Design and Optimization of Heat Exchangers Using Large Language Models

Sandeep Mishra[1,†], Vishal Jadhav[1,†], Shirish Karande[1] and Venkataramana Runkana[1,*]

[1] *Tata Research Development and Design Centre, Pune, Maharashtra, India*

Abstract

Heat exchangers (HEs) are essential in process industries for efficient thermal energy transfer. Their design and optimization are crucial for improving energy efficiency, reducing costs, and ensuring reliable system performance. However, these tasks are complex due to varying fluid properties, phase changes, and fouling. This study proposes the HxLLM framework, utilizing Large Language Models (LLMs) to aid in the design and optimization of HEs. The framework identifies the mathematical model for heat transfer in HEs, followed by retrieval-augmented generation (RAG) based code generation and correction. In this study, a repository was created by extracting mathematical models from relevant literature along with common errors observed in such tasks. These repositories, combined with carefully crafted prompts, were used to extract the mathematical model and generate the corresponding code within this framework. We observed that LLMs can effectively identify and generate initial code for mathematical models, though first responses often needed corrections. The RAG approach for code correction significantly enhanced code accuracy. This study demonstrates that LLMs, with a RAG framework, can automate and improve the design and optimization process of HEs, offering a promising tool for engineers and researchers to achieve better efficiency and cost-effectiveness.

Keywords

Heat Exchangers, Design and Optimization, Large Language Models, Retrieval Augmented Generation

1. Introduction

A heat exchanger is a device designed to transfer heat between two or more fluids. It plays a critical role in various industries such as chemical processing, power generation, HVAC systems, automotive, and renewable energy [1]. The design and optimization of heat exchangers are vital for enhancing energy efficiency, reducing operational costs, and minimizing environmental impact. Effective heat exchanger design ensures the efficient transfer of thermal energy, leading to significant energy savings and reduced greenhouse gas emissions [2]. Currently, methods such as the Log-Mean Temperature Difference (LMTD), Effectiveness-Number of Transfer Units (ϵ-NTU), Computational Fluid Dynamics (CFD) [3], heuristic search techniques like Genetic Algorithms and artificial intelligence (AI) and machine learning(ML) algorithms based techniques like artificial neural networks (ANNs) are used for the design and optimization of heat exchangers [4, 5]. The typical steps in design and optimization of heat exchanger involve problem definition, selection of design method, initial design, detailed analysis, optimization, validation and testing, iteration, and implementation. These steps ensure the heat exchanger operates at optimal efficiency, meeting both economic and environmental goals [5] .

Designing and optimizing heat exchangers are challenging tasks, especially since these systems operate in harsh environments such as power plants. Researchers have explored various surrogate modeling approaches, including machine learning [6, 7, 8, 9, 10] and deep learning [11, 12, 13, 14, 15, 16], to accelerate this process (further details on these approaches can be found in appendix 6). While these methods have demonstrated improvements in design and optimization, implementing them for new designs requires engineers to undergo extensive training. Consequently, applying these approaches to new design problems can be time-consuming.

Recently, large language models (LLMs) have shown significant potential in design and optimization within the engineering domain. Pluhacek et al. [17] used LLMs like GPT-4 to create new hybrid swarm intelligence optimization algorithms, highlighting their innovative role in tackling optimization challenges. Ma et al. [18] found that LLM-generated design solutions were more feasible and useful than crowdsourced ones, enhancing design quality through advanced natural language processing. Sabbatella et al., [19] focused on prompt optimization to improve LLM performance, emphasizing the importance of fine-tuning. Ma et al,[20] introduced LLaMoCo, a framework that fine-tunes LLMs for optimization tasks, showing superior performance compared to other models. Yang et al. [21] frame LLMs as optimizers, using them to generate candidate solutions from natural language prompts in an iterative loop. Liu et al.[22] sur-

KiL'24: Workshop on Knowledge-infused Learning co-located with 30th ACM KDD Conference, August 26, 2024, Barcelona, Spain

*Corresponding author.

† These authors contributed equally.

sandeepm.mishra@tcs.com (S. Mishra); vi.suja@tcs.com (V. Jadhav); shirish.karande@tcs.com (S. Karande); venkat.runkana@tcs.com (V. Runkana)

veyed ChatGPT-related research, highlighting LLM applications in education, medicine, and physics, driven by innovations in pre-training and fine-tuning. Kashefi and Mukerji [23] directly explore using ChatGPT [24] to generate code for numerical algorithms, with promising results. Yao et al[25] introduces frameworks like the "Tree of Thoughts" enable LLMs to explore multiple reasoning paths and make global decisions for challenging tasks. Wang et al.[26] propose an interactive "chain of repairing" approach where LLM agents work together to iteratively debug and improve generated code. Xu et al.[27] proposes integrating information retrieval into the LLM reasoning chain for knowledge-intensive tasks, enabling the model to dynamically incorporate relevant information and modify its reasoning trajectory. Additionally, Li and Mellou [28] introduce OptiGuide, a framework utilizing LLMs to bridge the gap between supply chain automation and human comprehension. Chen et al. [29] present OptiChat, a system equipped with a chatbot GUI for diagnosing infeasible optimization models using natural language interactions. Liu et al. [30] introduce CodeMind, a framework designed to evaluate the code reasoning abilities of LLMs. CodeMind supports three code reasoning tasks and shows that while LLMs can follow control flow constructs and explain how inputs evolve to output for simple programs, their performance drops for more complex code. Also Ni and Buehler [31] demonstrate how teams of interacting LLM agents can autonomously collaborate to solve mechanics problems, write code, and incorporate domain knowledge. Ahmadi Teshnizi et al.[32] propose OptiMUS, an LLM-based agent for formulating and solving mixed integer linear programming (MILP) problems from natural language descriptions. In summary, above studies have shown their effectiveness of LLMs in creating new algorithms, enhancing design solutions, improving optimization tasks, and generating code. Various frameworks and approaches, such as LLaMoCo, OptiGuide, and OptiChat, demonstrate LLMs' capabilities in tackling complex problems, including optimization, code reasoning, and collaborative tasks.

Inspired by these developments, this study explores the application of LLMs for automating the design and optimization of heat exchangers with HxLLM framework. The primary objective is to develop a system that can generate mathematical models and optimization algorithms based on user-input, leveraging the capabilities of LLMs to determine optimal parameter values and perform cost optimization. By extracting relevant information from research articles, LLMs could potentially generate models and propose optimized designs through an iterative, interactive process, reducing the manual effort required.

2. Methodology

Designing and optimizing a heat exchanger typically involves the following steps: defining the problem, selecting a design method, optimizing, validating and testing, iterating, and implementing. The design method selection includes the mathematical formulation describing heat transfer, while optimization involves finding the optimal design parameters by evaluating various pairs of heat transfer metrics and their corresponding design parameters.

To mimic real-world scenarios, this study relies on available literature on heat exchanger design and optimization. Initially, some literature texts were provided as context to the LLM model with prompts to extract mathematical model summaries. These summaries were then stored in the mathematical model summary repository. For each of these mathematical models, an expert wrote the Python script to evaluate the mathematical model equations. Also, the frequent errors were stored in the code error repository. The details of the mathematical model repository and code error repository are as follows:

Mathematical Model Repository Our repository, a CSV file, currently contains over 115 entries, organized into three columns: ID, Summary, and Code. The Summary section comprises summaries of articles, which were generated using prompts (Table 1) for research articles. After creating the code for each mathematical model, we processed it through an error correction framework to ensure its accuracy. Once refined, the code was added to the Code section corresponding to its respective summary.

Code Error Repository This repository is a collection of complex errors that the LLM had difficulty solving, documented as metadata. Each entry includes the error description and its solution, provided as the page content of a document object.

We propose the HxLLM framework for LLM-assisted design and optimization of heat exchangers. It has three main components: mathematical model identification, code retrieval for the mathematical model, and code generation and correction. In subsequent sections, we discuss the components of the HxLLM framework and how it utilizes the mathematical model repository and code error repository.

2.1. Mathematical model identification

This component extracts the mathematical model from the user input. User input can also include a research article that the user might want to use for a new design. Figure 1 (a) shows the mathematical model identification

Table 1
Prompt for extracting the mathematical model summary from user input

<paper> {raw_text} </paper> Please carefully read through the mathematical model section of the paper above. Identify and list all the unique equations that are used to construct the model. For each equation you find: - Give it a specific, descriptive name based on what it represents or calculates. For example, if there is an equation for "tube side heat transfer coefficient", name it something like "Tube side heat transfer coefficient (ht)". - Briefly describe how to calculate the equation based on the information in the paper. Avoid using generic labels like "Equation ()" for the names. Also avoid using phrases like "Calculated using equation ()" in your descriptions. Instead, just provide a general description of how each equation is calculated. Output your final list of named and described equations inside <equations>tags.

component. The user provides scientific text or a research article (in PDF file format), which is then processed by the LLM. The LLM is prompted (see table 1 for the prompt provided to the LLM) to summarize the mathematical model described in the article, focusing solely on its description instead of detailed equations. This approach was chosen due to challenges with accurately parsing and reading equations (mathematical expressions present in the article) using our current PyPDF reader for text extraction from the research article PDF file.

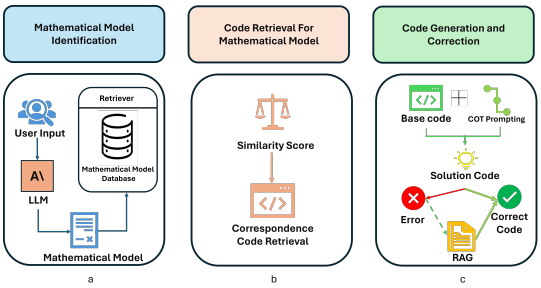

Figure 1: HxLLM Framework

Further, the generated mathematical model is compared with mathematical models available in mathematical model summary database. Using TF-IDF vectorizer cosine similarity score is evaluated between mathematical model summary generated by LLM and model summary in mathematical model summary repository.

2.2. Code retrieval for mathematical model

The highest similarity score obtained with mathematical model extraction component, is then compared with threshold similarity score. In this case, if the similarity score is greater than 75%, the summary corresponding to the highest similarity score is extracted. Subsequently, we retrieve the code associated with that summary, which serves as our sample base code for further processing. The details of this step are illustrated in Fig. 1 (b). If the similarity score is less than 75%,then the paper is new and does not match any summary in our repository, and an alternative approach is employed. This approach pairs a most similar research article mathematical model with its corresponding code and, when presented with a new research article, uses a sequential prompting technique to produce the desired output.This essentially represents a few-shot prompting example, as illustrated in Fig. 2

2.3. Code generation using LLM

The sample base code obtained from previous component, along with chain of thought prompting technique, enables the LLM to generate new code through sequential prompting (see prompts in table 2 and table 3). In this component, we created the error database for errors that were encountered while creating the code for each mathematical model (see Fig. 1(c)). It contains a list of potential errors encountered in mathematical models for heat exchangers. Code generated by LLM is executed, and if the execution results in an error , then RAG based technique is used to get similar error and its resolution . Then again the prompt with similar error and its resolution is given to the LLM (see prompt in table 4 and table prompt 5). This iteration continues until specific number of times (in this case 3) or terminates early if the execution is successful. In case code exits with error then human provides the resolution of the error and then once again the code generation loop continues.

2.4. Code Generation for Optimization Algorithm

The final stage in our methodology involves generating the code for the optimization algorithm, following a similar approach to the code generation component. We start by providing a sequential prompt to generate the initial code. This code then goes through an iterative correction process to identify and fix any complex errors, leveraging suggestions from the RAG system.Here are the two prompts(Table:6,Table:7) that we have used for generation of final optimisation algorithm.

After utilizing the Chain of Thought prompting to generate the code for the optimization algorithm using

Table 2
Prompt for code generation, by detecting which part of base code required change

Your task is to recreate the Python code for the specified research paper. The goal is to generate code that can be directly copied and pasted for execution on a user's system. You will be provided with a series of prompts to guide you through the process. <paper> {a} </paper> First, carefully study the mathematical models and equations provided in the paper above. Pay close attention to the details and notations used. <sample_code> {a1} </sample_code> Next, compare the sample code provided above with the original paper. Identify which functions need to be changed based on the equations and models described in the paper. When writing the code, make sure to write out the full formula. As an example, for equation (3) instead of simply representing it as a comment like # equation (3). This will ensure clarity and completeness. To optimize your work, only write the equations that are different and need to be changed. If a function is already the same as in the original paper, there's no need to rewrite it. Remember, it's crucial that any generated results are complete and accurate. Double-check your work to ensure the code aligns with the paper's specifications. Please provide your response inside <code>tags. <code> Your generated code goes here. </code> You don't need to provide the entire code in one go. I will provide more prompts in a sequence to guide you through the process. Let's work together to recreate the code accurately and efficiently.

Table 3
Prompt for Generating New Code for Mathematical Model

<original_code> {a1} </original_code> And here are the modified lines of code: <modified_code> {b} </modified_code> The specific lines or code blocks that need to be replaced from the modified code into the original code are: {b} Please carefully examine the original code and modified code provided. Your task is to surgically replace the lines/blocks specified in modified code from the <modified_code>into the appropriate locations in the <original_code>. When doing the replacements, make sure the modified code integrates properly with the surrounding original code. Fix any mismatched variable names, indentation issues, or other small inconsistencies introduced by inserting the modified code. Once you have an updated version of the code with the changes incorporated, your next step is to create a unified linking function. This function should contain all the key parameters used throughout the different code blocks. The goal is to have a single master function that can be called with all the necessary parameters, which will then appropriately call the other functions and code blocks in the proper sequence to produce the end result. When you have completed the unified linking function, test the code to make sure it runs without any errors. Debug if necessary. Finally, provide the full corrected code, from first line to last line, inside <corrected_code>tags. Format it cleanly so that I can directly copy and paste your code and run it without issues.

the previously created mathematical model as the objective function, we then pass this code through our final code correction framework. This framework employs the same RAG techniques to produce the correct final code.

2.5. HxLLM Workflow

In summary, HxLLM framework takes user input in the form of research article PDF and generates the code for mathematical model and optimization algorithm mentioned in the research article. The entire workflow is presented in Fig, 2. Initially, the workflow processes a research article focused primarily on heat exchanger models and their cost optimization. Using LLM and prompt engineering, we identify the mathematical model. We then compare the provided article with a database containing mathematical models from multiple papers to check for similarity. If a match is found, we retrieve the base code as a reference for the LLM to generate the mathematical model described in the paper. To ensure the accuracy of the generated code, we incorporate a code correction framework along with RAG techniques. If the base code is not present, we use few-shot examples to generate the mathematical model and subsequently the optimization model.

3. Results and Discussion

In this study, we used the Anthropic Claude 3 Opus model as the LLM, and several research articles, including [33] and [34], were utilized as user input.

We applied HxLLM framework to 115 research articles. In the following subsections, we describe representative results of our framework for research articles with similar

Table 4
Prompt for Error Correction

Here is a Python code that encountered an error when run: <code> {code} </code> The error message was: {error_message} Please carefully read the error message and identify which line number and what part of the code is causing the error. Then, make the necessary changes to fix the code so that it will run without any errors. Provide the full corrected code inside <fixed_code>tags. The pattern should be <fixed_code>Corrected version of the code </fixed_code>. Double check it so that it should not contain any type of other errors and should return results.

Table 5
Prompt for Error Correction with RAG

Here is a Python code that encountered an error when run: <code> {code} </code> The error message was: {error_message} The solution you can refer to solve this problem is: {Solution} Please carefully read the error message and identify which line number and what part of the code is causing the error. <thinking> Think step-by-step about: 1. What the error message is saying 2. Which line of code it references 3. How the provided solution addresses the issue </thinking> Then, make the necessary changes to fix the code so that it will run without any errors. Provide the full corrected code inside <fixed_code>tags.

mathematical models present in the mathematical model repository, as well as for research articles without similar models in the repository. Finally, we discuss optimization algorithm code generation. .

Table 6
Prompt - 1 for code generation for final optimisation algorithm

You will be optimizing the mathematical model of a heat exchanger based on provided code and an optimization algorithm from a research paper. Here is the code for the heat exchanger model: <code> {Code} </code> And here is the paper having the optimization algorithm: <paper> {raw_text} </paper> Please carefully read through the code and paper to understand the heat exchanger model and the proposed optimization approach. After reviewing the materials, please identify the target/design variables that the paper aims to optimize, along with any specified limits on those variables. List out each variable and its limits (if given). Next, find the unified linking function described in the paper. Show how to incorporate this linking function into the optimization algorithm. Then, write out the full code that performs the optimization of the heat exchanger model. Use the target/design variables, limits, and unified linking function you identified from the paper. Aim to closely follow the optimization approach from the paper while integrating it with the existing heat exchanger model code. Before outputting the final code, take a moment to double check that your code follows the optimization algorithm correctly and that you have used the right variables and limits. Think through the optimization process step-by-step to verify the logic. Finally, output the complete optimized code inside <optimized_code>tags. Also output the final optimized values of the target/design variables inside <optimized_values>tags. Remember, the goal is to implement the paper's optimization approach to find optimal values for the heat exchanger model's design variables. Let me know if you have any other questions!

3.1. Model Extraction and Code Generation for Similar Research Articles

3.1.1. Mathematical Model Extraction

As mentioned before, it is expected that user will provide the scientific text containing the mathematical model for heat exchanger design. This text is passed as context along with prompt (mentioned in table 1 to LLM for extraction of the mathematical model and its parameters. For this study, we have considered the research article by [33] as the scientific text example. The mathematical model extracted by LLM for this research article is given

Table 7
Prompt – 2 for code generation for final optimisation algorithm

You will be combining two code snippets, one for a mathematical model and another for an optimization algorithm, to produce a final output that can be copy-pasted and run.
Here is the code for the mathematical model:
<code1>
{Code}
</code1>
And here is the code for the optimization algorithm:
<code2>
{d}
</code2>
Please carefully review both code snippets and think about how they should be combined to produce a working final output. Write out your thought process for combining the code inside <scratchpad>tags.
<scratchpad>
</scratchpad>
Once you have determined the best way to combine the code snippets, please provide the final combined code that can be copy-pasted and run inside <combined_code>tags.
<combined_code>
</combined_code>

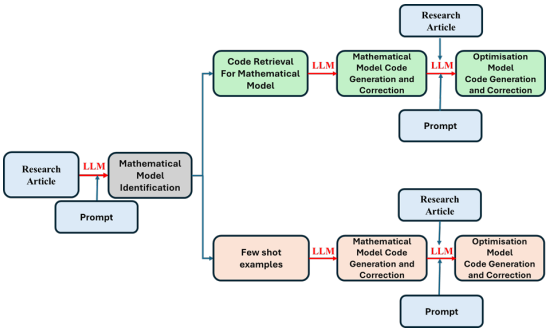

Figure 2: HxLLM Framework Workflow

in table 8.

3.1.2. Code Retrieval for Mathematical Model

Post extraction of the mathematical model summary, we identify the similar mathematical model present in our mathematical model repository. We experimented with several retrieval approaches like retrieval using LLM, parent document retriever and RAG fusion with query duplication and reformulation along with TF-IDF based retrieval method, to identify the similar mathematical model from repository. However, we observed that TF-IDF based retrieval was providing the satisfactory result

Table 8
LLM-Generated Summary for User-Provided Article

1. Tube side heat transfer coefficient for laminar flow (ht_laminar): Calculated using the tube inside diameter, thermal conductivity, tube side Reynolds number, tube side Prandtl number, and tube length.
2. Tube side heat transfer coefficient for transition flow (ht_transition): Calculated using the tube inside diameter, thermal conductivity, tube side Reynolds number, tube side Prandtl number, tube length, and the Darcy friction factor.
3. Tube side heat transfer coefficient for turbulent flow (ht_turbulent): Calculated using the tube outside diameter, thermal conductivity, tube side Reynolds number, tube side Prandtl number, tube side dynamic viscosity, and wall dynamic viscosity.
4. Darcy friction factor (ft): Calculated using the tube side Reynolds number.
5. Tube side Reynolds number (Ret): Calculated using the tube side fluid density, tube side flow velocity, tube inside diameter, and tube side dynamic viscosity.
6. Tube side flow velocity (vt): Calculated using the tube side mass flow rate, tube inside diameter, tube side fluid density, number of tube passes, and number of tubes.
7. Number of tubes (Nt): Calculated approximately using the shell inside diameter, tube outside diameter, number of tube passes, and coefficients that depend on the flow arrangement and number of passes.
8. Tube side Prandtl number (Prt): Calculated using the tube side dynamic viscosity, tube side specific heat, and tube side thermal conductivity.
9. Shell side heat transfer coefficient (hs): Calculated using the shell equivalent diameter, thermal conductivity, shell side Reynolds number, shell side Prandtl number, shell side dynamic viscosity, and wall dynamic viscosity.
...continued...

compared to other approaches. Hence, TF-IDF vectorizer was used to identify the similar mathematical model with cosine similarity as metric for the similarity. Threshold of 75% for the similarity metric (through trial and error) was selected to decide if the mathematical model extracted by LLM is similar to any of the model present in the repository.

Output of the TF-IDF vectorizer based retrieval for mathematical model similar to the extracted mathematical model is shown in table:9. It can be seen that identified similar mathematical model has similarity score of 0.924 as the elements like, tube side heat transfer coefficient, friction factor etc are common in both extracted mathematical model (see table 8) and identified similar mathematical model (see table 9).

3.1.3. Code generation and Correction

Post identifying the similar mathematical model from the repository, the python code corresponding to similar

Table 9
Output of TF-IDF vectorizer method showing the similarity scores and closest summary

Output of TF-IDF vectorizer method
Closest summary index: 11
Similarity score: 0.924
Closest summary:
1. Heat transfer area (A): Calculated using the heat transfer rate, overall heat transfer coefficient, LMTD correction factor, and logarithmic mean temperature difference. 2. Heat transfer rate (Q): Determined from an energy balance using the mass flow rates and specific heats of the hot and cold fluids along with their inlet and outlet temperatures. 3. Tube side flow velocity (Vi): Calculated from the tube side mass flow rate, fluid density, number of tubes, number of tube passes, and tube inner diameter. 4. Number of tubes (NT): Estimated using an empirical correlation based on the shell diameter, tube outer diameter, and coefficients that depend on the tube layout and number of passes. 5. Tube side Reynolds number (Re): Calculated from the tube side flow velocity, tube inner diameter, and kinematic viscosity of the tube side fluid. ...continued...

Table 10
List of functions to be updated to match key equations from the original paper

Block-2 function `calculate_fs()` - Replace with correct equation for friction factor `fs` from the paper. Block-7 function `Tube_side_heat_transfer_coefficient()` - Replace `ht` equations with correct ones from paper. Block-12 function `Shell_Side_heat_transfer_coefficient()` - Replace `hs` equation with correct one from paper. Block-13 function for overall heat transfer coefficient `U` - Replace with equation (2) from paper. Block-15 function `Correction_factor()` - Replace with correct `F` equation from paper. Block-19 function `calculate_Cod()` - Verify discounting equation matches paper. Block-20 function `Calculate_Total_Cost()` - Verify capital cost equation matches paper.

mathematical model was taken as base code (see appendix 5.1) for further processing. Since our aim is to get the code for extracted mathematical model, we first identify the changes required in base code by passing the base code as context with prompt mentioned in table 2 to LLM and ask LLM to identify the changes required in base code. The changes suggested by the LLM are given in table 10.

The changes mentioned in LLM response illustrates the equations present within the original codebase that necessitate alterations. It details which segments in the base code should be modified to ensure the accuracy and logical integrity of extracted mathematical model. Following this, we provided the next prompt (Table:3), instructing the model to include these changes in the base code and rewrite it coherently. The generated code for this step is given in appendix 5.2.

To ensure that, the generated code is accurate and runs without any errors, this code was passed to the code correction framework. The errors encountered during the execution of the code, are presented in figure 3. Simple errors (not present in the error repository) was resolved by prompting (see table 4) LLM and errors which was not so obvious for LLM (listed in error repository) were resolved through RAG based approach with prompt (given in table 5). It may happen, though, that errors may still not be resolved; in that case, the error was resolved manually, and the solution for the error was added to the error repository for further reference by LLM.

By learning from past errors in this way, the RAG system can iteratively improve the quality and accuracy of the code it produces. The final result of the error-free mathematical code generated by the LLM is provided in (Appendix:5.2).

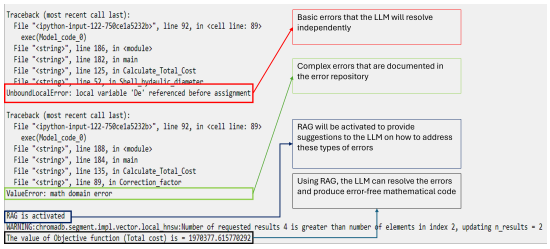

Figure 3: Error Correction Framework Response

3.2. Model Extraction and Code Generation for Non-Similar Research Articles

In the previous section 3.1, we described the steps for user input for which we had similar mathematical model. However, in practical scenario user may come up with models that may not be similar to model available in mathematical model repository. In such scenario, we adopted the different steps for code generation of mathematical models. These steps are described below:

3.2.1. Mathematical Model Extraction

We have selected the research article [34], for which similar mathematical model is not available in the model

Table 11
Summary generated by LLM for dissimilar paper

Shear stress constraint($C1(x)$): Ensures the shear stress is less than the maximum allowable shear stress S, calculated using the maximum load P_{max}, coil diameter D, and wire diameter d. Free length constraint($C2(x)$): Ensures the free length is greater than the minimum required length, calculated using the deflection δ, number of coils n, and wire diameter d. Minimum wire diameter constraint($C3(x)$): Ensures the wire diameter is greater than the minimum allowable wire diameter d_{min}. Continued...

repository. The provided paper mainly discusses the application of Rao algorithms to optimize mechanical system component designs, assessing their comparative effectiveness against established methods in addressing complex constraints and mixed-type variables. However, our database contains papers focused on the cost optimization of heat exchangers. We use the prompt (given in table 1) to generate the mathematical model summary. LLM generated summary is given in table 11.

3.2.2. Code Retrieval for Mathematical Model

The mathematical model summary (in table 11) when compared using TF-IDF vectorizer based retriever, we found that, similarity index was 0.512, indicating that no mathematical model closely matches the generated mathematical model summary. Hence, the sample code was taken as reference base code for the step of code generation.

3.2.3. Code generation and Correction

Since we are not using the base code which is similar to the mathematical model, in this case we ask the LLM to generate the code from scratch with sample code using few-shot example prompting approach (Table:12). We provided a closely matched reference paper, the mathematical model related to the paper, and associated it with the prompt in a way that the model learns from the example how to build the mathematical model for an optimization algorithm problem. The result generated with this approach is given in appendix 5.3.

This code demonstrates that Claude 3 Opus is a very capable model, able to learn from examples and write code in a manner required for our case. However, the code generated by the LLM needs to be executed to ensure correctness. For this, we sent the code to our code correction framework (as described in sub section 3.1.3), and the output of that framework after multiple correction iterations, is given in appendix:5.4.

Table 12
Mathematical code generation prompt for dissimilar paper

You will be recreating the Python code for a specified research paper. The goal is to generate code that can be directly copied and pasted for execution. I will provide you with a series of prompts to guide you through the process.
First, here is a sample paper for your reference:
`<sample_paper>`
`{SAMPLE_PAPER}`
`</sample_paper>`
And here is the sample code associated with the mathematical model from that paper:
`<sample_code>`
`{SAMPLE_CODE}`
`</sample_code>`
Note that the sample code may be entirely different from the paper you will be working with. It is only provided to give you an idea of how to proceed in creating the code for the mathematical model of the given paper. The mathematical expressions and logic in your paper may be different.
Now, here is the paper you will be recreating the code for:
`<paper>`
`{PAPER}`
`</paper>`
Please carefully study the mathematical models and equations provided in this paper.
Next, write out the equations for all the formulas in the paper as Python functions, similar to how it was done in the sample code. First, you will need to figure out the logic and all the formulas. Then, write out each formula as a Python function.
If there are no mathematical models or formulas given in the paper, simply state `"No mathematical models or formulas are present in this paper."`
When writing the code, make sure to write out the full formula for clarity. For example, instead of representing equation (3) as a comment like `# equation (3)`, write out the entire equation. This will ensure completeness.
It's crucial that any generated results are complete and accurate. Double-check your work to ensure the code aligns with the paper's specifications.
Please provide your generated code inside `<code>` tags like this:
`<code>`
`Your generated code goes here.`
`</code>`
You don't need to provide the entire code all at once. I will provide more prompts in a sequence to guide you through the process. Let's work together to recreate the code accurately and efficiently.

3.3. Optimisation code generation

Once the mathematical model code is generated, it can be integrated with optimization algorithm for obtaining the configuration with optimal cost. It is achieved with two steps, first code generation for optimization algorithm and second integration of mathematical model

with optimization algorithm. The first step is performed in this study, by providing the user provided research article as context to LLM with prompt (Table:6), to generate the code for optimization. Here we have illustrated the output of this approach for research article by Patel and Rao [33]. This article [33] describes the use of PSO (Particle Swarm Optimization) techniques, where the target variables are three design variables: shell internal diameter, outer tube diameter, and baffle spacing. These variables are considered for optimization. As per the prompt, the LLM correctly identified the design variables along with their limits. It also accurately replicated the PSO algorithm as described in the paper to optimize the cost function and determine the values of these target variables. The code generated by LLM is given in appendix:5.5.

In next step, prompt (given in table7), that instructs LLM to combine both the mathematical model and the optimization code to produce the final code for optimization. This combined code was then passed through the code correction framework. The output of code correction framework is presented in appendix:5.6.

The LLM performed remarkably well in identifying and writing most of the code logic accurately. However, the final answer did not exactly match the results described in the paper, as some of the parameter values were guessed by LLM since it was not present in the article provided by user. Also the code generated by LLM relied on the pdf text, which did not have accurate information of mathematical model equation due to limitation of pdf text parsing.

3.4. Limitations and Future Work

Research articles are often in pdf format, and parsing these pdfs to text can result in loss of equation information, table or graph image information. In present study, we have ignored this loss of information. However, the concrete methodology to parse the information in pdf files or any other similar files which contains multiple type of information needs to formulated for better comprehension of context by LLMs. Combination of methods like, use of neural networks [35], vision transformer based models [36] or LLM based models [37, 38] can be explored further for overcoming this limitation. The code generation capability of the LLMs have improved in recent times, however LLMs require human inputs (prompts) code generation of complete design and optimization of heat exchangers and can not be achieved in one shot. This further can be improved with agent based workflows for better code generation. Also, the current benchmark studies does not consider requirements of such design and optimization workflows and can't be used for evaluating the framework proposed in this study and human evaluation approach was selected for analysis. Also, the current framework relies heavily on the user input for the mathematical model, its parameters and optimization algorithm to be used. To reduce this dependency on user input, the external data repositories (like the mathematical model repository and code error repository mentioned in this framework) for material properties, optimization methodologies, program synthesis, domain specific information etc. can be integrated in this framework to obtain the reliable and comprehensive framework for design and optimization across different design problems. Developing such framework, is the future goal of the present authors.

4. Conclusion

This study introduced the HxLLM framework, leveraging Large Language Models (LLMs) to automate the design and optimization of heat exchangers (HEs). Our approach integrated mathematical model extraction, code generation, and error correction using a RAG framework. The LLMs effectively identified and generated initial code for the mathematical models, although initial responses often required corrections. Our results demonstrated that LLMs, when combined with RAG, offer a promising tool for automating the design and optimization processes of HEs. This can lead to increased energy efficiency and reduced costs in the process industry. However, the study also highlighted the limitations of current LLM capabilities, particularly in handling diverse mathematical models and optimizing complex designs without substantial human input. Future work should focus on enhancing the LLM's ability to parse complex information from various document formats and reducing reliance on user input by integrating extensive external data repositories. Additionally, incorporating agent-based workflows may further improve the code generation process. These advancements will help create a more comprehensive and reliable framework for designing and optimizing diverse engineering systems.

References

[1] J. B. B. Rao, V. R. Raju, Numerical and heat transfer analysis of shell and tube heat exchanger with circular and elliptical tubes, International Journal of Mechanical and Materials Engineering 11 (2016) 2198–2791. URL: https://doi.org/10.1186/s40712-016-0059-x. doi:10.1186/s40712-016-0059-x.

[2] S. Jung, H. Jung, Y. Ahn, Optimal economic–environmental design of heat exchanger network in naphtha cracking center considering fuel type and co2 emissions, Energies 15 (2022).

URL: https://www.mdpi.com/1996-1073/15/24/9538. doi:10.3390/en15249538.

[3] S. Hall, 2 - heat exchangers, in: S. Hall (Ed.), Branan's Rules of Thumb for Chemical Engineers (Fifth Edition), fifth edition ed., Butterworth-Heinemann, Oxford, 2012, pp. 27–57. URL: https://www.sciencedirect.com/science/article/pii/B9780123877857000025. doi:https://doi.org/10.1016/B978-0-12-387785-7.00002-5.

[4] K. J. Bell, Heat Exchanger Design for the Process Industries , Journal of Heat Transfer 126 (2005) 877–885. URL: https://doi.org/10.1115/1.1833366. doi:10.1115/1.1833366.

[5] S. Kharaji, Heat exchanger design and optimization, in: L. C. Gómez, V. M. V. Flores, M. N. Procopio (Eds.), Heat Exchangers, IntechOpen, Rijeka, 2021. URL: https://doi.org/10.5772/intechopen.100450. doi:10.5772/intechopen.100450.

[6] M. Saeed, A. S. Berrouk, Y. F. Al Wahedi, M. P. Singh, I. A. Dagga, I. Afgan, Performance enhancement of a c-shaped printed circuit heat exchanger in supercritical co2 brayton cycle: A machine learning-based optimization study, Case Studies in Thermal Engineering 38 (2022) 102276. URL: https://www.sciencedirect.com/science/article/pii/S2214157X22005226. doi:https://doi.org/10.1016/j.csite.2022.102276.

[7] H. Keramati, F. Hamdullahpur, M. Barzegari, Deep reinforcement learning for heat exchanger shape optimization, International Journal of Heat and Mass Transfer 194 (2022) 123112. URL: https://www.sciencedirect.com/science/article/pii/S001793102200583X. doi:https://doi.org/10.1016/j.ijheatmasstransfer.2022.123112.

[8] J. Zou, T. Hirokawa, J. An, L. Huang, J. Camm, Recent advances in the applications of machine learning methods for heat exchanger modeling—a review, Frontiers in Energy Research 11 (2023). URL: https://www.frontiersin.org/articles/10.3389/fenrg.2023.1294531. doi:10.3389/fenrg.2023.1294531.

[9] L. Huang, J. Zou, B. Liu, Z. Jin, J. Qian, Machine learning assisted microchannel geometric optimization—a case study of channel designs, Energies 17 (2024). URL: https://www.mdpi.com/1996-1073/17/1/44. doi:10.3390/en17010044.

[10] E. Efatinasab, N. Irannezhad, M. Rampazzo, A. Diani, Machine and deep learning driven models for the design of heat exchangers with micro-finned tubes, Energy and AI 16 (2024) 100370. URL: https://www.sciencedirect.com/science/article/pii/S2666546824000363. doi:https://doi.org/10.1016/j.egyai.2024.100370.

[11] A. Gupta, V. Jadhav, M. Patil, A. Deodhar, V. Runkana, Forecasting of Fouling in Air Pre-Heaters Through Deep Learning ASME 2021 Power Conference (2021). URL: https://doi.org/10.1115/POWER2021-64665. doi:10.1115/POWER2021-64665, v001T01A002.

[12] A. Gupta, V. Jadhav, A. Deodhar, V. Runkana, Physics-assisted long-short-term-memory network for forecasting of fouling in a regenerative heat exchanger, in: Heat Transfer Summer Conference, volume 85796, American Society of Mechanical Engineers, 2022, p. V001T20A001.

[13] S. Sundar, M. C. Rajagopal, H. Zhao, G. Kuntumalla, Y. Meng, H. C. Chang, C. Shao, P. Ferreira, N. Miljkovic, S. Sinha, S. Salapaka, Fouling modeling and prediction approach for heat exchangers using deep learning, International Journal of Heat and Mass Transfer 159 (2020) 120112. URL: https://www.sciencedirect.com/science/article/pii/S0017931020330489. doi:https://doi.org/10.1016/j.ijheatmasstransfer.2020.120112.

[14] V. Jadhav, A. Deodhar, A. Gupta, V. Runkana, Physics informed neural network for health monitoring of an air preheater PHM Society European Conference, 7(1) (2022). URL: https://doi.org/10.36001/phme.2022.v7i1.3343. doi:10.36001/phme.2022.v7i1.3343.

[15] R. Majumdar, V. Jadhav, A. Deodhar, S. Karande, L. Vig, V. Runkana, Real-time health monitoring of heat exchangers using hypernetworks and pinns, 2022. arXiv:2212.10032.

[16] Z. Wu, B. Zhang, H. Yu, J. Ren, M. Pan, C. He, Q. Chen, Accelerating heat exchanger design by combining physics-informed deep learning and transfer learning, Chemical Engineering Science 282 (2023) 119285. URL: https://www.sciencedirect.com/science/article/pii/S0009250923008412. doi:https://doi.org/10.1016/j.ces.2023.119285.

[17] M. Pluhacek, A. Kazikova, T. Kadavy, A. Viktorin, R. Senkerik, Leveraging large language models for the generation of novel metaheuristic optimization algorithms, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, p. 1812–1820. URL: https://doi.org/10.1145/3583133.3596401. doi:10.1145/3583133.3596401.

[18] K. Ma, D. Grandi, C. McComb, K. Goucher-Lambert, Conceptual design generation using large language models, 2023. arXiv:2306.01779.

[19] A. Sabbatella, A. Ponti, I. Giordani, A. Candelieri, F. Archetti, Prompt optimization in large language models, Mathematics 12 (2024). URL: https://www.mdpi.com/2227-7390/12/6/929. doi:10.3390/math12060929.

[20] Z. Ma, H. Guo, J. Chen, G. Peng, Z. Cao, Y. Ma,

Y.-J. Gong, Llamoco: Instruction tuning of large language models for optimization code generation, 2024. arXiv:2403.01131.

[21] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, X. Chen, Large language models as optimizers, 2024. arXiv:2309.03409.

[22] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu, Z. Wu, L. Zhao, D. Zhu, X. Li, N. Qiang, D. Shen, T. Liu, B. Ge, Summary of chatgpt-related research and perspective towards the future of large language models, Meta-Radiology 1 (2023) 100017. URL: https://www.sciencedirect.com/science/article/pii/S2950162823000176. doi:https://doi.org/10.1016/j.metrad.2023.100017.

[23] A. Kashefi, T. Mukerji, Chatgpt for programming numerical methods, 2023. arXiv:2303.12093.

[24] OpenAI, Chatgpt, https://www.openai.com/chatgpt, 2023. Accessed: 15 May 2024.

[25] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, 2023. arXiv:2305.10601.

[26] H. Wang, Z. Liu, S. Wang, G. Cui, N. Ding, Z. Liu, G. Yu, Intervenor: Prompting the coding ability of large language models with the interactive chain of repair, 2024. arXiv:2311.09868.

[27] S. Xu, L. Pang, H. Shen, X. Cheng, T.-S. Chua, Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks, 2024. arXiv:2304.14732.

[28] B. Li, K. Mellou, B. Zhang, J. Pathuri, I. Menache, Large language models for supply chain optimization, 2023. arXiv:2307.03875.

[29] H. Chen, G. E. Constante-Flores, C. Li, Diagnosing infeasible optimization problems using large language models, 2023. arXiv:2308.12923.

[30] C. Liu, S. D. Zhang, A. R. Ibrahimzada, R. Jabbarvand, Codemind: A framework to challenge large language models for code reasoning, 2024. arXiv:2402.09664.

[31] B. Ni, M. J. Buehler, Mechagents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge, 2023. arXiv:2311.08166.

[32] A. AhmadiTeshnizi, W. Gao, M. Udell, Optimus: Optimization modeling using mip solvers and large language models, 2023. arXiv:2310.06116.

[33] V. Patel, R. Rao, Design optimization of shell-and-tube heat exchanger using particle swarm optimization technique, Applied Thermal Engineering 30 (2010) 1417–1425. URL: https://www.sciencedirect.com/science/article/pii/S1359431110001080. doi:https://doi.org/10.1016/j.applthermaleng.2010.03.001.

[34] R. Venkata Rao, R. Pawar, Design of Mechanical Components Using Variants of Rao Algorithm, 2023, pp. 687–700. doi:10.1007/978-981-19-9285-8_64.

[35] Z. Wang, J.-C. Liu, Translating math formula images to latex sequences using deep neural networks with sequence-level training, International Journal on Document Analysis and Recognition (IJDAR) 24 (2021) 1–13. doi:10.1007/s10032-020-00360-2.

[36] L. Blecher, G. Cucurull, T. Scialom, R. Stojnic, Nougat: Neural optical understanding for academic documents, 2023. arXiv:2308.13418.

[37] LlamaIndex, Llamaparse, https://docs.llamaindex.ai/en/stable/module_guides/loading/connector/llama_parse/, 2024. Accessed: 15 May 2024.

[38] anthropic, claude3, https://anthropic.com/claude, 2024. Accessed: 15 May 2024.

5. Code Listing

5.1. Retrieved Base Code for similar research article

```
import math
import numpy as np

pie = 3.141

def Prandtl_number(mew,Cp,Kt):
   Pr = (mew*Cp)/Kt
   return Pr

def calculate_fs(Re):
   fs = 2 * 0.72 * Re**(-0.15) # equation (28)

   return fs

def Number_of_tubes(C, n, DG, do):
   NT = C * ((DG/do)**(-n)) # equation (4)
   return NT

def Flow_velocity_tube(m, di, rho, NT, s):
   Vi = m / ((math.pi/4) * (di**2) * rho * (
       NT/s))
   # equation (3)
   return Vi

def Reynolds_number_tube(rho_t,vt,di,mew_t):
   Re = (rho_t*vt*di)/mew_t
   return Re

def calculate_friction_factor(Re):
   ft = (1.82*math.log10(Re) - 1.64)**(-2) #
       equation (5)
   return ft
```

```
def Tube_side_heat_transfer_coefficient(di,
    Re, Pr, k):
   Nu = 0.023 * (Re**0.8) * (Pr**0.4) #
       equation (6)
   hi = (Nu * k) / di
   return hi

def Shell_hydaulic_diameter(St, do):
   if St == 1.25*do: # square pitch
      De = (1.27 / do) * (St**2 - 0.785*do**2)
      # equation (7a)
   else: # triangular pitch
      De = (1.10 / do) * (St**2 - 0.917*do**2)
      # equation (7b)
   return De

def Cross_section_area(St, do, e, DG):
   As = (St - do) * e * DG / St # equation (8)

   return As

def Flow_velocity_shell(ms,rho_s,As):
   vs = ms/(rho_s*As)
   return vs

def Reynolds_number_shell(ms,de,As,mew_s):
   re = (ms*de)/(As*mew_s)
   return re

def Shell_Side_heat_transfer_coefficient(k,
    De, Re, Pr, mu,
mu0, jh):
   Nu = jh * Re * (Pr**(1/3)) * ((mu/mu0)
       **0.14)
   # equation (11)
   ho = (Nu * k) / De
   return ho

def Overall_heat_transfer_coefficient(hi, di,
     do, Rfi, kw,
xw, Rfo, ho):
   K = 1 / ((do/(hi*di)) + (Rfi*do/di) + (xw/
       kw) + Rfo +
   (1/ho)) # equation (12)
   return K

def LMTD(Th_i,Th_o,Tc_i,Tc_o):
   deltaT1 = Th_i - Tc_o
   deltaT2 = Th_o - Tc_i
   lmtd = (deltaT1 - deltaT2) / math.log(
       deltaT1 / deltaT2)
   return lmtd

def Correction_factor(R, P):
   numerator = ((R**2 + 1)**0.5) / (R - 1)
   denominator = math.log((1 - P) / (1 - P*R))
        / math.log
   ((2 - P*(R + 1 - (R**2 + 1)**0.5)) /
   (2 - P*(R + 1 + (R**2 + 1)**0.5)))
   F = numerator * denominator # equation (20)

   return F

def Sensible_heat(Cp_h,mh,Th_i,Th_o):
   Q = mh*Cp_h*(Th_i-Th_o)
   return Q

def calculate_DPt(s, L, di, rho, Vi, mu, mu0,
     m):
   DPt = s * ((0.092 * (L/di) * ((rho * Vi**2)
       /2) *
   ((mu/mu0)**(-m))) + 2.5) * ((rho * Vi**2)
       /2)
   # equation (16)
   return DPt

def calculate_DPs(jf, DG, De, Le, rho, Vo, mu,
     mu0):
   DPs = 8 * jf * (DG/De) * (Le/DG) *
   ((rho * Vo**2)/2) * ((mu/mu0)**(-0.14))
   # equation (16)
   return DPs

def calculate_Cod(n, i, C_o):
   X = np.arange(1, n + 1) # Create an array
       of X
   values from 1 to n
   terms = C_o / (1 + i) ** X
   C_od = np.sum(terms)
   return C_od

def Calculate_Total_Cost(C, n, DG, do, m,
    rho_t, rho_s,
s, mew_t, L, Kt, St, e, ms, mew_s, Ks, Rfi,
    Rfo, Th_i,
Th_o, Tc_i, Tc_o, Cp_h, Cp_c, etta, ny, H, Ce,
     i, kw,
xw, jh, jf, mu0, m_exp):
   di = 0.8*do
   NT = Number_of_tubes(C, n, DG, do)
   Vi = Flow_velocity_tube(m, di, rho_t, NT,
       s)
   Re_t = Reynolds_number_tube(rho_t, Vi, di,
        mew_t)
   Pr_t = Prandtl_number(mew_t, Cp_c, Kt)
   hi = Tube_side_heat_transfer_coefficient(
       di, Re_t,
   Pr_t, Kt)

   De = Shell_hydaulic_diameter(St, do)
   As = Cross_section_area(St, do, e, DG)
   Vo = Flow_velocity_shell(ms, rho_s, As)
   Re_s = Reynolds_number_shell(ms, De, As,
       mew_s)
   Pr_s = Prandtl_number(mew_s, Cp_h, Ks)
   ho = Shell_Side_heat_transfer_coefficient(
       Ks, De, Re_s,
   Pr_s, mew_s, mu0, jh)
```

```
    K = Overall_heat_transfer_coefficient(hi,
        di, do, Rfi, kw,
    xw, Rfo, ho)
    lmtd = LMTD(Th_i, Th_o, Tc_i, Tc_o)
    R = (Th_i - Th_o) / (Tc_o - Tc_i)
    P = (Tc_o - Tc_i) / (Th_i - Tc_i)
    F = Correction_factor(R, P)
    A = (m*Cp_c*(Tc_o-Tc_i))/(K*F*lmtd)
    Span = (A/(math.pi*do*NT))

    DPt = calculate_DPt(s, Span, di, rho_t, Vi,
         mew_t, mu0,
    m_exp)
    DPs = calculate_DPs(jf, DG, De, L, rho_s,
        Vo, mew_s, mu0)

    a1 = 8000
    a2 = 259.2
    a3 = 0.93 # Exchanger made with Stainless
        steel for both
    shell and tubes
    C_i = a1 + a2 * (A**(a3/3))

    P = 1 / etta * (((m / rho_t) * DPt) + ((ms
         / rho_s) * DPs))
    C_o = P * Ce * H
    C_od = calculate_Cod(ny, i, C_o)

    C_tot = C_i + C_od
    return C_tot

# Unified linking function
def main():
    C = 0.158
    n = 2.263
    DG = 0.54
    do = 0.0254
    m = 18.80
    rho_t = 995
    rho_s = 850
    s = 4
    mew_t = 0.00358
    L = 4.88
    Kt = 0.13
    St = 1.25*do
    e = 0.127
    ms = 5.52
    mew_s = 0.0004
    Ks = 0.13
    Rfi = 0.00061
    Rfo = 0.00061
    Th_i = 199
    Th_o = 93.3
    Tc_i = 37.80
    Tc_o = 76.7
    Cp_h = 2.47
    Cp_c = 2.05
    etta = 0.9
    ny = 10
    H = 7000
    Ce = 0.12
    i = 10
    kw = 50
    xw = 0.0005
    jh = 0.0035
    jf = 0.0035
    mu0 = 0.0004
    m_exp = 0.25

    C_tot = Calculate_Total_Cost(C, n, DG, do,
         m, rho_t,
    rho_s, s, mew_t, L, Kt, St, e, ms, mew_s,
        Ks, Rfi, Rfo,
    Th_i, Th_o, Tc_i, Tc_o, Cp_h, Cp_c, etta,
        ny, H, Ce, i,
    kw, xw, jh, jf, mu0, m_exp)
    print(f""The value of Objective function (
        Total cost)
    is = {C_tot}"")

if __name__ == ""__main__"":
    main()
```

5.2. Final Mathematical code for similar research article

This is the final result of the error-free mathematical code generated by the LLM.

```
import math
import numpy as np

pie = 3.141

def Prandtl_number(mew,Cp,Kt):
    Pr = (mew*Cp)/Kt
    return Pr

def calculate_fs(b0, Re):
    fs = 2 * b0 * Re**(-0.15)
    return fs

def Number_of_tubes(C,n1,Ds,d0):
    Nt = C*((Ds/d0)**n1)
    return Nt

def Flow_velocity_tube(mt,dt,rho_t,n,Nt):
    vt = (mt*(n/Nt))/((pie/4)*(dt**2)*rho_t)
    return vt

def Reynolds_number_tube(rho_t,vt,di,mew_t):
    Re = (rho_t*vt*di)/mew_t
    return Re

def calculate_friction_factor(Re):
    ft = (1.82 * math.log10(Re) - 1.64)**(-2)
    return ft

def Tube_side_heat_transfer_coefficient(Kt,di
```

```
,Re,Pr,L,d0,mew_t,mew_w):
   if Re < 2300:
      ht = (Kt/di)*(3.657+(0.0677*(Re*Pr*(di/L
          ))
      **1.33)/(1+0.1*Pr*((Re*di/L)**0.3)))
   elif 2300 < Re < 10000:
      ft = calculate_friction_factor(Re)
      ht = (Kt/di)*((ft/8)*(Re-1000)*Pr)
          /(1+12.7
      *math.sqrt(ft/8)*(Pr**(2/3)-1))*(1+(di/L
          )**0.67)
   else:
      ht = 0.027*(Kt/d0)*(Re**0.8)*(Pr**(1/3))
          *
      ((mew_t/mew_w)**0.14)
   return ht

def Shell_hydaulic_diameter(Pt,d0):
   De = 4*(Pt**2 - (math.pi*d0**2)/4)/(math.
       pi*d0)
   return De

def Cross_section_area(Ds,b,C1):
   As = Ds*b*C1
   return As

def Flow_velocity_shell(ms,rho_s,As):
   vs = ms/(rho_s*As)
   return vs

def Reynolds_number_shell(ms,de,As,mew_s):
   re = (ms*de)/(As*mew_s)
   return re

def Shell_Side_heat_transfer_coefficient(Ks,
     De,
Re,Pr,mew_s,mew_w):
   hs = 0.36*(Ks/De)*(Re**0.55)*(Pr**(1/3))*
   ((mew_s/mew_w)**0.14)
   return hs

def LMTD(Tis,Tos,Tit,Tot):
   deltaT1 = Tis - Tot
   deltaT2 = Tos - Tit
   lmtd = (deltaT1 - deltaT2) /
   math.log(deltaT1 / deltaT2)
   return lmtd

def Correction_factor(Tis, Tos, Tit, Tot):
   R = (Tis - Tos) / (Tot - Tit)
   P = (Tot - Tit) / (Tis - Tit)
   numerator = math.sqrt(R**2 + 1) / (R - 1) *

   math.log((1 - P) / (1 - P*R))
   denominator = math.log((2 - P*(R + 1 -
   math.sqrt(R**2 + 1))) / (2 - P*(R + 1 +
   math.sqrt(R**2 + 1))))
   F = numerator / denominator
   return F

def Sensible_heat(ms,Cps,Tis,Tos):
   Q = ms*Cps*(Tis-Tos)
   return Q

def calculate_DPt(rho_t,vt,L,dt,ft,n):
   p = 4
   DPt = rho_t*vt**2/2 * ((L/dt)*ft + p) * n
   return DPt

def calculate_DPs(fs,rho_s,vs,L,B,Ds,De):
   DPs = fs * rho_s*vs**2/2 * (L/B) * (Ds/De)
   return DPs

def calculate_Cod(n, i, C_o):
   X = np.arange(1, n + 1)
   terms = C_o / (1 + i) ** X
   C_od = np.sum(terms)
   return C_od

def Calculate_Total_Cost(C, n1, Ds, mt, rho_t,

rho_s, n, mew_t, L, Kt, Pt, d0, B, ms, mew_s,
Ks, Rfs, Rft, Tis, Tos, Tit, Tot, Cp_h, Cp_c,
etta, ny, H, Ce, i, b, C1, mew_w):
   p = 4
   di = 0.8*d0
   De = Shell_hydaulic_diameter(Pt,d0)
   Re_s = Reynolds_number_shell(ms,De,
   Cross_section_area(Ds,b,C1),mew_s)
   Pr_s = Prandtl_number(mew_s,Cp_h,Ks)
   hs = Shell_Side_heat_transfer_coefficient
   (Ks,De,Re_s,Pr_s,mew_s,mew_w)
   Nt = Number_of_tubes(C, n1, Ds, d0)
   vt = Flow_velocity_tube(mt,di,rho_t,n,Nt)
   Re_t = Reynolds_number_tube(rho_t,vt,di,
       mew_t)
   Pr_t = Prandtl_number(mew_t,Cp_c,Kt)
   ht = Tube_side_heat_transfer_coefficient
   (Kt,di,Re_t,Pr_t,L,d0,mew_t,mew_w)
   U = 1/((1/hs)+Rfs+(d0/di)*(Rft+(1/ht)))
   lmtd = LMTD(Tis,Tos,Tit,Tot)
   F = Correction_factor(Tis,Tos,Tit,Tot)
   A = (mt*Cp_c*(Tot-Tit))/(U*F*lmtd)
   Span = (A/(pie*d0*Nt))
   ft = calculate_friction_factor(Re_t)
   DPt = calculate_DPt(rho_t,vt,Span,di,ft,n)
   b0 = 0.72
   fs = calculate_fs(b0, Re_s)
   vs = Flow_velocity_shell(ms,rho_s,
   Cross_section_area(Ds,b,C1))
   DPs = calculate_DPs(fs,rho_s,vs,L,B,Ds,De)
   a1 = 8000
   a2 = 259.2
   a3 = 0.93
   C_i = a1 + a2 * (A**(a3/3))
   P = 1 / etta * (((mt / rho_t) * DPt) +
   ((ms / rho_s) * DPs))
   C_o = P * Ce * H
   C_od = calculate_Cod(ny, i, C_o)
   C_tot = C_i + C_od
```

```
   return C_tot
```

5.3. First Instance Code generation for Non Similar research article

```
import math

# Rao-1 algorithm
def rao1_update(Xu_v_w, Xbest_v_w, Xworst_v_w,
     r1):
   return Xu_v_w + r1 * (Xbest_v_w -
       Xworst_v_w)

# Rao-2 algorithm
def rao2_update(Xu_v_w, Xbest_v_w, Xworst_v_w,
     r1, r2,
Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w):
   return Xu_v_w + r1 * (Xbest_v_w -
       Xworst_v_w) + r2
   * (abs(Xu_v_w_or_XU_v_w) - abs(
       XU_v_w_or_Xu_v_w))

# Rao-3 algorithm
def rao3_update(Xu_v_w, Xbest_v_w, Xworst_v_w,
     r1, r2,
Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w):
   return Xu_v_w + r1 * (Xbest_v_w - abs(
       Xworst_v_w))
   + r2 * (abs(Xu_v_w_or_XU_v_w) - (
       XU_v_w_or_Xu_v_w))

# Helical compression spring objective
    function
def helical_spring_volume(D, d, n):
   return math.pi**2 * D * d**2 * (n+2) / 4

# Hydrostatic thrust bearing objective
    function
def hydrostatic_thrust_bearing_power_loss(Q,
    Po, Ef):
   return Q * Po / 0.7 + Ef

# Multiple disc clutch brake objective
    function
def multiple_disc_clutch_brake_mass(ro, ri, t,
     z, rho):
   return math.pi * (ro**2 - ri**2) * t * (z
       +1) * rho

# Cylindrical roller bearing objective
    function
def
    cylindrical_roller_bearing_dynamic_capacity
    (bm, lv,
gamma, Dr, le, Z):
   return 207.9 * bm * lv *
   (1 + (1.04 * ((1-gamma)/(1+gamma))
       **(143/108))**
   (9/2))**(-2/9)
   *gamma**(2/9) * (1-gamma)**(29/27) * (1+
       gamma)
   **(1/4)
   * (le)**(7/9) * Dr**(29/27) * Z**(3/4)

# Spherical roller bearing objective function
def spherical_roller_bearing_dynamic_capacity
    (bm,
lv, gamma, Dr, le, Z, alpha):
   return 207.9 * bm * lv *
   (1 + (1.04 * ((1-gamma)/(1+gamma))
**(143/108))**(9/2))**
(-2/9) * gamma**(2/9) * (1-gamma)**(29/27)
 * (1+gamma)**(1/4) *
 (le*math.cos(alpha))**(7/9) * Z**(3/4) * Dr
     **(29/27)

 # Plate fin heat exchanger objective
     function
def
    plate_fin_heat_exchanger_entropy_generation

(epsilon, Tci, Thi, Rc, teh, Cph, dPh, Phi,
tec, Cpc, dPc, Pci):
   return (1-epsilon) * ((Tci-Thi)**2 /
   (Tci*Thi)) + (Rc*teh/Cph) * (dPh/Phi)
   + (Rc*tec/Cpc) * (dPc/Pci)

# Shell and tube heat exchanger objective
    function
def shell_and_tube_heat_exchanger_total_cost(
    Ci, Cod):
   return Ci + Cod

# Welded beam objective function
def welded_beam_cost(c1, x1, t, l, h, c2, b,
    L):
   return (1+c1) * (x1*t + l) * h**2 + c2*t*b
       *(l+L)

# Belt-pulley drive objective function
def belt_pulley_drive_weight(rho, b, d1,
t1, d2, t2, d1_1, t1_1, d1_2, t1_2):
   return math.pi * rho * b * (d1*t1 +
   d2*t2 + d1_1*t1_1 + d1_2*t1_2)

# Hollow shaft objective function
def hollow_shaft_weight(do, di, L, rho):
   return math.pi/4 * (do**2 - di**2) * L *
       rho

def run_optimization(Xu_v_w, Xbest_v_w,
Xworst_v_w, r1, r2, Xu_v_w_or_XU_v_w,
XU_v_w_or_Xu_v_w,D, d, n, Q, Po, Ef,
ro, ri, t, z, rho, bm, lv, gamma, Dr,
le, Z, alpha, epsilon, Tci, Thi, Rc,
teh, Cph, dPh, Phi, tec, Cpc, dPc,
Pci, Ci, Cod, c1, x1, l, h, c2, b, L, do, di):
```

```
"""
Runs the optimization process using
the Rao algorithms and objective functions.


Parameters:
- Xu_v_w, Xbest_v_w, Xworst_v_w,
r1, r2, Xu_v_w_or_XU_v_w,
XU_v_w_or_Xu_v_w: Parameters for Rao
    algorithms
- D, d, n: Parameters for helical
compression spring
- Q, Po, Ef: Parameters for hydrostatic
    thrust bearing
- ro, ri, t, z, rho: Parameters for
    multiple
disc clutch brake
- bm, lv, gamma, Dr, le, Z: Parameters for
cylindrical roller bearing
- alpha: Additional parameter for
    spherical
roller bearing
- epsilon, Tci, Thi, Rc, teh, Cph, dPh,
    Phi,
tec, Cpc, dPc, Pci: Parameters for plate
    fin
heat exchanger
- Ci, Cod: Parameters for shell and tube
    heat exchanger
- c1, x1, l, h, c2, b, L: Parameters for
    welded beam
- do, di: Additional parameters for hollow
     shaft
"""

# Run Rao algorithms
rao1_result = rao1_update(Xu_v_w,
    Xbest_v_w, Xworst_v_w, r1)
rao2_result = rao2_update(Xu_v_w,
    Xbest_v_w, Xworst_v_w,
r1, r2, Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w)

rao3_result = rao3_update(Xu_v_w,
    Xbest_v_w, Xworst_v_w,
r1, r2, Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w)


# Calculate objective functions
helical_spring_result =
    helical_spring_volume
(D, d, n)
hydrostatic_thrust_bearing_result =
    hydrostatic_
thrust_bearing_power_loss(Q, Po, Ef)
multiple_disc_clutch_brake_result =
multiple_disc_clutch_brake_mass(ro, ri, t,
     z, rho)
cylindrical_roller_bearing_result =
    cylindrical
_roller_bearing_dynamic_capacity
(bm, lv, gamma, Dr, le, Z)
spherical_roller_bearing_result =
    spherical
_roller_bearing_dynamic_capacity
(bm, lv, gamma, Dr, le, Z, alpha)
plate
_fin_heat_exchanger_result = plate
_fin_heat_exchanger_entropy_generation
(epsilon, Tci, Thi, Rc, teh, Cph, dPh, Phi,

tec, Cpc, dPc, Pci)
shell_and_tube_heat_exchanger_result =
    shell
_and_tube_heat_exchanger_total_cost
(Ci, Cod)
welded_beam_result = welded_beam_cost(c1,
x1, t, l, h, c2, b, L)
belt_pulley_drive_result =
    belt_pulley_drive_weight
(rho, b, d1, t1, d2, t2, d1_1, t1_1, d1_2,
     t1_2)
hollow_shaft_result = hollow_shaft_weight(
    do, di, L, rho)

# Return results
return {
   'rao1_result': rao1_result,
   'rao2_result': rao2_result,
   'rao3_result': rao3_result,
   'helical_spring_result':
   helical
   _spring_result,
   'hydrostatic
   _thrust_bearing_result': hydrostatic
   _thrust_bearing_result,
   'multiple_disc_clutch_brake_result':
       multiple
   _disc_clutch_brake_result,
   'cylindrical_roller_bearing_result':
       cylindrical
   _roller_bearing_result,
   'spherical_roller_bearing_result':
   spherical
   _roller_bearing_result,
   'plate_fin_heat_exchanger_result':
   plate_fin_heat_exchanger_result,
   'shell_and_tube_heat_exchanger_result':
       shell
   _and_tube_heat_exchanger_result,
   'welded_beam_result': welded_beam_result
       ,
   'belt_pulley_drive_result':
       belt_pulley_drive_result,
   'hollow_shaft_result':
       hollow_shaft_result
}
```

5.4. Final revised Mathematical model for the non-similar research article following the code correction framework

```
import math

# Rao-1 algorithm
def rao1_update(Xu_v_w, Xbest_v_w, Xworst_v_w,
     r1):
   return Xu_v_w + r1 * (Xbest_v_w -
       Xworst_v_w)

# Rao-2 algorithm
def rao2_update
(Xu_v_w, Xbest_v_w, Xworst_v_w, r1,
r2, Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w):
     return
   Xu_v_w + r1 * (Xbest_v_w - Xworst_v_w)
   + r2 * (abs(Xu_v_w_or_XU_v_w) - abs(
       XU_v_w_or_Xu_v_w))

# Rao-3 algorithm
def rao3_update(Xu_v_w, Xbest_v_w, Xworst_v_w,
     r1, r2,
Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w):return
Xu_v_w + r1 * (Xbest_v_w - abs(Xworst_v_w)) +
r2 * (abs(Xu_v_w_or_XU_v_w) - (
     XU_v_w_or_Xu_v_w))

# Helical compression spring objective
     function
def helical_spring_volume(D, d, n):
   return math.pi**2 * D * d**2 * (n+2) / 4

# Hydrostatic thrust bearing objective
     function
def hydrostatic_thrust_bearing_power_loss(Q,
     Po, Ef):
   return Q * Po / 0.7 + Ef

# Multiple disc clutch brake objective
     function
def multiple_disc_clutch_brake_mass(ro, ri, t,
      z, rho):
   return math.pi * (ro**2 - ri**2) * t * (z
       +1) * rho

# Cylindrical roller bearing objective
     function
def
     cylindrical_roller_bearing_dynamic_capacity

(bm, lv, gamma, Dr, le, Z):
   return 207.9 * bm * lv *
   (1 + (1.04 * ((1-gamma)/(1+gamma))
   **(143/108))**(9/2))**(-2/9)
   * gamma**(2/9) * (1-gamma)**(29/27) * (1+
       gamma)**(1/4)
   * (le)**(7/9) * Dr**(29/27) * Z**(3/4)

# Spherical roller bearing objective function
def spherical_roller_bearing_dynamic_capacity
(bm, lv, gamma, Dr, le, Z, alpha):return
207.9 * bm * lv * (1 +
(1.04 * ((1-gamma)/(1+gamma))**(143/108))
**(9/2))**(-2/9) * gamma**(2/9) * (1-gamma)
     **(29/27)
* (1+gamma)**(1/4) * (le*math.cos(alpha))
     **(7/9) * Z**(3/4) *
Dr**(29/27)

# Plate fin heat exchanger objective function
def
     plate_fin_heat_exchanger_entropy_generation

(epsilon, Tci, Thi, Rc, teh, Cph,
dPh, Phi, tec, Cpc, dPc, Pci):return
(1-epsilon) * ((Tci-Thi)**2 / (Tci*Thi)) +
(Rc*teh/Cph) * (dPh/Phi) + (Rc*tec/Cpc) * (
     dPc/Pci)

# Shell and tube heat exchanger objective
     function
def shell_and_tube_heat_exchanger_total_cost(
     Ci, Cod):
   return Ci + Cod

# Welded beam objective function
def welded_beam_cost(c1, x1, t, l, h, c2, b,
     L):
   return (1+c1) * (x1*t + l) * h**2 + c2*t*b
       *(l+L)

# Belt-pulley drive objective function
def belt_pulley_drive_weight(rho, b, d1, t1,
d2, t2, d1_1, t1_1, d1_2, t1_2):return
math.pi * rho * b * (d1*t1 + d2*t2 + d1_1*
     t1_1 + d1_2*t1_2)

# Hollow shaft objective function
def hollow_shaft_weight(do, di, L, rho):
   return math.pi/4 * (do**2 - di**2) * L *
       rho

def run_optimization(Xu_v_w, Xbest_v_w,
Xworst_v_w, r1, r2, Xu_v_w_or_XU_v_w,
     XU_v_w_or_Xu_v_w,
D, d, n, Q, Po, Ef, ro, ri, t, z, rho, bm,
lv, gamma, Dr, le, Z, alpha,, Tci, Thi, Rc,
     teh,
Cph, dPh, Phi, tec, Cpc, dPc, Pci, Ci, Cod,
c1, x1, l, h, c2, b, L, do, di):"""""""
   Runs the optimization process using the
       Rao algorithms
   and objective functions.

   Parameters:
   - Xu_v_w, Xbest_v_w, Xworst_v_w, r1, r2,
```

```
Xu_v_w_or_XU_v_w, XU_v_w_or_Xu_v_w:
    Parameters
for Rao algorithms
- D, d, n: Parameters
for helical compression spring
- Q, Po, Ef: Parameters
for hydrostatic thrust bearing
- ro, ri, t, z, rho: Parameters
for multiple disc clutch brake
- bm, lv, gamma, Dr, le, Z: Parameters
for cylindrical roller bearing
- alpha: Additional
parameter for spherical roller bearing
- epsilon, Tci, Thi, Rc,
teh, Cph, dPh, Phi, tec, Cpc, dPc, Pci:
    Parameters
for plate fin heat exchanger
- Ci, Cod: Parameters for shell and tube
    heat exchanger
- c1, x1, l, h, c2, b, L: Parameters for
    welded beam
- do, di: Additional parameters for hollow
     shaft
"""""""

# Run Rao algorithms
rao1_result = rao1_update(Xu_v_w,
    Xbest_v_w,
Xworst_v_w, r1)
rao2_result = rao2_update(Xu_v_w,
    Xbest_v_w,
Xworst_v_w, r1, r2, Xu_v_w_or_XU_v_w,
    XU_v_w_or_Xu_v_w)
rao3_result = rao3_update(Xu_v_w,
    Xbest_v_w,
Xworst_v_w, r1, r2, Xu_v_w_or_XU_v_w,
    XU_v_w_or_Xu_v_w)

# Calculate objective functions
helical_spring_result =
    helical_spring_volume
(D, d, n)
hydrostatic_thrust_bearing_
result =
    hydrostatic_thrust_bearing_power_loss
(Q, Po, Ef)
multiple_disc_clutch_brake_
result = multiple_disc_clutch_brake_mass
(ro, ri, t, z, rho)
cylindrical_roller_bearing_
result =
    cylindrical_roller_bearing_dynamic_capacity

(bm, lv, gamma, Dr, le, Z)
spherical_roller_bearing_
result =
    spherical_roller_bearing_dynamic_capacity

(bm, lv, gamma, Dr, le, Z, alpha)
plate_fin_heat_exchanger_
result = plate_fin_heat_exchanger_entropy_
generation
(epsilon, Tci, Thi, Rc, teh, Cph, dPh, Phi,
     tec, Cpc,
dPc, Pci)
shell_and_tube_heat_exchanger_
result =
    shell_and_tube_heat_exchanger_total_cost

(Ci, Cod)
welded_beam_result = welded_beam_cost
(c1, x1, t, l, h, c2, b, L)
belt_pulley_drive_result =
    belt_pulley_drive_weight
(rho, b, d1, t1, d2, t2, d1_1, t1_1, d1_2,
     t1_2)
hollow_shaft_result = hollow_shaft_weight
(do, di, L, rho)

# Return results
return {
   'rao1_result': rao1_result,
   'rao2_result': rao2_result,
   'rao3_result': rao3_result,
   'helical_spring_result':
       helical_spring_result,
   'hydrostatic_thrust_bearing_result':
       hydrostatic_
   thrust_bearing_result,
   'multiple_disc_clutch_brake_result':
       multiple_disc_
   clutch_brake_result,
   'cylindrical_roller_bearing_result':
       cylindrical_roller
   _bearing_result,
   'spherical_roller_bearing_result':
       spherical_
   roller_bearing_result,
   'plate_fin_heat_exchanger_result':
       plate_fin_
   heat_exchanger_result,
   'shell_and_tube_heat_exchanger_result':
       shell_and_tube_
   heat_exchanger_result,
   'welded_beam_result': welded_beam_result
       ,
   'belt_pulley_drive_result':
       belt_pulley_drive_result,
   'hollow_shaft_result':
       hollow_shaft_result
}
```

5.5. Initial code generation for optimization model outlined in the research article

```
def PSO
```

```
(num_particles, max_iterations, w, c1, c2,
    Ds_min,
Ds_max, do_min, do_max, B_min, B_max):

   particles = []
   velocities = []
   best_positions = []
   best_costs = []

   global_best_position = None
   global_best_cost = float('inf')

   for i in range(num_particles):
      Ds = random.uniform(Ds_min, Ds_max)
      do = random.uniform(do_min, do_max)
      B = random.uniform(B_min, B_max)

      particles.append([Ds, do, B])
      velocities.append([0, 0, 0])

      cost = Calculate_Total_Cost(0.158,
          2.263, Ds,
      18.80, 995, 850, 4, 0.00358, 4.88, 0.13,
           1.25*
      0.0254, do, B, 5.52, 0.0004, 0.13,
          0.00061,
      0.00061, 199, 93.3, 37.80, 76.7, 2.47,
          2.05,
      0.9, 10, 7000, 0.12, 10, 0.127, 0.4,
          0.0004)

      best_positions.append([Ds, do, B])
      best_costs.append(cost)

      if cost < global_best_cost:
         global_best_position = [Ds, do, B]
         global_best_cost = cost

   for iteration in range(max_iterations):
      for i in range(num_particles):
         r1 = random.random()
         r2 = random.random()

         velocities[i][0] = w*velocities[i][0]
              + c1*r1*
         (best_positions[i][0] - particles[i
             ][0]) + c2*r2*
         (global_best_position[0] - particles[
             i][0])
         velocities[i][1] = w*velocities[i][1]
              + c1*r1*
         (best_positions[i][1] - particles[i
             ][1]) + c2*r2*
         (global_best_position[1] - particles[
             i][1])
         velocities[i][2] = w*velocities[i][2]
              + c1*r1*
         (best_positions[i][2] - particles[i
             ][2]) + c2*r2*
         (global_best_position[2] - particles[
             i][2])

         particles[i][0] += velocities[i][0]
         particles[i][1] += velocities[i][1]
         particles[i][2] += velocities[i][2]

         particles[i][0] = max(min(particles[i
             ][0], Ds_max)
         , Ds_min)
         particles[i][1] = max(min(particles[i
             ][1], do_max)
         , do_min)
         particles[i][2] = max(min(particles[i
             ][2], B_max)
         , B_min)

         cost = Calculate_Total_Cost(params)

         if cost < best_costs[i]:
            best_positions[i] = particles[i].
                copy()
            best_costs[i] = cost

            if cost < global_best_cost:
               global_best_position =
                   particles[i].copy()
               global_best_cost = cost

   return global_best_position

# PSO parameters
num_particles = 50
max_iterations = 100
w = 0.7
c1 = 1.4
c2 = 1.4

# Variable limits
Ds_min = 0.1
Ds_max = 1.5
do_min = 0.015
do_max = 0.051
B_min = 0.05
B_max = 0.5

optimal_values = PSO(num_particles,
max_iterations, w, c1, c2, Ds_min, Ds_max,
    do_min,
do_max, B_min, B_max)
```

5.6. Final revised optimisation model following the code correction framework

```
import math
import numpy as np

pie = 3.141
```

```
def Prandtl_number(mew,Cp,Kt):
   Pr = (mew*Cp)/Kt
   return Pr

def calculate_fs(b0, Re):
   fs = 2 * b0 * Re**(-0.15)
   return fs

def Number_of_tubes(C,n1,Ds,d0):
   Nt = C*((Ds/d0)**n1)
   return Nt

def Flow_velocity_tube(mt,dt,rho_t,n,Nt):
   vt = (mt*(n/Nt))/((pie/4)*(dt**2)*rho_t)
   return vt

def Reynolds_number_tube(rho_t,vt,di,mew_t):
   Re = (rho_t*vt*di)/mew_t
   return Re

def calculate_friction_factor(Re):
   ft = (1.82 * math.log10(Re) - 1.64)**(-2)
   return ft

def Tube_side_heat_transfer_coefficient
(Kt,di,Re,Pr,L,d0,mew_t,mew_w):
   if Re < 2300:
      ht = (Kt/di)*(3.657+(0.0677*
      (Re*Pr*(di/L))**1.33)/(1+0.1*Pr*((Re*di/
          L)**0.3)))
   elif 2300 < Re < 10000:
      ft = calculate_friction_factor(Re)
      ht = (Kt/di)*((ft/8)*
      (Re-1000)*Pr)/(1+12.7*
      math.sqrt(ft/8)*(Pr**(2/3)-1))*(1+(di/L)
          **0.67)
   else:
      ht = 0.027*(Kt/d0)*(Re**0.8)*(Pr**(1/3))
      *((mew_t/mew_w)**0.14)
   return ht

def Shell_hydaulic_diameter(Pt,d0):
   De = 4*(Pt**2 - (math.pi*d0**2)/4)/(math.
       pi*d0)
   return De

def Cross_section_area(Ds,b,C1):
   As = Ds*b*C1
   return As

def Flow_velocity_shell(ms,rho_s,As):
   vs = ms/(rho_s*As)
   return vs

def Reynolds_number_shell(ms,de,As,mew_s):
   re = (ms*de)/(As*mew_s)
   return re

def Shell_Side_heat_transfer_coefficient
(Ks,De,Re,Pr,mew_s,mew_w):
   hs = 0.36*(Ks/De)*(Re**0.55)
   *(Pr**(1/3))*((mew_s/mew_w)**0.14)
   return hs

def LMTD(Tis,Tos,Tit,Tot):
   deltaT1 = Tis - Tot
   deltaT2 = Tos - Tit
   lmtd = (deltaT1 - deltaT2) / math.log(
       deltaT1 / deltaT2)
   return lmtd

def Correction_factor(Tis, Tos, Tit, Tot):
   R = (Tis - Tos) / (Tot - Tit)
   P = (Tot - Tit) / (Tis - Tit)
   numerator = math.sqrt(R**2 + 1) /
   (R - 1) * math.log((1 - P) / (1 - P*R))
   denominator = math.log((2 - P*
   (R + 1 - math.sqrt(R**2 + 1))) /
   (2 - P*(R + 1 + math.sqrt(R**2 + 1))))
   F = numerator / denominator
   return F

def Sensible_heat(ms,Cps,Tis,Tos):
   Q = ms*Cps*(Tis-Tos)
   return Q

def calculate_DPt(rho_t,vt,L,dt,ft,n):
   p = 4
   DPt = rho_t*vt**2/2 * ((L/dt)*ft + p) * n
   return DPt

def calculate_DPs(fs,rho_s,vs,L,B,Ds,De):
   DPs = fs * rho_s*vs**2/2 * (L/B) * (Ds/De)
   return DPs

def calculate_Cod(n, i, C_o):
   X = np.arange(1, n + 1)
   terms = C_o / (1 + i) ** X
   C_od = np.sum(terms)
   return C_od

def Calculate_Total_Cost(C, n1, Ds, mt, rho_t,

rho_s, n, mew_t, L, Kt, Pt, d0, B, ms, mew_s,
Ks, Rfs, Rft, Tis, Tos, Tit, Tot, Cp_h, Cp_c,
etta, ny, H, Ce, i, b, C1, mew_w):
   p = 4
   di = 0.8*d0
   De = Shell_hydaulic_diameter(Pt,d0)
   Re_s = Reynolds_number_shell(ms,De,
   Cross_section_area(Ds,b,C1),mew_s)
   Pr_s = Prandtl_number(mew_s,Cp_h,Ks)
   hs = Shell_Side_heat_transfer_coefficient
   (Ks,De,Re_s,Pr_s,mew_s,mew_w)
   Nt = Number_of_tubes(C, n1, Ds, d0)
   vt = Flow_velocity_tube(mt,di,rho_t,n,Nt)
   Re_t = Reynolds_number_tube(rho_t,vt,di,
       mew_t)
   Pr_t = Prandtl_number(mew_t,Cp_c,Kt)
```

```
   ht = Tube_side_heat_transfer_coefficient
   (Kt,di,Re_t,Pr_t,L,d0,mew_t,mew_w)
   U = 1/((1/hs)+Rfs+(d0/di)*(Rft+(1/ht)))
   lmtd = LMTD(Tis,Tos,Tit,Tot)
   F = Correction_factor(Tis,Tos,Tit,Tot)
   A = (mt*Cp_c*(Tot-Tit))/(U*F*lmtd)
   Span = (A/(pie*d0*Nt))
   ft = calculate_friction_factor(Re_t)
   DPt = calculate_DPt(rho_t,vt,Span,di,ft,n)
   b0 = 0.72
   fs = calculate_fs(b0, Re_s)
   vs = Flow_velocity_shell(ms,rho_s,
   Cross_section_area(Ds,b,C1))
   DPs = calculate_DPs(fs,rho_s,vs,L,B,Ds,De)
   a1 = 8000
   a2 = 259.2
   a3 = 0.93
   C_i = a1 + a2 * (A**(a3/3))
   P = 1 / etta * (((mt / rho_t) * DPt) +
   ((ms / rho_s) * DPs))
   C_o = P * Ce * H
   C_od = calculate_Cod(ny, i, C_o)
   C_tot = C_i + C_od
   return C_tot

import random

def PSO(num_particles, max_iterations, w,
c1, c2, Ds_min, Ds_max, do_min, do_max, B_min,
      B_max):

   particles = []
   velocities = []
   best_positions = []
   best_costs = []

   global_best_position = None
   global_best_cost = float('inf')

   for i in range(num_particles):
      Ds = random.uniform(Ds_min, Ds_max)
      do = random.uniform(do_min, do_max)
      B = random.uniform(B_min, B_max)

      particles.append([Ds, do, B])
      velocities.append([0, 0, 0])

      cost = Calculate_Total_Cost(0.158,
      2.263, Ds, 18.80, 995, 850, 4, 0.00358
      , 4.88, 0.13, 1.25*0.0254, do, B, 5.52
      , 0.0004, 0.13, 0.00061, 0.00061, 199,
      93.3, 37.80, 76.7, 2.47, 2.05, 0.9, 10
      , 7000, 0.12, 10, 0.127, 0.4, 0.0004)

      best_positions.append([Ds, do, B])
      best_costs.append(cost)

      if cost < global_best_cost:
         global_best_position = [Ds, do, B]
         global_best_cost = cost

for iteration in range(max_iterations):
   for i in range(num_particles):
      r1 = random.random()
      r2 = random.random()

      velocities[i][0] = w*velocities[i][0]
           + c1*r1*
      (best_positions[i][0] - particles[i
          ][0]) + c2*r2*
      (global_best_position[0] - particles[
          i][0])
      velocities[i][1] = w*velocities[i][1]
           + c1*r1*
      (best_positions[i][1] - particles[i
          ][1]) + c2*r2*
      (global_best_position[1] - particles[
          i][1])
      velocities[i][2] = w*velocities[i][2]
           + c1*r1*
      (best_positions[i][2] - particles[i
          ][2]) + c2*r2*
      (global_best_position[2] - particles[
          i][2])

      particles[i][0] += velocities[i][0]
      particles[i][1] += velocities[i][1]
      particles[i][2] += velocities[i][2]

      particles[i][0] = max(min(particles[i
          ][0], Ds_max),
      Ds_min)
      particles[i][1] = max(min(particles[i
          ][1], do_max),
      do_min)
      particles[i][2] = max(min(particles[i
          ][2], B_max),
      B_min)

      cost = Calculate_Total_Cost(0.158,
          2.263,
      particles[i][0], 18.80, 995, 850, 4,
          0.00358
      , 4.88, 0.13, 1.25*0.0254, particles[
          i][1],
      particles[i][2], 5.52, 0.0004, 0.13,
          0.00061
      , 0.00061, 199, 93.3, 37.80, 76.7,
          2.47, 2.05
      , 0.9, 10, 7000, 0.12, 10, 0.127,
          0.4, 0.0004)

      if cost < best_costs[i]:
         best_positions[i] = particles[i].
             copy()
         best_costs[i] = cost

         if cost < global_best_cost:
            global_best_position =
                particles[i].
```

```
                copy()
                global_best_cost = cost

    return global_best_position

# PSO parameters
num_particles = 50
max_iterations = 100
w = 0.7
c1 = 1.4
c2 = 1.4

# Variable limits
Ds_min = 0.1
Ds_max = 1.5
do_min = 0.015
do_max = 0.051
B_min = 0.05
B_max = 0.5

optimal_values, optimal_cost = PSO(
    num_particles,
max_iterations, w, c1, c2, Ds_min, Ds_max,
    do_min,
do_max, B_min, B_max)

print(f"Optimal values: Shell diameter =
{optimal_values[0]}, Tube outer diameter =
{optimal_values[1]}, Baffle spacing = {
    optimal_values[2]}")
```

6. Heat Exchanger Design with Surrogate Models

Researchers have explored various surrogate modeling approaches, including machine learning [6, 7, 8, 9, 10] and deep learning [11, 12, 13, 14, 15, 16], to accelerate the design and optimization of heat exchangers.

Fouling, the deposition of chemical compounds, reduces the heat transfer efficiency of heat exchangers and can lead to operational stoppages. Measuring the extent of fouling is challenging due to the lack of direct measurements. Recently, deep learning-based models have been developed to provide real-time visibility and forecast the health of heat exchangers, aiding in better planning and optimization of operations [11, 12, 13]. Physics-informed deep learning approaches have further enhanced real-time visibility into fouling severity, facilitating real-time operational optimizations in plants [14, 15]. These approaches have also been applied to accelerate the design process [16].

Saeed et al. [6] applied machine learning algorithms to improve the performance of a C-shaped printed circuit heat exchanger (PCHE) within a supercritical CO2 Brayton cycle. By analyzing 81 channel configurations using computational fluid dynamics (CFD) and subsequently training machine learning models, they demonstrated significant improvements in thermo-hydraulic performance using a multi-objective genetic algorithm. Concurrently, Keramati et al. [7] explored the potential of deep reinforcement learning (Deep RL) for heat exchanger shape optimization, integrating a deep neural network (DNN) with a CFD solver. Their approach, based on Proximal Policy Optimization (PPO), resulted in notable enhancements in heat transfer efficiency and pressure drop reduction. Zou et al. [8] reviewed the applications of machine learning methods for heat exchanger modeling, highlighting the effectiveness of these models for design optimization. Additionally, Long et al. [9] and EFATINASAB et al. [10] investigated the use of machine learning and deep learning models for microchannel and micro-finned tube heat exchangers, respectively, demonstrating the competitive performance and scalability of these techniques in design optimization.