

Usage of ChatGPT for Engineering Design and Analysis Tool Development

Kristopher C Pierson¹

NASA Glenn Research Center, Brook Park, Ohio 44135

Matthew J Ha²

HX5, LLC, NASA Glenn Research Center, Brook Park, Ohio 44135

ChatGPT, a generative AI large language model, has recently captured significant attention in both the computer science community and the broader public domain. It has demonstrated a wide range of capabilities, from answering simple questions to writing fully functional computer code. This study spotlights both the capabilities and limitations of ChatGPT when addressing engineering problems. The model's capacity to generate practical engineering tools is highlighted through an example of a prompt that leads to an interactive plotting tool, enabling the examination of the fluid boundary layer around a fan blade. Subsequently, the paper also uncovers potential pitfalls in ChatGPT's application, shown through an unsuccessful attempt to use ChatGPT to automate a process in Ansys Workbench through scripting. The research further investigates ChatGPT's proficiency in addressing inquiries and providing explanations about the functionalities of OpenMDAO, an open-source, multidisciplinary design, analysis, and optimization tool developed at NASA Glenn Research Center. Finally, an optimization methodology, developed with ChatGPT's help, is applied to the structural optimization of a fan blade. The developed optimization method utilizes T-Blade3 for geometry generation, Ansys Mechanical for meshing and finite element analysis, and sci-kit learn's MLPRegressor method to generate a trained neural network model of the design space. OpenMDAO is then used to find the optimal point within the design space. The outcome is a significant reduction in stress in the optimized model—less than one-fifth of the stress value in the baseline model.

I. Nomenclature

AI	=	Artificial Intelligence
CFD	=	Computational Fluid Dynamics
FEA	=	Finite Element Analysis
FoS	=	Factor of Safety
GPT	=	Generative Pre-trained Transformer
LHS	=	Latin Hypercube Sampling or Latin Hypercube Sample
LLM	=	Large Language Model
ML	=	Machine Learning
MLP	=	Multilayer Perceptron
NN	=	Neural Network

¹Research Engineer, Multiscale and Multiphysics Modeling Branch, AIAA Member.

²Design Research Engineer, Turbomachinery and Turboelectric Systems Branch, AIAA Member.

dm'_s	=	Normalized differential arc length of a streamline
m_s	=	Streamline position
r	=	radial coordinate in (x, r, θ) cylindrical coordinate system
r_s	=	Streamline r coordinate
x	=	Axial coordinate in (x, r, θ) cylindrical coordinate system
x_s	=	Streamline x coordinate
θ	=	Left-handed rotational coordinate in (x, r, θ) cylindrical coordinate system
$\sigma_{vm,max}$	=	Maximum von-Mises stress
σ_{ys}	=	Yield Strength
$\Delta m'_{25}$	=	Sweep control point at 25% span (similarly for other span location)
$\Delta \theta_{25}$	=	Lean control point at 25% span (similarly for other span locations)

II. Introduction

The incorporation of generative artificial intelligence (AI), particularly large language models (LLMs), into engineering tool development signifies a paradigm shift in engineering practices. ChatGPT is an AI chatbot developed by OpenAI [1] that has recently become a topic of interest in popular culture. There are several versions of it, some of which are available to use freely on the OpenAI website. The current free version of ChatGPT is based on Generative Pre-trained Transformer 3.5 (GPT-3.5). A professor at Wharton School at the University of Pennsylvania recently wrote a report on GPT-3.5 stating that the chatbot would score a B or B- on the final exam in their operations management course [2]. Despite this high praise for GPT-3.5, the authors have found that the newest version of ChatGPT, which currently requires a subscription, performs much better at programming tasks compared to GPT-3.5.

The newest version of ChatGPT is based on GPT-4; this is the model that is the focus of this paper. Further references to ChatGPT refer to the version based on GPT-4. GPT-4 has shown remarkable ability in a wide variety of academic fields. It has outscored, significantly in many instances, GPT-3.5 on every exam that the authors have seen reported. GPT-4 scored in the 90th percentile on the Uniform Bar Exam and the 80th percentile in the GRE Quantitative exam according to a March 2023 report by OpenAI [3]. GPT-4 also achieved passing grades on a mock 500-question neurosurgical written board examination [4].

These are some extremely impressive accolades for any person, much less an AI language model. However, it is still important to consider the limitations of such tools. Like a person, ChatGPT is only proficient in topics on which it has been sufficiently trained. Unlike most people, when ChatGPT is insufficiently trained on a topic, it often returns a response which appears to be legitimate but may be entirely made up. This is known as an AI hallucination [5], and it is a major hurdle to overcome so that these models can be even more useful in engineering work.

Still, ChatGPT has shown itself to be an incredibly valuable tool in this work. One key to success in the usage of ChatGPT is drafting an effective prompt. If a prompt is well-crafted, the user is much more likely to successfully achieve their objective. Prompts should be written clearly and concisely with information that is accurate and pertinent to the objective. The level of specificity depends highly on the type of output that is desired. Sometimes, one may simply desire that ChatGPT produces a general framework that can later be modified after the backbone has been created. Other times, it may be better to provide specific instructions, such as axis labels in a plot.

In the following sections, ChatGPT is presented as a valuable engineering resource with notable pitfalls. In the design optimization work highlighted in Section V to Section IX, the language model is used to explain the functions of OpenMDAO, an open-source multidisciplinary analysis and optimization framework developed at NASA Glenn Research Center [6]. ChatGPT is also used to write Python code that generates samples of the design space using Latin Hypercube Sampling (LHS). Finally, ChatGPT recommends the usage of a neural network (NN) as a method of representing the design space. The NN is trained using the design points of the LHS. This NN is then evaluated using a gradient based optimization algorithm. The design optimization methodology developed with ChatGPT's help is used to drive the factor of safety (FoS) as high as possible through modification of only the lean and sweep control points in T-Blade3 [7,8].

III. ChatGPT For Simple Tool Creation

The first example of ChatGPT's programming capability is the creation of a visualization tool to inspect the boundary layer of air on a fan blade. In this section, the specifics of the CFD simulation are not presented. The Python code produced by ChatGPT is the focus.

A data structure is already created and there is now interest in developing an effective way to visualize the data. The authors are well-versed python programmers, so they already know the packages that they wish to use to complete

this task. Those packages are Matplotlib [9], a matrix plotting library, and PyQt5 [10], a cross-platform GUI toolkit. However, the method of producing such a tool without using a GUI designer such as Qt Designer [11] is not well understood. This section shows that ChatGPT has no issue creating this type of tool using Python code alone.

The aim is to integrate the boundary layer plot into a PyQt5 window and leverage PyQt5 to select which region on the fan surface to visualize. Two contrasting examples are provided: one illustrating a brief, insufficient explanation, and another delivering a more comprehensive, satisfactory description of the desired Python program. The prompt for the brief explanation of the desired Python program is shown below.

Prompt:

I have data that I would like to plot in an interactive graph using PyQt5 with an embedded matplotlib plot. Please write a Python code to do this based on the following information.

The data to plot has the following format:

VelocityMagnitude[i_position, j_position, t_position]

DistanceFromWall[i_position, j_position]

I want to be able to toggle i_position and t_position in the GUI and plot all j points.

The output from this request produces a working interactive figure that plots the boundary layer. The interactive plot is shown in Fig. 1. Though the code functions as described in the prompt, the output leaves several things to be desired. First, ChatGPT creates a class for the interactive plot, but this class does not take any inputs. Therefore, it must be placed below the data definition within a script, and the data cannot be imported. This issue can easily be corrected by programming the class to take inputs of the variables that contain the velocity magnitude and the distance from the wall. Second, it placed the velocity magnitude on the y-axis and distance from the wall on the x-axis, which is the opposite of the typical way to show such data. This type of data is also often too fine to include markers, as they may overlap each other for certain positions. Finally, this data is always in the standard SI unit system, so it is desirable to include units in the plot.

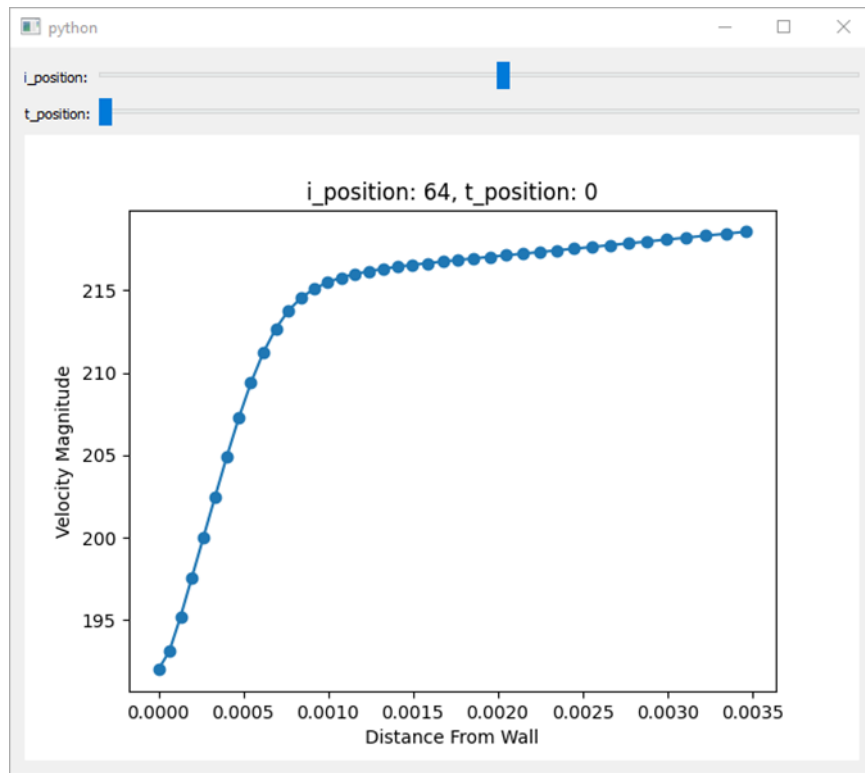


Fig. 1 Boundary Layer Visualizer produced with inadequate description of request

Based upon the deficiencies of the first plot, a more descriptive prompt is now used to produce an improved interactive figure. This prompt is much more detailed in describing the format that is desired, which results in a better tool produced by ChatGPT.

Prompt:

I have data that I would like to plot in an interactive matplotlib graph. I'd like the interactive plot to be controlled using PyQt5. Please write a Python code to do this based on the following information.

The data to plot has the following format:

VelocityMagnitude[i_position, j_position, t_position]

It also has a corresponding set of data, which is the distance from the wall.

DistanceFromWall[i_position, j_position]

I would like this to be its own class that takes inputs for VelocityMagnitude and DistanceFromWall as its inputs. I want to be able to toggle i_position and t_position in the GUI and plot all j points. The GUI should print the i_position and t_position to the screen. The toggles should be a horizontal sliders. The velocity variable should be plotted on the x-axis, and the distance should be plotted on the y-axis. Please add labels to them including units, this is all in standard mks. I would like the plot to be a line plot without markers.

I also want this to be easy to import, so please make a separate function that takes care of all the extra PyQt stuff like the show method.

The second boundary layer visualization tool functions exactly as desired. The class initialization now takes inputs of the variables used to generate the plot. The code also included a function that makes it very easy to import into any other script. The axes are aligned as requested and include labels with units. The code is included in Appendix A.

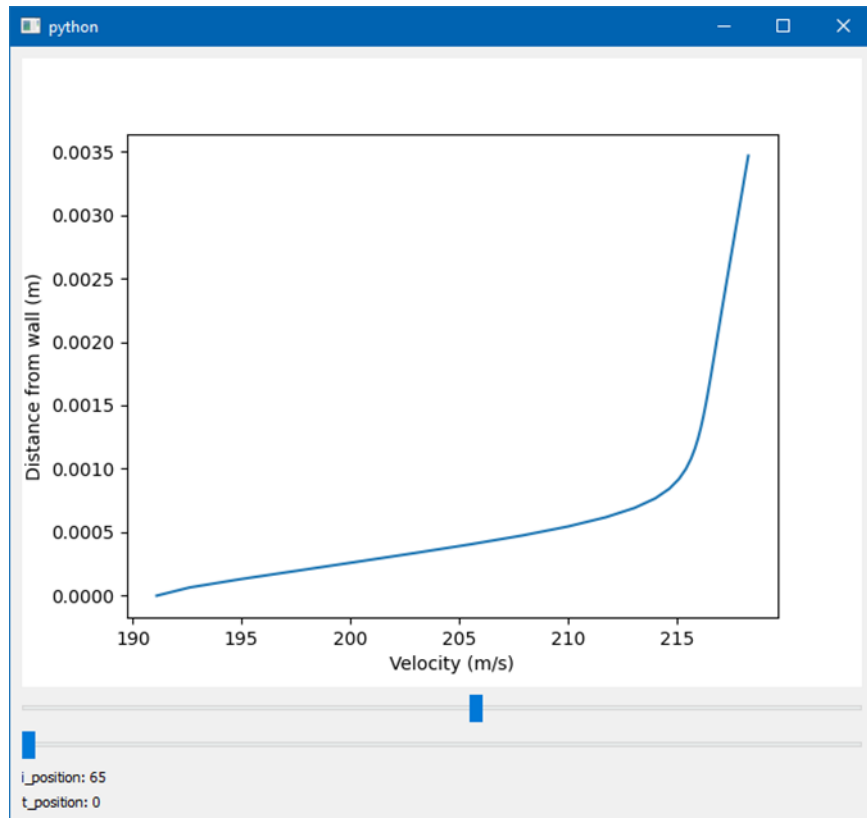


Fig. 2 Screen capture of ChatGPT boundary layer visualizer produced with adequate explanation of the request

The length of the Python code output is nearly at the maximum output size for a ChatGPT response, so it is difficult to create a more sophisticated program than this. Conceivably, the user could request that ChatGPT create various functions that could later be pieced together into a larger, more complicated program. However, the approach that the

authors have taken is to ask ChatGPT programming questions and then write the code themselves. This is discussed in Section V.

IV. ChatGPT Hallucination Issues

As part of the development of the optimization method that is described in Section VIII of this paper, it was necessary to automate an Ansys Mechanical analysis. The simulation automation was completed before ChatGPT (based on GPT-4) was released. So, a method of doing this task was already known to the authors. The most complicated portion of this process is changing the geometry of the blade. The process requires an external Python-based script that is an Ansys Workbench Journal File [12] with an embedded JavaScript [13] that is sent to Ansys DesignModeler. So, the Ansys Workbench script should contain a large string of JavaScript which controls the geometry update in DesignModeler.

Multiple prompts are attempted with the hopes that ChatGPT can write the portion of the code that changes the fan geometry's source file. Each of these prompts results in a hallucination [5] where ChatGPT outputs a solution that reads like it could be correct if one is not familiar with Ansys scripting, but code is entirely nonsense. The first prompt requests that ChatGPT write an Ansys Workbench Journal File that uses DesignModeler to change the geometry of a fan blade.

Prompt:

Please generate an Ansys Workbench Journal File to change geometry in Ansys DesignModeler. The geometry object that I wish to change is called "Blade" in the DesignModeler tree outline and the file describing the geometry is a STEP file.

The response produces a script and the assumptions made in creating it. ChatGPT finishes with important items to remember when running this type of script. However, the script is not functional. The commands are made up and, based on its response, ChatGPT does not even know what type of scripting language that is used in Ansys Workbench.

Response:

Sure, you want to generate an Ansys Workbench journal file to automate the change of a geometry object named "Blade" in the Ansys DesignModeler, which is originally described by a STEP file. Below is a simple example of how you might automate this task.

This journal assumes that:

You have an ANSYS Workbench project with a DesignModeler geometry component.

The geometry named "Blade" is already present in the DesignModeler tree.

You want to replace this geometry with another STEP file, let's assume it's named "NewBlade.step" and is located in the "C:\Users\YourUsername\Documents" directory.

```
! ANSYS Workbench Journal File for Updating Geometry in DesignModeler
```

```
! Open the active DesignModeler session
/activate ACT
```

```
! Open the DesignModeler application
AG.ACTivate DesignModeler
```

```
! Delete the existing "Blade" geometry
AG.DeleteItem("Blade")
```

```
! Import the new STEP file to replace the old geometry
AG.Import("C:\Users\YourUsername\Documents\NewBlade.step")
```

```
! Rename the newly imported geometry as "Blade"
AG.RenameItem("NewBlade", "Blade")
```

```
! Update and save the changes in the geometry
AG.Update()
AG.SaveProject()
```

```
! Close the DesignModeler session
AG.Exit()
```

Remember:

1. Replace C:\Users\YourUsername\Documents\NewBlade.step with the actual path to your new STEP file.
2. Journal files can be run from the ANSYS Workbench environment, or via command line. Depending on the specifics of your setup and what you're looking to accomplish, you might need to modify the provided script to better suit your needs.
3. Always back up your original project and data before running any script or automation to prevent accidental data loss.

This response is followed with a question, “What language is this written in?” ChatGPT responds by stating that “the code I provided is written in ANSYS DesignModeler scripting language, which is a part of the ANSYS Workbench scripting environment. This language is specifically designed for automating tasks within the ANSYS Workbench and DesignModeler environment. The commands, prefixed with AG, are special functions provided by ANSYS to interact with DesignModeler.”

So, a few things are obviously wrong with this response outside of the script. First, DesignModeler is not scripted in the same language as Workbench. Workbench has Python-based journaling, and DesignModeler scripting is done in JavaScript. Second, there is no AG object that exist within the Workbench scripting environment, and since the AG object does not even exist, it certainly does not send commands to DesignModeler as ChatGPT claims.

A second prompt is made where the problem is simplified for ChatGPT. In the second prompt, Ansys Workbench is not mentioned, and the goal is to simply create a script that will automate changing the geometry source in DesignModeler.

Prompt:

Please generate a script for me that I can use in Ansys DesignModeler to change the source of a geometry object. The name of the geometry object in the tree outline is "Blade", and the file format is STEP.

Response:

Certainly! Changing the source of a geometry object in Ansys DesignModeler would typically involve replacing the geometry with another source, such as importing a different STEP file.

To help you with the task, here is a script written in Ansys DesignModeler's scripting language, which is based on JavaScript:

```
// Define the main function
function ChangeSource() {
  // Access the geometry object named "Blade"
  var bladeObject = DM.FindObject("Blade");
  if (bladeObject === null) {
    DM.AddMessage("Error: 'Blade' object not found.", 2);
    return;
  }

  // Remove the original "Blade" object from the tree
  bladeObject.Delete();

  // Path to the new STEP file you want to use as the source.
  // Replace 'path_to_step_file.stp' with the actual path to your file.
  var newFilePath = "path_to_step_file.stp";

  // Import the new STEP file
  var importResult = DM.Import(newFilePath, "step");

  // Check if the import was successful
  if (importResult === 0) {
    DM.AddMessage("STEP file imported successfully.", 1);

    // Rename the imported geometry to "Blade"
    var importedGeometry = DM.ActiveObject();
    if (importedGeometry !== null) {
      importedGeometry.Rename("Blade");
    } else {
      DM.AddMessage("Error: Imported geometry could not be renamed.", 2);
    }
  }
}
```

```

    }
} else {
    DM.AddMessage("Error importing STEP file.", 2);
}
}

// Execute the main function
ChangeSource();

```

Instructions:

- Replace the text "path_to_step_file.stp" with the actual path to your new STEP file.
- Execute this script within Ansys DesignModeler.

Remember, this script assumes you already have an object named "Blade" in your tree. If the object doesn't exist, the script will give an error message. The script deletes the old "Blade" object and imports the new geometry from the specified STEP file, renaming it to "Blade".

This response is written in the correct language, and it follows the steps that are needed when scripting this process. A brief outline of operations in the script are listed below:

1. Find the current geometry object which is to be replaced
2. Delete the current blade geometry
3. Import the new geometry

So, ChatGPT interprets the request correctly. However, it is using objects that do not exist. For example, it is using an object called DM which does not exist in DesignModeler scripting. When the following question, "What is DM in the script above?", ChatGPT responded that "when scripting within Ansys DesignModeler, DM is a global object that provides access to the functions and features of the DesignModeler environment." None of this is true.

Several other attempts were made to successfully get ChatGPT to produce something that is usable. However, none of them were successful. One of the attempts that is not presented in this paper resulted in a rather comical creation of a new programming language called AGL, which ChatGPT claimed stands for ANSYS Geometry Interface Language. ChatGPT summarized AGL by stating "AGL is a scripting language developed by ANSYS for its DesignModeler tool. DesignModeler is a part of the ANSYS suite of simulation software, and it's focused on creating and editing 3D geometry for simulation." In case it needs to be stated, the reader should note that AGL is not something that actually exists.

The issue of hallucination [5] is among the most critical challenges to the creation of reliable generative AI systems. The current AI models tend to provide an answer to a question, regardless of whether that answer is correct. If the AI system has been sufficiently trained on the material that it is prompted about, the chance for a hallucination is reduced. However, for proprietary software that is not that well-documented with openly available information, the training is limited, thus the responses are less reliable.

V. ChatGPT Contribution to Optimization Scheme

While ChatGPT struggled to create anything usable from the prompts in the previous section, it proved to be extremely effective in explaining how to use OpenMDAO [6] and other open-source Python packages that are used in the optimization scheme presented in Section VIII and IX. ChatGPT is effective in both explaining how certain functions work in OpenMDAO as well as proposing additional ideas to improve the optimization process.

OpenMDAO has various libraries that can be used to set up an optimization routine. ChatGPT's primary role with respect to the OpenMDAO-based tool development was the explanations of the OpenMDAO Python objects to allow the first author to quickly come up to speed on the optimization tool. To show the utility of ChatGPT in explaining how Python code functions, ChatGPT is prompted with the custom OpenMDAO class used in the next section of this paper. This code is included in Appendix B.

Prompt:

Below is a Python code that I am going to ask you questions about. Please provide a brief summary of the code. I will also follow this prompt with additional questions about the code.
[In ChatGPT prompt, the code from Appendix B was pasted here.]

Response:

The provided Python code seems to be part of a simulation optimization setup possibly related to turbomachinery, judging by the naming conventions and libraries used. Here's a brief summary:

1. Import Statements:
 - The code imports necessary modules and classes including InputFileGenerator and FileParser from openmdao.utils.file_wrap, SimulationControlClass from TurboOpt3DSimControl, OpenMDAO's API (openmdao.api), shutil, os, time, and numpy.
2. Global Variable Declaration:
 - LastIteration is declared as a global variable to keep track of the last successful iteration in case the simulation process is interrupted.
3. Class Definition (TurboOPT3D):
 - TurboOPT3D class is defined which inherits from OpenMDAO's ExternalCodeComp class.
 - In the setup method:
 - Inputs (like lean and sweep values at different sections of a blade) and outputs (FoS - Factor of Safety, and W - possibly Weight) are defined.
 - Instances of InputFileGenerator, SimulationControlClass, and some other variables are initialized.
 - In the compute method:
 - The method orchestrates a simulation process by preparing input files, running a simulation, performing structural analysis, and logging the results to a file.
 - The results of each iteration are appended to a result file, which logs the iteration number, FoS, weight, and input parameters.
 - WriteTblade3File method:
 - Modifies a template input file for a simulation tool (presumably TBlade3) with updated values of lean and sweep at different blade sections.
 - StructuralAnalysis method:
 - Attempts to run a structural analysis (using a method from SimulationController) on the simulation results, handling any exceptions that occur, and logging failures to the result file.
4. Error Handling:
 - If a simulation failure occurs during structural analysis, it's logged in the results file, and if too many consecutive failures occur, the program will exit.

This code is structured to automate the process of running simulations, modifying input files, and logging results within an optimization framework provided by OpenMDAO. The TurboOPT3D class encapsulates the logic needed to interact with external simulation tools, handle file I/O, and manage the simulation process.

This response is an excellent summary of the functions of the Python class that was provided. The only mistake is confusing T-Blade3 for the analysis tool, which is an understandable error given the limited amount of information provided to ChatGPT for this exercise. If ChatGPT is provided with the information that T-Blade3 is a parametric geometry generating tool for use with turbomachinery, then it is better able to analyze the code, which results in a more informative response to the prompter.

The more information that ChatGPT has on a certain topic, either from its training or from a prompt, the better it is at answering questions. In terms of creating an effective prompt, it is ideal to be concise while still including a reasonable amount of pertinent information. If ChatGPT is not well-trained on a certain topic, the amount of background information that should be included in the prompt grows. T-Blade3 is not a well-known tool, so it is not surprising the ChatGPT is unfamiliar with it.

Prompt:

Just a bit of context for you, T-Blade3 is a parametric geometry generating tool for use with Turbomachinery. Can you please explain to me how editing the T-Blade 3 input file works in the TurboOPT3D class?

Response:

Certainly! The TurboOPT3D class within the provided code snippet has a method named WriteTblade3File that is responsible for modifying the input file for T-Blade3, which is a parametric geometry generating tool for turbomachinery.

Here's a step-by-step breakdown of how the WriteTblade3File method operates to edit the T-Blade3 input file:

1. Method Signature:
 - The WriteTblade3File method accepts two arguments: tbladeinputfile (which is the path to the original input file for T-Blade3) and inputs (which contains the new values for parameters like lean and sweep at different sections of a blade).
2. Generated File Naming:
 - A new file name (tbladeinputfile_generated) is created by appending _gen to the original file name tbladeinputfile. This new file will temporarily hold the updated input data before it overwrites the original file.
3. Input File Parser Initialization:
 - The method initializes the InputFileGenerator instance (self.writeParser) with the original input file (tbladeinputfile) as the template and the generated file name (tbladeinputfile_generated) as the output.
4. Modifying Lean Parameters:
 - self.writeParser is then used to find and replace the values of lean parameters at different blade sections:
 - The anchor is reset and marked at the line containing 'Control points for lean'.
 - The transfer_var method of self.writeParser is used to replace the existing values of lean parameters at different blade sections (lean_025, lean_050, lean_100) in the original file with the new values provided in inputs.
 - The generate method of self.writeParser is called to write these changes to the generated file.
5. Modifying Sweep Parameters:
 - Similar to the lean parameters, the sweep parameters are updated:
 - The anchor is reset and marked at the line containing 'Control points for sweep'.
 - The transfer_var method of self.writeParser is used to replace the existing values of sweep parameters at different blade sections (sweep_025, sweep_050, sweep_100) in the original file with the new values provided in inputs.
 - The generate method of self.writeParser is called again to write these changes to the generated file.
6. Overwriting Original File:
 - Finally, the shutil.copyfile method is used to overwrite the original input file (tbladeinputfile) with the updated data from the generated file (tbladeinputfile_generated).
 - The generated file (tbladeinputfile_generated) is deleted using os.remove to clean up.

Through this process, the WriteTblade3File method automates the editing of the T-Blade3 input file to reflect the new lean and sweep parameters for a given iteration, making it ready for the next simulation run. This automated editing of input files is crucial for facilitating parametric studies or optimization runs where the geometry parameters are varied systematically.

This is a much more detailed overview of the WriteTblade3File method. ChatGPT does an excellent job explaining how this works and makes no mistakes this time. This utility could prove to be very useful in getting engineers up to speed on how a code works. It could also be very useful in documenting code. There are a lot of possibilities with one significant hinderance. Submitting information like this through the typical web interface is not secure. This issue is briefly discussed in the conclusion.

The initial approach to optimization utilized a genetic algorithm (GA) directly on the fan blade FEA model. This proved to be difficult for this problem. The simulation time was too long for the GA implementation that was being used. An attempt to use a gradient based optimization approach led to difficulties with becoming stuck in local minima. Additionally, it proved to be difficult to restart the optimization process if there was a simulation failure. So, a new approach is desired. The intent is to run a set of cases and use that data to fit a polynomial to the objective function with the design variables as independent variables. Latin hypercube sampling (LHS) [14] is chosen as the method to survey the design space. ChatGPT is prompted to see if there is any way to do this with a Python package. It responds that there are three Python packages available to do LHS: PyDOE, SciPy and SALib. Since SciPy is the most commonly used package of those listed, it is chosen as the tool. Now, ChatGPT is tasked with writing the code that will generate the LHS and save it to a text file.

Prompt:

Please use SciPy to create a LHS with 6 design variables. I also want to be able to set the size of the range of the variables and the center point of the variables. The variables are called: lean_025, lean_050, lean_100, sweep_025, sweep_050, sweep_100. I want 1000 points in the sample.

After this prompt, ChatGPT successfully generates a code that creates an LHS, but the prompt did not request a save function, so the code did not save the sample. ChatGPT is asked to add a save function to the code. The following is the ChatGPT authored code that successfully generates an LHS of the design space and saves it to a text file.

Python Code Portion of Response:

```
import numpy as np
import pandas as pd
from scipy.stats.qmc import LatinHypercube

# Define bounds
c1, c2, c3, c4, c5, c6 = [0, 0, 0, 0, 0, 0] # Define your center points here
r1, r2, r3, r4, r5, r6 = [1, 1, 1, 1, 1, 1] # Define your range values here

bounds = [
    [c1-r1, c1+r1],
    [c2-r2, c2+r2],
    [c3-r3, c3+r3],
    [c4-r4, c4+r4],
    [c5-r5, c5+r5],
    [c6-r6, c6+r6]
]

# Generate LHS sample
lhs_obj = LatinHypercube(d=6)
samples = lhs_obj.random(n=1000)

# Adjust the samples to the specified bounds
samples_adjusted = np.array(bounds)[: , 0] + samples * (np.array(bounds)[: , 1] -
np.array(bounds)[: , 0])

# Convert samples to a DataFrame for easier saving
columns = ['lean_025', 'lean_050', 'lean_100', 'sweep_025', 'sweep_050', 'sweep_100']
df_samples = pd.DataFrame(samples_adjusted, columns=columns)

# Save function
def save_to_csv(df, filename="lhs_samples.csv"):
    df.to_csv(filename, index=False)
    print(f"Saved to {filename}")

# Save the data
save_to_csv(df_samples)
```

After the space is mapped, the cases are run using the points from the LHS. This process is described in more detail in Section VIII. The initial intent was to fit the space using polynomial regression. Initial attempts at this failed, resulting in strong oscillations for the curve fit of the FoS within the design space. So, ChatGPT is queried to see if it has any advice on fitting the points.

Response:

If you're experiencing oscillations in your polynomial regression fit, you might be dealing with overfitting, especially when using high-degree polynomials. Here are some alternative methods and approaches to consider:

1. **Regularization:** Introducing regularization like Ridge (L2) or Lasso (L1) can help to reduce overfitting in polynomial regression. By adding a penalty term to the cost function, the algorithm is discouraged from fitting the training data too closely.
2. **Lower the polynomial degree:** If you are using high-degree polynomials, try reducing the degree. It might give a smoother fit, albeit with a potential increase in bias.
3. **Piecewise Regression:** Instead of fitting a single polynomial over the entire domain, fit multiple lower-degree polynomials over partitions of the domain.
4. **Splines and Smoothing Splines:** Splines are piecewise polynomial functions. Smoothing splines can be particularly useful as they allow for a smooth curve that fits the data points with a penalty on the curve's roughness.
5. **Kernel Regression:** It's a non-parametric technique, meaning that the model doesn't assume any underlying distribution. It fits the data points using local approximations.
6. **Gaussian Processes (GPs):** GPs can be used for regression tasks and provide a probabilistic non-parametric model. They give both a mean function (the regression function) and a covariance function (giving uncertainty estimates).

7. Random Forests and Gradient Boosted Trees: Even though these are more commonly used for classification, they can be adapted for regression tasks. They work well with complex datasets and don't require the underlying relationship between variables to be of a specific form.
8. Neural Networks: A simple feed-forward neural network can be used for regression. They can capture complex non-linearities, but require careful tuning and can overfit if not handled properly.
9. Dimensionality Reduction: Given you have 6 design variables, consider techniques like Principal Component Analysis (PCA) to reduce the dimensionality of your problem. This can simplify the regression task, especially if the original variables have multicollinearity.
10. Cross-validation: No matter which method you choose, always use cross-validation to evaluate the model's performance on unseen data. This will give you an idea of how well your model might perform on new, unseen data points.
11. Data Visualization: Given that your design space has 6 variables, direct visualization can be challenging. However, consider techniques like pair plots, parallel coordinate plots, or t-SNE to visualize and understand relationships in your data.
12. Experiment with Interactions and Transformations: Instead of just using the raw variables, consider creating interaction terms (e.g., $x_1 \times x_2$) or transforming variables (e.g., log-transform). Sometimes, the relationship between the response and predictors can be captured better with transformations.

Remember, the choice of method should be based on the nature of your data, the relationships between variables, and the specific problem requirements. It's often a good idea to try multiple approaches and evaluate their performance.

Following this response, the concept of using a neural network to model the space was pursued which led to the usage of the Neural Network in Section VIII.D.

VI. Initial Design, Design Variables and Objective Function

The design selected for improvement through the optimization method described in the next section is called the Tail Cone Thruster (TCT) Design No. 7-28-8 [15], previously known as the aft boundary layer propulsor in the cited paper. The input files to generate this rotor can be found on the T-Blade3 Github in Ref. [16]. The TCT is a blade designed to handle the ingestion of the boundary layer towards the back of the airplane as depicted in Fig. 3.

For the purposes of this paper, this model has been selected simply for demonstration purposes and the boundary layer ingestion is not a consideration in the proposed optimization methodology. Rather the focus is on improvement in the factor of safety (FoS) through modification of the lean and sweep control points exclusively. The FEA simulation, which considers centrifugal loading only, is described in detail in Section VII.



Fig. 3 The Tail Cone Thruster (TCT) as depicted in Refs [15,17]

For the optimization presented in the next sections, the design variables selected for use in the optimization are lean and sweep control points at radial stations of 25%, 50% and 100%.

A. Lean and Sweep Control Points

The lean and sweep of the blade can be altered by a user-specified number of spanwise control points in the T-Blade3 input file [18]. B-spines are used to generate smooth geometry from these inputs. The stacking of the blade sections may be adjusted in the θ direction using lean control points and the m'_s direction using sweep control points.

Fig. 4 shows the (x, r) meridional plane of the turbomachine along with streamlines of the expected airflow. The third direction of the coordinate system that this plane lies on is the θ direction; together these three coordinates form the (x, r, θ) coordinate system with the rotation direction in θ defined using left-handed rotation. In this context, the streamlines are axisymmetric and should be thought of as construction curves. When revolved about the x -axis, they are construction surfaces. The position along the streamline is defined by m_s and the differential arc length of the streamline, dm_s , can be determined by Eq. (1).

$$dm_s = \sqrt{dx_s^2 + dr_s^2} \quad (1)$$

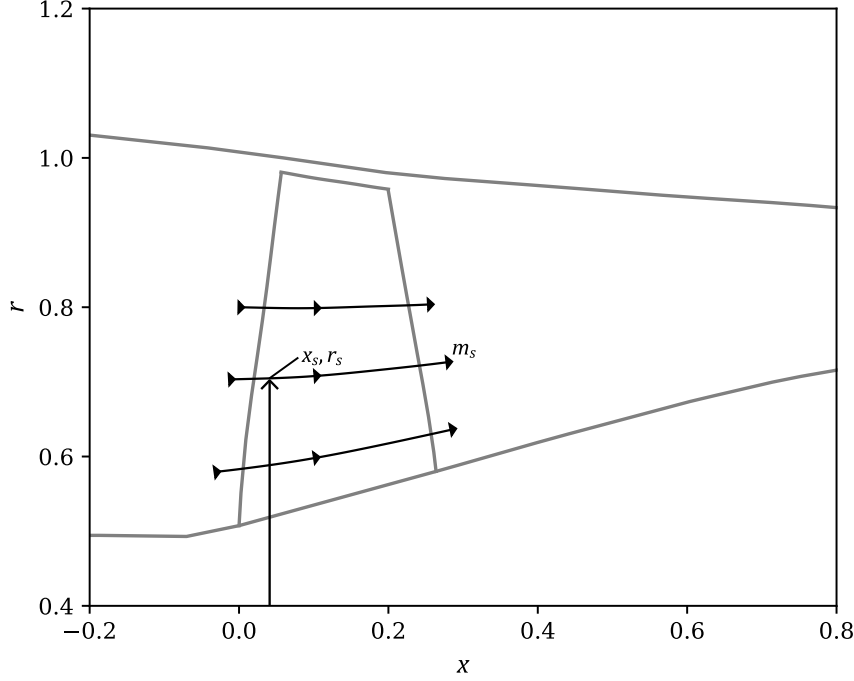


Fig. 4 The (x, r) meridional plain of the turbomachine. This figure also shows streamlines, which are used to define the m'_s coordinate.

For the purposes of design, it is useful to normalize the space to allow for effective scaling of the size. For this purpose, the normalized differential arc length, dm'_s , defined in Eq. (2) is used. Finally, the m'_s coordinate is determined through integration of dm'_s in Eq. (3).

$$dm'_s = \frac{dm_s}{r_s} \quad (2)$$

$$m'_s = \int \frac{\sqrt{(dx_s)^2 + (dr_s)^2}}{r_s} \quad (3)$$

In the T-Blade3 input file used for the optimization work presented in this paper, there are four control points defined at different positions along the blade span; these are at 0%, 25%, 50% and 100% span locations. Keep in mind that the number of these control points is determined by the user and can be set arbitrarily. The control points represent a shift in location from the profile defined in another portion of the T-Blade3 input file. Sweep control points are defined in the non-dimensional m'_s coordinate. These are referred to as $\Delta m'_0$, $\Delta m'_{25}$, $\Delta m'_{50}$ and $\Delta m'_{100}$. Lean control points produce a shift of the blade profile in the θ coordinate, which is rotation about the x -axis and has units of radians. Similar to the sweep control points, the lean control points are referred to as $\Delta\theta_0$, $\Delta\theta_{25}$, $\Delta\theta_{50}$ and $\Delta\theta_{100}$. These two parameters are used to shift the blade profile in the (m'_s, θ) coordinate system. The result is a change in the lean and sweep angles of the blade.

For the optimization work presented in this paper, the values of $\Delta\theta_0$ and $\Delta m'_0$ are always held at zero, while the other six parameters are considered design variables. There is an example below in Fig. 5 that shows the effect of

changing the lean and sweep control points. In Fig. 5, the new geometry is shown in green while the baseline geometry is shown in grey. Fig. 5a shows the result of applying $\Delta m'_{25} = 0.2$, resulting in a forward swept blade at the 25% span. Fig. 5b shows the result of applying $\Delta \theta_{25} = -0.2$, which has a strong effect on the lean of the blade around the 25% span. For additional background on T-Blade 3 and how it functions, please reference [7,8,16,18–23].

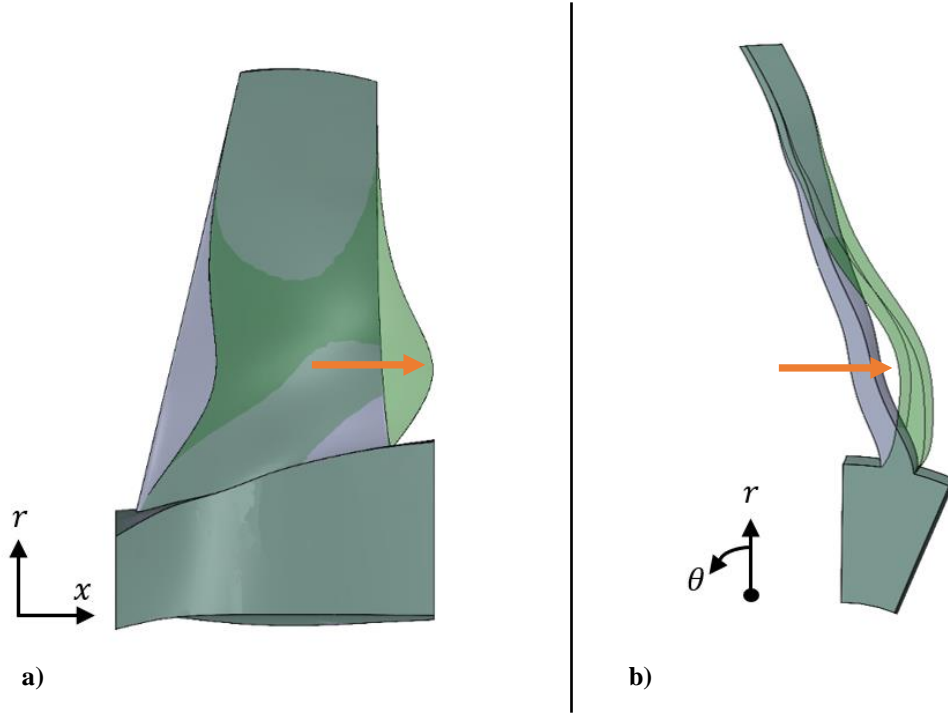


Fig. 5 Sweep (a) and lean(b) examples; a) positive sweep control point at 25% span ($\Delta m'_{25} = 0.2$); b) negative lean control point at 25% span ($\Delta \theta_{25} = -0.2$)

B. Objective Function

The objective function for the design optimization is the negative of the factor of safety (FoS). The FoS is defined as the maximum von-Mises, $\sigma_{vm,max}$, stress in the blade divided by the yield strength of the material. This is shown below in Eq. (4). The goal of the optimization is to minimize the objective function, which is the same as maximizing the FoS for this case. This is done through modification of the lean and sweep control points.

$$\text{FoS} = \frac{\sigma_{ys}}{\sigma_{vm,max}} \quad (4)$$

VII. FEA Model Settings and Starting Design Results

A. Baseline Model

As mentioned in the previous section the starting design for this optimization is the Tail Cone Thruster (TCT) Design No. 7-28-8 [15,16]. A slight alteration has been made to the version that is available on the T-Blade3 website; the lean and sweep at 0% span have been set to zero. The blade geometry is of a model previously designed for a wind tunnel test. The blade is approximately 150 [mm] in radius and its design rotational speed is 21,581 [RPM] [15]. The blade is constructed of Ti-6Al-4V. The properties for this material are given in Table 1.

Table 1 Properties of the titanium compound used for this analysis

Ti-6Al-4V Properties	
Density [kg]	7,850
Elastic Modulus [GPa]	200
Poisson Ratio	0.3
Tensile Strength [MPa]	250

The geometry of the blade design is shown in Fig. 6. This geometry does not contain a design for a disk. Instead, it is supported by a thick wedge. Disk design is outside the scope of this work, but it is a future topic of interest. A constant radius fillet sized 1/16 [in] (1.5875 [mm]) is added to the root of the blade in Ansys DesignModeler [24].

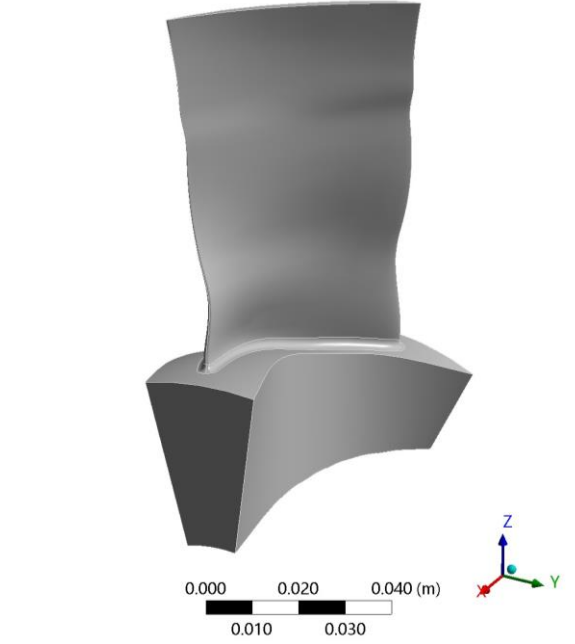


Fig. 6 TCT 7-28-8 with all lean and sweep parameters set to zero.

B. Mesh Settings

The mesh settings used in a typical analysis in the optimization scheme are listed below in Table 2. The topology is separated such that it allows for mapped face meshing on both the pressure and suction faces. Two edges run along the span (Spanwise Edges) and two edges run along the chord (Chordwise Edges) at both the root and top of the blade. The fillet region is also composed of two mappable faces, whose refinement is defined by the Chordwise Edge sizing and the Fillet Edge sizing. Table 3 shows the mesh parameters used by the Ansys mesh utility. Please see the Ansys Mechanical manual for details about these mesh parameters [25]. For the baseline geometry, the mesh has 148,914 nodes and 100,254 solid, second-order elements. Fig. 7 shows the resultant mesh.

Table 2 Mesh edge sizing for the centrifugal load safety analysis.

	Divisions	Bias Type	Bias Factor
Spanwise Edges	50	Towards Root	3
Chordwise Edges	50	Both Sides	3
Fillet Edges	5	None	N/A

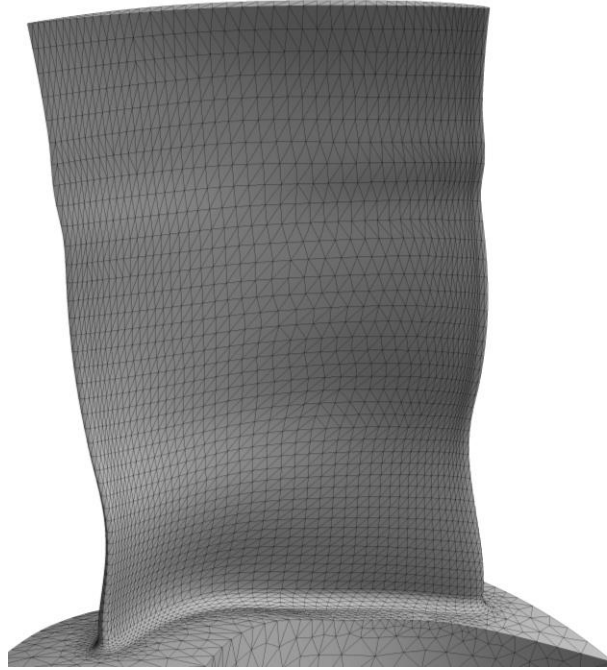


Fig. 7 Mesh of the baseline case.

Table 3 Ansys mesh parameters used for analyzing models for the optimization.

Ansys Mesher Settings	
Element Order	2nd
Element Size [m]	0.005
Growth Rate	1.85
Capture Curvature	Yes
Curvature Min Size [mm]	0.3
Curvature Normal Angle	12
Capture Proximity	Yes
Proximity Min Size [mm]	1

C. Loading and Constraints

A cyclic symmetry model is not used in order to conduct the simulation as quickly as possible. Instead, the model is fixed at the base of the wedge as well as the faces which would otherwise have cyclic symmetry boundaries applied. The design speed for the rotor is 21,581 [RPM], and the centrifugal load is the only load that is applied.

D. Mesh Convergence Study

A mesh refinement study is conducted to ensure that the chosen mesh is sufficiently refined. In this study, the number of divisions on the mapped faces are both coarsened and refined in order to evaluate for mesh convergence. The settings used in this study are shown below in Table 4.

Table 4 Mesh settings for mesh convergence study

Mesh Type	Edge Divisions			Growth Rate	Total Nodes	Total Elements
	Spanwise	Chordwise	Fillet			
Coarse	30	30	4	1.85	116,959	81,584
Regular	50	50	6	1.85	148,914	100,254
Fine	100	100	8	1.4	309,788	197,214

Table 5 shows the results of the mesh convergence study in terms of the FoS, which is the only component of the objective function and is defined in Equation 4. The results show that the regular and fine mesh, which has twice as many nodes as the regular mesh, have nearly the same results with a deviation of only 0.6%. The coarse mesh, on the other hand, has a result which deviates 10.6% from the fine mesh.

Table 5 FoS results of the refinement study

	FoS
Coarse	1.1182
Regular	1.0167
Fine	1.0107

The mesh convergence study results indicate that the regular mesh is sufficiently refined. Also, the regular mesh settings produce a model that can be meshed and solved in less than a minute, which is an excellent property given the large number of cases that need to be simulated. The reported meshing and simulation time is on a laptop using 4 cores, so the time on the workstation where the optimization is conducted is even less.

VIII. Optimization Methodology

In this section, the turbomachinery blade optimization system that was inspired by the recommendation of ChatGPT to utilize a neural network model is presented. The method is described in a flow chart in Fig. 8. The flow chart was fed to ChatGPT's image input function and asked to describe the process illustrated in the flow chart.

Response:

The flowchart outlines a design optimization process that employs both sampling techniques and neural networks. Here's a step-by-step explanation:

1. Choose a design space: This is the initial step where you define the boundaries and constraints of the problem you want to solve or optimize.
2. Sample that space with LHS: LHS stands for Latin Hypercube Sampling, which is a statistical method used for generating a sample of plausible collections of parameter values from a multidimensional distribution. It's a way to ensure that the entire design space is adequately explored.
3. Run each point from the sample: After sampling the design space, each sampled point or scenario is executed or evaluated. This could involve running a simulation, an experiment, or any method that gives you a result for that specific set of parameters.
4. Use the results to train a NN: NN stands for Neural Network, a type of machine learning algorithm. The results from the sampled points are used to train this neural network. Essentially, the neural network learns the relationship between the input parameters and the results.
5. Run an optimization on the NN model: Once the neural network is trained, you can use optimization techniques on it to find the best possible solution within the design space. This step tries to improve the performance based on what the neural network has learned.
6. Determine if result is satisfactory: After running the optimization, you check if the results meet your criteria for success. If they do, you proceed to finalize the optimization. If not, you go to the next step.
7. Reduce the design space and recenter it around the best point: If the result from the neural network optimization isn't satisfactory, the design space is narrowed down, focusing more closely around the best solution found so far. This refined design space is then sampled and evaluated again, effectively iterating on the optimization process.

Finally, once a satisfactory result is achieved, the process concludes with the statement "Optimization Complete."

In essence, this process is a mix of exploratory sampling, machine learning, and iterative refinement to find an optimal solution within a design space.

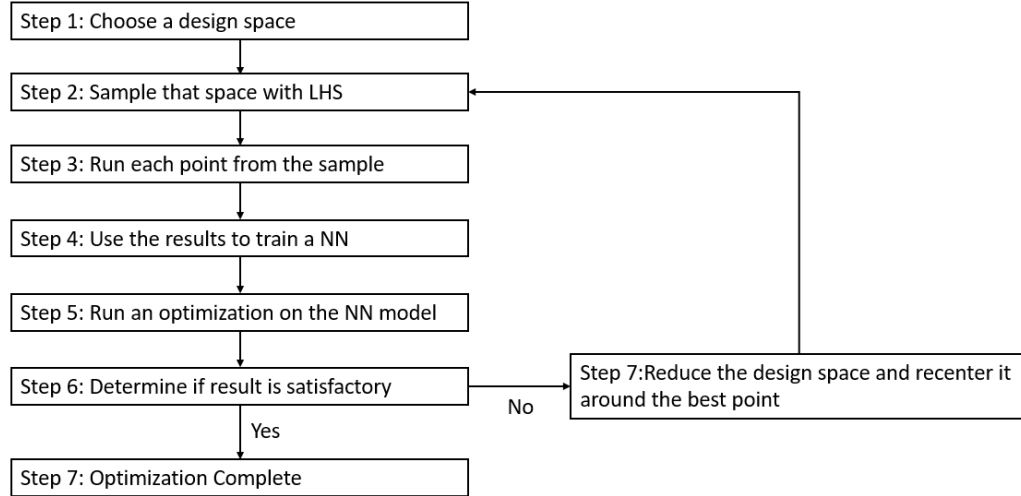


Fig. 8 Flow chart of optimization methodology

The following subsections explain the individual steps in further detail.

A. Initial Design Space

The choice of a design space is simple in concept but can be difficult in practice if the designer is unfamiliar with the design space. For example, if too large of a space is selected with many infeasible designs, it will reduce the understanding of the useful portion of the design space. In the case of the optimization of the Tail Cone Thruster (TCT), prior experience indicated that a range of -0.1 to 0.1 for all design variables would provide designs that are largely reasonable for this geometry. This is the space used for the initial iteration in the design and is also shown below in Table 6. Please see Section VI.A for a description of the lean and sweep control point design variables.

Table 6 Initial design and design space for initial iteration of optimization loop

	Initial Design	Design Space	
		Lower Limit	Upper Limit
$\Delta m'_{25}$	0.0	-0.1	0.1
$\Delta m'_{50}$	0.0	-0.1	0.1
$\Delta m'_{100}$	0.0	-0.1	0.1
$\Delta \theta_{25}$	0.0	-0.1	0.1
$\Delta \theta_{50}$	0.0	-0.1	0.1
$\Delta \theta_{100}$	0.0	-0.1	0.1

B. Latin Hypercube Sampling

The sampling of a high dimensional space, in this case a 6-dimensional space for the 6 design variables, is a problem that has been studied by statisticians. The goal is to fill the design space as uniformly as possible, the design of such a sampling is known as a space-filling design. One common method for filling such a space is called Latin Hypercube Sampling (LHS), which was introduced by McKay, Beckman and Conover in 1979 [26]. LHS ensures that each of the dimensions within the sampling space is sampled uniformly. This is described in Ref [14] as “Latin hypercube designs have one-dimensional uniformity in that, for each input variable, if its range is divided into the same number of equally-spaced intervals as the number of observations, there is exactly one observation in each interval.” However, this does not guarantee that combinations of variables will be sampled effectively, and the points could still be somewhat bunched together in some of the regions of the design space.

The LHS can be improved through optimization methods which produce a sample of the design space that is more even and avoids point bunching. SciPy’s LHS generator provides several optimization routines to do this. The ‘random-cd’ optimization method provided by SciPy conducts random permutations of coordinates to lower the centered discrepancy [27,28]. The centered discrepancy gives the largest difference between the number of sample points in any sub-interval and the expected number of sample points. Simply put, this optimization method reduces point bunching and pushes the sample towards a more uniform distribution of points in any sub-domain of the space.

For the work presented in this paper, SciPy’s Latin Hypercube Sampling is used to sample a 6-dimensional space with 1000 points. The sample is optimized using the ‘random-cd’ optimization parameter to reduce centered discrepancy.

There are also other methods known to the authors but not explored in this paper. One of these methods is Orthogonal-Array Latin Hypercube Sampling (OA-LHS) [14,29]. OA-LHS sampling has properties that reduce the variance of interaction. Consequently, OA-LHS can offer improved representation of the interactions between different design variables in many scenarios. However, the complexity of generating a set of OA-LHS points is greater compared to ordinary LHS. Since this type of sampling is freely available in SciPy, the complexity of generating the sample is not a concern. So, this is a very interesting topic for the focus of future research. Another interesting topic for future research is the number of points used to sample the design space.

C. Simulation Execution

In addition to the optimization routines included in OpenMDAO, it also contains a useful utility for editing input files and running simulations. These features are used to evaluate the set of 1000 cases produced by the LHS by first creating the geometry using T-Blade3 and then evaluating the design with the meshing parameters and conditions described in Section VII. If a particular configuration fails to produce a FoS result, either due to an issue in geometry generation or a failure of the FEM, the point is simply thrown out and not used to train the neural network. The types of cases that fail are typically in a range of the design space that performs very poorly, so the loss those points does not have much effect on the ability to optimize. The result and design are both saved to a text file. In the next step, the data is used to train the neural network.

D. The Neural Network

Prior to the recommendation to use a neural network from ChatGPT, neither of the authors had significant experience working with neural networks. Fortunately, the recommendations from ChatGPT for the model architecture and training parameters proved to train the model effectively with little tweaking. The model recommended for use by ChatGPT is sklearn’s Multi-Layer Perceptron (MLP) Regressor [30,31]. The model parameters used in this work are given in Table 2. These parameters are little changed from the recommendation from ChatGPT, with the only change in the hidden layer size. The original recommendation was for a size of (100, 100), but some experimentation with this setting found that increasing the size to (300, 200) produced a better result. An investigation into these neural network model and training parameters is outside of the scope of the current work, but it is a topic that will be considered in future research.

Table 7 Parameters used for the MLPRegressor model.

MLPRegressor Parameters	
Hidden Layer Size	300, 200
Activation	Relu
Solver	Adam
Alpha	0.001
Learning Rate	Constant
Random State	1
Max Iterations	500

E. Optimization of the Neural Network

After the neural network (NN) is created, an optimization is run on this network. OpenMDAO is used for this optimization. OpenMDAO contains several different optimization routines. The goal of the optimization is to drive the FoS as high as possible through modification of only the lean and sweep design variables. In this paper, the gradient-based optimization is used. OpenMDAO has a method called ScipyOptimizeDriver which is a wrapper for the scipy.optimize.minimize method [32,33]. The method for the minimization is Sequential Least Squares Quadratic Programming (SLSQP) [34].

A genetic algorithm (GA), OpenMDAO’s SimpleGADriver [35], has also been used during the course of this work. GAs have the advantage over gradient based methods in terms of resistance to becoming stuck in local minima. However, they are much slower compared to the gradient based methods. So far, the GA and gradient-based methods have produced similar results for optimization conducted on the NN. Exploration of the GA may be reported in the future if a benefit is found.

F. Determination of a Satisfactory Result

For this work, the initial design space is a large one with many regions that are well outside of a desirable design. So, the initial survey is a very coarse one. It is also possible that the optimal result lies somewhere outside of the design space.

Given those two ideas, as the design iterations increase, the design space should both be recentered around the optimal design point and the range reduced in order to produce a more accurate picture around the space of highest interest. If the optimal point lands on the edge of the design space, that is a good indication that the optimal point may be outside of the current design space and further iterations are warranted. Also, if the FoS prediction from the NN deviates significantly from the one determined from analyzing the same case with FEA, then that is also an indication that the design space may not be well represented by the NN.

With all of that said, much of the determination of a satisfactory result is case-by-case and requires engineering judgement. Best practices are still being determined. After each iteration in the current work, the design space is both recentered around the best point and reduced in size. For example, for the first iteration, the range for each variable is 0.2 while this range is reduced to 0.15 for the second iteration. The range is 0.1 for the third and final iteration. The center point for iterations 2 and 3 is the best point from the previous iteration.

IX. Optimization Results

The optimization process starts with the design space centered around zero values for all parameters, like in Table 6. For this case, the full optimization process lasts for three iterations (see Fig. 8 for a depiction of the process). The starting FoS, as defined in Equation 4, is 1.017. This is improved in each iteration with a final value 5.176. So, this process reduced the maximum stress in this blade to one fifth of the original value. The best result after each iteration is shown in Table 8, which also shows the lean and sweep control points used for each model.

During the optimization, it became necessary to make a slight alteration for the NN training methodology for iteration 3. The initial model, trained based on the method described in Section VIII.D, does not predict the optimal value. Instead, it drives the set of variables to a corner of the design space, predicting that minimum values for all variables would be the optimal point. After some guidance from ChatGPT, an early stopping criterion was added to the training parameters. Early stopping [36] is a useful tool to prevent overfitting of data by creating two training sets, one which is not used in training; instead, it is used exclusively for model validation. With early stopping enabled, training is stopped if the error in the validation set begins to increase even if the error in the training set continues to decrease.

It is not clear why the data for iteration three required an early stopping criterion while the other two iterations did not. However, it had a significant improvement on the result. A future research topic may explore this and other criteria of the neural network.

Results from the first and second iteration both show that the optimal point is located on the edge of the design space for that iteration. However, for iteration 3, the result is located very close to the center of the design space. One other thing to note, the difference in the result between the previous iteration and iteration 3 is much smaller compared to the prior iterations. This is a good indication that the design is near its best for the current set of design variable.

Contour plots of von-Mises stress on the blade surface are included in Fig. 9. The stress unit for the figure is [MPa]. One interesting thing to note is the change in the location of maximum stress. For the initial design, the maximum stress (shown in the figure using the max label) occurs in the fillet region while it occurs in the middle of the blade for the optimized model. The stress in the optimized model is also much more even around the filleted region.

Table 8: Results of the Optimization

Iteration	Initial	1	2	3
$\Delta m'_{25}$	0	0.0006	-0.0566	-0.0562
$\Delta m'_{50}$	0	-0.0654	-0.1022	-0.1017
$\Delta m'_{100}$	0	-0.09	-0.1079	-0.1083
$\Delta \theta_{25}$	0	-0.0278	-0.0953	-0.0949
$\Delta \theta_{50}$	0	-0.0855	-0.153	-0.1526
$\Delta \theta_{100}$	0	-0.09	-0.1549	-0.1554
FoS, ML	N/A	2.141	3.264	3.504
FoS, FEA	1.017	2.742	5.163	5.176

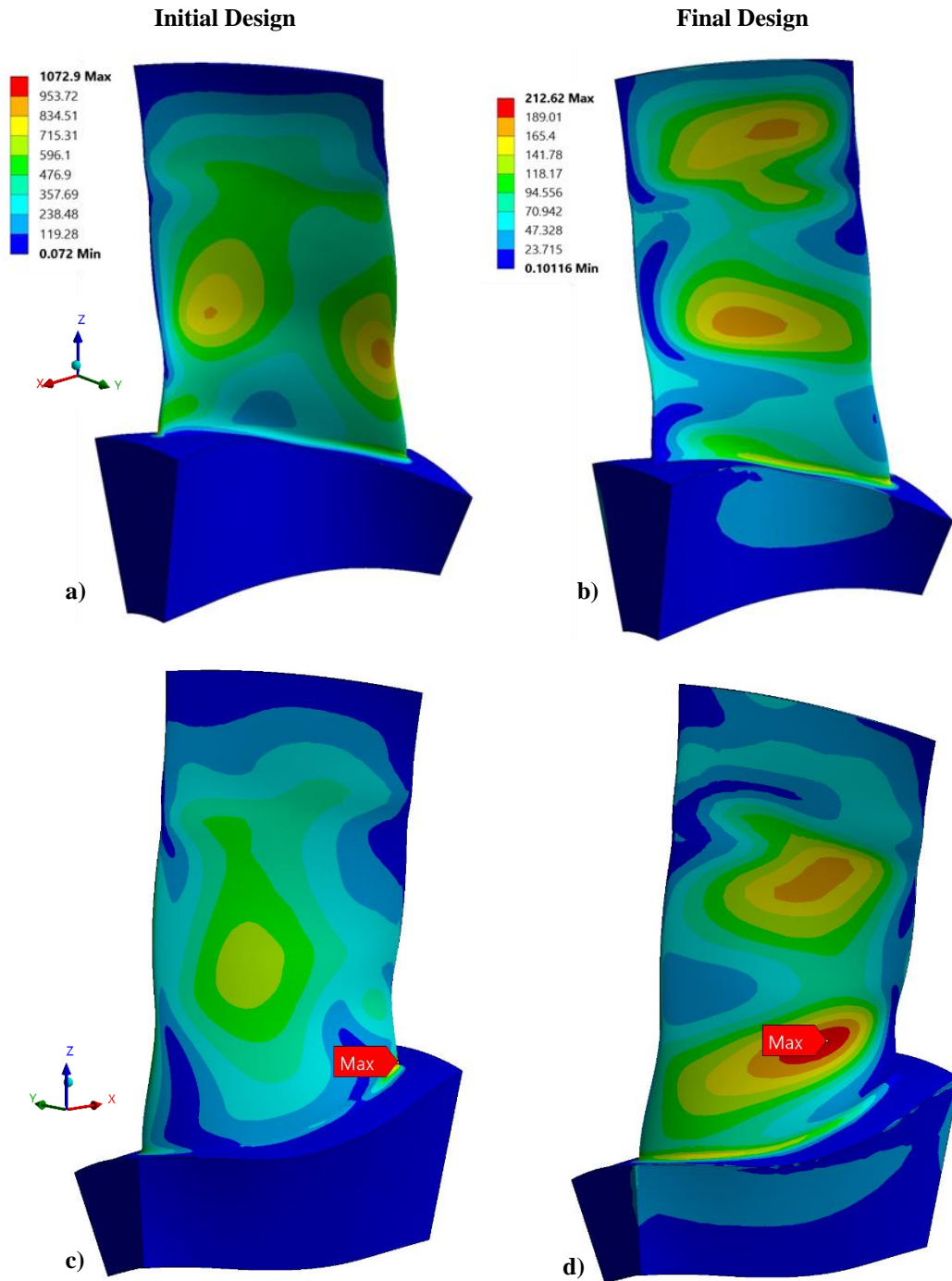


Fig. 9 Contours of von-Mises stress on the fan blade; a) initial design, pressure side; b) final design, pressure side; c) initial design, suction side; d) final design, suction side. Stress units are in [MPa].

X. Conclusions

This paper began with a focus on ChatGPT. ChatGPT played a crucial part in the development of the optimization scheme presented in the past few sections. It is a capable Python programmer and can even make fully functioning engineering tools as shown in Section III. This capability amazed the authors and inspired the submission of the draft manuscript for this paper.

ChatGPT does suffer from some significant issues, especially hallucinations, which are covered in Section IV. The issue of hallucinations makes it clear that one cannot yet use AI to replace human engineers. Also, any engineer making use of AI systems must be acutely aware that the output may not be accurate. As these large language models continue to be developed, hopefully the developers can find ways to have the system indicate that it is uncertain on inquiries for topics on which it is poorly trained. This would be more favorable than the creation of fake programming languages or other false responses.

Still, ChatGPT is incredibly useful. Its ability to provide the user with immediate examples of programming concepts aids the learning process for engineering tasks. For certain topics, it can be similar to a personal tutor guiding one through a new idea. This was the case when it recommended the usage of the neural network to model the design space instead of using polynomial regression. In this case, it not only provided an example, but it actually wrote the code that trained the first neural network used in this research. This was then used to map out a 2D design space that had already been fully explored, and it became immediately clear that this method had promise. ChatGPT allows for rapid consideration of many different tools to solve problems. The engineer can then use their judgement to explore the best of the options and even use ChatGPT to create prototypes in some cases.

Another use of ChatGPT, discussed in Section V, is its ability to document existing code. This clearly a very useful feature, and ChatGPT produced excellent documentation for the code used in the work. However, the use of this for proprietary source code documentation is not advised as ChatGPT is not a secure platform. It seems like an obvious business model, so any businesses interested in code documentation should contact OpenAI directly to see if they offer this service.

The optimization methodology developed with ChatGPT's help proved to be extremely effective, leading to a reduction in blade stress to less than one fifth of the original value. The space is incredibly nonlinear with lots of interaction between variables, which the method captured very well. The method is also very robust, and the failure of any single simulation has little negative effect on the result. There are many research opportunities to be explored in the future with respect to the mapping the design space and training parameters used for this the neural network. During the course of this research, the benefit of early stopping in the model training was encountered. The size of the training set should also be explored.

Appendix

A. ChatGPT Boundary Layer Visualizer Code

The following Python class was produced entirely by ChatGPT based on the second prompt given in Section III. This code was then copied to a desktop computer and executed locally.

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QSlider, QLabel, QWidget
from PyQt5.QtCore import Qt
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas

class InteractivePlot(QMainWindow):
    def __init__(self, VelocityMagnitude, DistanceFromWall):
        super().__init__()

        self.VelocityMagnitude = VelocityMagnitude
        self.DistanceFromWall = DistanceFromWall

        self.init_ui()

    def init_ui(self):
        # Create a QWidget as the central widget
        central_widget = QWidget(self)
        self.setCentralWidget(central_widget)

        # Create a QVBoxLayout
        vbox = QVBoxLayout()

        # Create and configure the plot canvas
        self.figure, self.ax = plt.subplots()
        self.canvas = FigureCanvas(self.figure)
        vbox.addWidget(self.canvas)

        # Create sliders for i_position and t_position
        self.i_position_slider = QSlider(Qt.Horizontal)
        self.i_position_slider.setMinimum(0)
        self.i_position_slider.setMaximum(len(self.VelocityMagnitude) - 1)
        self.i_position_slider.valueChanged.connect(self.update_plot)
```

```

vbox.addWidget(self.i_position_slider)

self.t_position_slider = QSlider(Qt.Horizontal)
self.t_position_slider.setMinimum(0)
self.t_position_slider.setMaximum(len(self.VelocityMagnitude[0, 0]) - 1)
self.t_position_slider.valueChanged.connect(self.update_plot)
vbox.addWidget(self.t_position_slider)

# Create labels for i_position and t_position
self.i_position_label = QLabel()
vbox.addWidget(self.i_position_label)
self.t_position_label = QLabel()
vbox.addWidget(self.t_position_label)

# Set the layout
central_widget.setLayout(vbox)

# Initial plot
self.update_plot()

def update_plot(self):
    # Get the current i_position and t_position from sliders
    i_position = self.i_position_slider.value()
    t_position = self.t_position_slider.value()

    # Update the labels
    self.i_position_label.setText(f'i_position: {i_position}')
    self.t_position_label.setText(f't_position: {t_position}')

    # Clear the previous plot
    self.ax.clear()

    # Plot the new data
    velocity_data = self.VelocityMagnitude[i_position, :, t_position]
    distance_data = self.DistanceFromWall[i_position, :]
    self.ax.plot(velocity_data, distance_data)

    # Add labels
    self.ax.set_xlabel('Velocity (m/s)')
    self.ax.set_ylabel('Distance from wall (m)')

    # Update the canvas
    self.canvas.draw()

def run_interactive_plot(VelocityMagnitude, DistanceFromWall):
    app = QApplication(sys.argv)
    main = InteractivePlot(VelocityMagnitude, DistanceFromWall)
    main.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    import numpy as np

    # Generate sample data
    VelocityMagnitude = np.random.random((10, 5, 3))
    DistanceFromWall = np.random.random((10, 5))

    run_interactive_plot(VelocityMagnitude, DistanceFromWall)

```

B. Optimization Class Sent to ChatGPT for Explanation

```

from openmdao.utils.file_wrap import InputFileGenerator, FileParser
from TurboOpt3DSimControl import SimulationControlClass
import openmdao.api as om
import shutil, os, time
import numpy as np

# Last iteration is used to restart. Set it to the most recent
# successful iteration if the solution process is interrupted.

LastIteration = 0

class TurboOPT3D(om.ExternalCodeComp):
    def setup(self):

        self.add_input('lean_025')
        self.add_input('lean_050')
        self.add_input('lean_100')
        self.add_input('sweep_025')

```

```

self.add_input('sweep_050')
self.add_input('sweep_100')
self.add_output('FoS')
self.add_output('W')
self.SimFailureCounter = 0
self.SimSuccess = True

self.writeParser = InputFileGenerator()
self.SimulationController = SimulationControlClass()

self.WorkingDir = self.SimulationController.WorkingDir
self.CaseResults = os.path.join(self.WorkingDir, 'TCT_Iteration2_OptimizedLHS.txt')
print('Opening %s'%self.CaseResults)
self.ResultsFile = open(self.CaseResults, 'a')
if LastIteration == 0:
    self.ResultsFile.write('%6s|%9s|%12s|%12s|%12s|%12s|%12s|%12s|%12s|%12s\n'%
        ('# Iter', 'FoS', 'W', 'lean_025', 'lean_050', 'lean_100', 'sweep_025', 'sweep_050', 'sweep_100'))
self.ResultsFile.close()
self.Iteration = LastIteration

def compute(self, inputs, outputs):

    WorkingDatFileDict, WorkingSpanCtrlDict = self.SimulationController.CopyTBlade3Files(Process=1)
    self.tbladeinputfile = WorkingDatFileDict['Rotor1']
    self.WriteTblade3File(self.tbladeinputfile, inputs)
    self.SimulationController.RunTblade3(Process=1)
    FoS, W = self.StructuralAnalysis()

    outputs['W'] = W
    outputs['FoS'] = FoS

    self.Iteration += 1
    self.ResultsFile = open(self.CaseResults, 'a')
    self.ResultsFile.write('%6i|%9.3f|%12.7f|%12.7f|%12.7f|%12.7f|%12.7f|%12.7f|%12.7f\n' %
        (self.Iteration, FoS, W,
        inputs['lean_025'][0],
        inputs['lean_050'][0],
        inputs['lean_100'][0],
        inputs['sweep_025'][0],
        inputs['sweep_050'][0],
        inputs['sweep_100'][0]
        ))

    if self.SimSuccess == True:
        self.SimulationController.CopySpinAnalysisArchive(Process=1,
            DesignNumber='TestCases_Iteration%i'%self.Iteration, SimSuccess=self.SimSuccess)
    else:
        self.SimulationController.CopySpinAnalysisArchive(Process=1,
            DesignNumber='TestCases_Iteration%i_Failed'%self.Iteration, SimSuccess=self.SimSuccess)
    self.ResultsFile.close()

def WriteTblade3File(self, tbladeinputfile, inputs):

    # tbladeinputfile is the template file.
    # We just treat it as if it is the file that we are going to use. If it needs
    # modified by the optimizer, then a version with _gen on the extension is created
    # the original file is then overwritten by the _gen file, and the _gen file is deleted
    # This will allow us to easily change to optimizing the IGVs or OVGs if desired.

    tbladeinputfile_generated = tbladeinputfile + '_gen'

    self.writeParser.set_template_file(tbladeinputfile)
    self.writeParser.set_generated_file(tbladeinputfile_generated)

    self.writeParser.reset_anchor()
    self.writeParser.mark_anchor('Control points for lean')
    self.writeParser.transfer_var(inputs['lean_025'][0], 4, 2)
    self.writeParser.transfer_var(inputs['lean_050'][0], 5, 2)
    self.writeParser.transfer_var(inputs['lean_100'][0], 6, 2)
    self.writeParser.generate()

    self.writeParser.reset_anchor()
    self.writeParser.mark_anchor('Control points for sweep')
    self.writeParser.transfer_var(inputs['sweep_025'][0], 4, 2)
    self.writeParser.transfer_var(inputs['sweep_050'][0], 5, 2)
    self.writeParser.transfer_var(inputs['sweep_100'][0], 6, 2)
    self.writeParser.generate()

    shutil.copyfile(tbladeinputfile_generated, tbladeinputfile)
    os.remove(tbladeinputfile_generated)

```



```

def StructuralAnalysis(self):
    if self.SimFailureCounter > 7:
        print('Too many failures in a row :(')
        quit()
    try:
        FoSResultDict, MassResultDict = self.SimulationController.RunSpinAnalysis(Process=1)
        FoS = FoSResultDict['Rotor1']
        W = MassResultDict['Rotor1']
        self.SimSuccess = True
        self.SimFailureCounter = 0
        return FoS, W
    except:
        file = open(self.CaseResults,'a')
        file.write('Failed Ansys Sim\n')
        file.close()
        FoS = 0
        W=0
        self.SimSuccess = False
        self.SimFailureCounter += 1

    return FoS, W

```

Acknowledgments

The authors would like to acknowledge that this research was conducted on work funded by NASA's Advanced Air Transport Technology (AATT) project and Transformational Tools and Technologies (TTT) project. The development of T-Blade3 by the University of Cincinnati's Gas Turbine Simulation Laboratory made this work possible. The guidance from Mark Turner (NASA-GRC) on T-Blade3 is much appreciated. This work would also not have been possible without the freely available libraries in Python. Therefore, a special thanks goes to all the code developers who have generously donated their time to bring sophisticated scientific and mathematical tools to the masses. OpenMDAO, created by our colleagues at Glenn Research Center, was instrumental in both running cases and finding the optimum point within the neural network model of the design space. Finally, the attention to detail by Rula Coroneos (NASA-GRC) during the internal review process improved this paper significantly.

References

- [1] OpenAI, "OpenAI Charter." Retrieved 15 May 2023. <https://openai.com/charter>
- [2] Terwiesch, C., "Would Chat GPT3 Get a Wharton MBA? A Prediction Based on Its Performance in the Operations Management Course," 2023.
- [3] OpenAI, "GPT-4 Technical Report," 2023. <https://doi.org/10.48550/arXiv.2303.08774>
- [4] Ali, R., Tang, O. Y., Connolly, I. D., Sullivan, P. L. Z., Shin, J. H., Fridley, J. S., Asaad, W. F., Cielo, D., Oyelese, A. A., Doberstein, C. E., Gokaslan, Z. L., and Telfeian, A. E., "Performance of ChatGPT and GPT-4 on Neurosurgery Written Board Examinations," 2023. <https://doi.org/10.1101/2023.03.25.23287743>
- [5] Alkaissi, H., and McFarlane, S. I., "Artificial Hallucinations in ChatGPT: Implications in Scientific Writing," *Cureus*, 2023. <https://doi.org/10.7759/cureus.35179>
- [6] Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A., "OpenMDAO: An Open-Source Framework for Multidisciplinary Design, Analysis, and Optimization," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. <https://doi.org/10.1007/s00158-019-02211-z>
- [7] Gas Turbine Simulation Laboratory - University of Cincinnati, "T-Blade3 Github." Retrieved 30 April 2023. <https://github.com/GTSL-UC/T-Blade3>
- [8] Siddappaji, K., Turner, M. G., Nemnem, A. F., and Mahmood, S. M. H., "T-Blade3: A Simple and Robust Blade Design Tool for Complex Energy Conversion Devices," 2016.
- [9] The Matplotlib development team, "Matplotlib: Visualization with Python," 2023. Retrieved 18 October 2023. <https://matplotlib.org/>
- [10] The Python Community, "PyQt," The Python Wiki, 2023. Retrieved 18 October 2023. <https://wiki.python.org/moin/PyQt>
- [11] The Qt Company, "Qt Designer Manual," Qt Documentation, 2023. Retrieved 18 October 2023. <https://doc.qt.io/qt-6/qt designer-manual.html>
- [12] ANSYS Inc., "Ansys Workbench Scripting Guide," 2023.

- [13] Netscape Press Relations, “Archived Webpage: Netscape and Sun Announce JavaScript, the Open, Cross-Platform Object Scripting Language for Enterprise Networks and the Internet,” Dec 04 1994. Retrieved 17 October 2023. <https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html>
- [14] Lin, C. D., and Tang, B., “Latin Hypercubes and Space-Filling Designs,” 2022. Retrieved 12 October 2023. <https://arxiv.org/abs/2203.06334>
- [15] Mandal, P., Holder, J., Turner, M. G., and Celestina, M. L., “Design and Optimization of a Boundary Layer Ingesting Propulsor,” 2020. <https://doi.org/10.1115/GT2020-15603>
- [16] “T-Blade3 Input Files: Case 7-28-8,” 2020. Retrieved 15 October 2023. <https://github.com/GTSL-UC/T-Blade3/tree/master/inputs/OpenCSM/Case7-28-8>
- [17] Flynn, M.-C., Jones, C. E., Norman, P. J., and Galloway, S. J., “Establishing Viable Fault Management Strategies for Distributed Electrical Propulsion Aircraft A Review of Protection and Design Challenges of Viable Fault Management Strategies,” 2017.
- [18] Siddappaji, K., Turner, M. G., and Merchant, A., “General Capability of Parametric 3D Blade Design Tool for Turbomachinery,” 2012. <https://doi.org/10.1115/GT2012-69756>
- [19] Mahmood, S. M. H., Turner, M. G., and Siddappaji, K., “Flow Characteristics of an Optimized Axial Compressor Rotor Using Smooth Design Parameters,” 2016. <https://doi.org/10.1115/GT2016-57028>
- [20] Nemmem, A. F., Turner, M. G., Siddappaji, K., and Galbraith, M., “A Smooth Curvature-Defined Meanline Section Option for a General Turbomachinery Geometry Generator,” 2014. <https://doi.org/10.1115/GT2014-26363>
- [21] Turner, M. G., Merchant, A., and Bruna, D., “A Turbomachinery Design Tool for Teaching Design Concepts for Axial-Flow Fans, Compressors, and Turbines,” *Journal of Turbomachinery*, Vol. 133, No. 3, 2011. <https://doi.org/10.1115/1.4001240>
- [22] Sharma, M., Dannenhoffer, J. F., Holder, J., and Turner, M. G., “Integration of an Open Source Blade Geometry Generator Using a Physics Based Parameterization With the Engineering Sketch Pad,” 2020. <https://doi.org/10.1115/GT2020-15943>
- [23] Ha, M., Holder, J., Ghimire, S., Ringheisen, A., and Turner, M. G., “Detailed Design and Optimization of the First Stage of an Axial Supercritical CO₂ Compressor,” 2022. <https://doi.org/10.1115/GT2022-82590>
- [24] Ansys, “Ansys DesignModeler User’s Guide,” 2022.
- [25] Ansys, “Ansys Mechanical.” Retrieved 15 May 2023. <https://www.ansys.com/products/structures/ansys-mechanical>
- [26] McKay, M. D., Beckman, R. J., and Conover, W. J., “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code,” *Technometrics*, Vol. 21, No. 2, 1979, p. 239. <https://doi.org/10.2307/1268522>
- [27] The SciPy community, “SciPy Latin Hypercube Sampling,” SciPy Documentation, 2023. Retrieved 12 October 2023. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.LatinHypercube.html>
- [28] Fang, K.-T., Ma, C.-X., and Winker, P., “Centered L_2 -Discrepancy of Random Sampling and Latin Hypercube Design, and Construction of Uniform Designs,” *Mathematics of Computation*, Vol. 71, No. 237, 2000, pp. 275–297. <https://doi.org/10.1090/S0025-5718-00-01281-3>
- [29] Tang, B., “Orthogonal Array-Based Latin Hypercubes,” *Journal of the American Statistical Association*, Vol. 88, No. 424, 1993, p. 1392. <https://doi.org/10.2307/2291282>
- [30] Scikit-Learn developers, “Sklearn.Neural_network.MLPRegressor,” Sckit Learn Documentation, 2023. Retrieved 12 October 2023. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
- [31] Collobert, R., and Bengio, S., “Links between Perceptrons, MLPs and SVMs,” 2004. <https://doi.org/10.1145/1015330.1015415>
- [32] The OpenMDAO Development Team, “ScipyOptimizeDriver,” OpenMDAO Web Documentation, 2022. Retrieved 15 October 2023. https://openmdao.org/newdocs/versions/latest/features/building_blocks/drivers/scipy_optimize_driver.html
- [33] The SciPy community, “Scipy.Optimize.Minimize,” 2023. Retrieved 15 October 2023. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html>
- [34] Kraft, D., “A Software Package for Sequential Quadratic Programming,” Cologne, 1988.
- [35] The OpenMDAO Development Team, “SimpleGADriver,” OpenMDAO Web Documentation, 2022. Retrieved 18 October 2023. https://openmdao.org/newdocs/versions/latest/features/building_blocks/drivers/genetic_algorithm.html

- [36] Gencay, R., and Min Qi, "Pricing and Hedging Derivative Securities with Neural Networks: Bayesian Regularization, Early Stopping, and Bagging," *IEEE Transactions on Neural Networks*, Vol. 12, No. 4, 2001, pp. 726–734. <https://doi.org/10.1109/72.935086>