimization process. This object is created at runtime with the selected optimizer information, the detected controllable parameters in Figure 15 (Chapter 2), and the target values for the antenna design. The Optimizer Integrator manages the dynamically at runtime selected optimizer interfaces, so that no distinction is made between selected optimizers by the AntennaCAT kernel.

The eleven optimizers in AntennaCAT include a Particle Swarm optimizer, a Cat Swarm optimizer, a Chicken Swarm optimizer, a Bayesian optimizer with a Gaussian Process kernel, and a MultiGLODS optimizer. Several variations of the swarm optimizers, and a sweep optimizer, are included in the optimizer suite. Each optimizer, and its variations, are discussed in the subsections below. Also discussed as part of the Bayesian optimizer is the surrogate model library. It is possible to use all optimizers except for the sweep optimizer with all surrogate models, though not all combinations will converge. In total, there are 90 optimizer-surrogate model combinations, and additional configurations such as boundary condition handling which may cause unique optimizer behavior.

The selection of optimizers has been integrated based largely on their popularity in literature. The swarm-based optimizers use the version of the optimizer as they were introduced into the field of electromagnetics. Likewise, the 'quantum inspired' set are based on the versions of the optimizers as they were initially introduced. The quantum inspired optimizers use random numbers to introduce uncertainty into the particle movement. This process is further explained in the Quantum-Inspired Optimizers section in this chapter. When possible, optimizers have been grouped together under single tabs to make GUI navigation easier. For instance, the optimizers in the 'Swarm Based' tab grouping are related to traditional PSO, but are modeled after other natural occurrences to utilize specific behavioral problem solutions. Any quantum inspired counterparts are listed under their own 'Quantum Inspired' tab. The MultiGLODS [73] optimizer is a comparatively newer (2018) algorithm, a multi-objective direct search algorithm with a merge function in the search step. MultiGLODS and the Bayesian optimizers [74, 75, 76] have their own tabs as they are unique in the optimizer set.

Gradient-based optimizers rely heavily on initial solution guesses and their efficiency for finding local optima. This brings up two issues: first, simulations are time and computationally expensive, thus having multiple initial solution guesses is not efficient; secondly, electromagnetics problems tend to have multiple local minima in addition to the global minima. Given this, it was decided to implement primarily derivative free optimizers in the AntennaCAT software suite. This decision had the additional benefit of finding solutions that are not influenced by the initialization or initial state of the optimizer.

Figure 28 shows the initial screen for *Swarm Based* optimization selection. It includes a traditional particle swarm optimizer [77, 78], a time modulated step particle swarm optimizer [79], a simple cat swarm optimizer [80, 81], a sand cat swarm optimizer [82, 83], and a chicken swarm optimizer [84, 85, 86]. Multiple boundary types (Random, Reflection, Absorb, Invisible) are available [87, 88, 89], though random boundary is the default. Implementation of each optimizer is explained later in this chapter.

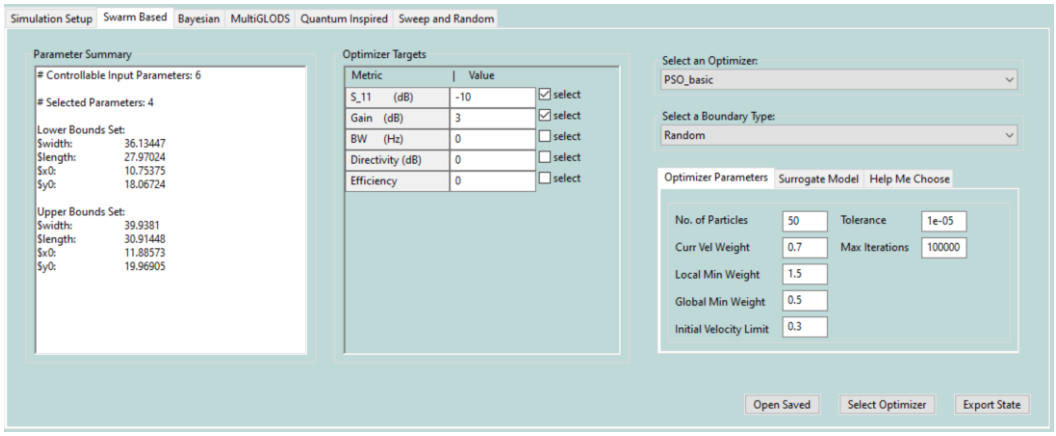


Figure 28 *The initial Swarm Based optimizer group customization tab on the Optimizer page.*

Figure 29 shows the MultiGLODS optimizer tab populated with the number of controllable input parameters, the number of selected parameters, and the lower and upper bounds. This field is populated when the 'Apply Configuration' on the detected controllable parameters box is clicked (see Figure 15).

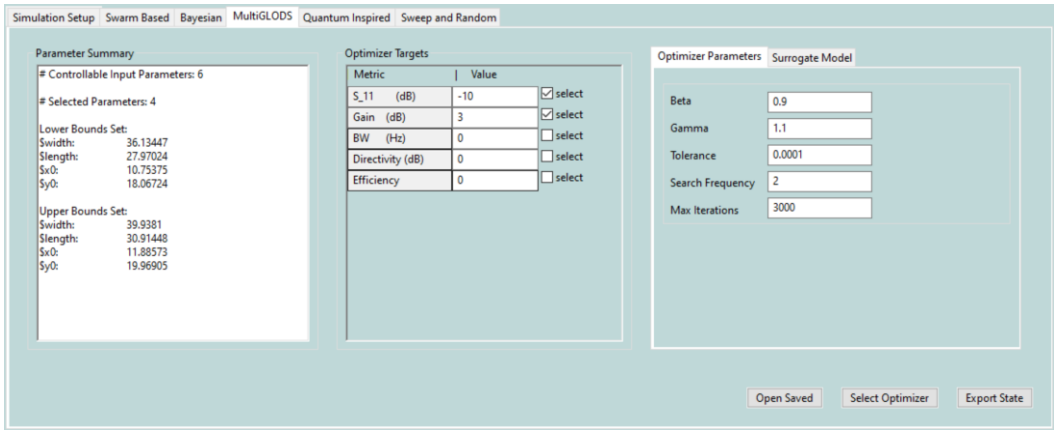


Figure 29 *The MultiGLODS optimizer tab on the Optimizer page. It shows the dynamically detected input parameters, the number of selected parameters, and the lower and upper bounds for the selected parameters.*

Unlike the MultiGLODS optimizer collection in Figure 29, the Swarm Based optimizer tab in Figure 28 includes the ‘Help Me Choose’ tab. On the ‘Help Me Choose' tab are suggestions for hyperparameters based on three factors: the selected optimizer, the number of selected controllable parameters, and the number of selected optimizer targets. AntennaCAT user can choose to use the suggested hyperparameters or not.

Optimizers demonstrating high parameter sensitivity during the data collection process described in Chapter 5 were excluded from the final dataset used to train the neural networks used to predict the hyperparameters. In the current optimizer set, the Bayesian Optimizer, the MultiGLODS optimizer, Sand Cat Swarm, and the sweep optimizer are excluded. Details on this are included in the chapter subsections relating to the respective optimizer. In Chapter 5, the data collection process and machine learning approaches used on the dataset are discussed.

There are five Optimizer Targets options available for the optimizer suite: S_{11} (dB), gain (dB), BW (bandwidth, Hz), directivity (dB), and efficiency. All five are available for all optimizers, though in experimentation it has not been efficient to use all five targets at once. Typically, S_{11} and gain have performed well enough for practical purposes. S_{11} is set to -10 dB by default. It is up to the user to set target values consistent with the output of the selected EM simulation software suite configurations. BW, directivity, and efficiency are set to 0 as defaults, and need to be changed before they can be used as target metrics. All target metrics are considered with the same priority, though they are evaluated in the same order they are listed, so if there are errors higher in the listed order, the later values will not be evaluated separately. In practice, using the variance of the simulation from the target resonant frequency, reflected power (S_{11}), and the overall gain of the antenna, is enough to utilize optimizers for topologies with few parameters and semi- analytical solutions, but the optimization performance drops as more features are added.

When multiple frequencies have been selected on the *Simulation Setup* tab, the target values will be applied to both frequencies. This allows for multiple frequencies to be evaluated simultaneously and applies to dual-band or tri-band designs in experimentation. Higher numbers of bands are possible but have not been evaluated for practicality reasons. It is suggested to optimize for features such as antenna footprint by controlling the physical parameters in the controllable parameters panel on the Optimizer page. Multi-band designs do not affect the problem bounds, but footprint minimization does as the lower and upper bounds for parameters are constrained.

The implementation of the presented optimizers does not require knowledge of specific topology. The only information provided to the optimizers are the selected target metrics (return loss, bandwidth, gain, directivity, efficiency), the target values, the controllable parameters extracted from the generated or imported scripts, and their upper and lower bounds. A mix of deterministic and stochastic optimizers are included to address a range of user needs, and to incorporate popular optimization methods for replication studies. Model-based optimizers have been explored, but due to their limited adaptability to unknown topologies, they are not the focus of the optimizer selection. By default, optimizers have pre-set tuning parameter values based on experimental usage. These values are not optimal for all topologies, nor are they expected to be [47], and Chapter 5 details how collected data has been used to train machine learning models on a range of objective functions to suggest initial hyperparameters based on the number of controllable parameters and targets. These hyperparameter suggestions are accessible through the 'Help Me Choose' tabs, and update dynamically based on user input.

Hyperparameters, more specifically model hyperparameters, are used in both the optimizers and the design of the pre-trained neural networks used to estimate initial optimizer hyperparameters. These hyperparameters are used by the optimizers to control the learning and navigation process through the state space. Manually tuning hyperparameters requires experimentation, is time-consuming, and is not easily reproducible across optimizers. To address this, many researchers are turning to automated hyperparameter tuning. In AntennaCAT, each optimizer selection has its own hyperparameters unique to that optimizer. For instance, PSO uses the number of particles, and several weight minimums and velocity limits, while the MultiGLODS optimizer utilizes a coefficient for step size contraction (beta), a coefficient for step size expansion (gamma), and a search frequency. All optimizers have tolerance and maximum iteration values, though what the tolerance controls may differ between optimizer groupings. For instance, in MultiGLODS, the tolerance controls the step size tolerance, and the radius is reduced in size as you approach the solution. However, in many other optimizers, the convergence criteria are based on distance from target, or the L2 norm of the standard metric on the function space output with respect to the target.

Default values for the hyperparameters in the Optimizer GUI are based on experimental examples, and are place holders, not optimized or globally well performing values. Respecting the no free lunch theorem [47], while an optimizer may perform well on one problem, it will not perform as

well on the set of all possible problems as no optimizers intrinsically have an advantage of being faster or more accurate on all problems. Likewise, no set of hyperparameters will perform well on every problem or for every optimizer. Optimizers must be selected based on compatible problem types, and then their hyperparameters must be tuned to the problem.

The next sections cover boundary and constraint handling mechanics for AntennaCAT compatible optimizers, and how optimization is automated in AntennaCAT. Following that, the individual optimizer implementations, including the surrogate model library, are discussed.

4.1 Boundary Condition Handling

All optimizers that generate or randomly sample the feasible decision space on an EM problem directly utilize boundary condition handling. For those that do not, including the Sweep grid search, hard boundaries may be used to restrict movement or to indicate the end of a search. These boundaries are statically set prior to starting an optimizer process. Boundary handling is conducted inside of the optimizer class, and not handled by AntennaCAT directly.

Optimizers using boundary condition handling are programmed with four types of bounds by default. In terms of the current optimizer suite, this includes all of the swarm-based optimizers and their quantum inspired counterparts. The Random bounds option will randomly respawn any particles or agents that leave the feasible decision space back into valid bounds. Reflection bounds will ‘bounce’ particles back into the valid space. Selecting Absorb for the boundary handling will cause the velocity of any particles to be ‘absorbed’, or set to zero, in that direction. Finally, the Invisible boundary conditions will cause any particles that have done out of bounds to no longer be evaluated. Depending on the optimizer implementation, these particles can either be set to inactive and their positions are no longer updated, or the particles can be set to inactive and their positions updated out of bounds but not considered for any calculations.

AntennaCAT optimizers use Random bounds by default, as it has experimentally proven to be the most stable. In cases where problem constraints are violated, but the problem bounds are not, the optimizer defaults to using random bounds logic.

4.2 Problem Constraint Handling

The optimizers used in the AntennaCAT require a constraints file to manage objective function constraints. If a file is not provided, then a default file with no constraints is used. This file is loaded into the optimizer constructor and cannot be changed during runtime. Examples can be found in the individual optimizer repositories on GitHub [79, 90-100]. When controlled by AntennaCAT, the default file is typically used. However, it is possible to add a custom constraints file. It is recommended to reference the README.md file for the GitHub repositories (including the AntennaCAT repository) for the creation of these files as they are highly problem specific and not included in AntennaCAT's error checking process. In the AntennaCAT Tuning process cycle, the objective function is replaced by an array containing the difference between the target values and the values exported from the simulation report values.

4.3 Single and Multi-Objective Optimization

All optimizers included in this section are compatible with both single and multi-objective functions. Not all optimizers will work equally well on both types of problems, but it is possible attempt problems regardless of input dimensions, or with any combination of the five target values that can be optimized with AntennaCAT (S_{11}, gain, etc.). AntennaCAT takes a 'no preference' method of multi-objective optimization but does not calculate a Pareto front. Instead, the 'best choice' is the smallest norm of output vectors (the L2 norm).

4.4 Objective Function Handling for Repository Examples

The stand-alone optimizer repositories [90-100] have a standardized structure to encourage experimentation. Every optimizer repository has three test objective functions included in the repository:

1. Himmelblau's function, which takes 2 inputs and has 1 output
2. A multi-objective function with 3 inputs and 2 outputs (see lundquist_3_var)
3. A single-objective function with 1 input and 1 output (see one_dim_x_test)

Each function has at least the following five files in a directory:

- configs_F.py - contains imports for the objective function and constraints, constant (static) assignments for functions and labeling, boundary ranges, the number of input variables, the number of output values, and the target values for the output.
- constr_F.py - contains a function with the problem constraints, both for the function and for error handling in the case of under/overflow.
- func_F.py - contains a function with the objective function.
- main_test.py – contains a script to run the optimizer and print out results to the terminal.
- main_test_graph.py - contains a script to graph the function for visualization.

These files follow the AntennaCAT compatible format, and the constr_F.py function can be imported into AntennaCAT to add constraints to project definitions. It is highly suggested that custom constraint function files be tested on an optimizer before being tested in AntennaCAT.

4.5 AntennaCAT Optimizer Compatibility

AntennaCAT compatible optimizers are functional as standalone programs for testing and publication transparency [90-100]. Specific details for each optimizer are included on their specific README.md pages, however AntennaCAT compatibility sets the following requirements:

1) Compatible optimizers (and integrated surrogate models) follow state machine logic to incorporate simulation as the objective function evaluation. With this process, the optimizer is initialized, and then a while loop is run until stop conditions are met. The stop condition can include a maximum number of iterations, or an evaluation within a specific error tolerance.
2) Boundaries are problem specific and enforced with lower and upper bounds set either by a configuration file (configs_F.py) or the AntennaCAT GUI. These are static and cannot be changed during the runtime of the optimizer.
3) Constraint files are used to enforce problem constraints (outside of those handled by the problem bounds enforcement). If a user does not specify a file, then a default file is used that always returns two Boolean true values.

4) The objective function, which can either be an objective function from a file (func_F.py) or a simulation, has two outputs. The first output is an array containing the function evaluation (or processed simulation export reports) corresponding to specific target values. When operating as a stand-alone program, the targets are included in the configuration file (configs_F.py). For single objective functions in the objective function suite, every target is 0. For multi-objective functions, target values were chosen to be values on the Pareto front. The second output of the objective function is a Boolean that represents if the objective function was successfully evaluated. This second Boolean is represented with a 'NoError' value in code, indicating that there has been no error in the evaluation of the constraints.
5) Optimizers use the standard format for function calls to step through the state machine, call the objective function, check if the optimizer has converged or otherwise met completion criteria and to return convergence data for logging. All optimizers use the format myOptimizer.complete(), where 'myOptimizer' is an arbitrary optimizer class object, to return a bool if completion criteria have been met. These criteria include at least a maximum number of iterations, and an L2 norm distance to the target. Only one of the criteria must be met. myOptimizer.step(*suppress_output*) is used to advance the state machine one step forward. *suppress_output* is a Boolean value controlling if detailed messages are displayed or not, a value of 'True' will suppress all messages from the optimizer class. myOptimizer.call_objective(*allow_update*) will attempt an objective function call if the optimizer is in a valid state for an objective function call. For most optimizers, this will trigger a call to the objective function. In some, such as MultiGLODS, an objective function call will not be made. The *allow_update* Boolean allows for the optimizer to be run through steps without calling the objective function. Finally, myOptimizer.get_convergence_data() is used to return a counter for the number of times the objective function has been called, and a best evaluation float value for the current best evaluation of the optimizer. These function calls can be seen in the main_test.py files for all optimizers.

In the following sections, the optimizer descriptions will cover the initialization and the call_objective() function, which allows the optimizer to call the objective function if the state machine is in a valid state for the call to be executed. The step() function call that causes the state machine to take one step forward is not discussed except in terms of state machine behavior. Images of optimizer functions are provided from their respective stand-alone repositories.

AntennaCAT does not currently graphically display optimizer details, particle position, surrogate model surface meshes, or other similar information.

4.6 Particle Swarm Optimizers

Particle Swarm Optimization (PSO) is a popular nature-inspired optimization algorithm introduced in "Particle Swarm Optimization" [101] in 1995. This algorithm was inspired by the social behavior of collaborative animal groups and is often compared to birds flocking or fish schooling. PSO is used to find approximate solutions to complex optimization problems and is often implemented on problems that have local minimum solutions in addition to a global solution, or multiple global solutions. Particle Swarm-based optimizers, in addition to the original Particle Swarm Optimization, are popular optimization techniques for multi-parameter, and multi-objective, problems, and are popular in studies using several of the featured replication study topologies. PSO consists of a population, or swarm, of candidate solutions called particles. Each particle moves through the search space influenced by its own best-evaluated position and the swarm global best.

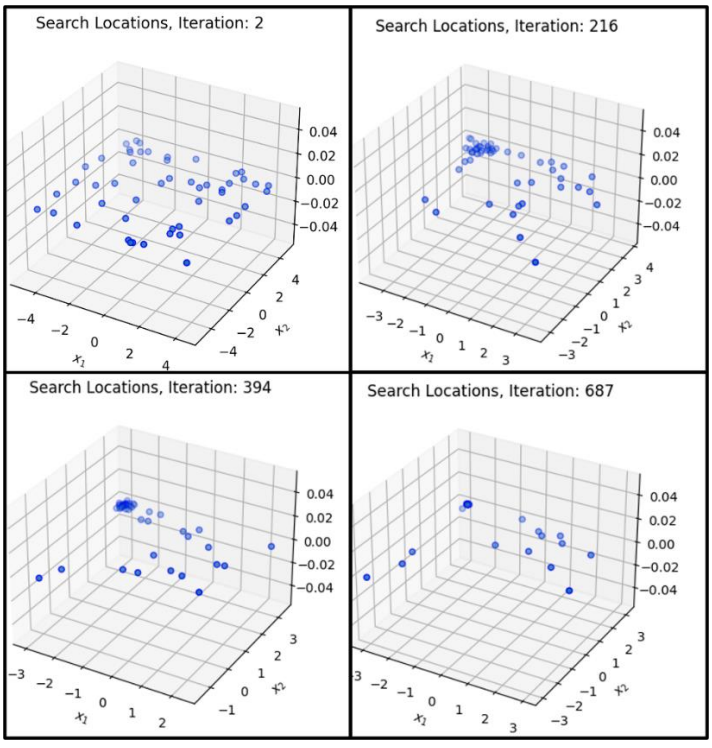


Figure 30 *Particles making up a 50-agent swarm in a traditional PSO algorithm converging on the single-objective Himmelblau's function global minima at 2, 216, 394, and 687 steps.*

Figure 30 shows the convergence of a traditional PSO algorithm on the single-objective function at steps 2, 216, 394, and 687. The 50 particles used in this example begin to converge visibly on a target by step 216 and have found a global minimum within an error tolerance of 10e-6 by iteration 687. The traditional particle swarm optimizer, as it was initially presented in literature, was used to find the global minima for Himmelblau's function, one of the objective functions included in Chapter 5 and the objective function library used for optimizer testing in AntennaCAT. Traditional PSO was used as the visual behavior of the swarm does not differ much from the other swarm-based algorithms in this section to the casual observer. However, there is a notable visual difference between Figure 30 and 31, using the same swarm configurations. Figure 31 uses a multi-objective test function from the test function suite that was designed to be fast to converge. To prevent convergence for demonstration purposes, the error tolerance for the solution was set to 10e-15.

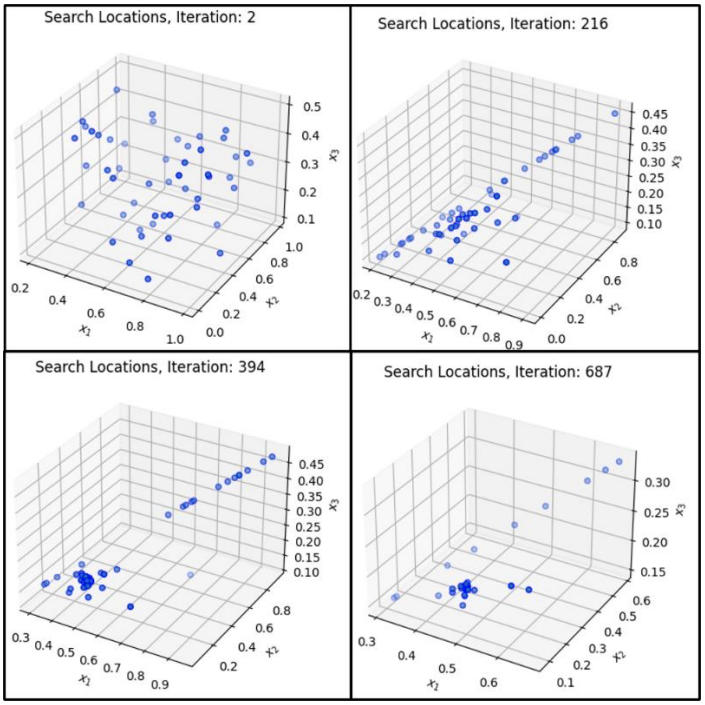


Figure 31 *Particles making up a 50-agent swarm in a traditional PSO algorithm converging on the multi-objective function target on the Pareto front 2, 216, 394, and 687 steps.*

In Figure 31, the swarm at step 2 is still randomly dispersed. By step 216, the swarm has begun to find the front of the feasible objective function space. In this example, a single point is Pareto Optimal (the target solution), which the swarm is converging on by iteration 687.

The AntennaCAT swarm-based optimizers track the global best position, the global best fitness (objective function evaluation), personal best position, personal best fitness, and the active/inactive status of the particles. The following subsections describe the initialization, optimizer step, and objective function call handling.

Traditional Particle Swarm

The traditional particle swarm optimizer included in AntennaCAT is based on the one proposed by [101] in 1995. It has no time-step modulation, no scalable search, no mutation, and other modifications made in the three decades since the algorithm was introduced. This optimizer is included in AntennaCAT to include a baseline for improvement on swarm-based optimizers, which is arguably the most populous optimizer type in electromagnetics.

It is available at: https://github.com/jonathan46000/pso_Python/tree/pso_basic

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Weights:
    - Inertia weight: previous movement impact
    - Cognitive Coefficient: individual exploration
    - Social Coefficient: group exploitation
- Velocity limit
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the traditional PSO optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Update the velocity vector
    3. Update the particle location
    4. Handle bounds

Particle Swarm with Time-Step Modulation

This particle swarm optimizer adds a time step modulation to the traditional PSO algorithm. It uses the mean absolute deviation of particle position as an adjustment to the time step to prevent the particle overshoot problem. This particle distribution is initialized to one when the swarm starts, so that the impact is boundary independent.

It is available at: https://github.com/jonathan46000/pso_python/tree/main

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)

- Weights:
    1. Inertia weight: previous movement impact
    2. Cognitive Coefficient: individual exploration
    3. Social Coefficient: group exploitation
- Velocity limit
- The number of output (target) values
- Numerical target values
- The time modulation parameter
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the traditional PSO optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Update the velocity vector
    3. Update the particle location, which uses the time modulation
    4. Handle bounds
- After all particles have been stepped through (1 per step()), update the delta t, which is the adaptive time step modulation value

4.7 Cat Swarm Optimizers

Cat Swarm Optimization is a nature-inspired algorithm based on the behaviors of cats. Introduced in the 2006 paper "Cat Swarm Optimization" [80, 81], this algorithm shares similarities with the traditional Particle Swarm Optimization algorithm, incorporating both location and velocity components. However, unlike PSO, the particles in Cat Swarm Optimization have two movement options during the update step, modeled after the behaviors of cats. In the traditional Cat Swarm Optimization, these are seeking and tracing. Sand Cat Swarm Optimization [82, 83] uses an attacking (exploitation) and a search (exploration) mode.

Traditional Cat Swarm

This optimizer uses the original algorithm proposed in [80, 81] with no modifications. Cat Swarm Optimization divides the population of candidate solutions (cats) into two groups: those in seeking mode, and those in tracing mode. Each cat can switch between these modes according to a specified probability.

1. Seeking Mode:

The seeking mode is responsible for exploring the search space to uncover new and potentially superior solutions. In this mode, cats simulate a behavior where they observe their environment and decide on new positions based on various potential moves. This process aids the algorithm in avoiding being trapped in local optima.

2. Tracing Mode:

The tracing mode is dedicated to exploiting the search space by pursuing the best solutions discovered by the cat swarm so far. In this mode, cats imitate a behavior where they move toward a promising position, similar to a cat chasing prey. This approach not only helps refine solutions and converge towards the global optimum but also ensures that the most promising regions of the search space are thoroughly explored and optimized.

It is available at: https://github.com/LC-Linkous/cat_swarm_python

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Weights, used for the velocity in tracing mode only
- Velocity limit
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection
- Mixture ratio (MR). Small value for tracing population size
- Seeking memory pool (SMP). The number of copies of cats made in seeking mode
- Seeking range of a selected dimension (SRD)
- Counts of dimensions to change (CDC), a mutation variable
- Self-position consideration (SPC), a Boolean for if the cat being copied in the seeking mode step should be considered in the copy count

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the Cat Swarm Optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Use the current particle's mode (seeking, tracing) to update location:
        2.1 Call tracing_mode()
        2.2 Call seeking_mode()
- Handle bounds

Sand Cat Swarm

In 2022, Sand Cat Swarm Optimization [82, 83] was proposed as an algorithm able to escape local minima while retaining a balance between the exploitation and exploration phases of the Cat Swarm Optimization algorithm. It has far fewer parameters and operators than other metaheuristic algorithms, including those in the AntennaCAT optimizer suite, making it easier to implement than some swarm algorithms. A trade off, however, is that this algorithm may be more computationally expensive in the short term due to the exploration phase of the algorithm in the step function call evaluating the objective function for all copies of the cats.

There are two stages for this optimizer:

1. Exploitation:

This stage is the 'attacking prey' phase where the particle is moved in the feasible decision space to introduce new possible solutions to the swarm. During this phase, when a particle's position is updated, both random numbers and random angles (from 0 to 360 degrees) are utilized to promote movement away from the current location. The new position and subsequent step are weighted according to the global best solution, ensuring that exploitation remains focused around high-performing regions.

2. Exploration:

The second stage is the 'search mode' stage of the algorithm. This stage selects a random particle from the swarm as a 'candidate'. This candidate is used to create several copies with similar characteristics that are then evaluated with the objective function to find the next potential best

position. The best evaluating copy is then used to move the cat. This mode encourages the herd behavior.

It is available at: https://github.com/LC-Linkous/cat_swarm_Python/tree/sand_cat_python

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Weights, used for the velocity in tracing mode only
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called. The step function will also call the objective function for any candidate cats in the search mode phase of the algorithm.

Optimizer Step:

When the optimizer step() function is called, the Sand Cat Swarm optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:

1. Check if the current location is a global local
2. Use the current particle's mode (exploitation, exploration) to update location:
   2.1 Call exploitation_mode()

   2.2 Call exploration_mode()

3. Handle bounds

4.8 Chicken Swarm Optimizer

The Chicken Swarm Optimization algorithm was introduced in 2014 by [84]. It is inspired by the hierarchy and behaviors observed in a swarm of chickens. Each type of bird (i.e., roosters, hens, and chicks), has its own unique movement rules and interactions based on two main types of hierarchy; flock and relational (maternal). There can be multiple swarms of chickens within a single particle swarm instance. Within each of the chicken swarms, the hens are also split into two types: hen and mother hen. Mother hens have at least one chick that follows their movement. Unlike the other swarm-based optimizers, there is an absence of an explicit random velocity component for each particle.

The general movement rules for each type of bird are as follows:

1. Roosters:

   Roosters have the best positions (fitness values) in the swarm. There is one rooster per chicken swarm, though there can be multiple roosters in the particle swarm implementation. The roosters move based on their current position and a random perturbation, which is used to reduce the chances of remaining in a local minimum. If a rooster gets stuck in a minimum, or another chicken in the swarm finds a new 'best' position, the rooster will be demoted in the hierarchy to a chicken, and a new rooster chosen.

2. Hens:

   Hens follow roosters. Hens in a chicken swarm update their positions based on the current position of their rooster and a randomly selected chicken. Following the rooster encourages

the swarm behavior of the algorithm, while using a randomly selected chicken's position encourages some variation to the movement, preventing the chicken swarm from converging on the rooster and not exploring an area.

3. Chicks:

   Chicks follow their mother hens. Not all hens have to be mother hens, but all mother hens have at least one chick. These particles update their position based on their mother's position and a random factor (float value) to simulate 'dependence'. Unlike the hens, which follow a rooster and another chicken, this dependence is used to make the chick's movement anchored on the mother hen, but not completely restricted.

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection
- Number of roosters (RN)
- Number of non-mother hens (HN)
- Number of mother hens (MN)
- Number of total chicks (CN)
- G, the generation value for how many iterations before the chicken designations in a swarm are shuffled.

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the Chicken Swarm Optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. If the current generation counter has reached a maximum, then reorganize the swarm
    3. Use the current particle's type (rooster, hen, chick) to update location:
        3.1 Call move_rooster()
        3.2 Call move_hen()
        3.3 Call move_chick()
    4. Handle bounds

4.9 Quantum-Inspired Optimizers

Quantum Particle Swarm Optimization (QPSO) was introduced in 2004 as an advancement on the traditional Particle Swarm Optimization algorithm [102, 103, 104, 105]. In [102], the authors describe traditional particle swarm as using Newtonian mechanics because a particle moves along a determined trajectory via a known position and a velocity vector. However, in quantum mechanics, the location and velocity vectors cannot be known simultaneously due to the uncertainty principle. By applying that principle to PSO by pulling inspiration from superposition and entanglement, these optimizes may have an advantage on some problem types.

Superposition, as it exists as a concept for the optimizers in this section, can be interpreted as a probability distribution over multiple states. In quantum mechanics, a particle can exist in a superposition of multiple states simultaneously. That is, if there are three possible states a particle can exist in, then there is a probability that it is in each one, but it is not possible to know with absolute certainty which state it is in. In QPSO, a particle's position is often updated using a probability distribution derived from the particle's personal best and the swarm's global best, to retain the swarming behavior. This uses the particle's personal best (which is not always the current location), as an anchor for this movement pattern. Using this method, rather than a deterministic position update, allows particles to explore the feasible decision space more effectively.

Entanglement in quantum physics is the phenomenon where particles become interconnected such that the state of one particle directly affects the state of another particle despite any amount of distance between the two. In QPSO, this relates to how particle position updates in a swarm influence others in the swarm due to each particle's location relying on the particle's personal best and the global best of the swarm.

The three quantum inspired swarm optimizers in this section (QPSO, Quantum Cat Swarm Optimization, and Quantum Chicken Swarm Optimization) are briefly discussed as they are of interest based on current literature, but for longevity of code they are not executed in the same way as current literature. Rather than using Qiskit, QuTIP, Pyquil, or other libraries that are either constantly adapting, new, or in early stages of documentation, the AntennaCAT versions of these optimizers use random numbers to introduce uncertainty into the movement model (superposition). These optimizers are based on a specific snapshot of literature, which the original QPSO algorithm introduced, and are meant for educational purposes.

Quantum Inspired Particle Swarm Optimizer

In contrast to conventional Particle Swarm Optimization, Quantum Particle Swarm Optimization (QPSO) does not employ a velocity vector. Instead, it updates particle positions directly using a probability distribution informed by the mean best position and a logarithmic factor, principles derived from quantum mechanics. The QPSO update mechanism utilizes quantum-inspired probabilistic movements to achieve a balanced exploration and exploitation strategy. By

integrating the strengths of both personal and global experiences with a stochastic element, QPSO effectively navigates complex optimization landscapes.

Updating the position vector of a particle is done in two steps. First, the mean best position is calculated from the swarm positions:

$$mb = \beta \cdot p + (1 - \beta) \cdot g \qquad (eq. 17)$$

Where β is a parameter controlling the influence between the personal best and global best positions.

The second step is the actual position update:

$$x_i(t + 1) = mb \pm \beta \cdot |p - g| \cdot \log(1/u) \qquad (eq. 18)$$

In eq. 18, *mb* is the mean best position from eq. 17, and β is the same user-defined parameter that controls the balance between personal best and global best. Some implementation may use different values for β, but the default for the AntennaCAT optimizer suite is to use the same value for both instances. p is the personal best of the particle, g is the global best of the swarm, u is a uniformly distributed random number between 0 and 1, bounds inclusive. The logarithmic term, $log(1/u)$, introduces a random factor in addition to the scaling β factor, which will be naturally biased towards smaller numbers because the random vector u is bounded [0,1]. That is, the logarithmic function naturally has a heavy-tailed distribution in this problem, so the particle movement will stay relatively controlled despite the random aspects to movement.

It is available at: https://github.com/LC-Linkous/pso_python/tree/pso_quantum

Unlike the other branches of the pso_python repository, the pso_quantum branch is not merged upstream. It is an experimental variation meant for educational purposes.

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Weights, used for the velocity in tracing mode only
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection
- The beta value for the position update (usually between 0 and 1)
- The number of input variables

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the QPSO optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Update the particle location using the mean best method
    3. Handle bounds

Quantum Inspired Cat Swarm Optimizer

The Quantum Inspired Cat Swarm Optimizer included in the AntennaCAT optimizer suite blends the traditional Cat Swarm Optimization algorithm with the mean best position update introduced in QPSO in the previous section. It does not use a quantum-based or quantum inspired library. In future work, other variations may be added to include other libraries designed for quantum inspired machine learning. The seeking and tracing modes of the cat swarm are implemented in this optimizer through the following.

In the quantum inspired seeking mode, the seeking algorithm is retained as much as possible. When a candidate particle is selected, copies are created with new positions using the mean best position update method described in the QPSO section previously. As with the original Cat Swarm algorithm, the fitness of the copies is evaluated, and the best preforming particle (the one with the lowest L2 norm fitness from the target) is selected.

The quantum inspired tracing mode uses a random vector u to update the particle movement towards the global best. This random vector is bounded [0,1].

It available at: https://github.com/LC-Linkous/cat_swarm_python/tree/cat_swarm_quantum

Unlike the other branches of the cat_swarm_python repository, the cat_swarm_quantum branch is not merged upstream. It is an experimental variation meant for educational purposes.

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Weights, used for the velocity in tracing mode only
- The number of output (target) values
- Numerical target values

- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection
- Mixture ratio (MR). Small value for tracing population size
- Seeking memory pool (SMP). The number of copies of cats made in seeking mode
- Seeking range of a selected dimension (SRD)
- Counts of dimensions to change (CDC), a mutation variable
- Self-position consideration (SPC), a Boolean for if the cat being copied in the seeking mode step should be considered in the copy count
- The beta value for the position update (usually between 0 and 1)
- The number of input variables

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the quantum inspired cat swarm optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Use the current particle's mode (seeking, tracing) to update location:
        2.1 Call tracing_mode()Call seeking_mode()
    3. Handle bounds

Quantum Inspired Chicken Swarm Optimizer

The Quantum Inspired Chicken Swarm Optimizer included in the AntennaCAT optimizer suite blends the mean best position update method with the movement models of the chicken swarm algorithm. The movement rules (in terms of hierarchy and relation) are preserved, the position changes are implemented as described in the following. This algorithm is a simplified blend of the two methods and is intended for experimentation purposes.

1. Rooster Movement:

Roosters are always the particles with the best evaluated fitness value in the swarm. These particles move based on their current position and a random vector in order to prevent stalling in a local minimum. The roosters in this algorithm can be toggled to move either with the original movement model as described in [84], or by using the mean best position update method as described in the QPSO section.

2. Hen Movement:

The hens in a chicken swarm follow their respective roosters. These particles update their positions in the quantum inspired algorithm using the mean best position update method. However, instead of using the global (for that flock) best position, the position of the flock's rooster is used.

3. Chick Movement:

Chicks in a traditional chicken swarm algorithm follow their assigned mother hen, and have a random vector added for modeling dependence. In the quantum inspired algorithm, like the hens, the chicks use the mean best position update. Instead of using a flock or rooster best, they use their mother hen as the anchor.

It available at:

https://github.com/LC-Linkous/chicken_swarm_python/tree/chicken_swarm_quantum

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of particles
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Boundary type selection
- Number of roosters (RN)
- Number of non-mother hens (HN)
- Number of mother hens (MN)
- Number of total chicks (CN)
- G, the generation value for how many iterations before the chicken designations in a swarm are shuffled.
- The beta value for the position update (usually between 0 and 1)
- The number of input variables
- A Boolean to use the classical or quantum inspired roster movement model

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called.

Optimizer Step:

When the optimizer step() function is called, the quantum inspired chicken swarm optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. If the current generation counter has reached a maximum, then reorganize the swarm
    3. Use the current particle's type (rooster, hen, chick) to update location:
        3.1 Call move_rooster()
        3.2 Call move_hen()
        3.3 Call move_chick()
    4. Handle bounds

4.10 Sweep Optimizer

The Sweep optimizer in the AntennaCAT optimization suite currently has two options for sweeping over an n-dimensional feasible decision space: random search, and grid search. Both options are compatible with single- and multi-particle search.

It is available at: https://github.com/LC-Linkous/sweep_python

Grid-Based Search

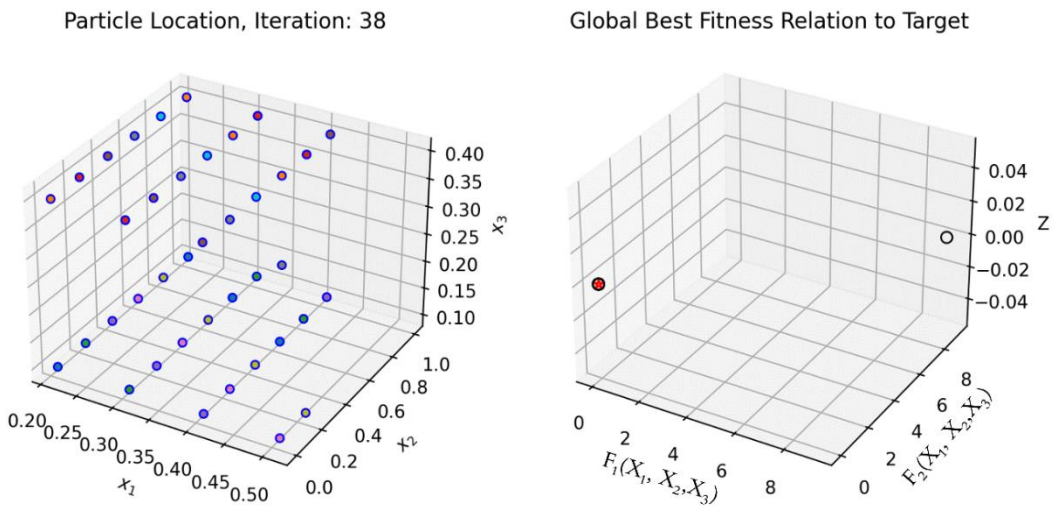


Figure 32 *The grid-based search option in the Sweep optimizer. On the right, the current and previous search locations. On the left, the red star is the global minima target. The black circle on the far right of the Global Best Fitness plot is the original evaluation, while the circle around the red star is the best evaluation.*

A grid-based sweep optimizer, often referred to as grid search, is a simple yet effective optimization technique commonly used for hyperparameter tuning in machine learning models. This method systematically explores a specified subset of the feasible decision space (and solution space). When using this method, an n-dimensional array is provided for the resolution in the n-directions that the particle will move in. If a single value is provided, it will be applied to all dimensions of the search space. The error tolerance for the example above in Figure 32 was reduced to 10e-15 to demonstrate the grid pattern. The plot on the left shows the current particle locations and the past grid search. These are updated in real time. The objective function surface is not shown with the example test class in the repository to avoid cluttering the plot. The global best fitness record is shown using the black circles on the plot on the right side of Figure 32. The red star represents the target value of the global minimum.

Random Search

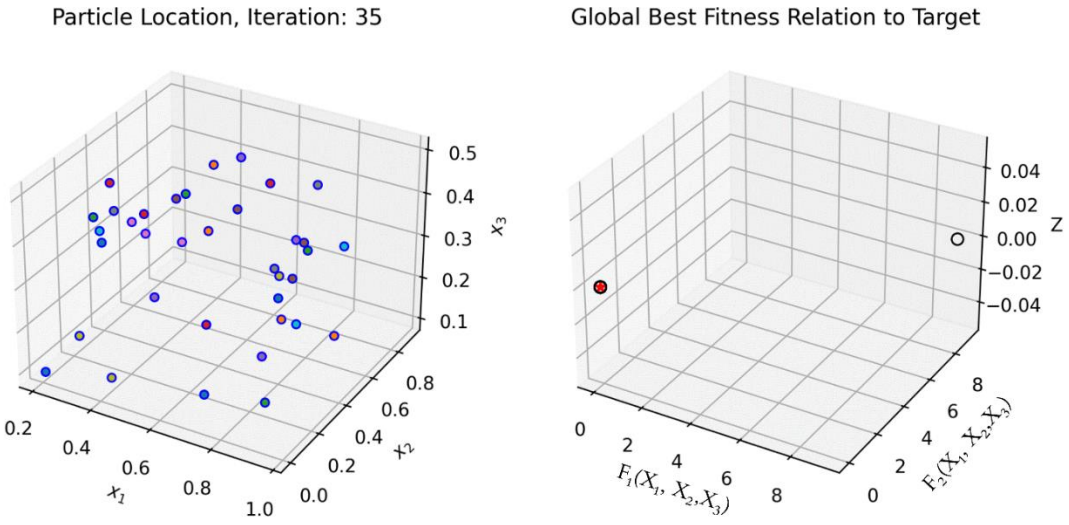


Figure 33 *The random search option in the Sweep optimizer. On the right, the current and previous search locations. On the left, the red star is the global minima target. The black circle on the far right of the Global Best Fitness plot is the original evaluation, while the circle around the red star is the best evaluation.*

The random search method employed in the Sweep optimizer is a fully random sample. This method is not iterative, or intelligent. It is meant to pull sampled from an n-dimensional space for evaluation. However, in small problem spaces it is not unusual for it to find a valid solution. The error tolerance for the example above in Figure 33 was reduced to 10e-15 to demonstrate the search pattern. The plot on the left shows the current particle locations and the past search locations search. These are updated in real time. The objective function surface is not shown with the example test class in the repository to avoid cluttering the plot. The global best fitness record is shown using the black circles on the plot on the right side of Figure 33. The red star represents the target value of the global minimum.

4.11 MultiGLODS

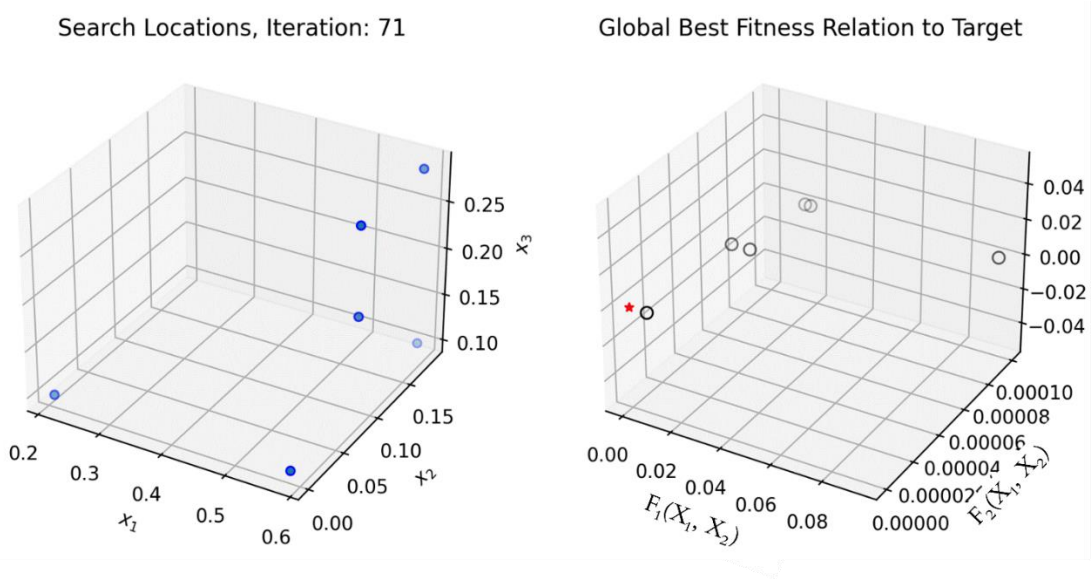


Figure 34 *The current search locations of six particles generated by MultiGLODS and the global best fitness record. On the right, the current and previous search locations. On the left, the red star is the global minima target. The black circle on the far right of the Global Best Fitness plot is the original evaluation, while the circle around the red star is the best evaluation.*

The *Multiobjective Optimization Global and Local Optimization using Direct Search* (MultiGLODS) algorithm is a derivative-free optimizer generalized for calculating the Pareto front of multi-objective, multimodal optimization problems [73]. MultiGLODS builds from the original GLODs algorithm proposed by the same authors in [106]. The multi-objective algorithm alternates between initializing new searches in the feasible objective space, using a 'multistart' strategy for selecting candidates likely to perform well, and balancing exploration with direct search. To reduce computation needs across valid, active points, it only compares points close to each other, rather than comparing one point with all other points active in the search.

The original MultiGLODS search was written in MATLAB by Dr. Ana Luise Custódio and J. F. A. Madeira at the Nova School of Science and Technology and at ISEL and IDMEC-IST, Lisbon respectively. The Python version used in the AntennaCAT optimizer suite was translated by Jonathan Lundquist at VCU [100]. The Python translation de-embeds the objective function using a state machine-based design. Several changes were made to counters to record how many times the objective function was called for AntennaCAT's standard log format, and the returns for the

objective function call were updated to match the compatibility requirements described in section 4.4. The core algorithm remains true to the original.

The Python version, and licensing information, are available on GitHub at:

https://github.com/Jonathan4600/multi_glods_python

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The number of input variables
- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- Numerical target values
- A radius-based convergence tolerance
- Maximum number of iterations
- Boundary type selection
- A beta parameter (BP) for step size tuning
- A gamma parameter (GP) for step size tuning
- A search frequency parameter (SF)

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included.

Optimizer Step and State Machine

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, but the state machine will not evaluate the objective function every iteration. MultiGLODS alternates between several modes of operation and may require several ‘steps’ before it is in a position to evaluate the objective function.

4.12 Bayesian Optimizer with Surrogate Model Kernel

The Bayesian Optimizer included in the AntennaCAT optimizer suite is compatible with a series of surrogate model kernels. Bayesian search, or Bayesian optimization, uses probabilistic models to optimize functions that may be computationally intensive or time expensive to evaluate. This optimizer iteratively updates a Bayesian model of the objective function based on sampled evaluations of the objective function. As part of the AntennaCAT suite it is compatible with single and multiple objective functions with a variety of inputs. Only a limited number of input/output dimension combinations have graphical outputs, such as the ones in Figure 35 and 36 due to dimensionality restrictions.

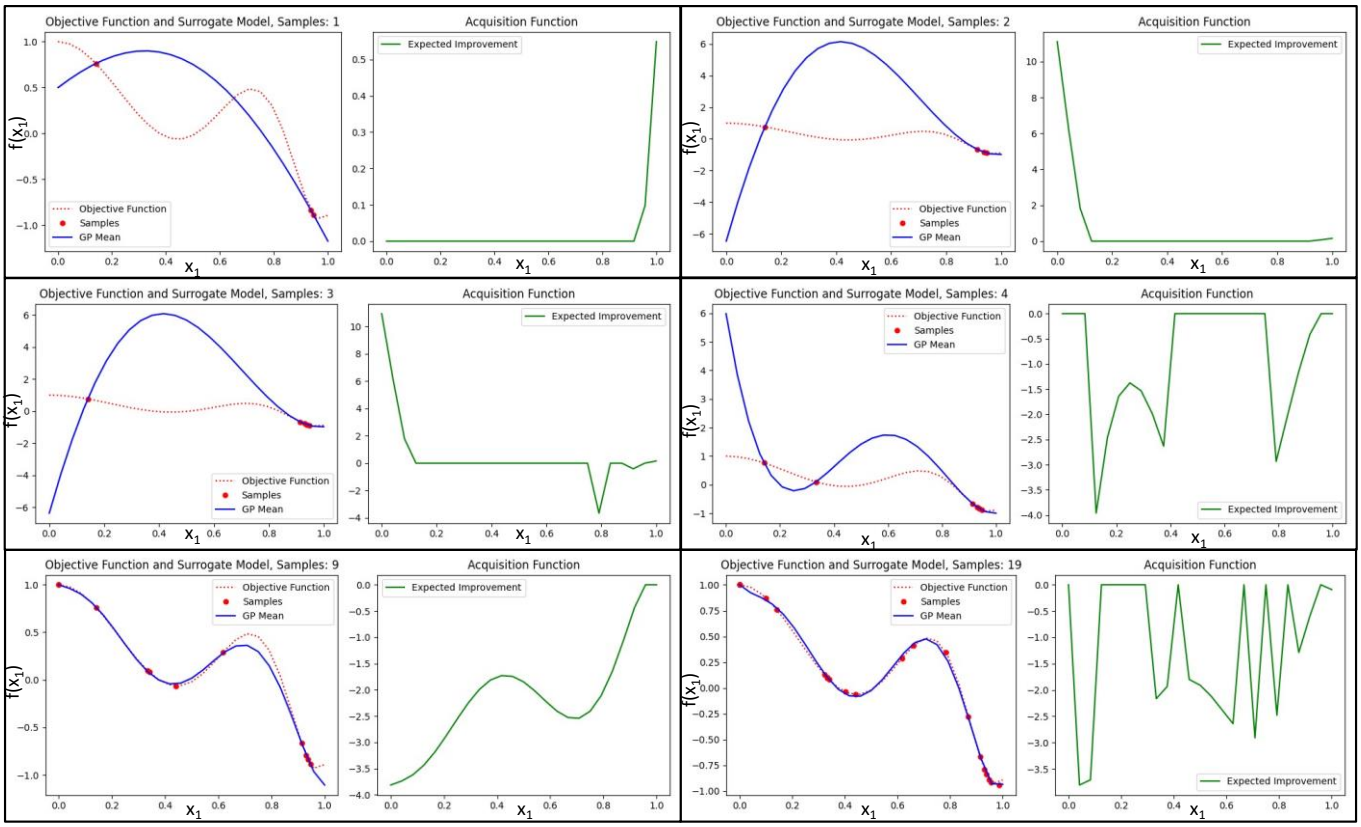


Figure 35 *The evolution of the surrogate model at 1, 2, 3, 4, 9, and 19 samples taken during the Bayesian optimization process on a single-input single-output objective function. On the left of each pair is the objective function ground truth represented by a dotted red line. The surrogate model (using a Gaussian Process Mean) prediction is drawn in blue. Sampled points are red dots. On the right, in green, is the plotted expected improvement of the acquisition function.*

In Figure 35 a single-input single-output equation has been used as an example. Each pair of graphs shows the progress of the optimizer as more samples are taken of the objective function. On the right-side plots, the objective function ground truth is shown with a dotted red line. The approximated surrogate model, in these examples using the Gaussian Process kernel, are shown in blue. Sampled points are shown as red dots. On the left-hand side plots, the Acquisition Function with the green Expected Improvement measure estimates which areas have the most potential for improvement based on the optimizer's current knowledge of the space. This isolation of areas of interest is more apparent in Figure 36, where the center image of each set of three plots is the expected improvement compared to the current samples shown in 2D.

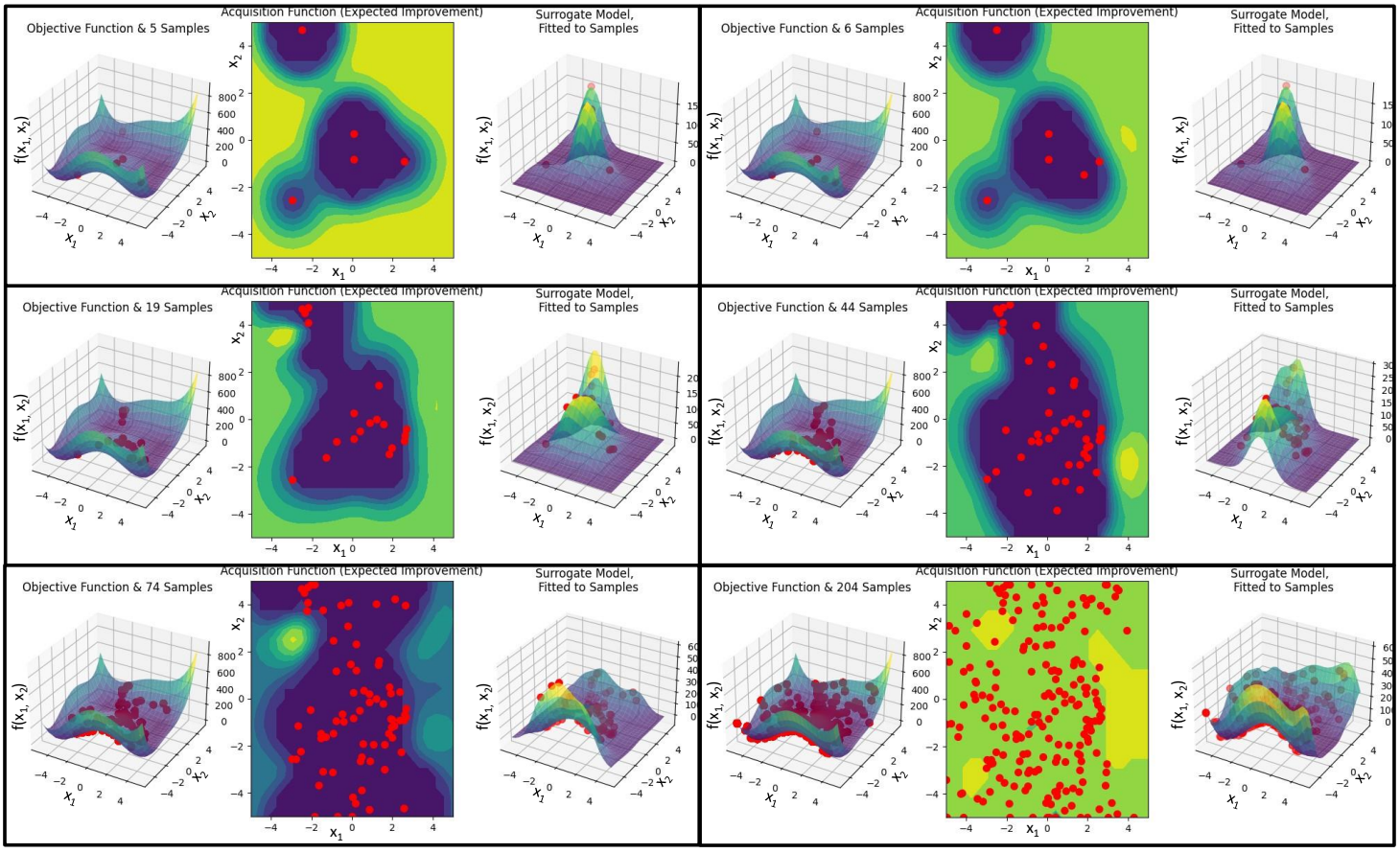


Figure 36 *The evolution of the Gaussian Process surrogate model at 5, 6, 19, 44, 74, and 204 samples taken during the Bayesian optimization process on a two-input single-output objective function. In each set of plots, the left plot is the objective function ground truth. The middle plot is the acquisition function, with the plotted expected improvement in 2D space. The right plot is the current surrogate model prediction of the objective function. In all three plots, the red dots indicate the samples taken from the original model.*

Figure 36 shows the development of the Bayesian Optimization process using the Gaussian Process surrogate model on the two-input single-output Himmelblau's function. This function was chosen to demonstrate the evolution of the surrogate model prediction against the image of the ground truth on the left of each set of plots. The error tolerance was lowered to 10e-20 to prolong the search for this example.

Surrogate models are often used in optimization as a proxy for the true objective function, which may be costly or impractical to evaluate directly. This model approximates the objective function's behavior through a simpler and computationally efficient framework, such as the Gaussian Process kernel used in this example. The surrogate model is refined iteratively based on the evaluations of the actual objective function, thereby enhancing its accuracy over time, which is demonstrated in both Figure 35 and Figure 36. This approach enables optimization algorithms to make informed decisions about subsequent exploration in the search space, effectively balancing the exploitation of known favorable regions with the exploration of potentially superior ones.

This optimization approach utilizes the surrogate modeling library discussed in the following subsections.

It is available at: https://github.com/LC-Linkous/bayesian_optimization_Python

Initialization:

The optimizer is initialized and controlled fully from an outside class. It is initialized with the following variables:

- The lower bounds of the problem (numeric constraints)
- The upper bounds of the problem (numeric constraints)
- The number of output (target) values
- Numerical target values
- Error tolerance, or acceptable deviation from target
- Maximum number of iterations
- Initial number of sample points
- xi, which controls the exploration of the optimizer
- the number of restarts when generating new points

During initialization, the objective function reference (pass by object) and the constraint function (pass by object) are also included. The class controlling the Bayesian optimizer also initializes the surrogate model object.

The state machine is simple, and steps forward every time the step function is called. The objective function is called once per iteration of the controlling class, and the objective function is executed every time it is called. When the Bayesian Optimizer is predicting the next point using the surrogate model, it calls a function in the parent control class, which then calls the surrogate model object. This separation is necessary to preserve the modularity of the optimizer and surrogate model libraries.

Optimizer Step:

When the optimizer step() function is called, the Bayesian optimizer executes the following process:

- Print a summary of the iteration values if suppress_output is False.
- For any active particles:
    1. Check if the current location is a global local
    2. Propose a new location through the surrogate model
        2.1 Generate a list of positions, for n number of restarts
            2.1.1 Calculate the expected improvement
        2.2 Use L2 norm to select the position with the best fitness

The following subsections describe the potential surrogate models included in the Bayesian Optimizer.

Radial Basis Function Network

A Radial Basis Function Network (RBFN) is a type of artificial neural network that employs radial basis functions as activation functions. It comprises three distinct layer types: an input layer, a hidden layer utilizing a non-linear RBF activation function (typically Gaussian), and a linear output layer. The implementation in the AntennaCAT surrogate model library adopts a straightforward approach, utilizing the numpy library and basic matrix mathematics rather than a machine learning-specific library. RBFNs are widely used for function approximation, time-series prediction, and classification tasks. They are highly regarded for their simplicity, ease of training, and capability to model complex, non-linear relationships with a smaller number of neurons compared to other neural network architectures, resulting in reduced matrix dimensions.

Gaussian Process

A Gaussian Process (GP) [107, 108] is a sophisticated probabilistic model widely employed in machine learning and optimization. It defines a distribution over functions, assigning a Gaussian distribution to each point in the function's domain. GPs are characterized by their mean function (often assumed to be zero) and covariance function (kernel), which dictates the relationships between different points. They serve as flexible and powerful tools for regression and uncertainty quantification, enabling predictions of both function values and the associated uncertainty. Due to their non-parametric nature, GPs can adapt to data of varying complexity, making them particularly useful for tasks that require a high degree of accuracy and reliability in predictive modeling.

Kriging

Kriging, similar to Gaussian Processes, is a technique for data interpolation and approximation. It leverages spatial correlation among data points to predict values at unsampled locations. By using a linear combination of data points with weights derived from spatial covariance functions (kriging models), Kriging estimates values and quantifies prediction uncertainty. This technique is particularly advantageous in fields such as spatial statistics, where it provides precise predictions and reliable uncertainty estimates based on the spatial relationships of known data points. Kriging is a specialized form of regression tailored for spatial datasets, emphasizing the spatial autocorrelation structure to enhance prediction accuracy in geospatial applications and beyond.

Polynomial Regression

Polynomial regression is a form of regression analysis that models the relationship between the independent variable x and the dependent variable y as an n-th degree polynomial function. Unlike linear regression, which assumes a linear relationship, polynomial regression can capture complex, non-linear relationships between variables. This approach enhances the model's ability to fit data with varying degrees of curvature, making it particularly useful for analyzing data with intricate patterns and trends that are not well-represented by linear models.

Polynomial Chaos Expansion

Polynomial Chaos Expansion (PCE) [109] is a sophisticated method employed in uncertainty quantification and sensitivity analysis. It represents the output of a stochastic model as a series expansion involving orthogonal polynomials, such as Hermite, Legendre, or other families,

depending on the underlying probability distribution. Each polynomial term corresponds to a different order of stochastic variables, capturing the variability and uncertainty inherent in the model's parameters or inputs. PCE facilitates the efficient computation of statistical moments, including mean and variance, and enables the quantification of how uncertainties in input parameters propagate through the model, thereby influencing output variability.

K-Nearest Neighbors Regression

Within an optimizer, the K-Nearest Neighbors (KNN) model predicts the objective function's value at a new point by leveraging the values of its k nearest neighbors in the sampled data. It estimates the function value using the distances and weights of these neighbors, thereby directing the optimizer to explore promising regions of the search space. KNN is valued for its simplicity and non-parametric nature, which provides flexibility across diverse optimization problems. However, its performance may be limited with high-dimensional data, or data with many local minima where the neighboring points can cluster.

Decision Tree Regression

Decision Tree Regression is a predictive modeling technique designed for continuous target variables. It operates by recursively partitioning the data into subsets according to the feature that minimizes the mean squared error (MSE) at each split. Each internal node corresponds to a feature, while each leaf node represents a predicted value, typically the mathematical mean of the target values within that region. This method effectively captures non-linear relationships and offers straightforward interpretability. However, it is prone to overfitting the sample data. To improve performance and enhance generalization, techniques such as pruning or the use of ensemble methods (e.g., Random Forests) are commonly employed.

CHAPTER 5
Machine Learning Assisted Optimization Data Collection and Training

Optimizers play a crucial role in machine learning by iteratively adjusting model parameters to minimize a specified objective function, thereby enabling models to discern patterns and make precise predictions. They are integral to achieving efficient convergence toward optimal solutions, which enhances model performance, generalization, and computational efficiency. The optimization of the hyperparameters of these optimizers is particularly significant, as different optimization algorithms exhibit varying efficacy in identifying the optimal set of hyperparameters for specific machine learning models and datasets. The optimizer objective function suite is designed for use with AntennaCAT's internal optimizer suite to offer a comprehensive benchmarking framework for assessing the performance of various optimizers in hyperparameter tuning tasks [110, 111]. Additionally, it assists in collecting data on optimizer performance to suggest hyperparameter values for the AntennaCAT optimizers based on knowledge about the number of controllable parameters, and the target metrics of which to optimize. The full objective function suite has over 120 function variations, with over 60 unique functions, provided for the automated data collection process. These objective functions have largely been pulled from literature so that optimizer performance can be compared across publications [73, 112-118]. In cases where existing, popular objective functions with desired dimensionality, such as with 10 inputs and 3 outputs, functions were created and included in the objective function suite. Which functions come from existing literature, and which were created for this work are noted extensively in the README.md on GitHub.

The objective function suite is available at:
https://github.com/LC-Linkous/objective_function_suite

This chapter contains a summary of the data collection process, the objective functions used to collect data and the criteria for their selection. The Sweep, Bayesian, Sand Cat Swarm, and MultiGLODS optimizers were ultimately excluded from the parameter sweep data collection.

These three optimizers were sensitive to parameter variation and problem space during experimentation, and so convergence data on them was sparse due to the automation process. A brief analysis is provided in Chapter 6 with the results of the training.

In the following objective function sections, the subsets chosen for data collection are grouped based on the dimensionality of their input and output. The groups are based on the organization of AntennaCAT's Hyperparameter Prediction Network library, which is described in the next section.

5.1. AntennaCAT MLAO Design Structure

The AntennaCAT software suite is designed to be modular and expandable. This is especially true for the integrated optimization suite and the hyperparameter tuning suggestions trained using the objective function library. The 'Help Me Choose' menu that accompanies most of the optimizers on the Optimizer page is a straightforward interface designed to provide hyperparameters predicted by a neural network library nested in a controller program. This approach has several notable benefits. Firstly, as optimizers are added to AntennaCAT, new machine learning models can be trained and added to the library without affecting existing models. Likewise, as AntennaCAT features are expanded, such as adding more target metrics to the optimizer GUI, the machine learning model library can be expanded to match. This is important because while there are a set, relatively finite, number of target metrics that can be optimized while designing an antenna, there are no such restrictions on the number of parameters in a design that users can load themselves.

Figure 37 shows a high-level relation between the known values provided from the GUI, the Hyperparameter Prediction Network (the 'Help Me Choose' network), and the predicted hyperparameter values. The known numerical values, the number of controllable parameters and the number of target values, are not explicitly user input to reduce the margin of error during operation. Instead, the number of controllable parameters is set when a user selects which controllable, detected parameters on the Optimizer page are allowed to be changed by AntennaCAT, and the number of targets are set when the optimizer is selected via the 'Select Optimizer' button. Navigating tabs will not trigger this process.

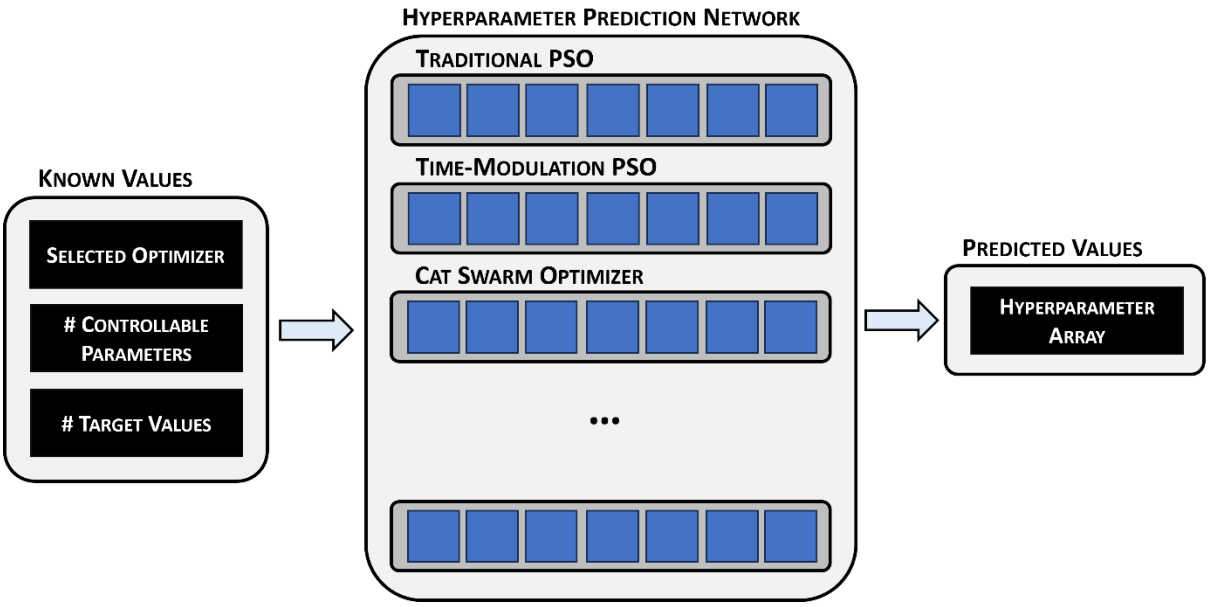


Figure 37 *A high level overview of how the 'Help me Choose' Optimization Page option where known values from the GUI are used to select which machine learning model will be used to predict the hyperparameters.*

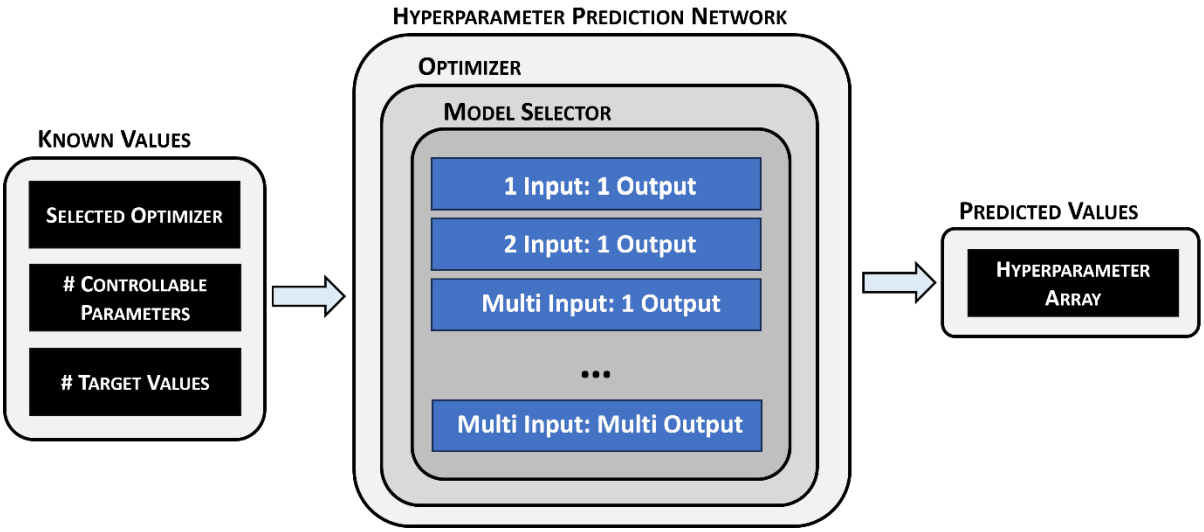


Figure 38 *The generalized process where the specific optimizer has been selected. The number of controllable parameters and number of target values are first used as inputs to the model selector to decide which machine learning model will be used to predict hyperparameters, and then as inputs to predict the hyperparameters.*

Related to Figure 37, Figure 38 shows a generalized process for using the known parameters to predict the optimizer hyper parameters. The selected optimizer when a 'Help Me Choose' option is selected will be used to select which optimizer on the Hyperparameter Prediction Network library will be used. When an optimizer is narrowed down, then the number of controllable parameters and number of target values are passed through as inputs to the model selector to decide

which machine learning models should be utilized with the known configuration. After the model has been selected, the number of controllable parameters and the number of target values are used as inputs to predict the hyperparameters, which are returned in an array back to the optimizer to be parsed into text. Neural networks were used due to their versatility, but a variety of machine learning models could be adapted to this problem. Due to how the library is organized, machine learning models for each optimizer, or even for each grouping of functions, do not need to be the same type.

The following sections discuss the data collection methodology, the objective functions used to collect the data, and training methodology for the machine learning models.

5.2 Data Collection Methodology

Of the eleven optimizers included in AntennaCAT's internal optimizer library, seven were used for data collection. Table 2 lists the total number of parameter combinations generated for each of the seven optimizers used. The parameter value ranges chosen were based on several criteria: are they practical for optimization using simulation instead of an objective function; are weight parameters practical for the size of the feasible decision space; and are any parameters known either in literature or experimentally to cause drastic variation in the optimizer performance.

Table 2 The total number of parameter combinations for the seven optimizers used with the objective function data collection.

Optimizer	**# Parameter Combinations**	**Optimizer**	**# Parameter Combinations**
Traditional PSO	6174	Quantum Insp. Cat Swarm	2560
Time modulated PSO	12348	Chicken Swarm	1334
Quantum Insp. PSO	2880	Quantum. Insp. Chicken Swarm	5376
Cat Swarm	2560		

For all swarm-based optimizers, weight values under 0.2 were not used, nor were increments smaller than 0.05. In the case of criteria 1, this led to the Chicken Swarm optimizer having a maximum of 3 flocks within a single swarm instance and limiting the total size of the swarm while attempting to balance the particles between the four categories. Likewise, the number of copies created by Cat Swarm Optimization were limited due to the cost of simulation needed to evaluate potential new location values on top of the total size of the swarm.

The Sweep, Bayesian, Sand Cat Swarm, and MultiGLODS optimizers were excluded due to their high sensitivity to the parameter selection and problem space, and how this was handled in the automated data collection process. The Sweep optimizer would converge on the first data sample on many objective functions due to the lower bounds of the problem being a target on the pareto front. This caused data collection for this optimizer to be sporadic, and due to the small feasible decision space, low resolution search grids always found an acceptable solution if the resolution value was set low enough. Convergence was sporadic for the Bayesian optimizer and Cat Swarm Optimizer on the selected objective function sets during the parameter sweep. Due to their limited input parameters, it was decided to recommend experimentally found values. The MultiGLODS algorithm was noted by the authors to be sensitive to parameter changes, and this was confirmed via experimentation. The authors' suggested parameters are used as a default.

Table three shows the default values used for the automated data collection scripts. In cases where other error tolerance variables were used, this was noted in Chapter 4 with the respective optimizer.

Table 3 *Constant values used in the automation process for the optimizer parameter sweep.*

Iterations Per Combination	Error Tolerance	Maximum Iterations	Boundary Selection
100	10e-25	10000	Random (1)

For all parameter configurations run on an objective function, 100 iterations were performed. When boundary conditions were enforced, a Random boundary was used as this was experimentally found to be the most stable options across optimizers and objective functions. The error tolerance was set to 10e-25, which ensured that in most cases the optimizer ran until the maximum iteration completion condition was triggered. On rare occasions, large enough swarms spawned particles close to global minima and the optimum solution was returned.

During the 100 iterations of each parameter combination, a log file recorded the following: parameter configuration index, trial number, objective function ID, number of input variables, number of output variables, the best fitness evaluation when the completion condition was triggered, the iteration that the best fitness was found, the optimized solution, the optimized outputs, the targe values, the objective function lower bounds, and the objective function upper bounds. Also logged were the optimizer specific parameters.

5.3 Objective Functions: Single Input, Single Output

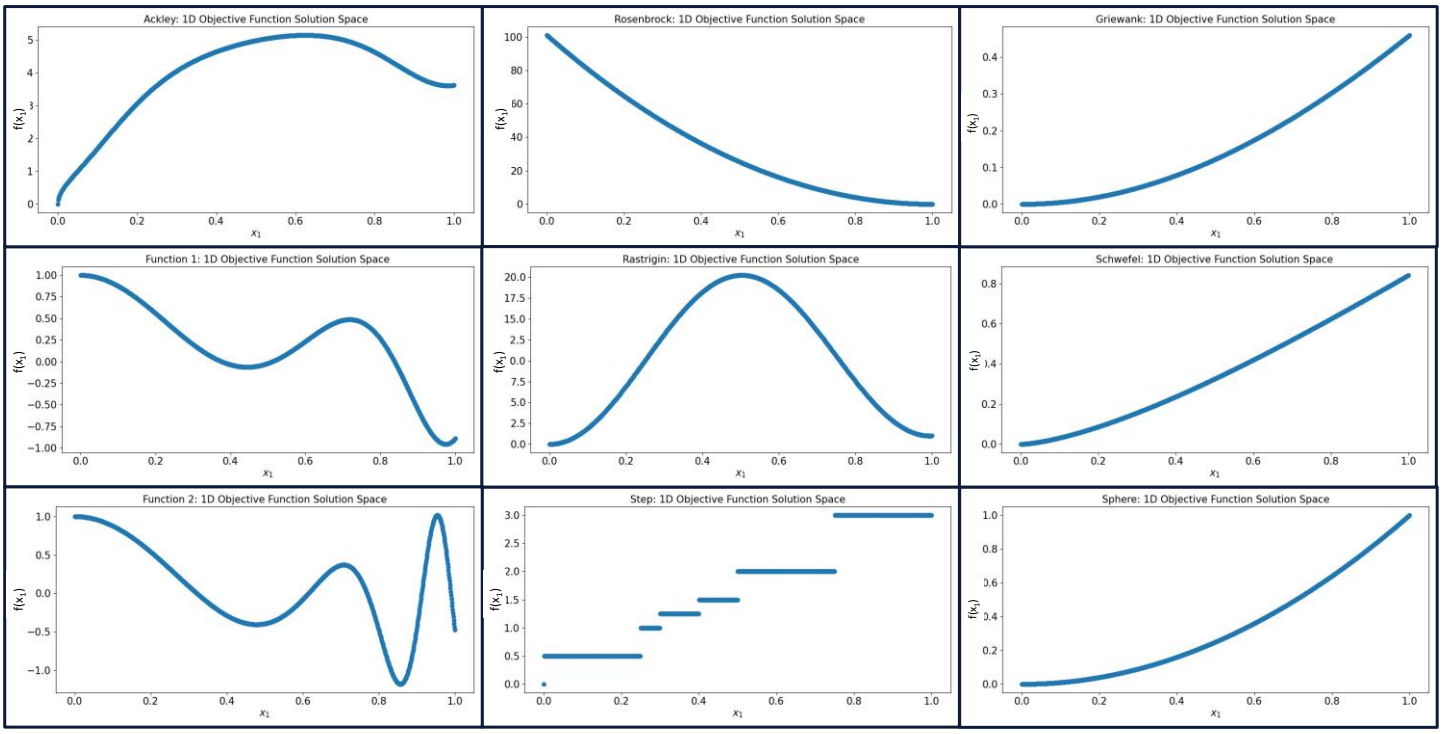


Figure 39 *The nine functions used in the single-input, single-output objective function subset for data collection. Top: eq. 19, eq. 20, eq. 21. Middle: eq. 22, eq. 23, eq. 24. Bottom: eq. 25, eq. 26, eq.27.*

There are nine functions used in the single-input, single-output objective function subset. Figure 39 shows the bounded evaluation of these functions where x_1 is evaluated on the interval [0,1]. Below are the equations for these objective functions.

Ackley Function:

$$f(x) = -20.0 \cdot exp\left(-0.2\sqrt{|x|}\right) - exp\big(cos(2\pi x)\big) + 20 + e \qquad (eq.19)$$

Rosenbrock Function:

$$f(x) = (1-x)^2 + 100(x-1)^2 \qquad (eq.20)$$

Griewank Function:

$$f(x) = \frac{x^2}{4000} - \cos\left(\frac{x}{\sqrt{1}}\right) + 1 \qquad (eq.21)$$

AntennaCAT Function 1:

$$f(x) = \sin(5x^3) + \cos(5x)\,(1 - \tanh(x^2)) \qquad (eq.22)$$

Rastrigin Function:

$$f(x) = x^2 - 10\cos(2\pi x) + 10 \qquad (eq.23)$$

Schwefel Function:

$$f(x) = x\sin\left(\sqrt{|x|}\right) \qquad (eq.24)$$

AntennaCAT Function 2:

$$f(x) = \sin(10x^5) + \cos(5x)\,(1 - \tanh(x^3)) \qquad (eq.25)$$

AntennaCAT Step, 1-Dimensional Function:

$$f(x) = \begin{cases} 1, & x < 0 \\ 0, & x = 0 \\ 0.5, & 0 < x \leq 0.25 \\ 1, & 0.25 < x \leq 0.3 \\ 1.25, & 0.3 < x \leq 0.4 \\ 1.5, & 0.4 < x \leq 0.5 \\ 2, & 0.5 < x \leq 0.75 \\ 3, & x > 0.75 \end{cases} \qquad (eq.26)$$

Sphere Function:

$$f(x) = x^2 \qquad (eq.27)$$

5.4 Objective Functions: Two Input, One Output

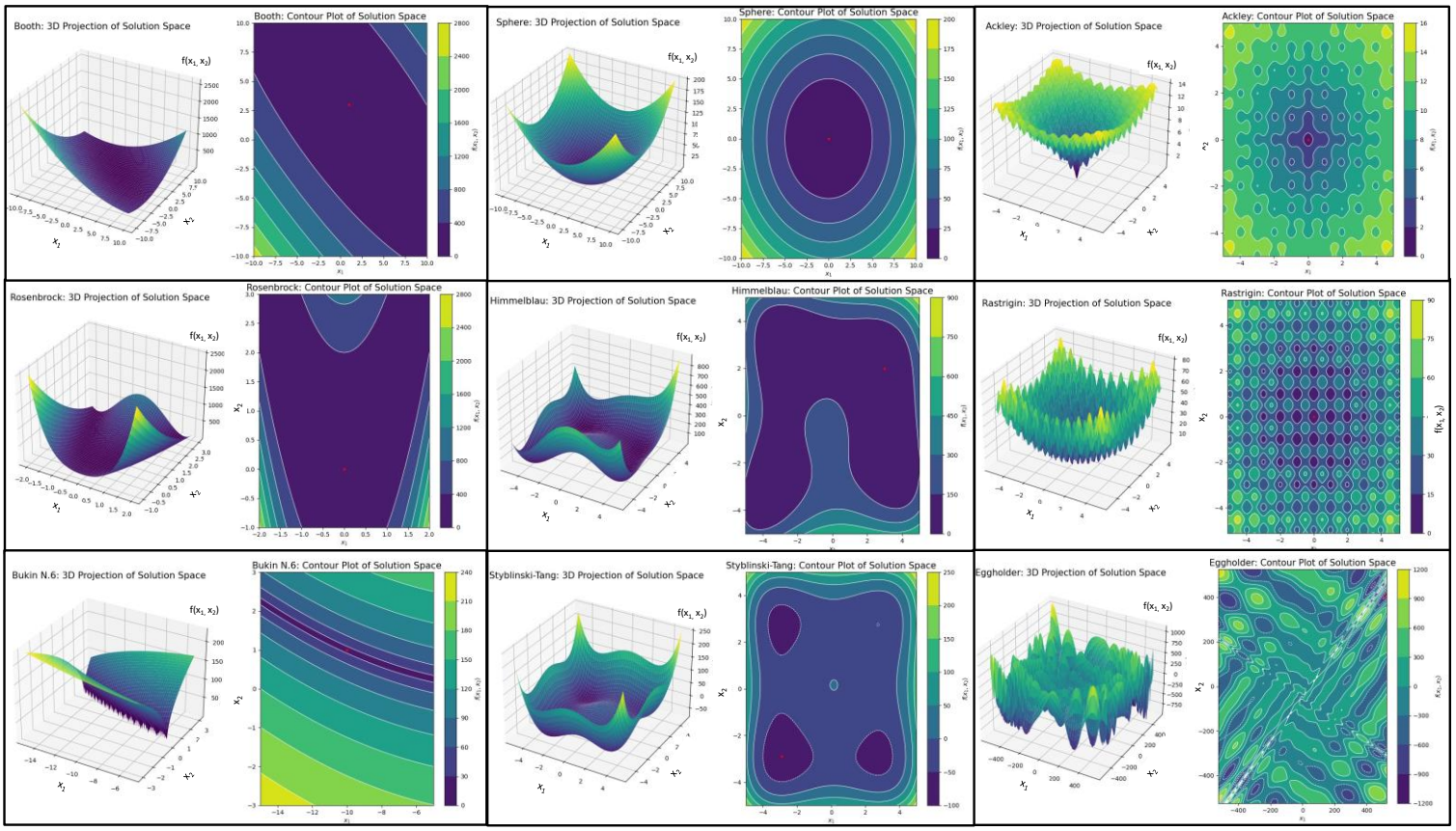


Figure 40 *The nine functions used in the two-input, single-output objective function subset for data collection. Top: eq. 28, eq. 29, eq. 30. Middle: eq. 31, eq. 32, eq. 33. Bottom: eq. 34, eq. 35, eq. 36. For each function, the left plot shows a 3D projection of the solution space, and the plot on the right shows a top view of the solution space, with the global minima marked in red.*

There are nine functions used in the two-input, single-output objective function subset. For each equation, Figure 40 shows the 3D projection of the solution space on the left and the top-view contour plot with the global minimum (or a global minimum candidate) marked in red. These equations are bound based on the descriptions provided by their respective authors. For more information, view the README.md on the objective function library repository on GitHub [49]. Below are the equations for these objective functions.

Booth Function:

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \qquad (eq.\,28)$$

Sphere 2-Dimensional Function:

$$f(x_1, x_2) = {x_1}^2 + {x_2}^2 \qquad (eq.\,29)$$

Ackley 2-Dimensional Function:

$$f(x_1, x_2) = -a \exp\left(-b\sqrt{\frac{{x_1}^2 + {x_2}^2}{2}}\right) - \exp\left(\frac{\cos(cx_1) + \cos(cx_2)}{2}\right) + a + \exp(1) \qquad (eq.30)$$

Rosenbrock Function:

$$f(x_1, x_2) = (a - x_1)^2 + b({x_2} - {x_1}^2)^2 \qquad (eq.31)$$

Himmelblau's Function:

$$f(x_1, x_2) = ({x_1}^2 + x_2 - 11)^2 + (x_1 + {x_2}^2 - 7)^2 \qquad (eq.32)$$

Rastrigin Function:

$$f(x_1, x_2) = 20 + ({x_1}^2 - 10\cos(2\pi x_1)) + ({x_2}^2 - 10\cos(2\pi x_2)) \qquad (eq.33)$$

Bukin N. 6 Function:

$$f(x_1, x_2) = 100\sqrt{|x_2 - 0.01{x_1}^2|} + 0.01|x_1 + 10| \qquad (eq.34)$$

Styblinski-Tang Function:

$$f(x_1, x_2) = 0.5[({x_1}^4 - 16{x_1}^2 + 5x_1) + ({x_2}^4 - 16{x_2}^2 + 5x_2)] \qquad (eq.35)$$

Eggholder Function:

$$f(x_1, x_2) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right) \qquad (eq.36)$$

5.5 Objective Functions: Other Multiple Input, One Output

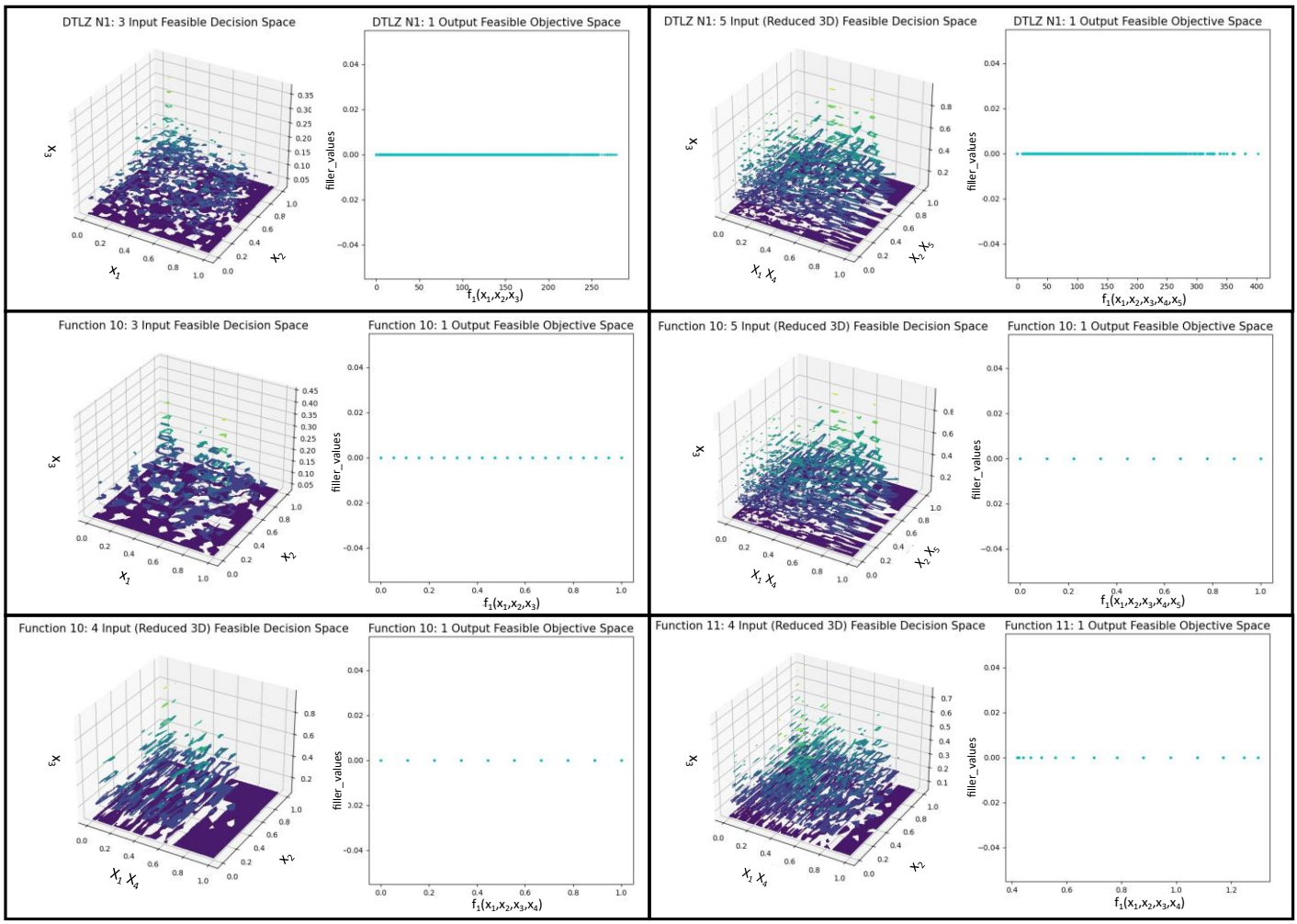


Figure 41 *The three functions, with multiple variations, used for the multi-input one-output objective function subset for data collection. Top row: DTLZ N1 (eqs. 37-39). Middle Row: AntennaCAT Function 10 (eqs. 40-41) Bottom Row: AntennaCAT Function 10 (eqs. 40-41) and AntennaCAT Function 11 (eq .42). For each function, the left plot shows a 3D representation of the feasible decision space, some of which have been reduced to 3D space. The right plot shows the feasible objective space for the function.*

There are three functions used in the multi-input, single-output objective function subset. Each function has several variations for input dimensionality. This is an unusual configuration that is possible using AntennaCAT, but is likely to overdefine the design problem. For each function, Figure 41 shows the feasible decision space on the left, and the feasible objective space on the right. These equations are bound based on the descriptions provided by their respective authors. For more information, view the README.md on the objective function library repository on GitHub [49]. Below are the equations for these objective functions.

DTLZ N1:

$$f_i = (1+g)\prod_{j=1}^{M-i}\cos\left(\frac{\pi x_j}{2}\right) \quad \text{for } i = 1,\dots,M-1 \qquad (eq.37)$$

$$f_M = (1+g)\prod_{j=1}^{M-1}\cos\left(\frac{\pi x_j}{2}\right) \qquad (eq.38)$$

$$g = \sum_{j=M-1}^{N-1}\left(x_j - 0.5\right)^2 \qquad (eq.39)$$

AntennaCAT Function 10:

$$f_i = \sin\left(\prod_{j=0}^{M-i} x_j\right) + co\,s(x_0^3) \quad \text{for } i = 0,1,\dots,M-1 \qquad (eq.40)$$

$$f_M = 1 - x_0 \qquad (eq.41)$$

AntennaCAT Function 11:

$$f_i = 0.5\cdot\sin\left(\prod_{j=0}^{M} X_j + X_0^3\right) + \sin(X_0^2) \quad \text{for } i = 0,1,\dots,M \qquad (eq.42)$$

5.6 Objective Functions: One Input, Two Output

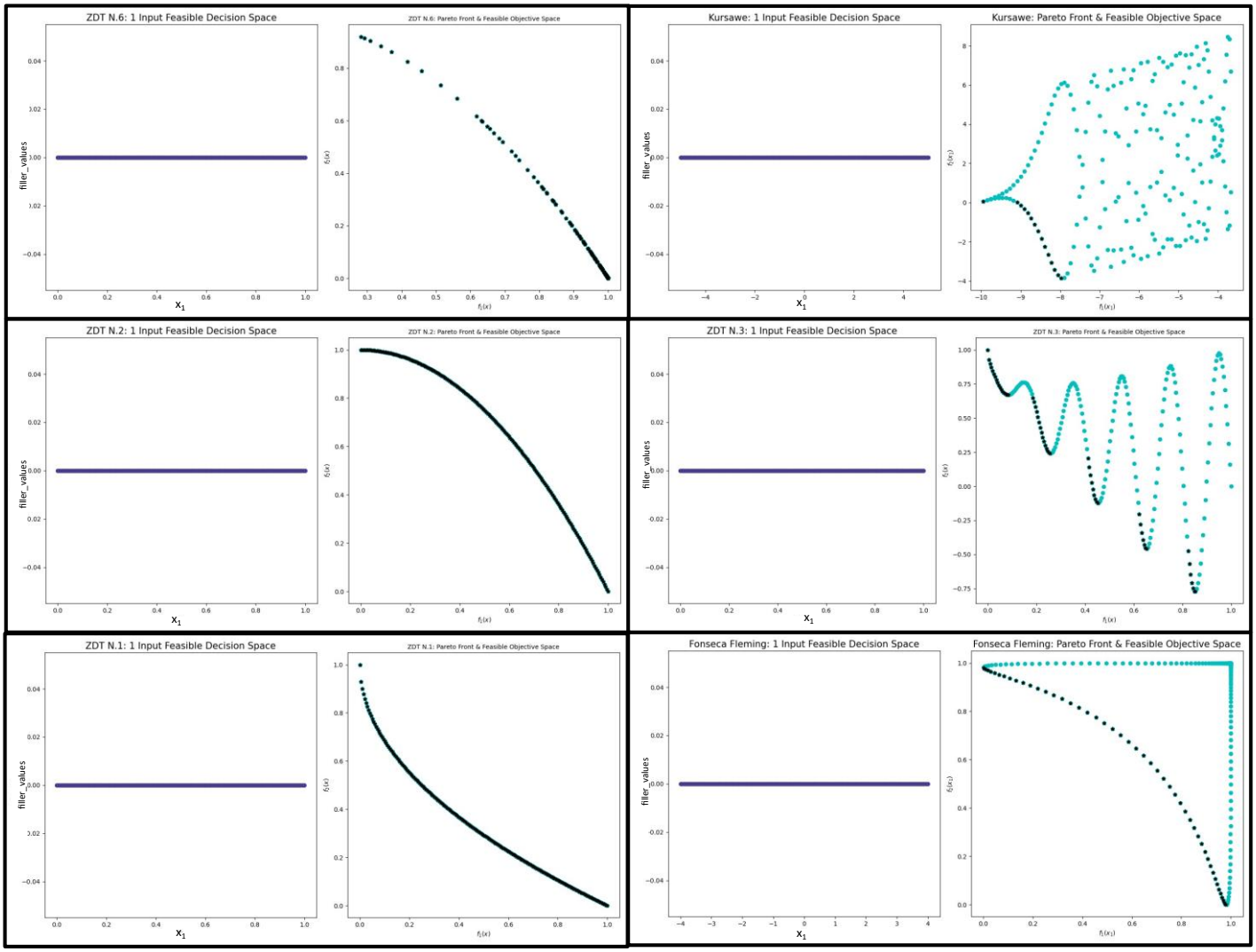


Figure 42 *The six functions used in the single-input, two-output objective function subset for data collection. Top: ZDT N.6 (eqs. 43-45), Kursawe (eqs. 46-47). Middle: ZDT N.2. (eqs. 48-50), ZDT N.3 (eqs. 51-53), Bottom: ZDT N.1 (eqs. 54 -56), Fonseca Fleming (eqs. 57-58). For each function, the left plot shows a 2D projection of the Feasible Decision Space where the Y-axis is comprised of filler values for graphical purposes, and the plot on the right shows the Feasible Objective Space with the Pareto front marked in black.*

There are six functions used in the two-input, single-output objective function subset. For each equation, Figure 42 has a set of two plots per equation where the plot on the left is the Feasible Decision Space with the1-dimesional input plotted on the x-axis, and filler values on the y-axis for visualization purposes. The plot on the right of each set is the Feasible Objective Space with the Pareto front marked in black. These equations are bound based on the descriptions provided by their respective authors. For more information, view the README.md on the objective function library repository on GitHub [49]. Below are the equations for these objective functions.

ZDT N.6:

$$f_0 = 1 - \exp\left(-4\frac{x_0}{(1+g)^2}\right) \quad (eq.43)$$

$$f_1 = 1 - \frac{f_1^2}{1+g} \quad (eq.44)$$

$$g = 1 + 9\left(\frac{\sum_{i=2}^{n} x_i}{n-1}\right)^{0.25} \quad (eq.45)$$

Kursawe:

$$f_0 = \sum_{i=0}^{N-2}\left[-10\exp\left(-0.2\sqrt{X_i^2 + X_{i+1}^2}\right)\right] \quad (eq.46)$$

$$f_1 = \sum_{i=0}^{N-1}\left[|X_i|^{0.8} + 5\sin\left(X_i^3\right)\right] \quad (eq.47)$$

ZDT N.2:

$$f_0 = x_0 \quad (eq.48)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right)(1 + \sin(10\pi f_0)) \quad (eq.49)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.50)$$

ZDT N.3:

$$\mathrm{f_0 = x_0} \quad (eq.51)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right)(1 + \sin(10\pi f_0)) \quad (eq.52)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.53)$$

ZDT N.1:

$$f_0 = x_0 \quad (eq.54)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right) \quad (eq.55)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.56)$$

Fonseca Fleming:

$$f_0 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i - \frac{1}{\sqrt{n}}\right)^2\right) \quad (eq.57)$$

$$f_1 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i + \frac{1}{\sqrt{n}}\right)^2\right) \quad (eq.58)$$

5.7 Objective Functions: Two Input, Two Output

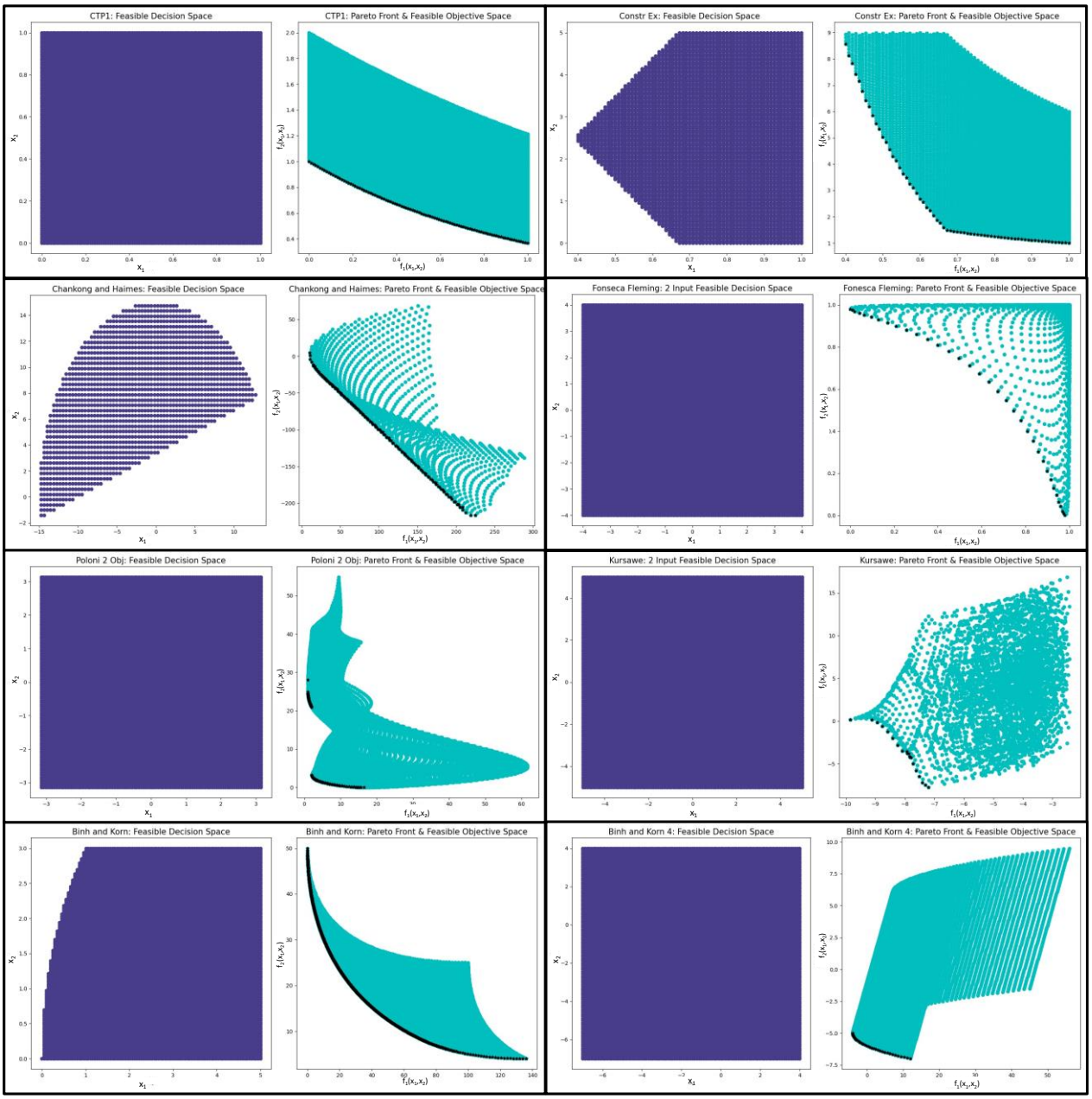


Figure 43 *The eight functions used in the two-input, two-output objective function subset for data collection. First Row: CTP1 (eqs. 59-60), Constr Ex (eqs. 61-62). Second Row: Chankong and Haimes (eqs. 63-64), Fonseca Fleming (eqs. 65-66), Third Row: Poloni 2-Objective (eqs.67-72), Kursawe (eqs.73-74). Forth Row: Binh and Korn (eqs.75-76), Binh and Korn Test Function 4 (eqs.77-78) For each function, the left plot shows a 2D projection of the Feasible Decision Space, and the plot on the right shows the Feasible Objective Space with the Pareto front marked in black.*

There are six functions used in the two-input, single-output objective function subset. For each equation, Figure 43 has a set of two plots where the plot on the left is the Feasible Decision Space.

The plot on the right of each set is the Feasible Objective Space with the Pareto front marked in black. These equations are bound based on the descriptions provided by their respective authors. For more information, view the README.md on the objective function library repository on GitHub [49]. Below are the equations for these objective functions.

CTP1:

$$f_0 = x_0 \tag{eq.59}$$

$$f_1 = (1 + x_1)\exp\left(-\frac{x_0}{1 + x_1}\right) \tag{eq.60}$$

Constr Ex:

$$f_0 = x_0 \tag{eq.61}$$

$$f_1 = \frac{1 + x_2}{x_1} \tag{eq.62}$$

Chankong and Haimes:

$$f_0 = 2 + (x_0 - 2)^2 + (x_1 - 1)^2 \tag{eq.63}$$

$$f_1 = 9x_0 - (x_1 - 1)^2 \tag{eq.64}$$

Fonseca Fleming:

$$f_0 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i - \frac{1}{\sqrt{n}}\right)^2\right) \tag{eq.65}$$

$$f_1 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i + \frac{1}{\sqrt{n}}\right)^2\right) \tag{eq.66}$$

Poloni Two-Objective Function:

$$A_1 = 0.5\sin(1) - 2\cos(1) + \sin(2) - 1.5\cos(2) \tag{eq.67}$$

$$A_2 = 1.5\sin(1) - \cos(1) + 2\sin(2) - 0.5\cos(2) \tag{eq.68}$$

$$B_1 = 0.5\sin(x_1) - 2\cos(x_1) + \sin(x_2) - 1.5\cos(x_2) \tag{eq.69}$$

$$B_2 = 1.5\sin(x_1) - \cos(x_1) + 2\sin(x_2) - 0.5\cos(x_2) \tag{eq.70}$$

$$f_0 = 1 + (A_1 - B_1)^2 + (A_2 - B_2)^2 \quad (eq.71)$$

$$f_1 = (x_1 + 3)^2 + (x_2 + 1)^2 \quad (eq.72)$$

Kursawe Function:

$$f_0 = \sum_{i=0}^{N-2} \left[-10 \exp\left(-0.2 \sqrt{X_i^2 + X_{i+1}^2} \right) \right] \quad (eq.73)$$

$$f_1 = \sum_{i=0}^{N-1} \left[|X_i|^{0.8} + 5 \sin\left(X_i^3\right) \right] \quad (eq.74)$$

Binh and Korn Function:

$$f_0 = 4x_0^2 + 4x_1^2 \quad (eq.75)$$

$$f_1 = (x_0 - 5)^2 + (x_1 - 5)^2 \quad (eq.76)$$

Binh and Korn Test Function 4:

$$\mathrm{f_0} = \mathrm{x_0^2} - \mathrm{x_1} \quad (\mathrm{eq.}\,77)$$

$$\mathrm{f_1} = -0.5\mathrm{x_0} - \mathrm{x_1} - 1 \quad (\mathrm{eq.}\,78)$$

5.8 Objective Functions: Multiple Input, Two Output

The following equations are used in the data set for multiple-input, two-output objective functions. This section was used to create a dataset for training one Hyperparameter Prediction Network model designed to take more than 3 inputs. It was trained with inputs of 3, 4, 5, 6, 10, and 20 dimensional inputs. All the functions except for the Viennet, and Osyczka and Kundu functions were used to collect data on all the input dimension variations. The Viennet function was used only for 3 input dimensions, while the Osyczka and Kundu function was used for 6 input dimensions.

Kursawe Function:

$$f_0 = \sum_{i=0}^{N-2} \left[-10 \exp\left(-0.2 \sqrt{X_i^2 + X_{i+1}^2} \right) \right] \quad (eq.79)$$

$$f_1 = \sum_{i=0}^{N-1} \left[|X_i|^{0.8} + 5 \sin\left(X_i^3\right) \right] \quad (eq.80)$$

Fonseca Fleming:

$$f_0 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i - \frac{1}{\sqrt{n}}\right)^2\right) \quad (eq.81)$$

$$f_1 = 1 - \exp\left(-\sum_{i=1}^{n}\left(X_i + \frac{1}{\sqrt{n}}\right)^2\right) \quad (eq.82)$$

Osyczka and Kundu:

$$f_0 = -25(x_0 - 2)^2 - (x_1 - 2)^2 - (x_2 - 1)^2 - (x_3 - 4)^2 - (x_4 - 1)^2 \quad (eq.83)$$

$$f_1 = \sum_{i=1}^{n} x_i^2 \quad (eq.84)$$

Viennet:

$$f_0 = 0.5(x_0^2 + x_1^2) + \sin(x_0^2 + x_1^2) \quad (eq.85)$$

$$f_1 = \frac{(3x_0 - 2x_1 + 4)^2}{8} + \frac{(x_0 - x_1 + 1)^2}{27} + 15 \quad (eq.86)$$

$$f_2 = \frac{1}{x_0^2 + x_1^2 + 1} - 1.1\exp\left(-(x_0^2 + x_1^2)\right) \quad (eq.87)$$

ZDT N.1:

$$f_0 = x_0 \quad (eq.88)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right) \quad (eq.89)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.90)$$

ZDT N.2:

$$f_0 = x_0 \quad (eq.91)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right)(1 + \sin(10\pi f_0)) \quad (eq.92)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.93)$$

ZDT N.3:

$$\mathrm{f_0 = x_0} \quad (eq.94)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_0}{g}}\right)(1 + \sin(10\pi f_0)) \quad (eq.95)$$

$$g = 1 + 9\frac{\sum_{i=2}^{n} x_i}{n-1} \quad (eq.96)$$

ZDT N.4:

$$f_0 = x_0 \quad (eq.97)$$

$$f_1 = g\left(1 - \sqrt{\frac{f_1}{g}}\right)\left(1 - \left(\frac{f_1}{g}\right)^2\right) \quad (eq.98)$$

$$g = 1 + 10(n-2) + \sum_{i=2}^{n}\left(x_i^2 - 10\cos(4\pi x_i)\right) \quad (eq.99)$$

ZDT N.6:

$$f_0 = 1 - \exp\left(-4\frac{x_0}{(1+g)^2}\right) \quad (eq.100)$$

$$f_1 = 1 - \frac{f_1^2}{1+g} \quad (eq.101)$$

$$g = 1 + 9\left(\frac{\sum_{i=2}^{n} x_i}{n-1}\right)^{0.25} \quad (eq.102)$$

5.9 Objective Functions: Multiple Input, Multiple Output

The following equations are used in the data set for multiple-input, multiple-output objective functions. This section was used to create a dataset for training one Hyperparameter Prediction Network model designed to take more than 3 inputs and have more than 2 outputs. The DTLZ function set is designed to take broad number of input output dimensionality. The set, however, performs better for optimization benchmarking when there is a minimum of four more inputs than outputs. Several custom functions were created to supplement this set. These functions are included below, and in the objective function library.

There are 5 target outputs available for AntennaCAT automated optimization, which naturally limits data collection requirements. In Table 4, the input and output dimensionality combinations used for the objective functions in this section, are sorted by their output dimensionality. Emphasis was placed on lower dimensionality inputs, as those have been more commonly used in experimentation with AntennaCAT.

Table 4 *The input and output dimensionality combinations for the multiple-input, multiple-output objective function dataset*

# Inputs	# Outputs	# Inputs	# Outputs	# Inputs	# Outputs
3	3	4	4	5	5
4	3	5	4	6	5
5	3	6	4	10	5
6	3	10	4	12	5
12	3	12	4	15	5
15	3	15	4	20	5
20	3	20	4	25	5

DTLZ N.1:

$$f_i = 0.5 \cdot x_0 \cdot (1 - x_1) \cdot \big(1 + g(\boldsymbol{x_M})\big), \quad \text{for } i = 0,1,\dots,M \qquad (eq.103)$$

$$\boldsymbol{f}_M = 0.5 \cdot (1 - x_0) \cdot \big(1 + g(\boldsymbol{x_M})\big) \qquad (eq.104)$$

$$g(\boldsymbol{x_M}) = 100\left(|\mathbf{x}| + \sum_{i=0}^{M}(x_i - 0.5)^2 - cos\big(20\pi(x_i - 0.5)\big)\right) \qquad (eq.105)$$

DTLZ N.2:

$$f_i = \big(1 + g(\boldsymbol{x_M})\big)\prod_{j}^{M-i}\cos^2\left(\frac{\pi x_j}{2}\right)\prod_{j=M-i+1}^{M-1}\sin^2\left(\frac{\pi x_j}{2}\right), \text{ for } i = 0,1,\dots,M-1 \qquad (eq.106)$$

$$\boldsymbol{f}_M = \big(1 + g(\boldsymbol{x_M})\big) \cdot sin(x_0\pi/2) \qquad (eq.107)$$

$$g(\boldsymbol{x_M}) = \sum_{x_i \in \boldsymbol{x_M}}(x_i - 0.5)^2 \qquad (eq.108)$$

DTLZ N.3:

$$f_i = \left(1 + g(\boldsymbol{x_M})\right) \cos\left(\frac{\pi x_0}{2}\right) \prod_{j=1}^{M-2} \sin\left(\frac{\pi x_j}{2}\right), \text{for } i = 0,1,\dots,M-1 \qquad (eq.109)$$

$$\boldsymbol{f}_M = \left(1 + g(\boldsymbol{x_M})\right) \cdot sin(x_0 \pi/2) \qquad (eq.110)$$

$$g(\boldsymbol{x_M}) = 100\left(|\mathbf{x}| + \sum_{i=0}^{M} (x_i - 0.5)^2 - cos\big(20\pi(x_i - 0.5)\big)\right) \qquad (eq.111)$$

AntennaCAT Function 12:

$$f_i = \sum_{j=1}^{M-1} \cos\left(\frac{\pi x_j}{2}\right) + \sin\left(\frac{\pi x_i}{2}\right), \quad \text{for } i = 0,1,\dots,M-1 \qquad (eq.112)$$

$$f_M = \sin\left(\frac{\pi x_{M-1}}{2}\right) \qquad (eq.113)$$

AntennaCAT Function 13:

$$f_i = \frac{1}{M}\left[\sum_{j=0}^{M} \left(\cos\left(\frac{i\pi x_j}{M}\right) + \sin\left(\frac{i\pi x_j}{M}\right)\right)\right], \text{for } i = 0,1,,\dots,M \qquad (eq.114)$$

5.10 Summarized Model Design, Training, and Results

A total of 49 neural networks were trained for seven optimizers. For brevity, this section covers the general methodology for designing the Hyperparameter Prediction Network models, the training, and acceptable result thresholds. Selected examples are shown here, but details on the most current version of each machine learning model will be available in the AntennaCAT software documentation included on GitHub. Here, these models are referred to as neural networks as that was the topology used during development of this process. The currently implemented neural networks may be replaced by other topologies or mixed with other topologies in the model library.

Data was separated first by optimizer, and then by the objective function grouping. Logged data is .csv compatible and can be read directly into a Python program using the *pandas* library. Data was read in as a DataFrame and evaluated. In instances where NaN, nulls, or other error values were recorded, that instance was removed from the set. These instances were rare and are suspected to occur when the automated data collection process was interrupted. The remaining valid data was

then separated so that the input and output dimensionality were the inputs to the neural network, and the optimizer hyperparameters were the outputs. The L2 norm difference of the optimal solution from the target was used in the Mean Squared Error (MSE) error calculation to incorporate the convergence performance. The sklearn (scikit-learn) library was used to separate data into 85% train, and 15% test. Training occurred with the torch (PyTorch) library, using the *nn* package with multiple hidden layers. ReLu activation functions were used on every layer of the forward pass. At least two hidden layers were used for all networks. In instances where the input and output dimensionality were high (e.g., multiple input, multiple output), more hidden layers were used. Hidden layers with more nodes performed better on these sets, using 64-96 nodes.

Network performance was first evaluated by the accuracy, MSE, and rate of false classifications. The final validation for if a network was well trained was entering the predicted hyperparameter values into the AntennaCAT GUI and testing on a low-dimensionality replication study (typically the rectangular patch). If the optimizer failed to converge with an error tolerance of less than 10e-3, with a maximum iteration limit of 600, on more than three attempts, the training process was restarted and the network potentially redesigned. Ultimately, networks that had an accuracy over 95%, even if they indicated some bias towards specific objective functions were considered as the predictions were close enough for practical purposes. For instance, when predicting the bird categories for chicken swarm, there is no practical distinction between having one swarm with a rooster, three hens, five mother hens, and four chicks, versus a swarm that has five chicks in terms of short-term objective cost. However, when applied to traditional PSO, it was observed that the balance between the predicted weights had a higher impact on the performance than the exact weight values. That is, for traditional PSO, with closely predicted velocity limits and within the typically predicted 10-13 particle swarm size, it was not atypical for predicted weights to range from 0.7-0.85, with at least two of the three weights being within 3% of each other.

Valid predictions also tended to be within a specific range, rather than specific values. For instance, when training the neural network for the two-input, two-output objective function group with the traditional particle swarm optimizer, mathematically the MSE varied by as much as 3 on predictions where if all variables except for the number of particles (qty. 10-13) were within 2%. For PSO, unless the time and computational cost of the simulation are high, there is limited practical difference between 10 and 11, or even 10 and 13 particles. This suggests that there may

be some benefit to timing the simulation run time during AntennaCAT's tuning process and adjusting hyperparameters based on the length of the simulation time. This utility is further discussed in the next section as a near-future expansion.

5.11 Hyperparameter Prediction Network Model Expansion

The objective function and optimizer combinations used to collect data for these models have been heavily covered in this chapter. During the design and testing of these models, consideration was given to addressing two design necessities. Firstly, between the prediction of the hyperparameter values and the output of the 'Help Me Choose' tab on the GUI, processing must be done to ensure that predicted values are valid inputs into the optimizer. While a properly trained ML model should not output values that are drastically outside of the expected, due to the modular nature of the library, the potential for unforeseen update factors as the library expands, and that the optimizers expect specific value formats, this layer of error checking is necessary. For instance, the swarm-based optimizers expect integers for the size of the swarm. While it is trivial to convert a float to an integer at the optimizer, as the Hyperparameter Prediction Network becomes more intelligent, it will be highly relevant for the controller to have access to the values that are run in the optimizer instances.

Secondly, while basic validation could be evaluated with accuracy predicting a test subset of data, the MSE for the predictions while training and testing, and analyzing the rate of false classifications, the final validation of a network could only be done with simulation. During testing and validation, with 49 neural networks in the model library, it was not reasonable to test every potential model candidate with multiple antenna topologies and multiple trials of optimization with simulation. The parameters were tested on the optimizers, but this remains a partial validation of the problem.

With these in mind, the trends discussed in section 5.10 regarding prediction ranges suggest that the next step for the Hyperparameter Prediction Network models, controller, and the 'Help Me Choose' GUI element is to integrate live tuning based on simulation results, runtime, and if possible, knowledge of computational resources. Previously, AntennaCAT has not tracked the time for simulations to run to completion, but by tracking the simulation time and using it as a proxy

for the objective function cost, the Hyperparameter Prediction Network could make adaptive predictions to further streamline the optimization process.

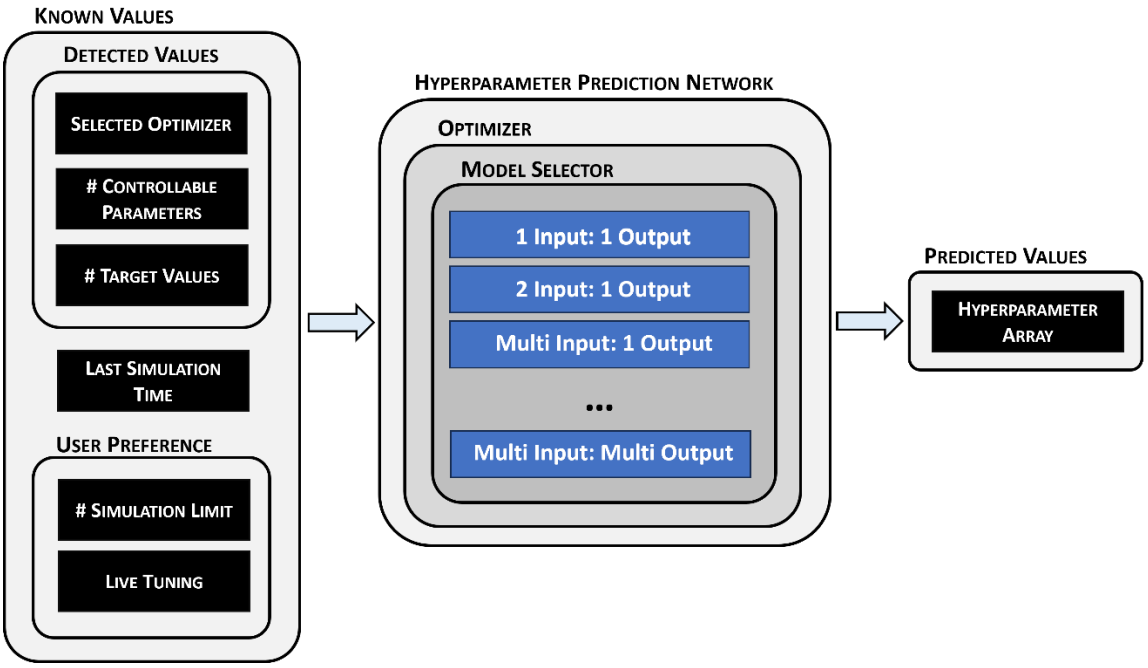


Figure 44 *The generalized process where the specific optimizer has been selected. The Known Values have been expanded from Figure 37 to include information that supports an expansion of the Hyperparameter Prediction Network.*

Figure 44 includes the expansion to the 'Known Variables' in the MLAO design structure shown in Figure 37. Rather than relying only on knowledge of the selected optimizer, the number of controllable parameters, and the number of target values, it would be beneficial to include the last simulation time, a live tuning option for users to select static or adaptive hyperparameters, and other variables that set optimizer-specific limits. For example, for optimizers such as the swarm-based optimizers, limiting the size of the swarm could be an additional input to the neural network library, which would act as another reference point for predicting hyperparameters, and set a maximum limit for prediction values, which would in turn set a limit on the maximum cost per generation of a swarm.

CHAPTER 6
Applications for Antenna Design

Presented thus far has been the motivation, design, and implementation of the AntennaCAT software suite. This chapter presents two examples of the utilization of AntennaCAT's automated tuning capabilities. Antenna measurements in this section were conducted in two ways. The S_{11} measurements were taken on a Keysight FieldFox Series N9912A Network Analyzer. 3D gain measurements were taken in isolation using an ETS Lindgren anechoic chamber and automated control system.

The first design presented in this section, a Wi-Fi 6E probe-fed planar antenna, was created using the imported .DXF from Figure 6, and then manually edited to include extra parameters. This design was then imported back into AntennaCAT. The second design, a dual-band 5 GHz and 6 GHz Wi-Fi antenna was imported as an existing planar antenna. To simplify the experimentation process, both designs were limited to the following controllable parameters: the scale of the imported .DXF design in the X and Y directions, and the X and Y locations of the probe feed. These designs had been previously optimized manually, so it was known that these were the minimum parameters needed to reach the set target values. The target S_{11} values were -10 dB or lower, and the minimum acceptable gain was set to 2 dB. The target values were chosen to increase the likelihood of the PSO algorithm converging to a solution. Each design was optimized five times, starting with the original configuration, and the best result was selected for discussion. All trials converged within 25 generations (with 5 particles), and results between trials were negligible given the easily met target metrics.

6.1 Wi-Fi 6E Automated Tuning

To create this design, the AntennaCAT logo was imported as a .DXF and used to create a simple planar antenna with a copper ground plane. Some manual editing was done to remove stray lines in HFSS due to the conversion between .TIFF and .DXF leaving relics. Additional parameterization was added to the .DXF import for the X and Y axis directions to add a scaling factor. Addressing polygon simplification is beyond the scope of the AntennaCAT software as it currently exists, but may be possible in the future. The planar antenna was simulated as is shown in Figure 45 c), where the orange represents the copper conductor, and the blue area was milled

out in construction. Double-sided copper FR4 with a permittivity of 4.4 and a thickness of 1.6 mm was used in simulation.

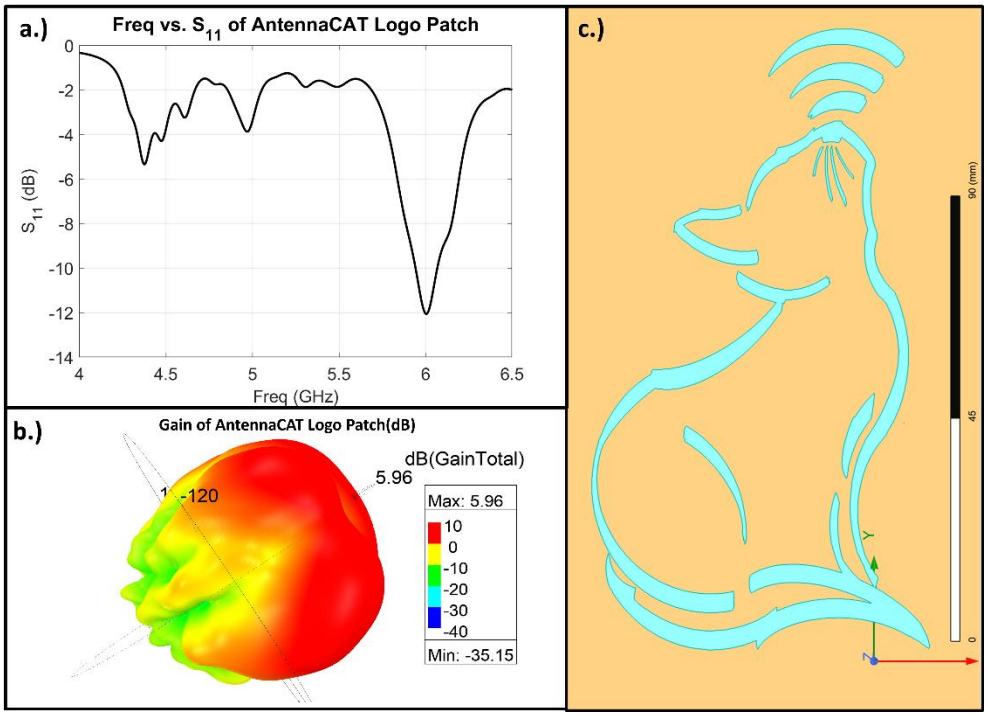


Figure 45 *The 6E Wi-Fi antenna created with the AntennaCAT logo. a.) The optimized resonant frequency of -12 dB at 6 GHz, b.) The gain of the AntennaCAT logo patch antenna, c.) a front view of the Ansys HFSS simulated patch antenna.*

The antenna in Figure 45 was simulated with a probe fed through the copper ground plane on the back. This antenna had a simulated resonance of -12 dB at 6 GHz, and a gain of 5.96 GHz. To verify the simulated design, the patch antenna was milled using double-sided FR4.

Figure 46 *Six versions of epoxy-treated AntennaCAT logo cat-shaped antennas used for validating optimizer results.*

Figure 46 shows 6 variations where the milled antennas were treated with dyed epoxy. Five of the antennas, aside from the dark pink (3rd from the right) were treated with glow-in-the-dark UV-cured epoxy. The dark pink antenna was treated with an acrylic-based epoxy and left to air-dry. The probe feed location is visible in the center with the silver solder. In experimentation, the epoxy added to the milled areas did not shift the frequency in the measured bands. Figure 47 shows the five antennas treated with glow-in-the-dark epoxy excited by UV light.

Figure 47 *The five AntennaCAT logo antennas treated with glow-in-the-dark epoxy being excited by UV light in a dark room.*

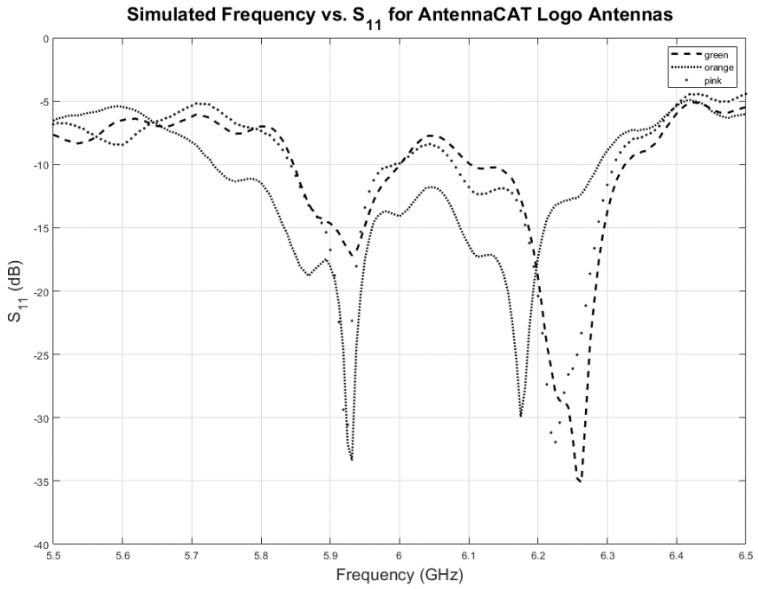


Figure 48 *The measured S_{11} of three of the glow-in-the-dark epoxy-treated milled AntennaCAT logos antennas.*

The return loss (S_{11}) of the AntennaCAT logo patch antennas is plotted in Figure 48. Three examples, the green, orange, and pink glow-in-the-dark epoxy treated antennas, are included in

this plot. These three antennas are in good agreement with each other, and all three show the slight shift upwards in frequency that occurred in the manufacturing process with this set, though the return loss for all three is at or below -10 dB at 6 GHz. The epoxy did not affect the S_{11} in the 6 GHz band. The epoxy treatment was experimentally noted to shift frequencies lower at 10 GHz and above with this design. However, the antenna was not designed for bands above 6 GHz, and further experimentation would need to be conducted to determine if that shift is reliable. Figure 49, below, shows the measured gain of the orange epoxy treated AntennaCAT logo antenna. The maximum gain occurred at 6085 MHz, at 1.8 dB. This was typical across this set, and it is strongly suspected to be caused by a slight shift in the probe location caused while drilling through the FR4. This design is extremely sensitive to shifts in the probe location due to the complex geometry. However, this design still demonstrates the ability of AntennaCAT to import and optimize designs.

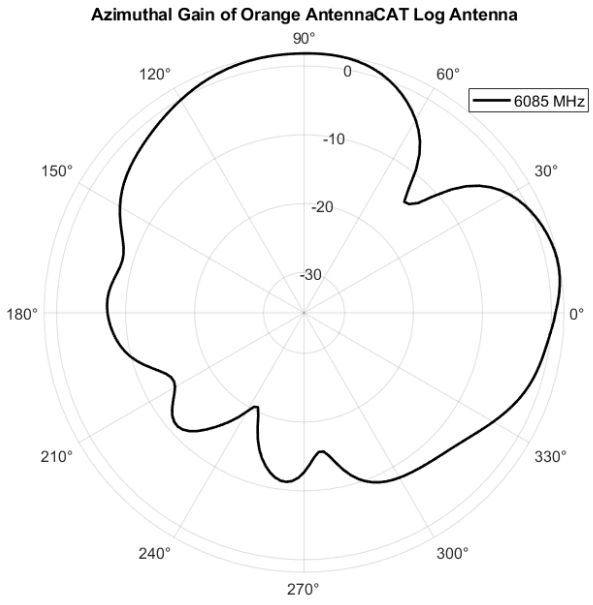


Figure 49 *The measured gain of the milled AntennaCAT logo antenna treated with orange epoxy. The maximum gain occurred with the azimuthal measurement at 6085 MHz, at 1.8 dB.*

6.2 Dual-Band 5 GHz, 6 GHz Wi-Fi Antennas

This planar design was created and parameterized manually. It uses the same probe-fed planar antenna base (1.6 mm FR4 substrate with copper ground plane) that AntennaCAT uses as a default planar setup to retain some consistency between the presented examples. The .DXF was imported and intersected with a copper sheet to create the conductor, and parameterization was added for the X and Y axis scaling. The probe was placed towards the left ear to encourage placement for cosmetic reasons. The planar antenna as it was simulated is shown in Figure 50 c), where the orange represents the copper conductor, and the yellow is the visible FR4 substrate. Double-sided copper FR4 with a permittivity of 4.4 and a thickness of 1.6 mm was used in simulation and construction of the antenna.

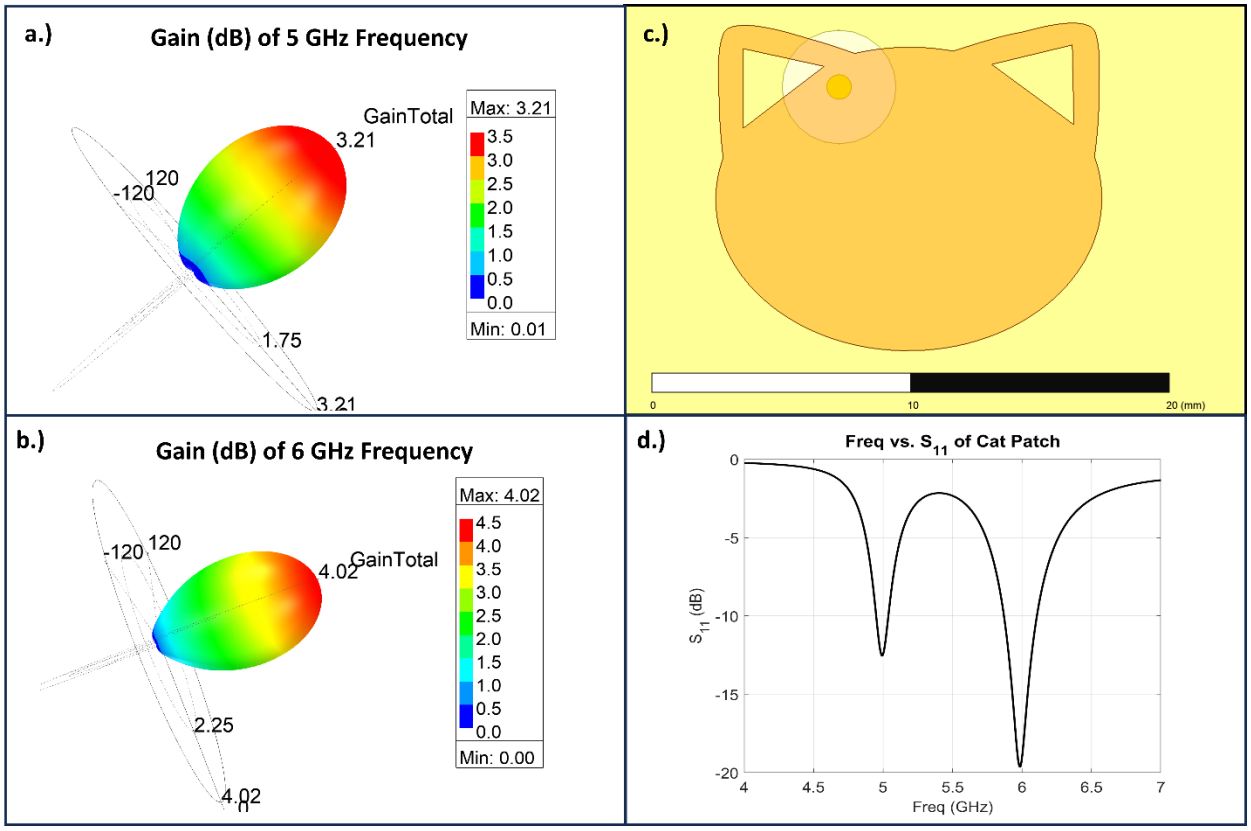


Figure 50 *The dual-band 5 GHz and 6 GHz Wi-Fi antenna created with the simplified cat patch planar antenna. a.) the gain plot for the 5 GHz frequency, b.) the gain plot for the 6 GHz frequency, c.) a front view of the Ansys HFSS simulated patch antenna, and d.) the simulated* S_{11}.

Figure 51 shows the milled cat patch antenna on 1.6 mm double-sided copper clad FR4, on the right with the rectangular ground plane. The design on the left of Figure 51 is the tuned version of

the rectangular patch placed on to a circular ground plane. Both designs are probe-fed from the back. The results of measurements with the FieldFox and anechoic chamber are shown in Figures 52 and 53, where the measured and simulated results are compared for the resonant frequency and gain, respectively.

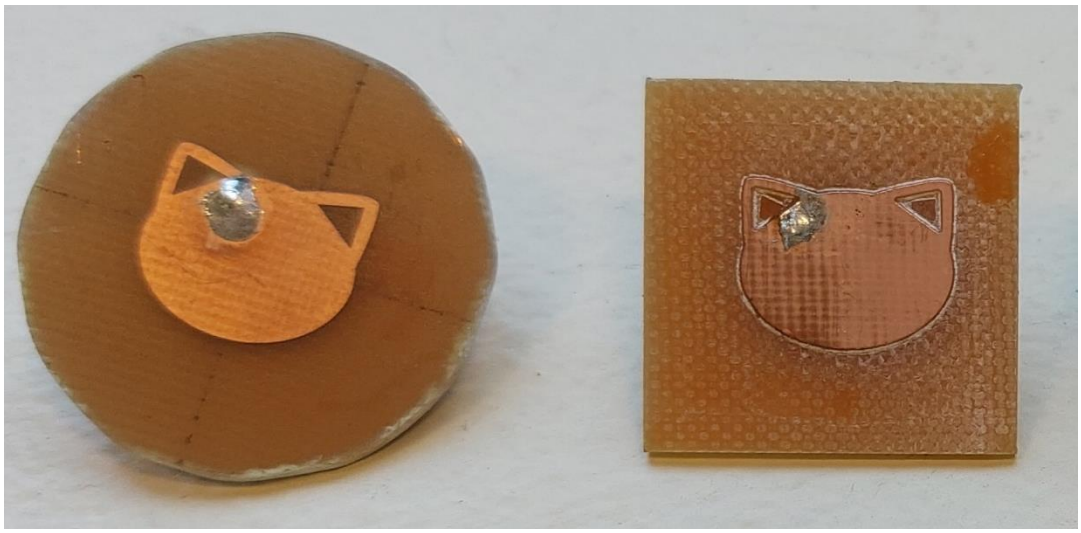


Figure 51 *Two milled variations of the dual-band 5 GHz and 6 GHz cat-shaped patch antenna used for verifying optimizer results. The rectangular ground plane version was simulated and tuned using the AntennaCAT software. The round version on the left differs only in ground plane construction.*

In Figure 52, both the rectangular and circular ground planes are measured to demonstrate the higher tolerance of this design to manufacturing changes. Unlike the complex AntennaCAT logo design, this design is tolerant of probe location and milling resolution. The resonant frequencies in Figure 52 are in good agreement, though the 5 GHz return loss of the rectangular patch performed better. The gain of the rectangular patch taken as the azimuth and elevation angles shows the performance at 5 and 6 GHz. The results are in good agreement, though there is some variation due to the variation caused by hand drilling the FR4.

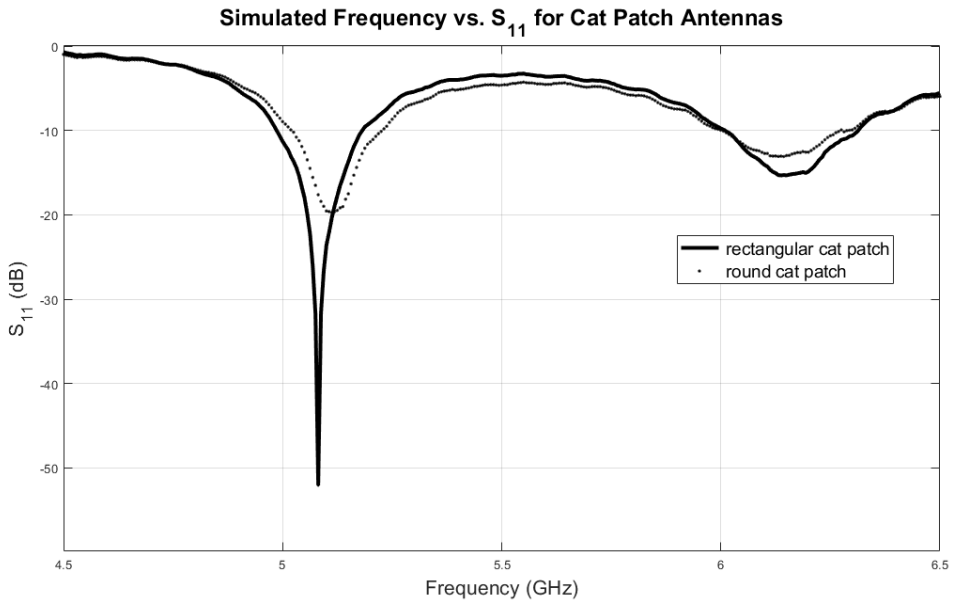


Figure 52 *The measured S_{11} of the milled cat-shaped patch antennas for 5 GHz and 6 GHz. Two variations were simulated: the cat-shape design implemented with a rectangular ground plane, and the cat-shape design with a round ground plane.*

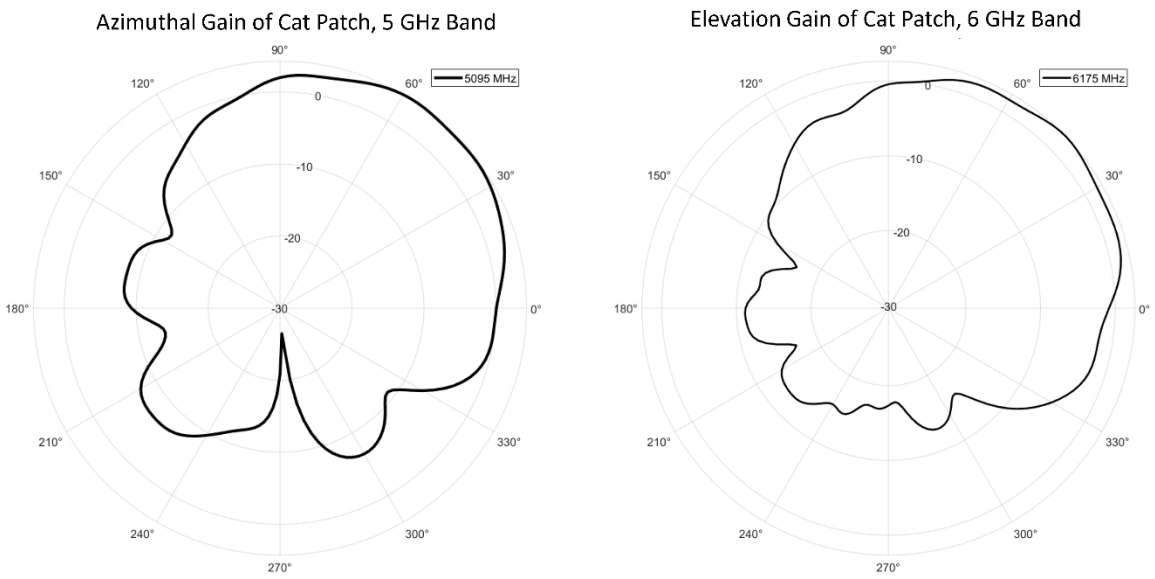


Figure 53 *The measured gain of the milled cat-shaped patch antennas. Left, the azimuthal value measuring the 5 GHz frequency polarized from cheek-to-cheek. Right, the elevation measuring the top-to-chin polarized 6 GHz band.*

CHAPTER 7
Conclusion

The Antenna Calculation and Autotuning Tool (AntennaCAT) software suite represents a significant advancement in the field of antenna design by automating the entire design, CAD, simulation, and optimization process compatible with several EM simulation software suites. It is the first comprehensive implementation of machine learning in this context. In particular, this work includes the capability to create and export structured datasets from the aforementioned EM software for iterative improvement and includes an expandable selection of optimizers. AntennaCAT promotes research replication with its antenna calculator for three topologies (rectangular patch antenna, half-wave monopole, and quarter-wave dipole) and an expandable internal library of parameterized replication study designs. The software supports importing existing scripts and loading in project references for file modification. All calculated, replicated, imported, and loaded projects are compatible with the eleven optimizers included in the internal optimizer suite, which include eight swarm-based optimizers, a Sweep optimizer with random and grid search options, a Bayesian optimizer, and optional surrogate model kernels to reduce computational needs. In total, there are 90 optimizer-surrogate model combinations, and additional configurations such as boundary condition handling which may cause unique optimizer behavior. With the internal design options, this creates more than 1,500 combination options for users to choose from, and then customize.

The AntennaCAT software suite addresses the broad range of design optimization options by leveraging data collected from single-objective and multi-objective benchmark functions to train neural networks in the Hyperparameter Prediction Network models. These networks suggest optimizer hyperparameters based on the number of controllable problem variables and the number of target values being optimized. The near-future expansion of this feature to include tracking simulation times and adjusting parameters such as maximum swarm size, or weights, as a proxy variable for assessing computational cost was discussed in Chapter 5.

While AntennaCAT is designed to be a constantly evolving project that will continue to incorporate more replication studies, optimizers, compatible EM simulation software suites, and other features, this document has detailed the methodology behind AntennaCAT from conception to execution.

It has also described the process of the generation, customization, scripting, CAD creation, simulation, and optimization that can be used to replicate existing research or create unique custom designs. In Chapter 6, two examples of probe-fed planar antennas were created using the AntennaCAT software suite, simulated, manufactured, and then evaluated in an anechoic chamber. The manufactured antennas were in good agreement with the simulated designs. Variations between simulation and measurement can be largely attributed to variations in permittivity between the simulated 4.4 and the actual material, hand drilling probe locations for the small patch antennas, and manufacturing tolerance with the milling process.

CHAPTER 8
References

[1] C. A. Balanis, *Antenna Theory:* Analysis and Design. Hoboken, NJ: Wiley, 2016.

[2] W. L. Stutzman and G. A. Thiele, Antenna Theory and Design. Hoboken, NJ: Wiley, 2013.

[3] J. L. Volakis, Antenna Engineering Handbook, 4th ed. New York, NY: McGraw-Hill Education, 2007.

[4] J. Lilja, P. Salonen, T. Kaija and P. de Maagt, "Design and Manufacturing of Robust Textile Antennas for Harsh Environments," in *IEEE Transactions on Antennas and Propagation*, vol. 60, no. 9, pp. 4130-4140, Sept. 2012, doi: 10.1109/TAP.2012.2207035.

[5] J. Lundquist *et al*., "Textile-Based Inkjet-Printed RFIDs: Exploring wearable antennas in the real world [Bioelectromagnetics]," in *IEEE Antennas and Propagation Magazine*, vol. 66, no. 1, pp. 50-62, Feb. 2024, doi: 10.1109/MAP.2023.3334671

[6] J. D. Lundquist, L. Linkous, U. Hasni and E. Topsakal, "Indirect Applications of Additive Manufacturing for Antennas," in *IEEE Open Journal of Antennas and Propagation*, vol. 4, pp. 434-445, 2023, doi: 10.1109/OJAP.2023.3265691.

[7] T. Rylander, P. Ingelström, and A. Bondeson, Computational Electromagnetics. New York, NY: Springer New York, 2013.

[8] K. Sankaran, “Are you using the right tools in computational electromagnetics?,” Engineering Reports, vol. 1, no. 3, Oct. 2019. doi:10.1002/eng2.12041

[9] U. Jakobus and G. Smith, "State of the art of electromagnetic modelling in FEKO," *2012 6th European Conference on Antennas and Propagation (EUCAP)*, Prague, Czech Republic, 2012, pp. 853-854, doi: 10.1109/EuCAP.2012.6206582.

[10] Q. Wu, Y. Cao, H. Wang and W. Hong, "Machine-learning-assisted optimization and its application to antenna designs: Opportunities and challenges," in *China Communications*, vol. 17, no. 4, pp. 152-164, April 2020, doi: 10.23919/JCC.2020.04.014.

[11] “A Novel Optimization Approach Using optiSLang and HFSS Analytical Derivatives,” Ansys, https://www.ansys.com/resource-center/webinar/novel-optimization-approach-using-optislang-and-hfss-analytical-derivatives (accessed Mar. 23, 2024).

[12] Ansoft Corp., “Parametrics and Optimization Using Ansoft HFSS,” Microwave Journal, Ansoft Corp., Nov. 1999

[13] “Antenna magus: Antenna Design Software - Simulia by Dassault Systèmes®,” Antenna Magus | Antenna design software - SIMULIA by Dassault Systèmes®. [Online]. Available: https://www.3ds.com/products-services/simulia/products/antenna-magus/. [Accessed: 14-Mar-2024].

[14] “Optimisation in Feko,” 2022.help.altair.com, https://2022.help.altair.com/2022/feko/topics/feko/user_guide/optimisation/optimisation_intro_feko_c.htm (accessed Mar. 22, 2024).

[15] C. Piersall, “Scripting HFSS inside MATLAB,” Scripting HFSS Inside MATLAB - HFSS Lib 0.0.0 documentation, 2015. [Online]. Available: https://arrc.ou.edu/~cody/hfsslib/matlab_examples/. [Accessed: 21-Mar-2024].

[16] Yuip, “Yuip/hfss-API: A HFSS API to control HFSS from MATLAB,” GitHub. [Online]. Available: https://github.com/yuip/hfss-api. [Accessed: 21-Mar-2024].

[17] S. Tariq, "Automation of reflectarrays in HFSS using visual basic scripting," *2018 Texas Symposium on Wireless and Microwave Circuits and Systems (WMCS)*, Waco, TX, USA, 2018, pp. 1-4, doi: 10.1109/WMCaS.2018.8400640.

[18] P. Monk, *Finite element methods for Maxwell’s equations*. Oxford ; New York: Clarendon Press, 2003.

[19] T. Rylander and A. Bondeson, “Stable FEM-FDTD hybrid method for Maxwell’s equations,” *Computer Physics Communications*, vol. 125, no. 1–3, pp. 75–82, Mar. 2000, doi: https://doi.org/10.1016/s0010-4655(99)00463-4

[20] Microstrip Patch Antenna Calculator. [Online]. Available: https://www.pasternack.com/t-calculator-microstrip-ant.aspx. [Accessed: 17-Mar-2024].

[21] Ł. Zaborowska, “Dipole calculator,” Antenna Length Calculator, 14-Nov-2022. [Online]. Available: https://www.omnicalculator.com/physics/dipole. [Accessed: 14-Mar-2023].

[22] “Antenna Wavelength Calculator,” Antenna Wavelength Calculator | Southwest Antennas - High Performance RF and Microwave Antennas & Custom Antenna Manufacturing. [Online]. Available: https://www.southwestantennas.com/calculator/antenna-wavelength. [Accessed: 14-Mar-2023].

[23] “Dipole Antenna Length Calculator,” Dipole Antenna Length Calculator - Everything RF. [Online]. Available: https://www.everythingrf.com/rf-calculators/dipole-antenna-length-calculator. [Accessed: 14-Mar-2023].

[24] Ansys (pyansys), “PyAEDT,” GitHub. [Online]. Available: https://github.com/pyansys/pyaedt. [Accessed: 14-Mar-2023].

[25] “Cadfeko API,” 2021.help.altair.com. [Online]. Available: https://2021.help.altair.com/2021.1/feko/topics/feko/user_guide/appendix/api_cadfeko_feko_c.htm. [Accessed: 27-Mar-2023].

[26] “Example Postfeko API Script,” 2021.help.altair.com. [Online]. Available: https://2021.help.altair.com/2021/feko/topics/feko/user_guide/scripting/example_postfeko_api_script_feko_c.htm. [Accessed: 27-Mar-2023].

[27] “Introduction to scripting and the API,” 2021.help.altair.com. [Online]. Available: https://2021.help.altair.com/2021.1/feko/topics/feko/user_guide/scripting/intro_script_api_feko_c.htm. [Accessed: 27-Mar-2023].

[28] “List of environment variables,” 2021.help.altair.com. [Online]. Available: https://2021.help.altair.com/2021/feko/topics/feko/user_guide/appendix/environment_variables_feko_r.htm. [Accessed: 27-Mar-2023].

[29] C. Piersall, “Scripting HFSS inside MATLAB,” Scripting HFSS Inside MATLAB - HFSS Lib 0.0.0 documentation, 2015. [Online]. Available: https://arrc.ou.edu/~cody/hfsslib/matlab_examples/. [Accessed: 14-Mar-2023].

[30] Yuip, “Yuip/hfss-API: A HFSS API to control HFSS from MATLAB,” GitHub. [Online]. Available: https://github.com/yuip/hfss-api. [Accessed: 14-Mar-2023].

[31] A. Z. Hood and E. Topsakal, "Particle swarm optimization for dual-band implantable antennas," 2007 IEEE Antennas and Propagation Society International Symposium, Honolulu, HI, USA, 2007, pp. 3209-3212, doi: 10.1109/APS.2007.4396219.

[32] E. K. Dahbi, T. Elhamadi, and N. Amar Touhami, “Optimization of the SIW cavity-backed slots antenna for X-band applications using the particle swarm optimization algorithm,” Journal of Electromagnetic Waves and Applications, vol. 36, no. 7, pp. 928–939, 2021. doi:10.1080/09205071.2021.1996278

[33] J. B. Romdhane Hajri et al., "Fast and Automatic RF Design Based on MATLAB-HFSS Control Applied on Magnetic Absorber with Metasurface," 2019 Photonics & Electromagnetics Research Symposium - Fall (PIERS - Fall), Xiamen, China, 2019, pp. 1339-1342, doi: 10.1109/PIERS-Fall48861.2019.9021573.

[34] S. Montoya Villada, E. Reyes Vera, and M. Arias-Correa, “Animage: A MATLAB-based tool

for generating microstrip antennas with complex shapes," SoftwareX, vol. 23, 2023. doi:10.2139/ssrn.4455828

[35] R. Banks, Q. Nguyen, R. Fenner and A. Zaghloul, "Pixelated Metamaterial Determination Using Genetic Algorithm and HFSS/Matlab Integration," 2023 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting (USNC-URSI), Portland, OR, USA, 2023, pp. 1595-1596, doi: 10.1109/USNC-URSI52151.2023.10237688.

[36] MPh-py, "MPh: Pythonic scripting interface for Comsol Multiphysics," *GitHub.* [Online]. Available: https://github.com/MPh-py/MPh. [Accessed: 14-Mar-2023].

[37] Fellobos, "cmphy," *GitHub*. [Online]. Available: https://fellobos.github.io/cmphy/. [Accessed: 14-Mar-2023].

[38] "Run COMSOL MULTIPHYSICS® simulations with MATLAB®," *COMSOL*. [Online]. Available: https://www.comsol.com/livelink-for-matlab. [Accessed: 27-Mar-2023].

[39] "Automate your modeling tasks with the COMSOL API for use with Java®," *COMSOL*. [Online]. Available: https://www.comsol.com/blogs/automate-modeling-tasks-comsol-api-use-java/. [Accessed: 27-Mar-2023].

[40] M. O. Akinsolu, K. K. Mistry, B. Liu, P. I. Lazaridis and P. Excell, "Machine Learning-assisted Antenna Design optimization: A Review and the State-of-the-art," *2020 14th European Conference on Antennas and Propagation (EuCAP)*, Copenhagen, Denmark, 2020, pp. 1-5, doi: 10.23919/EuCAP48036.2020.9135936.

[41] N. Sarker, P. Podder, M. R. H. Mondal, S. S. Shafin and J. Kamruzzaman, "Applications of Machine Learning and Deep Learning in Antenna Design, Optimization, and Selection: A Review," in *IEEE Access*, vol. 11, pp. 103890-103915, 2023, doi: 10.1109/ACCESS.2023.3317371.

[42] Q. Wu, W. Chen, C. Yu, H. Wang and W. Hong, "Machine-Learning-Assisted Optimization for Antenna Geometry Design," in *IEEE Transactions on Antennas and* Propagation, vol. 72, no. 3, pp. 2083-2095, March 2024, doi: 10.1109/TAP.2023.3346493

[43] Y. Sharma, H. H. Zhang, and H. Xin, "Machine Learning Techniques for Optimizing Design of Double T-Shaped Monopole Antenna," in IEEE Transactions on Antennas and Propagation, vol. 68, no. 7, pp. 5658-5663, July 2020, doi: 10.1109/TAP.2020.2966051.

[44] A. Srivastava, H. Gupta, A. Kumar Dwivedi, K. Kanth Varma Penmatsa, P. Ranjan, and A. Sharma, "Aperture coupled dielectric resonator antenna optimization using Machine Learning

Techniques," *AEU - International Journal of Electronics and Communications, vol.* 154, p. 154302, Sep. 2022.

[45] H. M. E. Misilmani and T. Naous, "Machine Learning in Antenna Design: An Overview on Machine Learning Concept and Algorithms," 2019 International Conference on High Performance Computing & Simulation (HPCS), Dublin, Ireland, 2019, pp. 600-607, doi: 10.1109/HPCS48598.2019.9188224.

[46] W. Chen, Q. Wu, C. Yu, H. Wang, and W. Hong, "Multibranch Machine Learning-Assisted Optimization and Its Application to Antenna Design," in *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 7, pp. 4985-4996, July 2022, doi: 10.1109/TAP.2022.3179597.

[47] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67-82, April 1997, doi: 10.1109/4235.585893.

[48] M. Baker, "1,500 scientists lift the lid on reproducibility," Nature, vol. 533, no. 7604, pp. 452–454, May 2016. doi:10.1038/533452a

[49] L. Linkous, *objective_function_suite* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/objective_function_suite

[50] L. Linkous, E. Karincic, J. Lundquist and E. Topsakal, "Automated Antenna Calculation, Design and Tuning Tool for HFSS," *2023 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2023, pp. 229-230, doi: 10.23919/USNC-URSINRSM57470.2023.10043119.

[51] L. Linkous, J. Lundquist and E. Topsakal, "AntennaCAT: Automated Antenna Design and Tuning Tool," *2023 IEEE USNC-URSI Radio Science Meeting (Joint with AP-S Symposium)*, Portland, OR, USA, 2023, pp. 89-90, doi: 10.23919/USNC-URSI54200.2023.10289238.

[52] L. Linkous, (LC-Linkous) (2022) AntennaCAT (Version 2024) [source code] https://github.com/LC-Linkous/AntennaCalculationAutotuningTool

[53] E. Karincic, E. Topsakal, and L. Linkous, "Patch Antenna Calculations and Fabrication Made Simple for Cyber Security Research," in *2023 ASEE Annual Conference & Exposition*, 2023.

[54] E. Karincic, (Dollarhyde) (2022) Antenna Calculator (Version 2.0) [source code]. https://github.com/Dollarhyde/AntennaCalculator

[55] Highway Surveying Manual, M22-97, Washington State Department of Transportation, WA, USA, 2005, pp. 135-142. [online] https://www.wsdot.wa.gov/publications/manuals/fulltext/M22-

97/highwaysurvey.pdf
[56] L. Linkous, *GeneticCAT* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/GeneticCAT
[57] A. Papathanasopoulos, P. A. Apostolopoulos and Y. Rahmat-Samii, "Optimization Assisted by Neural Network-Based Machine Learning in Electromagnetic Applications," in IEEE Transactions on Antennas and Propagation, doi: 10.1109/TAP.2023.3269883.
[58] U. Hasni and E. Topsakal, "Wearable Antennas for On-Body Motion Detection," 2020 IEEE USNC-CNC-URSI North American Radio Science Meeting (Joint with AP-S Symposium), Montreal, QC, Canada, 2020, pp. 1-2, doi: 10.23919/USNC/URSI49741.2020.9321663.
[59] A. Z. Hood and E. Topsakal, "Particle swarm optimization for dual-band implantable antennas," 2007 IEEE Antennas and Propagation Society International Symposium, Honolulu, HI, USA, 2007, pp. 3209-3212, doi: 10.1109/APS.2007.4396219.
[60] T. Karacolak, A. Z. Hood and E. Topsakal, "Design of a Dual-Band Implantable Antenna and Development of Skin Mimicking Gels for Continuous Glucose Monitoring," in IEEE Transactions on Microwave Theory and Techniques, vol. 56, no. 4, pp. 1001-1008, April 2008, doi: 10.1109/TMTT.2008.919373.
[61]S. Sukhija, R. K. Sarin, and N. Kashyap, “Design of compact wideband serpentine patch antenna for ingestible endoscopic applications,” Progress In Electromagnetics Research M, vol. 66, pp. 53–63, 2018. doi:10.2528/pierm17120101
[62] N. Sharma and S. S. Bhatia, “Design of antenna by amalgamating staircase and hexagonal ring-shaped structures with the modified ground plane for multi-standard wireless applications,” Journal of Electromagnetic Waves and Applications, vol. 36, no. 7, pp. 893–911, 2021. doi:10.1080/09205071.2021.1995898
[63] U. Hasni, M. E. Piper, J. Lundquist and E. Topsakal, "Screen-Printed Fabric Antennas for Wearable Applications," in IEEE Open Journal of Antennas and Propagation, vol. 2, pp. 591-598, 2021, doi: 10.1109/OJAP.2021.3070919.
[64] A. Nunnally and E. Topsakal, "Dual-Band FSS for WMTS and CBRS for Wearable Wireless Medical Telemetry," 2023 IEEE USNC-URSI Radio Science Meeting (Joint with AP-S Symposium), Portland, OR, USA, 2023, pp. 113-114, doi: 10.23919/USNC-URSI54200.2023.10289424.
[65] N. Jin and Y. Rahmat-Samii, "Parallel particle swarm optimization and finite- difference time-

domain (PSO/FDTD) algorithm for multiband and wide-band patch antenna designs," in IEEE Transactions on Antennas and Propagation, vol. 53, no. 11, pp. 3459-3468, Nov. 2005, doi: 10.1109/TAP.2005.858842.

[66] A. Aldhafeeri and Y. Rahmat-Samii, "Brain Storm Optimization for Electromagnetic Applications: Continuous and Discrete," in IEEE Transactions on Antennas and Propagation, vol. 67, no. 4, pp. 2710-2722, April 2019, doi: 10.1109/TAP.2019.2894318.

[67] A. Z. Hood, T. Karacolak and E. Topsakal, "A Small Antipodal Vivaldi Antenna for Ultrawide-Band Applications," in IEEE Antennas and Wireless Propagation Letters, vol. 7, pp. 656-660, 2008, doi: 10.1109/LAWP.2008.921352.

[68] R. B. Green and E. Topsakal, "Biocompatible Antennas for Implantable Biosensor Systems," 2019 International Workshop on Antenna Technology (iWAT), Miami, FL, USA, 2019, pp. 70-72, doi: 10.1109/IWAT.2019.8730633.

[69] T. Karacolak and E. Topsakal, "A Double-Sided Rounded Bow-Tie Antenna (DSRBA) for UWB Communication," in IEEE Antennas and Wireless Propagation Letters, vol. 5, pp. 446-449, 2006, doi: 10.1109/LAWP.2006.885013.

[70] M. Asili, R. Green, S. Seran and E. Topsakal, "A Small Implantable Antenna for MedRadio and ISM Bands," in IEEE Antennas and Wireless Propagation Letters, vol. 11, pp. 1683-1685, 2012, doi: 10.1109/LAWP.2013.2241723.

[71] A. Deb, J. S. Roy and B. Gupta, "Performance Comparison of Differential Evolution, Particle Swarm Optimization and Genetic Algorithm in the Design of Circularly Polarized Microstrip Antennas," in IEEE Transactions on Antennas and Propagation, vol. 62, no. 8, pp. 3920-3928, Aug. 2014, doi: 10.1109/TAP.2014.2322880.

[72] Z. Wu, Y. Yang and Z. Yao, "Multi-Parameter Modeling with ANN for Antenna Design," 2018 IEEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting, Boston, MA, USA, 2018, pp. 2381-2382, doi: 10.1109/APUSNCURSINRSM.2018.8608587.

[73] A. L. Custódio and J. F. A. Madeira, “MultiGLODS: global and local multiobjective optimization using direct search,” Journal of Global Optimization, vol. 72, no. 2, pp. 323–345, Feb. 2018, doi: https://doi.org/10.1007/s10898-018-0618-1.

[74] B. Bischl et al., “Hyperparameter optimization: Foundations, Algorithms, best practices, and open challenges,” WIREs Data Mining and Knowledge Discovery, vol. 13, no. 2, 2023.

doi:10.1002/widm.1484
[75] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, “Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization,” Journal of Electronic Science and Technology, vol. 17, no. 1, pp. 26–40, Mar. 2019, doi: https://doi.org/10.11989/JEST.1674-862X.80904120.
[76] J. Luo, W. Xu and J. Chen, "A Novel Radial Basis Function (RBF) Network for Bayesian Optimization," 2021 IEEE 7th International Conference on Cloud Computing and Intelligent Systems (CCIS), Xi'an, China, 2021, pp. 250-254, doi: 10.1109/CCIS53392.2021.9754629.
[77] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," in IEEE Transactions on Antennas and Propagation, vol. 52, no. 2, pp. 397-407, Feb. 2004, doi: 10.1109/TAP.2004.823969.
[78] Z. -H. Zhan, J. Zhang, Y. Li and H. S. -H. Chung, "Adaptive Particle Swarm Optimization," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 39, no. 6, pp. 1362-1381, Dec. 2009, doi: 10.1109/TSMCB.2009.2015956.
[79] J. Lundquist, L. Linkous, *pso_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/jonathan46000/pso_Python
[80] S.-C. Chu, P. Tsai, and J.-S. Pan, “Cat swarm optimization,” Lecture Notes in Computer Science, pp. 854–858, 2006. doi:10.1007/978-3-540-36668-3_94
[81] A. M. Ahmed, T. A. Rashid, and S. Ab. Saeed, “Cat Swarm Optimization Algorithm: A survey and performance evaluation,” Computational Intelligence and Neuroscience, vol. 2020, pp. 1–20, 2020. doi:10.1155/2020/4854895
[82] A. Seyyedabbasi and F. Kiani, “Sand Cat swarm optimization: a nature-inspired algorithm to solve global optimization problems,” Engineering with Computers, Apr. 2022, doi: https://doi.org/10.1007/s00366-022-01604-x.
[83] “Sand Cat swarm optimization,” www.mathworks.com. https://www.mathworks.com/matlabcentral/fileexchange/110185-sand-cat-swarm-optimization (accessed Jun. 19, 2024).
[84] X. B. Meng, Y. Liu, X. Gao, and H. Zhang, "A new bio-inspired algorithm: Chicken swarm optimization," in Proc. Int. Conf. Swarm Intell. Cham, Switzerland, Springer, 2014, pp. 86–94.
[85] S. Liang, Z. Fang, G. Sun, Y. Liu, G. Qu and Y. Zhang, "Sidelobe Reductions of Antenna Arrays via an Improved Chicken Swarm Optimization Approach," in IEEE Access, vol. 8, pp.

37664-37683, 2020, doi: 10.1109/ACCESS.2020.2976127.

[86] Z. Qiuqiao, B. Wang, L. Wei and W. Haishan, "Chicken swarm optimization algorithm based on quantum behavior and its convergence analysis," 2020 39th Chinese Control Conference (CCC), Shenyang, China, 2020, pp. 2107-2112, doi: 10.23919/CCC50068.2020.9189572.

[87] W. Chu, X. Gao, and S. Sorooshian, "Handling boundary constraints for particle swarm optimization in high-dimensional search space," Information Sciences, vol. 181, no. 20, pp. 4569–4581, 2011. doi:10.1016/j.ins.2010.11.030

[88] T. Huang and A. S. Mohan, "A hybrid boundary condition for robust particle swarm optimization," in IEEE Antennas and Wireless Propagation Letters, vol. 4, pp. 112-117, 2005, doi: 10.1109/LAWP.2005.846166.

[89] S. Xu and Y. Rahmat-Samii, "Boundary Conditions in Particle Swarm Optimization Revisited," in IEEE Transactions on Antennas and Propagation, vol. 55, no. 3, pp. 760-765, March 2007, doi: 10.1109/TAP.2007.891562.

[90] L. Linkous, J. Lundquist, *pso_basic* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/pso_Python/tree/pso_basic

[91] L. Linkous, *pso_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/pso_Python/tree/pso_quantum

[92] L. Linkous, J. Lundquist, *cat_swarm_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python

[93] L. Linkous, J. Lundquist, *sand_cat_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python/tree/sand_cat_Python

[94] L. Linkous, *cat_swarm_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python/tree/cat_swarm_quantum

[95] L. Linkous, J. Lundquist, *chicken_swarm_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/chicken_swarm_Python

[96] L. Linkous, *chicken_swarm_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/chicken_swarm_Python/tree/chicken_swarm_quantum

[97] L. Linkous, *sweep_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/sweep_Python

[98] L. Linkous, *bayesian_optimization_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/bayesian_optimization_Python

[99] L. Linkous, *surrogate_modeling_optimization* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/surrogate_modeling_optimization

[100] J. Lundquist, L. Linkous, *multi_glods_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/jonathan46000/multi_glods_Python

[101] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[102] Jun Sun, Bin Feng and Wenbo Xu, "Particle swarm optimization with particles having quantum behavior," Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 325-331 Vol.1, doi: 10.1109/CEC.2004.1330875.

[103] Jun Sun, Wenbo Xu and Bin Feng, "A global search strategy of quantum-behaved particle swarm optimization," IEEE Conference on Cybernetics and Intelligent Systems, 2004., Singapore, 2004, pp. 111-116 vol.1, doi: 10.1109/ICCIS.2004.1460396.

[104] L. dos Santos Coelho and P. Alotto, "Global Optimization of Electromagnetic Devices Using an Exponential Quantum-Behaved Particle Swarm Optimizer," in IEEE Transactions on Magnetics, vol. 44, no. 6, pp. 1074-1077, June 2008, doi: 10.1109/TMAG.2007.916032.

[105] Shuyuan Yang, Min Wang and Licheng jiao, "A quantum particle swarm optimization," Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), Portland, OR, USA, 2004, pp. 320-324 Vol.1, doi: 10.1109/CEC.2004.1330874.

[106] A. L. Custódio and J. F. A. Madeira, "GLODS: Global and Local Optimization using Direct Search," Journal of Global Optimization, vol. 62, no. 1, pp. 1–28, Aug. 2014, doi: https://doi.org/10.1007/s10898-014-0224-9.

[107] Carl Edward Rasmussen; Christopher K. I. Williams, "Relationships between GPs and Other Models," in Gaussian Processes for Machine Learning , MIT Press, 2005, pp.129-150.

[108] Carl Edward Rasmussen; Christopher K. I. Williams, "Model Selection and Adaptation of Hyperparameters," in Gaussian Processes for Machine Learning , MIT Press, 2005, pp.105-128.

[109] D. Xiu and G. Karniadakis, "The Wiener-Askey Polynomial Chaos for Stochastic Differential Equations," *SIAM J. Sci. Comput*., 2002, doi: https://doi.org/10.1137/S1064827501387826.

[110] L. Linkous and E. Topsakal, "Machine Learning Assisted Optimization Methods for Automated Antenna Design," *2024 United States National Committee of URSI National Radio*

Science Meeting (USNC-URSI NRSM), Boulder, CO, USA, 2024, pp. 377-378, doi: 10.23919/USNC-URSINRSM60317.2024.10464597.

[111] L. Linkous, J. Lundquist, M. Suche, and E. Topsakal, "Machine Learning Assisted Hyperparameter Tuning for Optimization," *2024 IEEE International Symposium on Antennas and Propagation and ITNC-USNC-URSI Radio Science Meeting*, Florence, Italy, 2024

[112] J. D. Lundquist, L. Linkous, K. Dyson, R. Ayala, and E. Topsakal, "MultiGLODS in Electromagnetics", in IEEE Transactions on Antennas and Propagation

[113] B. Y. Qu, J. J. Liang, Z. Y. Wang, Q. Chen, and P. N. Suganthan, "Novel benchmark functions for continuous multimodal optimization with comparative results," Swarm and Evolutionary Computation, vol. 26, pp. 23–34, 2016. doi:10.1016/j.swevo.2015.07.003

[114] K. Deb, L. Thiele, M. Laumanns, and Eckart Zitzler, "Scalable Test Problems for Evolutionary Multiobjective Optimization," *Springer eBooks*, pp. 105–145, Jan. 2005, doi: https://doi.org/10.1007/1-84628-137-7_6.

[115] D. Bingham, "Virtual Library of Simulation Experiments," Optimization Test Functions and Datasets, https://www.sfu.ca/~ssurjano/optimization.html (accessed Mar. 31, 2024).

[116] T. T. Binh and U. Korn, "Scalar optimization with linear and Nonlinear Constraints using Evolution Strategies," *Lecture notes in computer science,* pp. 381–392, Jan. 1997, doi: https://doi.org/10.1007/3-540-62868-1_130.

[117] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, Jun. 2000, doi: https://doi.org/10.1162/106365600568202.

[118] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," Evolutionary Computation, vol. 3, no. 1, pp. 1–16, Mar. 1995. doi:10.1162/evco.1995.3.1.1

CHAPTER 9
Related Publications and Open-Source

9.1 Journals and Magazines

Related Publications

L. Linkous, J. Lundquist, M. Suche, and E. Topsakal, " AntennaCAT: Automated Antenna Design with Machine Learning Assisted Optimization," in *IEEE Antennas and Propagation Magazine* [**under review**]

Other Publications

Laura Ellwein Fix, J. D. Khoury, R. R. Moores, L. Linkous, M. C. Brandes, and H. J. Rozycki, “Theoretical open-loop model of respiratory mechanics in the extremely preterm infant,” vol. 13, no. 6, pp. e0198425–e0198425, Jun. 2018, doi: https://doi.org/10.1371/journal.pone.0198425.

R. Eini, L. Linkous, N. Zohrabi, and S. Abdelwahed, “Smart building management system: Performance specifications and design requirements,” *Journal of Building Engineering*, vol. 39, p. 102222, Jul. 2021, doi: https://doi.org/10.1016/j.jobe.2021.102222.

J. D. Lundquist, L. Linkous, U. Hasni and E. Topsakal, "Indirect Applications of Additive Manufacturing for Antennas," in *IEEE Open Journal of Antennas and Propagation*, vol. 4, pp. 434-445, 2023, doi: 10.1109/OJAP.2023.3265691.

J. Jones, L. Linkous, L. Mathews-Ailsworth, R. Vazquez-Miller, E. Chance, J. Carter, I. Saneda, “Smart Little Campus Food Pantries: Addressing food insecurity at Virginia Commonwealth University,” *Journal of agriculture, food systems, and community development*, pp. 1–17, Apr. 2024, doi: https://doi.org/10.5304/jafscd.2024.133.016.

J. Lundquist, L. Linkous, MKE. Piper, Z. Sickey, K. Zimmet, I. Mendoza, S. Suresh, and E. Topsakal, "Textile-Based Inkjet-Printed RFIDs: Exploring wearable antennas in the real world [Bioelectromagnetics]," in *IEEE Antennas and Propagation Magazine*, vol. 66, no. 1, pp. 50-62, Feb. 2024, doi: 10.1109/MAP.2023.3334671.

J. Lundquist, L. Linkous and E. Topsakal, "Programmable Liquid Microwave GRIN Lens," *2023 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2023, pp. 66-67, doi: 10.23919/USNC-URSINRSM57470.2023.10043112.

9.2 Conference Papers

Related Publications

L. Linkous, E. Karincic, J. Lundquist and E. Topsakal, "Automated Antenna Calculation, Design and Tuning Tool for HFSS," *2023 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2023, pp. 229-230, doi: 10.23919/USNC-URSINRSM57470.2023.10043119.

E. Karincic, E. Topsakal, and L. Linkous, "Patch Antenna Calculations and Fabrication Made Simple for Cyber Security Research," in *2023 ASEE Annual Conference & Exposition*, 2023.

L. Linkous, J. Lundquist and E. Topsakal, "AntennaCAT: Automated Antenna Design and Tuning Tool," *2023 IEEE USNC-URSI Radio Science Meeting (Joint with AP-S Symposium)*, Portland, OR, USA, 2023, pp. 89-90, doi: 10.23919/USNC-URSI54200.2023.10289238.

L. Linkous and E. Topsakal, "Machine Learning Assisted Optimization Methods for Automated Antenna Design," *2024 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2024, pp. 377-378, doi: 10.23919/USNC-URSINRSM60317.2024.10464597.

L. Linkous, J. Lundquist, M. Suche, and E. Topsakal, "Machine Learning Assisted Hyperparameter Tuning for Optimization," *2024 IEEE International Symposium on Antennas and Propagation and ITNC-USNC-URSI Radio Science Meeting*, Florence, Italy, 2024

Other Publications

R. Eini, L. Linkous, N. Zohrabi, and S. Abdelwahed, “A testbed for a smart building,” *Proceedings of the Fourth Workshop on International Science of Smart City Operations and Platforms Engineering - SCOPE ’19*, 2019, doi: https://doi.org/10.1145/3313237.3313296.

L. Linkous, N. Zohrabi and S. Abdelwahed, "Health Monitoring in Smart Homes Utilizing Internet of Things," *2019 IEEE/ACM International Conference on Connected Health: Applications,*

Systems and Engineering Technologies (CHASE), Arlington, VA, USA, 2019, pp. 29-34, doi: 10.1109/CHASE48038.2019.00020.

N. Zohrabi, P. Martin, M. Kuzlu, L. Linkous, *et. al*, "OpenCity: An Open Architecture Testbed for Smart Cities," *2021 IEEE International Smart Cities Conference (ISC2)*, Manchester, United Kingdom, 2021, pp. 1-7, doi: 10.1109/ISC253183.2021.9562813.

N. Zohrabi, L. Linkous, *et al*., "Towards Sustainable Food Security: An Interdisciplinary Approach," *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, Atlanta, GA, USA, 2021, pp. 463-470, doi: 10.1109/SWC50871.2021.00069.

J. Lundquist, L. Linkous and E. Topsakal, "Mechanically Configurable, Capacitively Coupled, Disk Loaded Monopole Driven Corner Reflector," *2022 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2022, pp. 12-13, doi: 10.23919/USNC-URSINRSM57467.2022.9881449.

E. Karincic, E. Topsakal, L. Linkous, “Patch Antenna Calculations and Fabrication Made Simple for Cyber Security Research”, 2023 ASEE Annual Conference & Exposition, 2023

J. Lundquist, L. Linkous and E. Topsakal, "Active Dual Band Frequency Selective Surface for 2.4 GHz Wi-Fi and Wi-Fi 6E," *2023 7th International Electromagnetic Compatibility Conference (EMC Turkiye)*, İstanbul, Turkiye, 2023, pp. 1-4, doi: 10.1109/EMCTurkiye59424.2023.10287499.

E. Karincic, L. Linkous, E. Topsakal “From Classroom to Career with Practical Network Training”, 2024 ASEE Annual Conference & Exposition, Portland, OR, 2024.

M. Suche, L. Linkous and E. Topsakal, "Textile RFIDs for Healthcare Applications," *2024 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*, Boulder, CO, USA, 2024, pp. 143-144, doi: 10.23919/USNC-URSINRSM60317.2024.10464636.

J. Lundquist, L. Linkous and E. Topsakal, "Reconfigurable Diode-Based and Liquid Metal Antenna for 5 GHz Wi-Fi," *2024 United States National Committee of URSI National Radio*

Science Meeting (USNC-URSI NRSM), Boulder, CO, USA, 2024, pp. 287-288, doi: 10.23919/USNC-URSINRSM60317.2024.10464775.

9.3 Public Repositories

All software, aside from the proprietary electromagnetics software used for simulations, is available open source online. The software repositories summarized here are written in Python, though versions in other languages may be available from their respective authors.

AntennaCAT Software:

L. Linkous, *AntennaCalculationAutotuningTool* (Version 2024), GitHub, [source code]. Available: https://github.com/LC-Linkous/AntennaCalculationAutotuningTool

The AntennaCalculator:

Command Line Version: E. Karincic, L. Linkous, *AntennaCalculator* (Version 1.0), GitHub, [source code]. Available: https://github.com/Dollarhyde/AntennaCalculator

GUI Version: L. Linkous, E. Karincic, *AntennaCalculator* (Version 2.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/AntennaCalculator

Particle Swarm Optimizers:

J. Lundquist, L. Linkous, *pso_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/jonathan46000/pso_Python

L. Linkous, J. Lundquist, *pso_basic* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/pso_Python/tree/pso_basic

L. Linkous, *pso_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/pso_Python/tree/pso_quantum

Cat Swarm Optimizers:

L. Linkous, J. Lundquist, *cat_swarm_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python

L. Linkous, J. Lundquist, *sand_cat_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python/tree/sand_cat_Python

L. Linkous, *cat_swarm_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/cat_swarm_Python/tree/cat_swarm_quantum

Chicken Swarm Optimizer:

L. Linkous, J. Lundquist, *chicken_swarm_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/chicken_swarm_Python

L. Linkous, *chicken_swarm_quantum* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/chicken_swarm_Python/tree/chicken_swarm_quantum

Sweep/Grid Search:

L. Linkous, *sweep_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/sweep_Python

Bayesian Optimizer with Surrogate Models:

L. Linkous, *bayesian_optimization_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/bayesian_optimization_Python

L. Linkous, *surrogate_modeling_optimization* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/surrogate_modeling_optimization

MultiGLODS Optimizer:

J. Lundquist, L. Linkous, *multi_glods_Python* (Version 1.0), GitHub, [source code]. Available: https://github.com/jonathan46000/multi_glods_Python

Optimization Function Library:

L. Linkous, *objective_function_suite* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/objective_function_suite

Conference Paper Repositories:

L. Linkous, *2024_URSI_NRSM_1265* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/2024-URSI-NRSM-1265

L. Linkous, *2024_APS_URSI_3323* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/2024-APS-URSI-3323

GeneticCAT Repository:

L. Linkous, *GeneticCAT* (Version 1.0), GitHub, [source code]. Available: https://github.com/LC-Linkous/GeneticCAT