engineering proceedings

MDPI

Proceeding Paper

AI Assistant for Rapid Modelling and Design of Aircraft †

Sergio Jimeno Altelarrea *, Utkarsh Gupta and Atif Riaz

Faculty of Engineering and Applied Sciences, Cranfield University, Cranfield MK43 0AL, UK; utkarsh.gupta@cranfield.ac.uk (U.G.); a.riaz@cranfield.ac.uk (A.R.)

* Correspondence: s.jimeno@cranfield.ac.uk

† Presented at the 15th EASN International Conference, Madrid, Spain, 14–17 October 2025.

Abstract

Collaborative aircraft design environments face significant challenges in intuitive geometry manipulation and tool integration. This research develops an AI-assisted interface using the Model Context Protocol (MCP) to bridge natural language commands with established aerospace tools. The approach integrates large language models with three specialized applications for optimization, visualization, and aircraft geometry modification. Results demonstrate successful implementation, enabling designers to accomplish complex tasks such as multi-objective optimization and empennage reconfiguration through conversational prompts. While occasional AI misinterpretations required prompt refinement, the system proved effective at translating intent into precise tool operations. The study concludes that MCP provides a viable framework for creating intuitive design interfaces while maintaining accuracy via integration with domain-specific computational methods.

Keywords: artificial intelligence; model context protocol; natural language interface; aircraft design optimization; human–computer interaction; computational design; AI-assisted engineering

1. Introduction

The aviation industry faces increasing pressure to accelerate design cycles while maintaining rigorous performance standards, creating demand for more intuitive computational interfaces. Modern collaborative design environments promise rapid prototyping and enhanced teamwork, but their effectiveness is often hampered by traditional software tools. These existing applications, designed for mouse and keyboard interaction, create barriers to fluid collaboration and can disrupt creative workflows.

Natural language interaction presents a promising alternative to conventional input methods. While natural language processing historically presented challenges, recent advances in large language models (LLMs) have enabled new possibilities for interfacing with established domain-specific engineering tools. The approach leverages the strengths of both worlds: the powerful natural language processing capabilities of LLMs for intuitive interaction, and the proven reliability of specialized aerospace software for tasks such as accurate aerodynamic analysis and structural optimization.

The Model Context Protocol (MCP) [1] offers standardized communication between large language models and external software applications. This study investigates MCP integration with in-house software tools that have demonstrated value in previous research projects. These tools include AirCADia Composer for computational model orchestration and optimization; AirCADia Vision for result visualization; and AirCADia Geo for rapid

Academic Editors: Spiros Pantelakis, Andreas Strohmayer and Gustavo Alonso

Published: 19 May 2026

aircraft geometry creation and modification. The research also examines the implementation costs associated with MCP integration and identifies factors that influence the complexity of adapting existing tools to this protocol.

2. Materials and Methods

2.1. The Model Context Protocol

The Model Context Protocol (MCP) [1] is an open communication standard designed to facilitate structured interaction between large language models (LLMs) and external software applications. MCP communication operates through a client-server architecture.

The MCP server integrates with a software application and exposes its functionalities as callable tools. Each tool represents a discrete operation the application can perform (e.g., aerodynamic analysis, geometry modification) and is defined with precise specifications for input and output parameter formats, including data types and structures.

The MCP client interacts with these exposed tools, typically using a large language model to interpret natural language user requests and translate them into appropriate MCP tool calls. The client also handles server responses and formats the results for user presentation.

2.2. MCP Implementation Architecture

The objective of implementing the MCP in this research is to provide an MCP interface that is functionally equivalent to the UI of existing software applications. Consider, for example, an optimization application. In a traditional approach, users would define the objective function, design variables, and constraints through mouse and keyboard inputs via a graphical user interface, which provides visual feedback about the software state.

In contrast, an MCP implementation of the same application exposes these functionalities through MCP tool calls. Users employ natural language commands instead of direct manipulation, while the underlying definition of objective functions, design variables, constraints, and optimization algorithms remains consistent. Feedback regarding the optimization problem state is provided both through the application UI and via natural language responses from the MCP client, as illustrated in Figure 1.

As shown in Figure 1, the MCP architecture includes three new components: (1) an MCP server that communicates with the application and exposes its functionalities as tools; (2) an LLM that interprets user commands and determines appropriate tool calls; and (3) an MCP client that mediates between the user, LLM, and MCP server. For this study, the MCP server and client operate locally in separate processes, while the LLM executes in the cloud. Alternative architectures exist, such as integrating the MCP chat client directly within the application or hosting the LLM locally.

2.3. Existing Software Applications and Use Cases

This research aimed to cover a broad spectrum of aerospace engineering tasks by integrating MCP with three established software applications from the AirCADia suite:

- **AirCADia Composer** [2]: a software for orchestrating computational models and executing studies including design of experiments and multi-objective optimization.
- **AirCADia Vision** [2]: a data visualization application supporting multiple plot types (scatter, contour, parallel coordinates) with extensive customization options.
- **AirCADia Geo** [3]: a geometry modeling tool for aircraft configuration definition and modification, utilizing simple geometric primitives (cuboids, ellipsoids, cylinders) and CST parameterization for airfoils and body cross-sections [4].

https://doi.org/10.3390/engproc2026133156

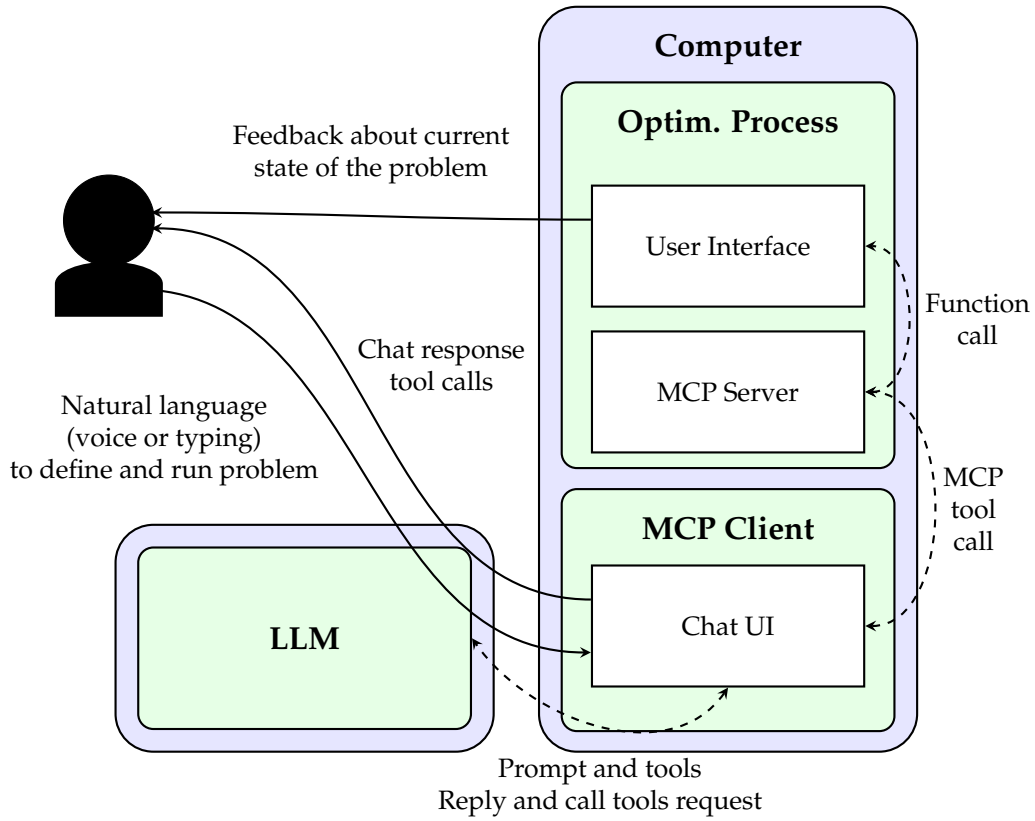


Figure 1. MCP Approach.

A distinct use case was defined for each application to evaluate the MCP integration across different engineering contexts. The specific details of each use case are elaborated in the following subsections.

2.3.1. Wind Turbine Optimization

AirCADia Composer was evaluated for its capability to configure and execute a multi-objective optimization study for wind turbine design, following the original problem proposed in [5]. The optimization problem was defined as:

Design a wind turbine with maximum power and minimum cost, and a useful life greater than 25 years.

$$p = \frac{d^2 v^3}{2000} \tag{1}$$

$$c = pd \tag{2}$$

$$l = \frac{100}{e^{\sqrt{\frac{p+\frac{d}{10}}{100}}}} \tag{3}$$

where d: diameter of wind turbine [m]; v: wind velocity [m/s]; p: power [kW]; c: cost [k£]; l: life [years].

The LLM was expected to: create data objects for all variables; define mathematical models for power, cost, and life calculations; configure the optimization study with appropriate design variables, objectives, and constraints; and execute the optimization. The MCP tools exposed by AirCADia Composer are detailed in Table 1.

Table 1. List of MPC tools by application.

Tool	Description
	AirCADia Composer
AddVariable	Adds a new data variable to the current project
AddModelCSharp	Adds a new C# model to the current project
AddModelPython	Adds a new python model to the current project
AddWorkflow	Adds a new workflow to the current project
AddMultiObjective-OptimizationStudy	Adds a new multi-objective study to the current project
AddFullFactorial-DoEStudy	Adds a new full factorial study to the current project
SetValues	Adds a new model to the current project
RunModel	Runs a model
RunStudy	Runs a model
CreateProject	Create a new composer project
LoadProject	Loads an existing composer project
GetProjectData	Gets the data (variables, models, workflows, and studies) of the current project
	AirCADia Vision
LoadDataset	Loads a dataset from a .csv file into the current project
GetDatasets	Gets the datasets of the current project
RemoveDataset	Removes the dataset with the given id
GetWidgetTypes	Gets the available widget types
CreateWidget	Creates a widget of the desired type, with the desired id
GetWidgets	Gets the widgets of the current project
UpdateScatter2DPlot	Update the desired properties of the plot
UpdateScatter3DPlot	Update the desired properties of the plot
UpdateParallel-CoordinatesPlot	Update the desired properties of the plot
UpdateContourPlot	Update the desired properties of the plot
EnableParetoClassification	Enable pareto classification.
	AirCADia Geo
GetComponent	Returns a specific component from the current aircraft
GetComponents	Returns all the components of the current aircraft
RemoveComponent	Removes a specific component from the current aircraft
AddOrUpdateFuselage	Adds or updates a fuselage in the current aircraft
AddOrUpdateWing	Adds or updates a wing in the current aircraft
AddOrUpdate-HorizontalTail	Adds or updates a horizontal tail in the current aircraft
AddOrUpdate-VerticalTail	Adds or updates a vertical tail in the current aircraft
AddOrUpdatePylon	Adds or updates a pylon in the current aircraft
AddOrUpdateNacelle	Adds or updates a nacelle in the current aircraft

2.3.2. Visualization of Wind Turbine Optimization Results

This use case evaluated the MCP ability to generate and customize visualizations through natural language commands. The LLM was tasked with creating three scatter plots showing variable relationships in pairs; a parallel coordinates plot displaying all variables; coloring points according to Pareto optimality; and customizing the plot appearance.

This use case was anticipated to be more complex as users can perceive the plots and all their graphical elements, while the MCP interface only provides access to a limited set of plot attributes and their values. The available tools in AirCADia Vision are listed in Table 1.

2.3.3. Aircraft Geometry Modification

AirCADia Geo MCP was assessed through a geometry modification task involving the conversion of an aircraft empennage from a conventional to a T-tail configuration. Starting

with the baseline geometry from [3], the LLM was instructed to reduce the vertical tail span by 20% and reposition the horizontal tail to achieve the T-tail arrangement.

This is expected to be the most complex use case as it involves spatial relationships and complex parameterization using CST parameters. The MCP tools provided by AirCADia Geo are summarized in Table 1.

2.4. MCP Server Development and Experimental Setup

Since the tools are written in C#, the MCP servers were developed using the official C# SDK for the Model Context Protocol Toolkit [6]. Each host application was registered as a dependency injection service, making its functionalities and data structures accessible within tool implementations. This architecture enables the MCP server to leverage existing application capabilities and update the UI in response to tool executions.

Communication between MCP servers and clients was implemented using HTTP transport. All tool inputs and outputs were serialized as JSON objects [7], which introduces limitations due to JSON's restricted type system supporting only numbers, strings, booleans, arrays, objects, and null values. Managing conversion between JSON and domain-specific data types added time and complexity to the MCP server implementation.

All experiments were conducted using Visual Studio Code's AI chat functionality [8] operating in agent mode [9], with GPT-4.1 [10] as the underlying large language model. The AirCADia MCP tools were registered with the chat environment. Users provided natural language commands, which the LLM processed to generate appropriate MCP calls. Server responses were subsequently interpreted by the LLM and formatted for user presentation.

3. Results

3.1. Wind Turbine Optimization

To configure the wind turbine optimization study, the LLM was prompted with: *Use Composer to design a wind turbine with maximum power and minimum cost, and a useful life greater than 25 years. Variables: d: diameter of wind turbine [m]; v: wind velocity [m/s]; p: power [kW]; c: cost [k£]; l: life [years]. Models: p = (d^2 v^3)/2000 c = pd l = 100/e^sqrt((p + d/10)/100)*

The LLM successfully initiated the process by creating variables using *AddVariable*, then defined mathematical models using *AddModelPython*. The initial attempt to configure the optimization study via *AddMultiObjectiveOptimizationStudy* failed due to missing workflow composition. After recognizing this error, the LLM created an appropriate workflow using *AddWorkflow* and successfully re-executed *AddMultiObjectiveOptimizationStudy*. The optimization was finally executed using *RunStudy*.

Figure 2 shows the resulting object dependency relationships and computational workflow structure.

3.2. Visualization of Wind Turbine Optimization Results

The initial prompt for visualization requested: *I want to see plots of the wind turbine results using Vision. I want to see v vs. d, v vs. p and c vs. d.*

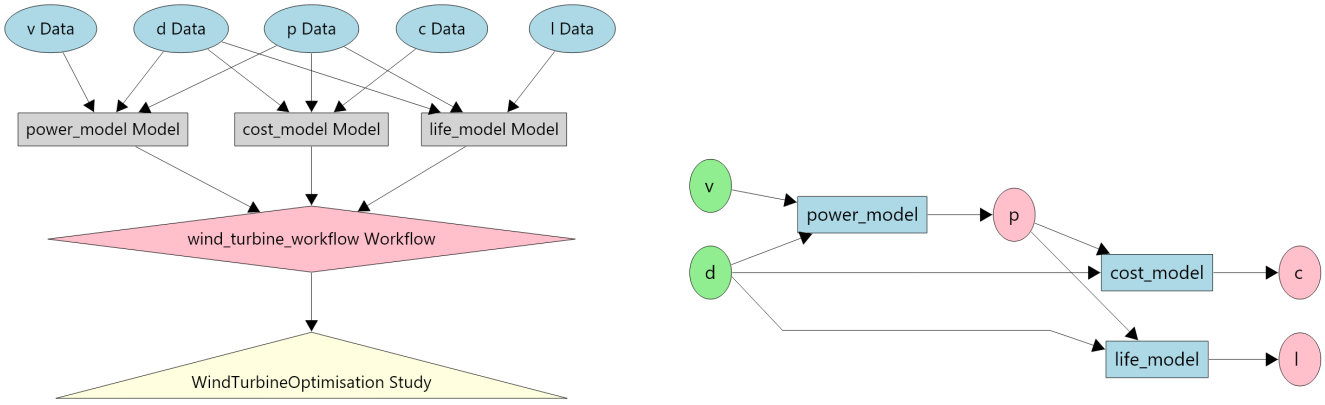
The first *CreateWidget* call specified an unsupported 'Scatter2DPlot' type. After receiving error details listing valid widget types, the LLM successfully created three scatter plots using the correct 'Scatter2D' type. Since default axes were set to dataset columns *ID* and *Generation*, three subsequent *UpdateScatter2DPlot* calls selected the desired variables.

A second prompt requested: *I also want to see a parallel coordinated plot with all variables selected. I want to see which points are pareto.*

The LLM successfully created a parallel coordinates plot via *CreateWidget* and applied Pareto classification using *EnableParetoClassification*, but failed to select all variables.

https://doi.org/10.3390/engproc2026133156

After prompt refinement to explicitly request variable selection, the desired visualization was achieved.



(**a**) (**b**)

Figure 2. Graphical representation of AirCADia composer results: (**a**) object graph. (**b**) computational workflow.

A final prompt customized appearance: *I want to change the background of all plots to white and then update the axis and borders so they show good contrast, Also make sure the fonts are bigger.*

Calls to *UpdateScatter2DPlot* and *UpdateParallelCoordinatesPlot* successfully modified background colors, axis properties, and font sizes across all plots. However, the axis bound labels in the parallel coordinates plot remained white and became illegible against the white background. Figure 3 shows the final output, where plot arrangement was manually configured but all other elements were generated through AI interaction.

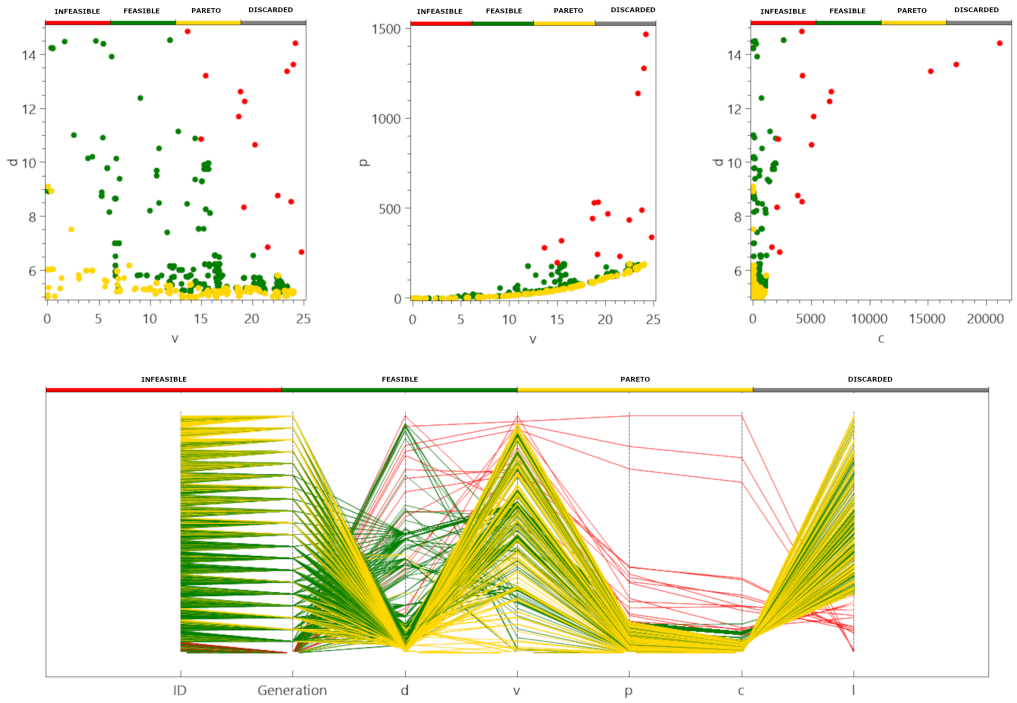


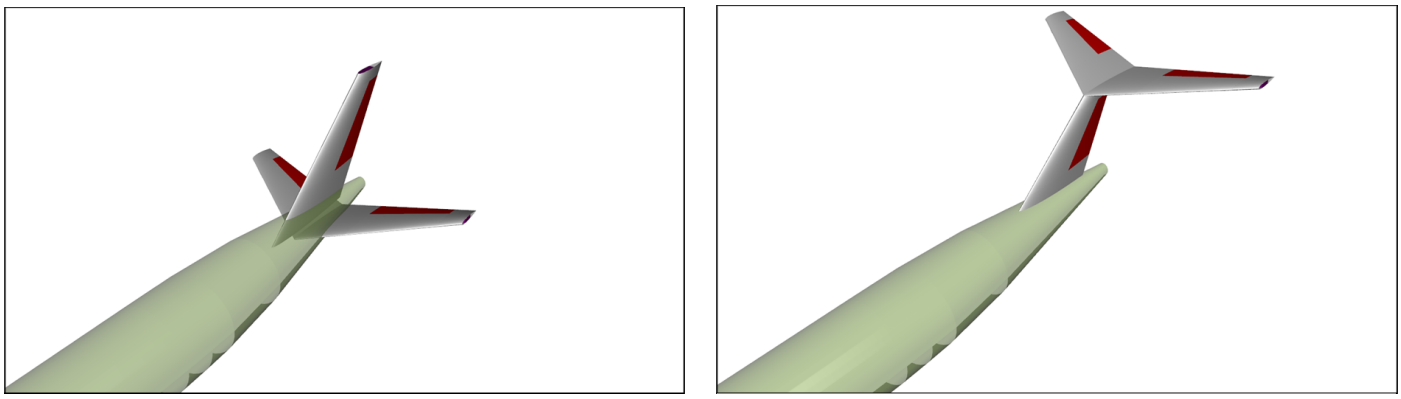
Figure 3. AirCADia Vision results showing scatter plots and parallel coordinates plot of wind turbine optimization results.

https://doi.org/10.3390/engproc2026133156

3.3. Aircraft Geometry Modification

The geometry modification prompt specified: *I want to convert the current aircraft's empennage into a T-tail. For this purpose, the vertical tail should be 20% shorter and the horizontal one should be moved on top. The position of the horizontal tail should match the new (shorter) vertical tail's tip leading edge.*

The LLM began by calling *GetComponents* to acquire necessary geometric context; reduced the vertical tail span from 10 m to 8 m using *AddOrUpdateVerticalTail*; and, finally, calculated the appropriate position and relocated the horizontal tail using *AddOrUpdate-HorizontalTail*. Figure 4 compares the aircraft geometry before and after modification.



(**a**) (**b**)

Figure 4. Comparison of AirCADia Geo tail geometry: (**a**) before modification. (**b**) after modification.

4. Discussion

The results demonstrate that MCP can effectively interface LLMs with specialized aerospace engineering software. A sensible design of MCP outputs for failure cases helped the LLM recover from errors with minimal human intervention. However, in the visualization use case, a prompt required refinement to achieve the desired result. Furthermore, in the geometry modification use case, the prompt itself needed to be quite detailed, as experience showed that simpler requests such as 'convert the tail to a T-tail' failed to produce consistent, high-quality results.

The most common error types involved misunderstandings of how the tools work, the correct sequence for calling them, and hallucinated values such as unsupported widget types. Explanatory error messages helped the LLM to independently recover from these situations. Errors related to a failure to correctly grasp the problem context, such as geometric relationships, were more likely to require human intervention for resolution.

The complexity of implementing MCP servers for existing software applications largely depends on the underlying application architecture and design. It can range from simply wrapping existing APIs to significant codebase refactoring. It may not even be feasible for closed-source applications that lack sufficient interfaces for MCP integration.

Applications with modular designs and well-defined APIs, such as AirCADia Composer, were easier to adapt to MCP. The server could leverage existing interfaces to expose functionalities as MCP tools, with only additional error handling required. In contrast, for AirCADia Vision and Geo, the UI components were tightly coupled with the application logic, requiring a redesign of the application architecture and the creation of specific classes for JSON serialization. The benefits of this redesign extend beyond MCP integration, but their implementation required weeks of additional development time—a critical factor to consider when planning MCP implementations.

https://doi.org/10.3390/engproc2026133156

Future work could explore more complex use cases, alternative architectures for the MCP server and client, and tighter integration with other software tools and collaborative environments such as extended reality. Investigating effective methods for making the LLM aware of the broader application context is also a promising direction, as this appears to be the source of the more challenging errors.

Author Contributions: Conceptualization, S.J.A., U.G. and A.R.; methodology, S.J.A., U.G. and A.R.; software, S.J.A. and U.G.; validation, S.J.A., U.G. and A.R.; writing—original draft preparation, S.J.A. and U.G.; writing—review and editing, S.J.A., U.G. and A.R.; supervision, A.R.; funding acquisition, A.R. All authors have read and a greed to the published version of the manuscript.

Funding: The research leading to the results in Section 4 has received funding from Innovate UK, under the Out of Cycle Next Generation Highly Efficient Air Transport (ONEHeart) project (Innovate UK proposal number 10003388).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
CST	Class-Shape Transformation
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
SDK	Software Development Kit
UI	User Interface

References

1. What Is the Model Context Protocol (MCP)? Available online: https://web.archive.org/web/20251126033954/https://modelcontextprotocol.io/docs/getting-started/intro (accessed on 5 May 2026).
2. Guenov, M.D.; Nunez, M.; Molina-Cristóbal, A.; Datta, V.C.; Riaz, A. AirCADia—An Interactive Tool for the Composition and Exploration of Aircraft Computational Studies at Early Design Stage. In Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences, St. Petersburg, Russia, 7–12 September 2014.
3. Mura, G.L.; Riaz, A.; Guenov, M.; Molina-Cristobal, A.; Chen, X.; Sharma, S.; Nicolls, K.; Lynas, C. Early Stage Design of High-Lift Devices with System and Manufacturing Constraints. In *AIAA Scitech 2019 Forum*; AIAA SciTech Forum; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2019. [CrossRef]
4. Kulfan, B.; Bussoletti, J. "Fundamental" Parameteric Geometry Representations for Aircraft Component Shapes. In *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2006. [CrossRef]
5. Riaz, A. Lecture 1. Introduction to Computational Optimisation Design. 2020, *unpublished lecture notes*.
6. Anthropic and Contributors. ModelContextProtocol. AspNetCore. 2025. Available online: https://web.archive.org/web/20260417025100/https://github.com/modelcontextprotocol/csharp-sdk (accessed on 5 May 2026).
7. Introducing JSON. Available online: https://web.archive.org/web/20251128163343/https://www.json.org/json-en.html (accessed on 5 May 2026).
8. Visual Studio Code. Available online: https://web.archive.org/web/20251127153147/https://code.visualstudio.com/ (accessed on 5 May 2026).

9. Visual Studio Code. Agent Mode Chat. Available online: https://web.archive.org/web/20251011230858/https://code.visualstudio.com/docs/copilot/chat/chat-agent-mode (accessed on 5 May 2026).
10. Introducing GPT-4.1 in the API. Available online: https://web.archive.org/web/20251002105158/https://openai.com/index/gpt-4-1/ (accessed on 5 May 2026).