Aerodynamic Design and Optimization via a Specialized Agentic Generative AI Framework

Conan Lee[1,2], Joaquim RRA Martins[2], Gokcin Cinar[*2]

[1]Department of Mechanical and Aerospace Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong.
[2]Aerospace Engineering Department, University of Michigan, 1320 Beal Ave., Ann Arbor, 48109-2140, MI, USA.

Contributing authors: cleeay@connect.ust.hk; jrram@umich.edu; cinar@umich.edu;

Abstract

The practical implementation of dynamic digital twins in aerospace demands rapid multidisciplinary design analysis and optimization (MDAO), yet this process is currently limited by slow, manual workflows. This paper introduces an agentic Generative Artificial Intelligence framework to automate the synthesis of complex MDAO workflows from high-level natural language commands. The framework employs a multi-agent architecture that autonomously generates, executes, and analyzes aerodynamic optimization scripts using the OpenAeroStruct tool. This process establishes a robust digital thread from requirement to result, providing a foundational component for the near-real-time model updates demanded by the digital twin paradigm. Experiments demonstrate that the framework successfully converges on optimized aircraft wing designs for given sets of constraints, showcasing accuracy and efficiency superior to single-shot language model prompting. This work presents a foundational enabling technology for the digital twin paradigm, providing a method to significantly accelerate the design cycle and make the exploration of novel aircraft concepts more efficient and accessible.

Keywords: Agentic Framework, Generative AI, Large Language Models, Multidisciplinary Design Optimization, Wing Design

1 Introduction

The digital twin paradigm is poised to reshape aerospace engineering by moving beyond traditional simulation to enable dynamic, bidirectional feedback between a physical asset and its virtual representation. A key principle is that the virtual model must be "fit-for-purpose", tailored to the specific decisions it supports rather than being an exquisite replica of the physical system (Ferrari and Willcox 2024; Willcox and Segundo 2024). Realizing this vision requires the ability to rapidly conduct multidisciplinary design analysis and optimization (MDAO) to update the virtual model. However, a significant barrier remains in moving digital twins from bespoke, artisanal products to robust, industrial-scale applications (Niederer et al. 2021).

This scaling challenge is rooted in the complexity of current engineering design processes. While MDAO provides a systematic framework for navigating complex multidisciplinary design

[*]Corresponding author: cinar@umich.edu

trade-offs (Martins and Lambe 2013), its adoption is often hindered by a steep learning curve and high domain expertise requirements. Consequently, critical updates to the digital twin in response to operational changes or new requirements remain "largely human-driven and manual" (Ferrari and Willcox 2024). This bottleneck fundamentally limits the dynamic, real-time coupling that defines a true digital twin, forming a significant barrier to the technology's widespread adoption.

Recent advancements in Generative Artificial Intelligence (GenAI), particularly Large Language Models (LLMs), offer a promising path to automate these complex workflows through their capabilities in code generation (Jiang et al. 2024), natural language understanding (Raj et al. 2025), and complex problem-solving (Ramírez-Rueda et al. 2024). These models are increasingly being applied across engineering disciplines for tasks such as hardware design and testing, giving answers towards questions in material science, 3D printing quality control, alloy discovery, and heat exchanger design (Blocklove et al. 2024; Buehler 2023; Farimani et al. 2024; Ghafarollahi and Buehler 2024; Mishra et al. 2024). In aerospace, LLMs have been explored for applications including non-safety-critical decision support in air traffic management, design space exploration for unmanned aerial vehicles, automating Computational Fluid Dynamics (CFD) workflows, and agentic frameworks for controller gain tuning (Abdulhak et al. 2024; Samplawski et al. 2025; Pandey et al. 2025; Guo et al. 2024). Furthermore, some LLMs have been enhanced with tool-using capabilities (Qu et al. 2025), allowing interaction with external resources for numerical computation and Web-based information retrieval, while others have employed augmented extraction (RAG) to ground responses in large databases of technical and academic documents using advanced embedding and retrieval methods (Cho et al. 2024).

Despite this progress, a critical gap persists. Most current frameworks use agents to execute well-defined, single-discipline tasks—such as generating input files or tuning a handful of parameters, or following procedural, pre-scripted plans. (Wu et al. 2023; Hong et al. 2024; Zhang et al. 2025). They are not yet equipped for the creative, multidisciplinary analysis required to *formulate* an MDAO problem from first principles. Bridging this gap requires new methods that align the general strengths of LLMs with the domain-specific demands of advanced multidisciplinary aerospace design and optimization.

To address the MDAO automation gap, this paper introduces a framework that applies Generative AI to automate the use of physics-based simulation tools. This approach is presented as a practical application of Scientific Machine Learning (SciML), where the goal is not to replace the solver, but to intelligently automate the entire scientific workflow surrounding it. The framework uses OpenAeroStruct (OAS) (Jasa et al. 2018), a Python-based tool built upon OpenMDAO (Gray et al. 2019) for aerostructural analysis and optimization, as a case study. The framework translates high-level design goals expressed in natural language into executable code, interprets simulation results, and facilitates iterative design refinement by generating detailed reports with RAG-enhanced analysis for human review or automatic iteration. To ensure the reliability and accuracy necessary for engineering tasks, the proposed approach moves beyond single-shot generation, which can be prone to errors. Instead, the framework employs a multi-agent architecture where specialized agents collaborate on tasks such as task understanding, code generation, and analysis, breaking down the complexity into smaller and more manageable components while maintaining transparency in each component.

While significant research in engineering applications of LLMs has focused on models such as OpenAI's GPT family (OpenAI 2024) and Meta's Llama (Touvron et al. 2023), there is less research specifically exploring Google's Gemini family (Google 2025) in aerospace engineering design. This study explores Gemini due to its cost-effectiveness and the availability of free API access, making it well-suited for initial research and open access. Gemini 2.0 Flash was selected over more complex alternatives, such as Gemini 2.5 Pro, to minimize computational overhead, as higher-capacity models tend to consume substantially more tokens even for relatively simple tasks.

The paper is structured as follows: Section 2 describes the agentic framework developed in this work. Key results are presented in Section 3, followed by a detailed discussion of the findings and their broader implications in Section 4. Section 5 concludes the paper. Appendix A provides the prompts used to guide the LLM agents in the work.

2 Methodology

This section details the agentic GenAI framework developed to automate the synthesis of complex MDAO workflows, demonstrated through the application of aerodynamic wing design. The architecture is engineered to replace slow, manual design processes with a dynamic, agent-driven approach, establishing a robust digital thread from a high-level natural language command to a validated, optimized result. Using the OAS toolset as a case study, the framework translates design goals into executable Python code, interprets simulation results, and facilitates iterative refinement through detailed multi-modal reporting.

2.1 Multi-Agent Framework Architecture

The architecture of the multi-agent framework is illustrated in Figure 1. The agents employed within this architecture are categorized into two distinct classes, which are color-coded in the figure's icons for clarity: general-purpose agents (Red) and specialized coding agents (Blue). This distinction is defined by their prompting style, which is crucial for their function. As detailed in Appendix (Section A), general-purpose agents, responsible for tasks like reformulation and analysis, are instructed to output text in specific formats (e.g., Python data structures and LaTeX), while coding agents are provided with commented sample code to ground their generation process, and instructed to output structured, commented code to prevent hallucinations.

As illustrated in Figure 1, the process is initiated by a user's design query, which specifies top-level requirements such as geometric wing constraints (e.g., span and area), design variables (e.g., sweep angle, dihedral, and taper ratio), flow conditions, optimization objectives, and desired plotting outputs. This design query is processed by a **Task Reformulation Agent** (a general-purpose agent) which interprets the natural language input, structures it into a deterministic Python data structure, and validates the problem formulation to identify missing constraints or potential errors at the outset.

Subsequently, a team of three specialized coding agents collaborates to generate the executable script for the analysis: The **Mesh Agent** generates the code for the wing's numerical mesh based on the specified geometry, while the **Geometry Agent** defines the wing's geometry and sets up the design variables to be explored during optimization. Meanwhile, the **Optimization Agent** writes the code that defines the objective function, constraints, and solver settings for the MDAO problem.

The integrated script is then executed, producing numerical results and reporting in HTML format. These reports are subsequently converted to PDF, a format more suitable for multi-modal LLM analysis, which allows the framework to extract and select graphical information for inclusion in the final summary. An **Initial Analysis Agent** (a general-purpose agent) parses these outputs to extract key findings on optimization performance, convergence, and design variable states. It also provides additional observations regarding factors such as manufacturability. These insights are drawn from the model's general knowledge, which is enhanced with domain-specific information from the RAG module. This module uses an **Information Retrieval Agent** to query a vectorized knowledge base for the physical meaning of the variables, as detailed in Section 2.3.2. Finally, a **Report Writing Agent** (a general-purpose agent) synthesizes these findings into a structured LaTeX report that presents a clear summary of the optimized design.

2.2 Foundational Model and Rationale

The selection of the foundational LLM was based on a trade-off analysis of performance, computational cost, and accessibility. This study utilizes Gemini 2.0 Flash. This model was selected because it provides robust capabilities in code generation

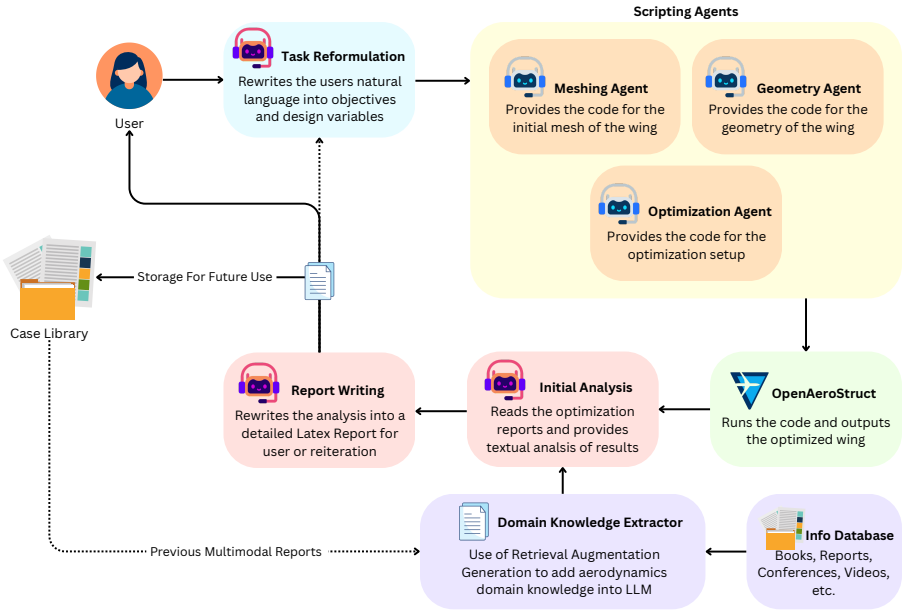

Fig. 1: Multi-Agent LLM Framework for OpenAeroStruct (Red icons denote general purpose agents, while blue icons denote specialized coding agents).

and technical reasoning while remaining computationally efficient. Minimizing computational overhead is critical for the framework's intended application within the digital twin paradigm, which demands rapid, iterative design cycles. The model's efficiency and API accessibility are expected to facilitate broader adoption by the academic and industrial aerospace communities upon releasing the framework.

2.3 Error Mitigation and Domain Grounding

To ensure the reliability necessary for engineering applications, the framework incorporates two key strategies for enhancing the accuracy and physical understanding of the generated outputs.

2.3.1 Structured Data Transfer via Schema

To maintain robustness and prevent data-passing errors between agents, communication is governed by rigid output schemas. Each agent is required to format its output as a structured object, ensuring consistent and correct information transfer throughout the workflow, as detailed in Appendix A). For simple mathematical operations, the agents currently perform basic arithmetic—division, multiplication, addition, and subtraction—such as independently calculating a rectangular wing's root chord from its area and span. Testing has demonstrated their stability and reliability for these straightforward calculations and has shown that no additional tools are required for implementation.

2.3.2 Retrieval Augmentation Generation (RAG)

To ground the agents' analyses in correct physical principles, the framework integrates a Retrieval-Augmented Generation (RAG) module (see the "Domain Knowledge Extractor" in Figure 1). When the Initial Analysis Agent evaluates an optimization result, the Information Retrieval Agent accesses information regarding relevant physical variables by querying a vector database. This process involves a similarity search within a vectorized knowledge base containing persistent embeddings. The knowledge base is populated with documentation on the governing physics of the simulation tool—such as the relationship between lift and angle of attack in the Vortex Lattice Method, which the tool is built upon, as well as

notes on aerodynamic optimization. By augmenting the agent's context with this domain-specific information, the RAG module improves the accuracy of the final analysis, enabling more physically consistent design recommendations and more effective troubleshooting of failed optimizations.

3 Results

To validate the proposed agentic framework, its performance was evaluated on two representative aerostructural optimization test cases. These cases were designed to assess the framework's end-to-end capabilities, from interpreting natural language commands to generating and analyzing executable MDAO workflows. The results demonstrate the framework's robustness and highlight the advantages of a multi-agent architecture over conventional single-shot prompting methods towards existing publicly available LLM tools.

3.1 Framework Validation on Aerostructural Test Cases

The framework was evaluated using two distinct wing design optimization problems, which are canonical examples in aerodynamic instruction. The first case involved a standard drag minimization problem for a given lift coefficient ($C_L = 0.5$), with taper and sweep as design variables. In a single, automated execution, the framework successfully interpreted the requirements, generated a valid and executable OpenAeroStruct script, and performed the optimization without manual intervention. The modular approach, where specialized agents generated distinct code segments, produced accurate and well-commented scripts that simplify potential debugging efforts. Following the optimization run, the Initial Analysis Agent correctly extracted key findings, identified that design variables remained within their bounds, and confirmed that the resulting lift distribution approached the expected elliptical shape for minimum induced drag . A comprehensive report, automatically generated by the Report Writing Agent, is available in the accompanying repository hosted on the University of Michigan IDEAS Lab Github page (Lee and Cinar 2025).

The second test case was designed to evaluate the framework's diagnostic capabilities in the event of a failed optimization. The problem required minimizing drag at a very high lift coefficient ($C_L = 2.0$), a condition that is physically challenging to meet with the initial design variable constraints. While the framework generated a valid script, the optimization solver failed to converge, as expected. This test highlights a critical strength of the framework: its ability to analyze and report on failed runs. The Initial Analysis Agent correctly identified the failure, noted that design variables had reached their specified limits, and recommended problem reformulation and constraint relaxation as corrective actions. This diagnostic capability is essential for practical engineering design, where identifying the root cause of non-converging optimizations is a common and time-consuming task. The failure also underscores the importance of the RAG module, whose implementation helps to ground the agents with the physical knowledge (e.g., the relationship between angle of attack and lift generation) needed to suggest more precise corrective actions, such as expanding the bounds on the angle of attack for lift increase.

3.2 Comparative Analysis of Agentic Framework vs. Single-Shot Prompting

To contextualize the performance of the multi-agent framework, a comparative analysis was conducted against Gemini 2.5 Pro, a highly ranked reasoning model for intelligence that employs a Chain-of-Thought (CoT) approach to problem-solving. For a comprehensive comparison, the same prompts were also tested using Cursor, an AI-powered code editor that leverages a generative model, to represent a different class of AI coding assistant (Anysphere, Inc. 2025). The results, summarized in Table 1, highlight the fundamental limitations of applying large, monolithic models to complex, multi-step engineering workflows, and demonstrate the advantages of the proposed framework.

When presented with the design prompt alone, the single-shot model generated syntactically incorrect code that was a hybrid of OpenMDAO and OpenAeroStruct syntax and was therefore unrunnable. In a second attempt, providing the model with a correct sample script

led to arbitrary variable name changes, which caused connection errors within the OpenMDAO architecture, which resulted in uncompilable code. Furthermore, attempting to provide the entire OpenAeroStruct library as context to ground the model exceeded its one million token limit, and could not yield any runnable or analyzable results.

These findings underscore the value of the proposed agentic framework. By decomposing the complex task of MDAO workflow synthesis into manageable sub-tasks handled by specialized agents, the framework overcomes the brittleness and context limitations of single-shot models. This modular approach not only generates correct and executable code but also automates the analysis of results, achieving the entire workflow at a significantly lower computational cost and in less time than a single, failed attempt by a large reasoning model.

4 Discussion

The results demonstrate that the agentic GenAI framework provides a robust and efficient method for automating MDAO workflows from natural language inputs. This automation can be viewed as a form of *process order reduction*, where a complex, multi-step human-driven workflow is reduced to a single, agent-driven command. While general-purpose multi-agent frameworks like MetaGPT (Hong et al. 2024) are powerful for software development, the key contribution of this work is the design and application of a specialized, 'fit-for-purpose' agentic framework tailored to the unique challenges of scientific computing and digital twin construction. This specialization is critical for navigating the specific syntax, object model, and complex data-passing requirements of a physics-based tool like OpenAeroStruct, a task for which general-purpose agents are not equipped.

This tailored approach successfully overcomes the brittleness observed in single-shot LLMs when faced with domain-specific tasks. Furthermore, its capability to not only execute successful optimizations, but also to diagnose and report failed runs, constitutes a significant step towards creating more autonomous and resilient engineering design tools.

	Prompt/Method	Code executable?	Outcome
This (Agentic) Framework	Natural Language Prompt	Yes	Successful
Gemini 2.5 Pro (Reasoning)	Prompt Only (CoT)	No	Failure: Syntactically incorrect code
Gemini 2.5 Pro (Reasoning)	Prompt + Sample Script (CoT)	No	Failure: Code with critical variable connection errors
Gemini 2.5 Pro (Reasoning)	Prompt + Full OAS Codebase (CoT)	No	Failure: Exceeded context token limit
Cursor	Prompt + Full OAS Codebase	No	Failure: Use of non-existent variables

Table 1: Comparison of MDAO Script Generation Approaches For the Test Case

By establishing a fully automated digital thread from natural language requirement to optimized result, this work serves as an enabling technology for dynamic digital twins. A true digital twin requires that the virtual model be rapidly updated in response to new data or changing requirements. The framework presented herein provides the mechanism for this rapid synthesis, allowing the virtual model to be rebuilt and re-optimized on demand. This tight, closed-loop integration of a generative model with a physics-based solver is a defining application of SciML, moving beyond simple code generation to achieve intelligent process automation.

While the current implementation focuses on a specific set of aerodynamic optimizations, the architecture is extensible. The presented results serve as a strong proof-of-concept for a new class of specialized AI-driven tools that can significantly accelerate the design cycle for complex engineering systems.

5 Conclusion

This paper presented a specialized generative AI agentic framework designed to automate the synthesis of complex MDAO workflows from natural language commands. The multi-agent architecture, tailored specifically for the OpenAeroStruct toolset, successfully generates and executes aerostructural optimization scripts, demonstrating a resilience and fidelity superior to single-shot LLM prompting.

The key contribution of this work is a "fit-for-purpose" multi-agent framework that establishes a robust digital thread from high-level requirements to a validated, optimized design. By successfully automating tasks that are challenging for general-purpose agentic systems, this research provides a foundational enabling technology for the digital twin paradigm, offering a method to significantly accelerate the design cycle and make the exploration of novel aircraft concepts more efficient and accessible.

This framework, developed as a proof-of-concept, currently leverages a subset of OAS capabilities. Users can primarily manipulate key aerodynamic variables such as angle of attack, taper ratio, span, dihedral angle, sweep angle, chord distribution, and twist distribution, with lift coefficient (C_L) and drag coefficient (C_D) as the primary output variables. Future development will focus on expanding the framework's functionalities both within OAS and in its integration with other aerospace engineering tools. Improvements within OAS include incorporating multi-point optimization, structural components, and explicit components for user-defined equations. Beyond OAS, potential enhancements include incorporating more tools to support design and analysis of a full aircraft.

Appendix A Selected Prompts For Instructing Agents and Sample Outputs

A selection of prompts used to guide the various agents in the framework is provided below. It is important to note that the structure of the prompts generally follows a consistent format, which includes: (1) the goal assigned to the language model, (2) a step-by-step breakdown of tasks, (3) the expected outputs, (4) code examples where applicable (for coding agents), and (5) supplementary materials such as PDFs or retrieved texts in cases involving RAG.

Reformulator Agent Prompt

Your goal is to rephrase the user's input into a format that OpenAeroStruct can understand and output using the provided schema, which is an object.

Your tasks are:
1. Read the user's input.
2. Identify the key information and requirements.
3. Use the provided schema to structure your response.
4. Generate a response that clearly outlines the key requirements and constraints for the OpenAeroStruct optimization process.
5. Ensure that the response is well-formatted and easy to understand.
6. Use the format below to structure your response.

Include the following information in your response as applicable:

Objective Function: Maximize or minimize objectives.
Trim Condition: If applicable; for example, minimize drag at $C_L = 0.5$ or $\alpha = 3.0$ deg.
Geometric Constraints: Wing area (S), root chord, tip chord, aspect ratio, etc.
Design Variables: Sweep, twist, taper, chord, etc.
Baseline Wing Mesh: Unless specified, use a rectangular mesh. If using a common research model, use CRM.
Optimization Algorithm: Unless specified, use SLSQP.
Plotting Requirements: If any, otherwise state none.
Errors: Note any errors in the input, such as missing key information (e.g., no objective specified), or if the number of constraints exceeds the number of design variables.

Mesh Agent Prompt

Your goal is to implement the instructions provided and write OpenAeroStruct mesh code. Follow the instructions and the code samples carefully, and output using the provided schema as text.

Your tasks are:
1. Read the inputs. You will be given the wing type and geometrical configuration. If the requirements call for a change, update the span, root chord, and wing type, then output the code.
2. Identify the requirements.
3. Use the provided code sample to structure your code.
4. Ensure that the response is well-formatted and easy to understand.

You may need to perform basic math calculations for mesh generation. For a rectangular wing, if span and area are given, calculate the root chord as follows: root_chord = area / span.

Follow the explained code sample provided. Do not add new fields—directly edit the code and explain any changes in the calculations and in the explanation field.

Sample mesh code with commented instructions

Geometry Agent Prompt

Your goal is to follow the instructions provided and write OpenAeroStruct geometry code. Carefully follow the instructions and code samples, and output using the provided schema as text.

Your tasks are:
1. Read the inputs. You will be given the variables that can be optimized. Update the geometry preparation code and uncomment variables that can be optimized.
2. Identify the requirements.
3. Use the provided code sample to structure your code.
4. Ensure that the response is well-formatted and easy to understand.

Follow the explained code sample provided. Do not add new fields—directly edit the code and explain your changes in the calculations and explanation field.

Sample geometry code with commented instructions

Optimization Agent Prompt

Your goal is to follow the instructions provided and write OpenAeroStruct optimization code. Carefully follow the instructions and code samples, and output using the provided schema as text.

Your tasks are:
1. Read the inputs. You will be given the variables that can be optimized. Update the optimization code to include the variables for optimization.
2. Identify the requirements.
3. Use the provided code sample to structure your code.
4. Ensure that the response is well-formatted and easy to understand.

Sample optimization code with commented instructions

Information Retrieval Agent Prompt

Your goal is to provide a query to retrieve information from a RAG system, using the provided schema (an object) for your output.

You need to craft a query so that another agent can use it to understand the physical variables involved in optimization.

Your tasks are:
1. Read the refined user input.
2. Identify the variables used.
3. Write a query that will be used to retrieve information from a RAG system. Include only the wing geometry variables used in optimization. Ignore variables not used in optimization.
4. Use the provided schema to structure your response.

Example output:
Please provide information for the following variables used in aerodynamic wing optimization: Taper ratio, Twist angle, Sweep angle, Lift coefficient (CL), Wing area (S), Wing span (b), and Drag.

Initial Analysis Agent Prompt

Your goal is to review the results of the OpenAeroStruct optimization and provide recommendations. Output should use the provided schema, which is an object. The initial problem statement and two PDF reports of the results will be provided.

Your tasks are:
1. Read the PDFs of the OpenAeroStruct optimization results.
2. Identify the key information and results.
3. Use the provided schema to structure your response.

Structure your response in four parts:
1) Analysis: Assess whether the optimization was successful, and explain what the results mean.
2) Recommendations: Suggest next steps, such as further optimizations, adjustments to design variables, or other considerations. Evaluate whether the results are reasonable; if not, explain why.
3) Optimization Performance: Discuss the computational performance, including the number of iterations, computation time, and, if unconverged, suggest an alternative optimization algorithm.
4) Unrelated Observations: Note any other observations not directly related to the optimization, such as manufacturability or non-design variables.

Include the following details:
1) Does the optimization achieve the objectives? If not, why? State numerical values of the objective.
2) List key design variables and their values: Do they reach their limits? Are they physically reasonable?
3) Analyze graphical plots: Do they make sense? Are there anomalies? For drag minimization, is lift distribution elliptical? How close is it to ideal?
4) Report computational performance.

Report Writing Agent Prompt

Your goal is to rewrite the LLM output into a report format, using the schema provided (which is an object). You will be given the textual analysis from another LLM.

Your tasks are:
1. Read the analysis and recommendations.
2. Identify the key information and requirements.
3. Use the provided schema to structure your response.
4. Generate a response that answers the user's question in paragraph form, formatted in properly written LaTeX.

Use all the information given to write a detailed analysis of the results and recommendations.

Please include this figure in the report:
The file path is "Figures/Optimized_Wing.pdf", which contains the optimized wing visualization. Reference this figure in your analysis, as it will also be provided.

Acknowledgments The authors wish to thank Prof. Joaquim R.R.A. Martins. The framework presented herein was first conceived as a project in his MDO course, and his instruction was foundational to its development.

Author contributions Conan Lee: Conceptualization, Methodology, Software, Visualization, Writing - Original Draft; Joaquim Martins: Conceptualization, Supervision; Gokcin Cinar: Conceptualization, Investigation, Supervision, Writing - Review & Editing.

Data availability The code developed for this study is available via the University of Michigan IDEAS Lab GitHub repository (Lee and Cinar 2025). The prompts used to guide the agents are included in Appendix A of the manuscript. Due to copyright restrictions, the document used for

retrieval-augmented generation is not publicly distributed. However, users are encouraged to upload a relevant wing design document and follow the instructions in the repository to generate their own embeddings.

Funding This research received no external funding and was conducted as independent research.

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

Ethics approval and Consent to participate Not applicable.

Replication of Results The necessary information to replicate the results of this study are provided in the manuscript. Access to the code used in this study is available on the University of Michigan IDEAS Lab Github page (Lee and Cinar 2025).

References

Abdulhak, S., Hubbard, W., Gopalakrishnan, K., Li, M.Z.: CHATATC: Large Language Model-Driven Conversational Agents for Supporting Strategic Air Traffic Flow Management. arXiv:2402.14850 (2024). https://doi.org/10.48550/arXiv.2402.14850

Anysphere, Inc.: Cursor: The AI-first Code Editor. https://cursor.sh. Accessed: 2025-07-31 (2025)

Blocklove, J., Garg, S., Karri, R., Pearce, H.: Evaluating LLMs for Hardware Design and Test. arXiv:2405.02326v2 (2024). https://arxiv.org/abs/2405.02326v2

Buehler, M.J.: MechGPT, a language-based strategy for mechanics and materials modeling that connects knowledge across scales, disciplines and modalities. arXiv:2310.10445v1 (2023). https://arxiv.org/abs/2310.10445v1

Cho, J., Mahata, D., Irsoy, O., He, Y., Bansal, M.: M3DocRAG: Multi-modal Retrieval is What You Need for Multi-page Multi-document Understanding. arXiv:2411.04952 (2024). https://doi.org/10.48550/arXiv.2411.04952

Farimani, A.B., Jadhav, Y., Pak, P.: LLM-3D Print: Large Language Models To Monitor and Control 3D Printing. Research Square (2024). https://doi.org/10.21203/rs.3.rs-4945366/v1

Ferrari, A., Willcox, K.: Digital twins in mechanical and aerospace engineering. Nature Computational Science **4**(3), 178–183 (2024) https://doi.org/10.1038/s43588-024-00613-8

Ghafarollahi, A., Buehler, M.J.: AtomAgents: Alloy design and discovery through physics-aware multi-modal multi-agent artificial intelligence. arXiv:2407.10022 (2024). https://doi.org/10.48550/arXiv.2407.10022

Gray, J.S., Hwang, J.T., Martins, J.R.R.A., Moore, K.T., Naylor, B.A.: Openmdao: an open-source framework for multidisciplinary design, analysis, and optimization. Structual and Multidisciplinary Optimization **59**(4), 1075–1104 (2019) https://doi.org/10.1007/s00158-019-02211-z

Guo, X., Keivan, D., Syed, U., Qin, L., Zhang, H., Dullerud, G., et al.: ControlAgent: Automating Control System Design via Novel Integration of LLM Agents and Domain Expertise. arXiv:2410.19811 (2024). https://doi.org/10.48550/arXiv.2410.19811

Google: Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 (2025). https://doi.org/10.48550/arXiv.2312.11805

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., et al.: MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 (2024). https://doi.org/10.48550/arXiv.2308.00352

Jasa, J.P., Hwang, J.T., Martins, J.R.R.A.: Open-source coupled aerostructural optimization using python. Structural and Multidisciplinary Optimization **57**(4), 1815–27 (2018) https://doi.org/10.1007/s00158-018-1912-8

Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S.: A Survey on Large Language Models for Code Generation. arXiv:2406.00515 (2024). https://doi.org/10.48550/arXiv.2406.00515

Lee, C., Cinar, G.: OpenAeroStruct LLM

Agent Repository. https://github.com/ideas-um/openaerostruct-llm-agent. Accessed: 2025-07-31 (2025)

Mishra, S., Jadhav, V.S., Karande, S., Runkana, V.: Design and optimization of heat exchangers using large language models. In: Fourth Workshop on Knowledge-infused Learning (2024). https://openreview.net/forum?id=2CJ9GNV8dL

Martins, J.R.R.A., Lambe, A.B.: Multidisciplinary design optimization: A survey of architectures. AIAA Journal **51**(9), 2049–75 (2013) https://doi.org/10.2514/1.J051895

Niederer, S.A., Sacks, M.S., Girolami, M., Willcox, K.: Scaling digital twins from the artisanal to the industrial. Nature Computational Science **1**(5), 313–20 (2021) https://doi.org/10.1038/s43588-021-00072-5

OpenAI: GPT-4 Technical Report. arXiv:2303.08774 (2024). https://doi.org/10.48550/arXiv.2303.08774

Pandey, S., Xu, R., Wang, W., Chu, X.: OpenFOAMGPT: a RAG-Augmented LLM Agent for OpenFOAM-Based Computational Fluid Dynamics. arXiv:2501.06327 (2025). https://doi.org/10.48550/arXiv.2501.06327

Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., Wen, J.-r.: Tool learning with large language models: a survey. Frontiers of Computer Science **19**(8), 198343 (2025) https://doi.org/10.1007/s11704-024-40678-2

Raj, H., Gupta, V., Rosati, D., Majumdar, S.: Semantic Consistency for Assuring Reliability of Large Language Models. arXiv:2308.09138 (2025). https://doi.org/10.48550/arXiv.2308.09138

Ramírez-Rueda, R., Benítez-Guerrero, E., Mezura-Godoy, C., Bárcenas, E.: Transforming software development: A study on the integration of multi-agent systems and large language models for automatic code generation. In: 12th International Conference in Software Engineering Research and Innovation (CONISOFT), pp. 11–20 (2024). https://doi.org/10.1109/CONISOFT63288.2024.00013

Samplawski, C., Cobb, A.D., Jha, S.: AGENT: An Aerial Vehicle Generation and Design Tool Using Large Language Models. arXiv:2504.08981 (2025). https://doi.org/10.48550/arXiv.2504.08981

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., et al.: LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 (2023). https://doi.org/10.48550/arXiv.2302.13971

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., et al.: AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 (2023). https://doi.org/10.48550/arXiv.2308.08155

Willcox, K., Segundo, B.: The role of computational science in digital twins. Nature Computational Science **4**(3), 147–9 (2024) https://doi.org/10.1038/s43588-024-00609-4

Zhang, T., Liu, Z., Xin, Y., Jiao, Y.: MooseAgent: A LLM Based Multi-agent Framework for Automating Moose Simulation. arXiv:2504.08621 (2025). https://doi.org/10.48550/arXiv.2504.08621