DUCTILE: Agentic Large Language Model Orchestration of Engineering Analysis in Product Development Design Practice

Citation for the original published paper (version of record):

Pradas Gómez, A., Brahma, A., Isaksson, O. (2026). DUCTILE: Agentic Large Language Model Orchestration of Engineering Analysis in Product
Development Design Practice. Journal of Mechanical Design - Transactions of the ASME, 148(12).
http://dx.doi.org/10.1115/1.4072279

ASME Journal of Mechanical Design
Online journal at:
https://asmedigitalcollection.asme.org/mechanicaldesign

DUCTILE: Agentic Large Language Model Orchestration of Engineering Analysis in Product Development Design Practice

Alejandro Pradas-Gómez[1]
Department of Mechanical Engineering,
Chalmers University of Technology,
SE-412 96 Gothenburg, Sweden;
GKN Aerospace Engine Systems,
SE-461 81 Trollhättan, Sweden
e-mail: alejandro.pradas@chalmers.se

Arindam Brahma
Department of Mechanical Engineering,
Chalmers University of Technology,
SE-412 96 Gothenburg, Sweden
e-mail: brahma@chalmers.se

Ola Isaksson
Department of Mechanical Engineering,
Chalmers University of Technology,
SE-412 96 Gothenburg, Sweden
e-mail: ola.isaksson@chalmers.se

Engineering analysis automation in product development relies on rigid interfaces between tools, data formats, and documented processes. When these interfaces change, as they routinely do as the product evolves in the engineering ecosystem, the automation support breaks. This article presents a DUCTILE (delegated, user-supervised coordination of tool- and document-integrated large language model (LLM)-enabled) agentic orchestration, an approach for developing, executing, and evaluating LLM-based agentic automation support of engineering analysis tasks. The approach separates adaptive orchestration, performed by the LLM agent, from deterministic execution, performed by verified engineering tools. The agent interprets documented design practices, inspects input data, and adapts the processing path, while the engineer supervises and exercises final judgment. DUCTILE is demonstrated on a constrained but realistic industrial structural analysis task at an aerospace manufacturer, where the agent handled input deviations in format, units, naming conventions, and methodology that would break traditional scripted pipelines. Evaluation against expert-defined acceptance criteria and deployment with practicing engineers confirm that the approach produces correct, methodologically compliant results across 10 repeated independent runs. The article discusses the paradigm shift and the practical consequences of adopting agentic automation, including unintended effects on the nature of engineering work when removing mundane tasks and creating an exhausting supervisory role. [DOI: 10.1115/1.4072279]

Keywords: large language models, agent-based design, design automation, design methodology, design process, product development, expert systems, design evaluation

1 Introduction

Product development relies on an engineering ecosystem with many interacting elements: engineers, data, methods, processes, and specialist tools [1–3]. In engineering analyses, organizations often try to automate how these interactions occur, for example, through scripts and tool-specific workflows. In practice, engineers repeatedly move between context-specific tools, scripts, and methods, adapting the workflow as the requirements, models, and interfaces change [3,4]. However, such changes, even if minor, can often lead to significant disruptions creating knock-on effects in downstream design activities or changes within the product [5,6]. For example, because of a tool update, a revised requirement, or an organizational change. As a consequence, the engineer works in a brittle analysis process [7], where significant time is spent on data wrangling [8,9] and tool orchestration [10,11], rather than on value-added decisions that directly affect product performance.

In practice, organizations have addressed the brittleness of engineering automation through two main strategies. One is to extend the deterministic automations to add more rules and cover more scenarios, which increases complexity and creates new failure modes as the variation grows [12–14]. If the automation itself cannot cover the changes, the second approach is to rely on expert engineers to adapt on a per-case basis. This second approach is slow and concentrates critical knowledge in individuals, creating an organizational vulnerability when those individuals are not available [14–16]. Neither approach scales well in environments where the workflow evolves alongside the product over a decade-long development process. Approaches integrating machine learning with expert validation have improved knowledge sourcing efficiency in well-structured domains [15], but remain dependent on predefined data structures, expert-curated feature sets, and rigid methodological pipelines, leaving the underlying brittleness of the workflow unaddressed.

In this article, a third approach is proposed: large language model (LLM)-based agents as orchestration layers that connect

[1]Corresponding author.

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received March 16, 2026; final manuscript received June 30, 2026; published online July 30, 2026. Assoc. Editor: Xingang Li.

Table 1 Requirements for DUCTILE and supporting literature

Req.	Requirement name	Description	Supporting literature
R1	Inspectability	Agentic application (prompts, plans, tool calls, and intermediate artifacts) must be directly inspectable by engineers/auditors; no hidden state	[17,18]
R2	Reproducibility	Model, prompt, tool versions, and inputs must be pinned and logged; runs are reproducible or cause-attributable despite stochastic LLM steps	[17,18]
R3	Deterministic execution boundary	Safety/engineering computations run in verified, deterministic tools; the LLM performs orchestration and code generation only	[18,44]
R4	Traceability and auditability	Every artifact is trace-linked to the generating step (arguments, stdout/stderr, checksums, and timestamps) to support do/check/approve reviews	[17,18]
R5	Data governance	Deployment mode (on-premises or application programming interfaces (API)) and data flows comply with confidentiality/export/privacy rules and are justified	[17,18]
R6	Human oversight and accountability	Engineers review plans and sign off on outputs; autonomy is limited to orchestration, not engineering judgment	[17]
R7	Robustness to routine variability	Orchestration handles foreseeable variation (formats, units, naming, and minor method updates) without modifying certified tools	[5,13,44–48]
R8	Observability	The system emits traces for model calls and tool invocations (arguments, latency, and tokens/cost) sufficient to diagnose failures	[49,50]
R9	Minimal coupling	Tool integration is lightweight/externalized (command-line interfaces (CLIs)/typed APIs); generated code is readable and runnable by engineers	[12,13,47,48]
R10	Evaluation and pass^k	Reliability is demonstrated on curated cases with deterministic checks and/or LLM-as-a-judge, using repeated independent runs (pass^k) proportionate to risk	[51–53]
R11	Change control	Any change to model, prompts, tools, or design practice triggers re-evaluation on the same cases before deployment	[17,18]
R12	Documentation quality	Tool/method documentation is sufficiently structured for agents to follow procedures; ambiguities are revised or clarified	[54,55]

engineers to the verified and trusted domain-specific engineering tools. The agent does not replace the domain-specific software, but instead, interprets the documented design practice, inspects the available data, and adapts to the particular context. In doing so, it generates and executes processing code at the engineer's request through the same verified and trusted tools that the engineer would use manually. By adding a now standardized and universal orchestration layer on the automation, this approach minimizes the development (and maintenance) burden of tailored workflows and frameworks. With this approach, engineers remain responsible for defining the goal, reviewing the plan, supervising execution, and validating outputs as per the latest artificial intelligence (AI) policies and standards of aerospace and automotive certification bodies [17–19].

This article makes two contributions. First, it presents delegated, user-supervised coordination of tool- and document-integrated large language model-enabled agentic orchestration (DUCTILE), an approach for developing, executing, and evaluating LLM-based agentic automation of engineering analysis tasks, grounded in the separation of adaptive orchestration from deterministic, verified tool execution [20]. Second, it provides a test case and evaluates the approach on a constrained but realistic industrial load processing task at an aerospace manufacturer, where the agent handled four input deviations that would break traditional automation, across 10 independent runs and two engineers' interactions with different agentic supervision styles. The novelty lies in demonstrating that LLM agents can orchestrate real aerospace analysis workflows transparently and compatibly with existing verified tools, while absorbing the input variability. The case, tools, and evaluation are provided as open-source for other researchers to replicate and improve.

2 Background

This section frames the engineering challenge that motivates the present work. Section 2.1 describes the operational context in practice. Section 2.2 reviews the families of computational support developed in the literature, their characteristic limitations, and the orchestration principle and requirements (Table 1) that follow from them.

2.1 The Tool Adoption Challenge at Aerospace Engineering Companies. Of specific interest in this article are the challenges of automation in the mechanical engineering analysis, particularly in the aerospace domain [15,21]. The engines division of GKN Aerospace (hereafter, the case company) is a Swedish manufacturer of jet engine components and is a risk and revenue-sharing partner for various jet engine Original Equipment Manufacturers. The case company routinely performs structural analysis activities to substantiate airworthiness and certification activities, an essential part of the safety-focused industry [22]. These activities follow documented design practices and rely on a combination of commercial solvers, legacy scripts, and an internally developed and evolving ecosystem of modular PYTHON tools [23] to preprocess and postprocess product data that are the basis for the requirements justification. The variety of tools in the ecosystem creates a practical challenge that directly relates to the user's experience level. A new engineer, for example, may understand the engineering principles, but may face a steep learning curve in connecting that knowledge to the specific tools and sequences unique to the company's established processes. On the other hand, an experienced engineer may know the methodology intimately but may lack the time to explore the documentation and capabilities of a frequently updated tool.

Both these cases highlight a significant and ever-increasing gap between the growing capability of computational tools and the engineering judgment required to use them effectively [24]. The engineer's value lies in understanding what the analysis must achieve and whether the inputs, method, and results are coherent and correctly interpreted. However, in practice, when a design change needs to be evaluated, a significant amount of time is spent on nonengineering tasks such as debugging data format mismatches between modules.

2.2 Approaches to Engineering Design Automation and Their Limitations. The challenge described at the case company is not unique. Engineering design problems are ill-defined and context-dependent, often described as *wicked problems* [25], where the problem becomes fully clear only as the solution emerges [26]. Gericke and Eckert [1] further note that design methods are embedded in ecosystems of representations, tools, and purposes. Any automation must therefore accommodate variability, incomplete

information, and evolving understanding. In aerospace, this is a known bottleneck both in design [27] and manufacturing [28]. Several families of computational support have been developed to address it, each with characteristic limitations.

Knowledge-based engineering (KBE) systems encode expert rules and geometric primitives to automate routine design tasks, but require ontologies [29] and design primitives to be defined in advance [12]. KBE developers and the engineers performing the analysis are typically not the same people [30], and the specialized knowledge to build and maintain them limits their use to high-performing teams on early concepts where the design space is sufficiently constrained [31]. Within such constraints, they support multidisciplinary optimization (MDO) and design space exploration [32]. For detailed design and certification phases, where data formats and methods constantly evolve, KBE approaches are rarely viable [33]. Even with fully explicit logic, the engineers using them describe the resulting rule chains as black boxes whose complexity exceeds what a practitioner can trace [13,33].

Process integration and design optimization platforms and scripted pipelines connect heterogeneous solvers through graphical workflow editors (e.g., modeFRONTIER, ANSYS WORKBENCH) or sequential scripts, enabling parametric and MDO studies [34,35]. Both rely on deterministic interfaces specified in advance: a format change in an OEM delivery, or a parameter not foreseen at design time, produces errors. Similarly to KBE, these workflows are seen as black boxes by other engineers unrelated to their development.

Machine learning approaches address parts of the same challenge through models that learn from data rather than encoded rules. Deep generative models such as generative adversarial networks (GANs), variational autoencoders (VAEs), and diffusion [36] produce new design candidates or surrogate predictions [37]. Early uses of LLMs followed the same pattern, as tools within an otherwise scripted process, for example, for textual product descriptions [38] or basic design synthesis [39,40]. In all these cases, the product performance computation happens within the model and the output is a candidate or a prediction. The orchestration of certified analysis processes, where documented procedures must be followed, tools called in defined sequences, and inputs and outputs traced through the quality chain [41,42], lies outside their scope.

LLM-based agents represent a different role for the same technology: instead of producing a design output, they orchestrate the use of existing tools and follow documented processes. Massoudi and Fuge [43] demonstrate this for conceptual systems engineering tasks (requirements analysis, functional decomposition, and architecture trade-offs), but their scope is the early design phase, where outputs are conceptual and integration with in-house tools and methods is not emphasized. Mustapha [44] characterizes the shift from tool to orchestrator as a paradigm change from deterministic to stochastic, and notes the adaptability or *malleability* of these systems, a concept close to ductility. These works signal the direction, but stop short of the detailed design and certification setting addressed in this article. Empirical studies on engineering change propagation show that even minor interface shifts can trigger disproportionate rework [5,45]. This is the brittleness that the present work targets. The central insight from the literature is summarized as the following core principle that is used to guide the development of the method:

Central Orchestration Principle: Effective engineering orchestration requires an intermediary capable of accommodating variability in data, tools, and practices, rather than relying on rigid, predetermined interfaces.

From the survey of literature, a set of requirements for engineering automation can be extracted; Table 1 summarizes 12 requirements, R1–R12 with brief descriptions and supporting references.

3 Large Language Model Agents: Technical Foundations

Building on the central orchestration principle articulated in Sec. 2.2, this section examines how LLMs within agents provide such an adaptive intermediary. So far, we have avoided defining the term Agent. Historically in AI, the term has been defined along different lines: by interaction style, as a *personal assistant* that the user collaborates with through indirect management rather than direct manipulation [56,57]; and by a set of properties, namely autonomy, social ability, reactivity, and pro-activeness [58]. In the context of LLMs, as the technology matures, the term continues to evolve reflecting different characteristics and challenges. Earlier work defined LLM agents as a *[…] task oriented combination of LLM models and the infrastructure to access tools, retrieve context information, and store memory of its interactions.* [59]. While every LLM frontier lab and agentic framework provider has a variation of this definition, this article uses the term that is gaining traction lately, which involves a description of what it does rather than what it is: *An LLM agent runs tools in a loop to achieve a goal* [60]. This action-oriented view is close to the historical *softbot* (software robot), an agent that senses and acts upon a software environment by issuing commands and interpreting their feedback, for example, UNIX shell commands [61,62]. In this article, we distinguish between the *agentic framework*, the software toolkit (or harness) that provides the inference loop and tool calling infrastructure (e.g., Claude Code and Pydantic-AI [63]), and the *agentic application*, the configured agentic framework that defines a specific LLM, system prompt, and tool connections for a given engineering context. Section 4 describes how an agentic application is developed, executed, and evaluated.

We focus solely on external, off-the-shelf large language models, recognizing that most engineering organizations, especially medium and small firms, lack the resources to develop or fine-tune such models in-house. The context window is central to this capability: since externally developed models have no built-in knowledge of an organization's internal tools and processes, the information available in the context window determines what the model can draw upon when generating each token.

Only a few years ago, LLMs were pejoratively referred to as "stochastic parrots" [64]. Today, LLMs offer a flexible inference mechanism that can interpret context, reconcile heterogeneous representations, and mediate between evolving tools and engineering intent [44]. To convey to the mechanical engineering community how rapidly this shift has occurred, the following subsections introduce the fundamentals of inference and response generation (Sec. 3.1), together with two capabilities that have matured rapidly: structured reasoning modes (Sec. 3.2), often referred to as *thinking* or *deliberate reasoning*, and tool calling (Sec. 3.3), which enables controlled interactions with deterministic external software.

3.1 Response Generation. A transformer-based LLM [65] sequentially generates a token at a time through an autoregressive process, see Fig. 1. At each step, the model takes the entire context window: the system prompt, the user message, and all tokens generated so far; and produces a probability distribution over the vocabulary of possible next tokens. A sampling strategy selects one token from this distribution, and the selected token is appended to the context window for the next iteration. This loop continues until a stop token is sampled, at which point the accumulated sequence of tokens is returned as the response. Other recent architectural language models, namely diffusion [66], substitute the autoregressive sequential token generation with a series of refinement steps. Both the architecture and the process are fundamentally nondeterministic: the same input can produce different responses depending on the sampling strategy and its stochastic parameters.

3.2 Thinking Mode. The transformer autoregressive nature of token generation introduces a limitation: once the model commits to an initial direction, subsequent tokens are conditioned on that choice, making it difficult to reverse course [67]. This can lead to fixations or hallucinations [68]. Thinking mode is a

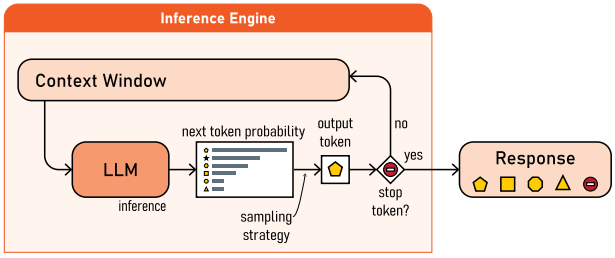

Fig. 1 Schematic of the LLM inference loop. The model takes the full context window as input, produces a probability distribution over the next token, samples one token according to a sampling strategy, appends it to the context window, and repeats until a stop token (<EOS>) is generated. The resulting sequence of tokens forms the response.

mitigation strategy developed during model post-training using reinforcement learning [69,70], and first experienced by the general public with OpenAI's o1 model in Sep. 2024 [71] and subsequent competitor models. Practically, the so-called thinking mode provides the model with a scratchpad in which self-critique is rewarded. Specific token sequences allow the model to reconsider its initial direction, explore alternative interpretations, and select the most appropriate response before committing to a final answer [72], see Fig. 2. For engineering orchestration, thinking mode is important because it allows the model to reason at inference time about ambiguous or unexpected situations before acting on them, increasing the probability of a successful final answer [70].

3.3 Tool Calling. Tool calling enables the model to ground its responses on external data and computations, beyond what is encoded in its training weights or present in the context window [73]. Models are trained to recognize when a question requires external information and to generate a structured output (a function name and arguments) instead of a direct answer. The model does not execute the tool itself. In practice, it performs a classification task, selecting which of the available tools is best suited for the current step and inferring the appropriate arguments from the context window. Figure 3 illustrates this with an engineering example.

Wei et al. [74] identify three styles of tool integration in agentic systems: (a) in-context, where the model reads documentation and generates calls zero-shot; (b) post-training, where the tool-use strategies are fine-tuned into model weights, and (c) orchestration-based where a planning layer coordinates multitool workflows. The present work uses the first and third; the agent reads tool documentation to learn about unfamiliar application programming interfaces

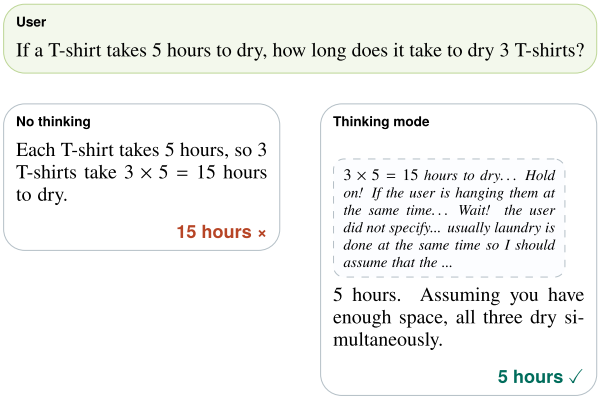

Fig. 2 Comparison of LLM responses with and without thinking mode enabled. The thinking scratchpad (dashed box) allows the model to self-critique and reconsider the autoregressive initial response before committing to a final answer.

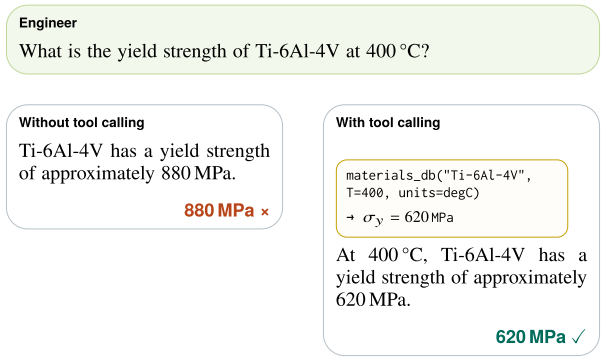

Fig. 3 Comparison of LLM responses with and without tool calling. Without tools, the model relies on parametric knowledge from training weights, returning a room-temperature value. With tool calling, the model queries an external materials database for temperature-dependent data, grounding its response on verified external information.

(APIs) and then plans multistep workflows across multiple tools. Post-training integration is deliberately excluded because engineering tools evolve and must remain independently verified and versioned. Fine-tuning the model to internalize the tool signature or APIs is, as mentioned earlier, possible but unrealistic in practice for most engineering companies.

3.4 Orchestration. These three mechanisms, inference, thinking mode, and tool calling, establish a two-layer architecture. The LLM selects actions and adapts to context; the external tools execute deterministically. Yao et al. [75] formalized this interleaving of reasoning traces and tool actions in the ReAct framework, showing that the model decides at each step whether to reason further, act on the environment, or return a final answer. This pattern has been demonstrated beyond engineering: Shen et al. [76] showed that an LLM can act as a controller that plans tasks, selects specialized AI models, and orchestrates their execution across vision, speech, and language domains.

Thinking and tool calling capabilities are prerequisites for orchestration. Agentic reasoning literature distinguishes two forms of scaling at inference time [74]: *scaling test-time computation*, where the model reasons longer within its context window (mapping to thinking) and *scaling test-time interaction*, where the model interacts with external tools and receives feedback from the environment.

3.5 Large Language Model Agents in Engineering Practice. Developments in the application of LLMs also come from commercial software vendors, who have begun embedding LLM-based assistants within their products. Simulation platforms such as ANSYS [77] or COMSOL [78] now offer interfaces that support engineer interactivity within the graphical user interface (GUI).

These assistants are, however, tightly coupled to their host application. They have access to the model being edited but cannot access external design practice repositories, in-house tools, or data from other stages of the analysis process. This closed-ecosystem constraint limits their applicability to the engineering orchestration problem described in Sec. 2.1, where the challenge is to bridge across multiple tools and data sources. Emerging service vendors such as COSMON [79] are developing agentic interfaces that connect LLMs to commercial computer-aided design (CAD) and simulation software. These solutions are interesting for a no-code solution. However, due to the requirements imposed on inspectability (R1), reproducibility (R2), and traceability and auditability (R4), these solutions are discarded as the

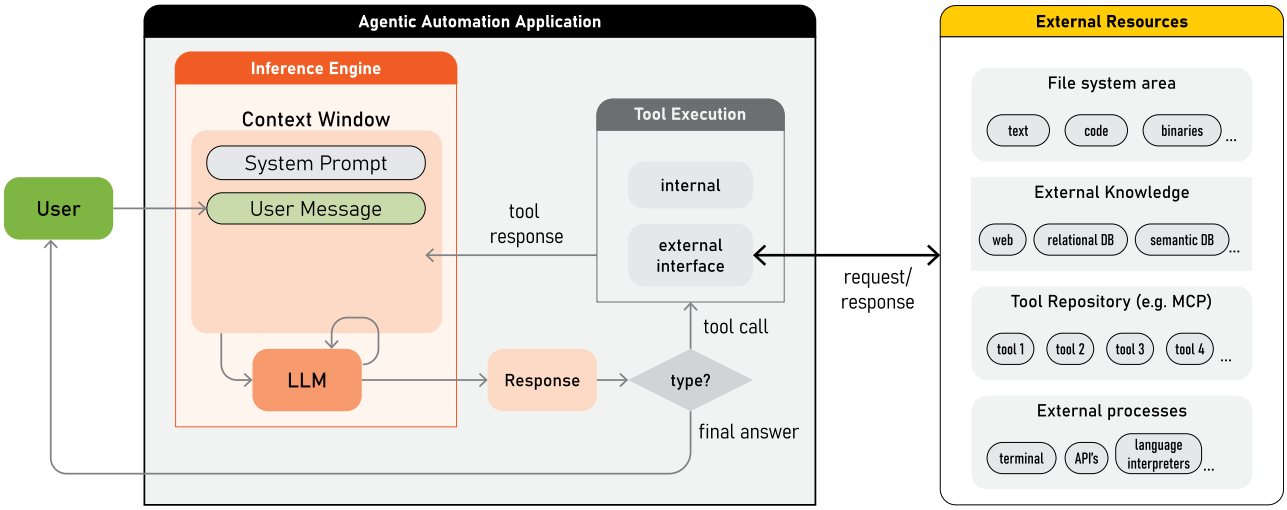

Fig. 4 Architecture of an agentic application. The inference engine processes user messages through the LLM within a context window, generating either final answers or tool calls. Tool execution is handled internally or through external interfaces that connect to external resources including file systems, knowledge bases, tool repositories, and system interfaces.

companies do not have direct access to adapting and therefore adding another potential layer of brittleness.

To address this gap, this article argues that a minimal and generic agentic application shall be developed at each company.

4 The DUCTILE Approach to Agentic Orchestration

This section addresses the gap identified in the previous sections by introducing the DUCTILE approach to agentic orchestration, whose core concept is to separate interpretation from computation. As argued in Sec. 2.2, brittleness arises when adaptation is encoded in fixed rules. DUCTILE addresses this by assigning interpretation and orchestration to an LLM and delegating all calculations to verified tools, improving robustness to interface and input changes without relaxing engineering rigor. This separation also supports traceability and certification compliance, which are necessary for the aerospace context. It is aligned with the requirements R1–R12 summarized in Table 1.

The following subsections describe how the approach is implemented. The description is divided into three parts: developing the agentic implementation (Sec. 4.1), executing the agentic analysis (Sec. 4.2), and evaluating the agentic performance (Sec. 4.3). Figure 4 shows the architecture of the agentic application used to execute the analysis.

4.1 Developing the Agentic Implementation.

In the development stage, the agentic application is established so that orchestration can be observed, reproduced, and integrated with existing engineering tools. In line with the requirements (Table 1), the agentic application implementation is shaped by two considerations: (i) the *character* of the agentic framework used to host the agent (addresses R1, R2, and R9) and (ii) the *configuration* of its core connections to models, prompts, and tools (addresses R2–R5 and R8).

4.1.1 Framework Character. The agentic framework exhibits inspectable and reproducible behavior, transparent configuration with minimal hidden state, and low coupling to the analysis stack. In practice, this corresponds to open-source, appropriately licensed solutions that surface prompts, traces, and file operations to the engineer, and that connect to external documents, tools, and analysis codes with minimal glue (addresses R1, R2, and R9).

4.1.2 Configuration Scope. Within such a framework, an agentic application is organized around three connective elements as shown in Fig. 4: *(a) the LLM model connection*, *(b) the system prompt*, and *(c) the tool interfaces*. The model connection covers the chosen model family and deployment mode (on-premises or API), together with version pinning and environment capture so that runs are attributable across iterations (addresses R2 and R5). The system prompt provides stable interaction conventions and output contracts, while domain procedures and method rules are provided *at run time* through documents and skills rather than embedded as static instructions; this keeps behavior auditable and portable across projects (addresses R1 and R9).

Selecting and Connecting to an LLM. Model choice for engineering analysis is typically shaped by a three-way trade-off between capability profile, computational cost, and data-governance constraints. Capability is commonly profiled along three dimensions: *tool use* (selection and invocation of external tools), as reflected in τ-bench [80]; *agentic reasoning* (multistep planning and recovery on error), as reflected in AgentBench [51]; and *code generation* quality, e.g., SWE-bench [81]. The choice of on-premises versus externally hosted model deployment further conditions feasibility and governance. On-premises execution offers full control over data flows, while externally hosted models reduce infrastructure burden and broaden immediate model availability. If an externally hosted solution is chosen, it shall be bounded by confidentiality requirements, export-control rules, and applicable data-protection frameworks (e.g., General Data Protection Regulation (GDPR)). For exploratory or nonsensitive workloads, however, external providers often function as a practical starting point. Regardless of the choice, model and version pinning enable run-to-run attribution (addresses R2 and R5).

System Prompt. Model behavior is influenced by the system prompt [82], with design that can evolve iteratively or via systematic methods [82,83]. Guidance from major providers (OpenAI [84], Anthropic [85], Google [86], Meta [87], and Mistral [88]) offers different advice, but has common sections that are always recommended to include:

- *Goal*: purpose and scope of responsibility.
- *Style*: tone and interaction conventions for communicating with the user.
- *Available tools*: typically injected automatically by the framework.
- *Reasoning strategy*: approach to task decomposition, depth of diagnosis, and information exhaustiveness.
- *Output format*: a verbal schema or a structured contract (e.g., a final tool call).

Within the present approach, the prompt is kept task-agnostic (addresses R1 and R9). Company-specific procedures, methods, and design practices are not encoded statically; instead, they are retrieved at run time from knowledge resources or disclosed progressively via skills [89]. This keeps behavior auditable and portable, and allows the user to state the task (e.g., *perform task X using method Y, revision Z*) while the agent assembles the relevant organizational information (addresses R1, R9, and R12).

Configuring Tool Access. Tool access can be provided internally or via external resources (Fig. 4). The approach favors connecting to external resources wherever feasible to minimize framework complexity and maximize reusability across agent instances (addresses R9). Interfaces expose deterministic, versioned capabilities with strict I/O contracts (paths, formats, and units) and controlled write locations; generated scripts and invocation logs (arguments and stdout/stderr) are persisted alongside inputs and outputs to preserve traceability and support inspection (addresses R3 and R4).

4.2 Executing the Agentic Analysis. Execution proceeds as an alternation between stochastic planning and deterministic tool use. The engineer provides the design context, inputs, and task objective; the application incorporates these into the context window. The inference process described in Sec. 3 proposes the next step toward the goal. Here, the stochasticity inherent to the model is an advantage: the model adapts inputs and company knowledge to the particular design context and goals (see Fig. 4). Once the model proposes a tool call, the agentic framework executes it deterministically and appends the result to the context window. The agent repeats this cycle, calling successive tools until the task objective is met (addresses R3, R4, and R7).

This loop connects the agent to the same tools an engineer would execute manually or hardcoded in a scripted pipeline, but the orchestration logic is no longer hardcoded. The LLM translates a *natural language defined* process description from a document into a sequence of tool invocations, selecting each step based on the task objective, intermediate results, and available tools (addresses R1 and R9). This adaptability is reported to the engineer in the final response and remains fully visible in the model traces and generated artifacts.

At any time in the parallel or sequential call of tools iterations, the agent may return the control to the user [75]. The decision to finish the iteration is made by the LLM based on its training behaviors and the content of the context window. A few examples of when the LLM can choose (via the likelihood of the next token and subsequent generation of a response) to finish its turn are:

- –When the request of the user can be answered directly, without the need to call tools.
- –When the model misses critical information to carry out the task.
- –When the tool calls have provided enough information to complete the task.
- –When a roadblock in the execution plan stops the model from getting closer to the goal (e.g., permission access, and tool failure) and reasonable efforts have been made to circumvent those limitations within the independence scope defined in the context window.

4.2.1 Final Response and User Intervention. When stopping, the agent may summarize previous activities in the agent's turn. The agentic application parses the results and then finalizes the process by returning the last message to the user. Current agentic applications stream to the engineer directly the intermediate tool calls and thinking steps to follow the task progress. This allows for engineers to stop the agentic execution if it is deviating from the goal or stalling with an activity (addresses R6).

4.2.2 Traceability and Observability. Messages, tool calls, and tool results at minimum shall be sent for recording to an observability platform via standard payloads (e.g., OpenTelemetry). If a company external service is used, the same data sharing considerations shall be evaluated. Generated code is stored next to inputs/outputs to preserve traceability (addresses R3 and R4). Where available, telemetry captures call latency and token/cost for model steps, supporting diagnosis, evaluation, and change control over time (addresses R8, R10, and R11). The resulting traces provide the evidence required for do/check/approve review and for the evaluation described in the next section.

4.3 Evaluating the Agentic Application Performance. Method adoption in engineering practice depends on evidence that the method behaves as expected, with a high degree of confidence [90]. Furthermore, the system needs to be validated like any in-house engineering software with rigor proportional to the risk of the activity performed [91,92]. However, agentic systems introduce challenges beyond those of traditional software.

First, the context window evolves across tool calls and intermediate reasoning steps, making the internal state difficult to inspect and debug. Observability platforms that trace each model invocation, tool call, and token exchange, such as Logfire [49], are essential for diagnosing failures (addresses R8). Second, outputs can vary structurally across identical inputs and yet can be equally valid. Because the DUCTILE approach intentionally uses flexibility to adapt to diverse engineering contexts, suppressing variability is not the goal. Instead, evaluation is performed against curated datasets of representative problems, including edge cases, similar to the held-out test sets used to assess generalization in machine learning [93]. These datasets verify that the agentic application produces correct and compliant outputs across the expected range of inputs, independent of the particular response path taken (addresses R10).

Unlike machine learning datasets containing only input $\mathbf{x}$ and expected output $\mathbf{y}$, each evaluation case must specify:

(1) The *design environment*: the system prompt, the engineer query, the executable tools, and reference documentation.
(2) The *acceptance criteria*: expected outcomes expressed as quantitative tolerances, qualitative descriptions, or both.
(3) The *evaluation method*: a deterministic check against reference values, a human expert review, or an LLM-as-a-judge assessment.

The evaluation must cover both standard interactions and edge cases, both measuring quantifiable expected engineering values and agentic behavior under high uncertainty design conditions. For example, the expectation to ask for clarification on underconstrained problems, or to explicitly report the intermediate assumptions made to engineers. Expected answers shall be developed and maintained by domain experts; in practice, these use cases grow to the order of hundreds of scenarios. Because model responses are probabilistic, each use case must be repeated to establish confidence in the agent's ability. DUCTILE adopts the pass^k metric [52], defined as whether the agent correctly performs the same task in k independent repetitions. The choice of k depends on the evaluation scenario and the criticality of the task.

A useful analogy comes from material qualification in aerospace structures, where allowable properties are defined probabilistically as x/y: the xth population percentile at y% confidence. Three standard levels exist: S, B, and A-basis [53]. Treating each evaluation case as an independent Bernoulli pass/fail event, the Clopper–Pearson exact bound [94] for k successes in k trials yields the minimum number of consecutive passes required at each confidence level, as shown in Table 2.

The basis-level k values serve as a reference for comparison with established engineering practice. Note that the resulting basis values therefore describe confidence in the agent's behavior for the specific configuration tested and do not extrapolate to other prompts, tools, documents, or scenarios.

In practice, DUCTILE delegates execution but not responsibility. The engineer retains accountability for reviewing outputs, and existing engineering quality processes remain in place. Since

Table 2 Minimum consecutive passes k for the pass^k metric at 95% confidence, derived from the Clopper–Pearson bound $p_{\text{lower}} = \alpha^{1/k}$ with $\alpha = 0.05$

	S-basis	B-basis	A-basis
Pass probability p	≥ 0.50	≥ 0.90	≥ 0.99
Required k	5	29	299
DUCTILE recommended values			
Development		$k = 3$	
Deployment		$k \geq 10$	

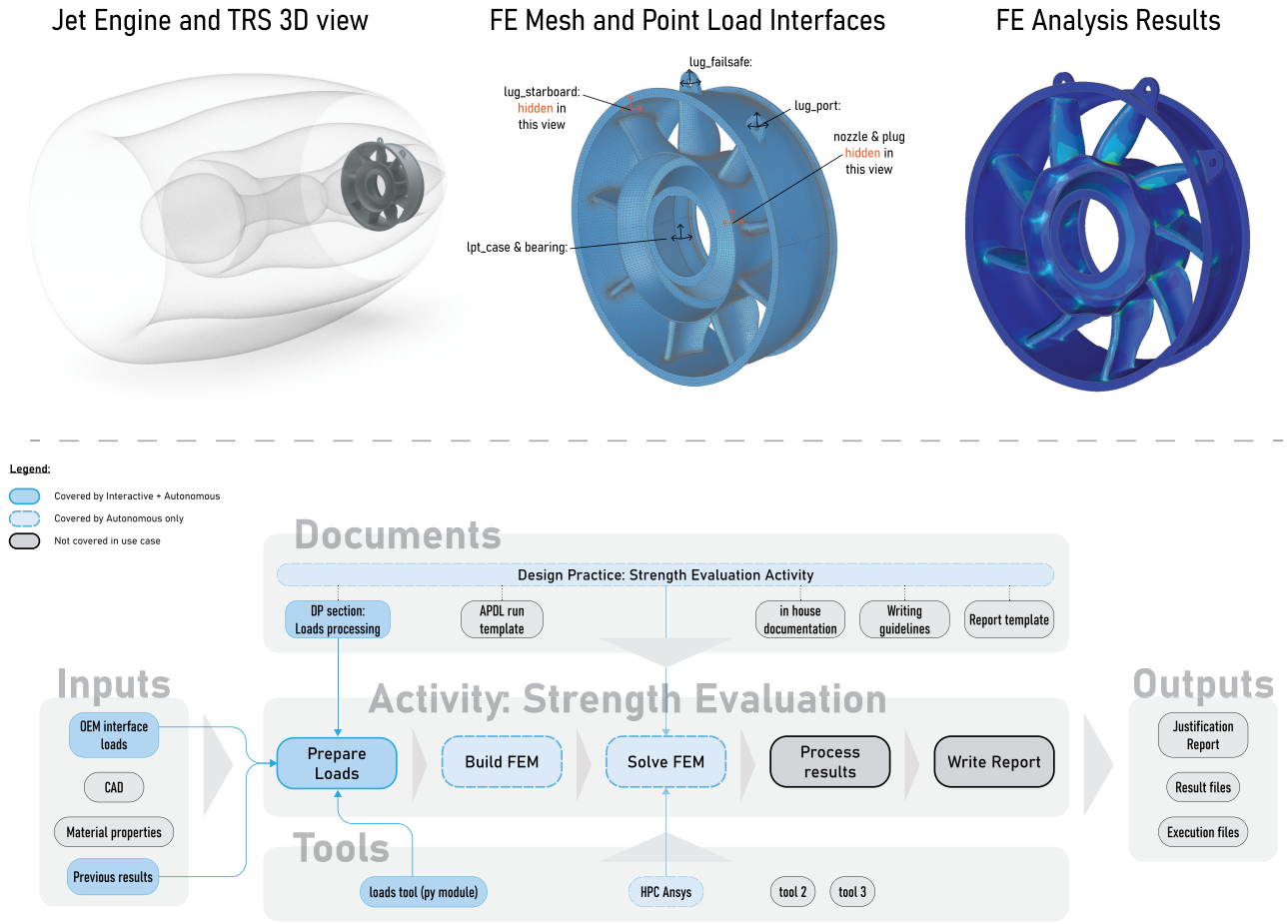

Fig. 5 Top: views of the component in the physical domain. Bottom: activity view of the TRS strength evaluation. Only a subset of the inputs, documents, and tools are shown for visual clarity. The three first steps: Prepare Loads, Build FEM and Solve FEM (blue in color) covers the autonomus evaluation, while the evaluation environment with the agent–engineer interaction covers only the first step.

each repetition consumes computational resources and cost, the DUCTILE approach recommends $k = 3$ during development and $k \geq 10$ for deployment, increasing k as the criticality or autonomy of the agent grows.

Evaluations must be rerun whenever the system prompt, tools, documentation, or underlying model changes. At the scale of hundreds of cases repeated k times, human evaluation becomes unfeasible. Automated assessment via deterministic checks or LLM-as-a-judge is therefore essential, with trust metrics analogous to inter-rater reliability established for the latter.

5 Industrial Application of Agentic Load Processing on a Realistic Case

The DUCTILE approach described in Sec. 4 is applied to a structural strength evaluation on a realistic case at the case company. The component in this case is a turbine rear structure (TRS), shown in Fig. 5. While the case company routinely performs static strength analyses on the TRS as part of the certification and continuous product development activities, the jet engine OEM generally owns the global stiffness and loads models at the whole-engine integration level, and delegates structural compliance checks to component partners. A design practice document at the case company defines the steps, methods, and expected outputs for this type of analysis.*The Use Case Description.*

The engineering use case concerns a new structural evaluation triggered by a design change in a neighboring engine component. The geometry, finite element mesh, and analysis methods remain unchanged; only the loading input from the OEM is new. A new, company-maintained PYTHON-based loads processing tool is used together with the established design practice.

The new OEM delivery introduces four deviations compared to the previous analysis. They have been selected as realistic representatives of input and method deviations that consume engineering hours, and that the case company recognizes as a major

Table 3 Deviations in the new analysis scenario

Deviation	What changed	Impact on automation
File format	YAML instead of JSON	Requires new parser
Unit system	Imperial instead of SI	Silent error
Node naming	`right/left` instead of `port/starboard`	Finite element model (FEM) run fails downstream
Method change	1.04 factor on Fx forces	Silent error

Note: The first three are input format changes; the fourth is an OEM-communicated correction specified in the task description.

contributor to brittleness in day-to-day analysis cycles. Three of these deviations are input format changes, a common example of variability when data crosses organizational boundaries; the fourth is a correction method communicated by the OEM requiring a 1.04 factor on all Fx forces, separate from the raw data. In a traditional scripted pipeline, each deviation would require manual intervention or code modification. Table 3 summarizes the four deviations.

The remainder of this section describes the configuration as shown in Sec. 5.1, the execution with two engineers (Sec. 5.2), and the systematic evaluation of the agent's performance, in Sec. 5.3.

5.1 Case Setup. Following Sec. 4.1, the agentic implementation was built using Claude Code [95], a terminal-based agentic framework. An extensive description of the implementation is available in the Supplemental Materials on the ASME Digital Collection. Claude Code operates within the engineer's existing integrated development environment (IDE) and file system. Since the engineers at the case company already work with scripts, input files, and terminal commands, the agentic application fits within the existing workflow rather than imposing a new one. The engineer can inspect every file the agent reads or writes (addresses R1 and R2). To maximize transparency and user experience, engineering product data information was placed in the working file system during the experiments. The design practice was served via an Model Context Protocol (MCP) document server, giving the agent version-controlled access to the methodology. The loads processing tool was distributed as a PYTHON package via PyPI, with API documentation published in both human-readable and LLM-consumable formats (`llms.txt`) on the corresponding GitHub Pages.

Two main evaluation environments were set up to gather performance measurements. Due to the limited time availability of engineers, the first environment was an interactive agent-engineer set up covering only the *Prepare Loads* step of Fig. 5 highlighted in blue. In contrast, a second automated environment was set up covering the same task description but requesting the agent to rerun the Finite element model (FEM) with the new loads. This wider scope triggered the execution of the first three steps defined in Fig. 5.

The task description was prepared prior to agent testing, targeting both an experienced engineer or an inexperienced graduate joining the company; no refinements to the prompt or integration were required after initial testing. The design practice document captures the logic of the activities to be performed. Figure 6 shows the workflow derived by the authors from this document. This flowchart was not available to the agent. By reading the document and inspecting the inputs, the agent was expected to infer and implement a comparable sequence of activities (addresses R1 and R12).

5.2 Execution. For the interactive environment, two engineers at the case company were assigned the same task with the same inputs and deviations described in Table 3. Both had access to the agent and were encouraged to interact with it and verify its outputs as they saw fit. The transcripts and output files are available in the Supplemental Material.

5.2.1 Engineer 1: Delegation. Engineer 1, familiar with agentic workflows, delegated the task to the agent and checked the outputs after the full activity was completed. Figure 7 summarizes the interaction.

The engineer pointed the agent to the task description and requested a plan. The agent read the task description .pdf file, fetched the design practice via the MCP document server, explored the working directory, and examined the previous analysis run and input files. During this reading phase, the agent identified all four deviations in its reasoning trace: the YAML format incompatible with the tool's JSON parser, the imperial unit system, the changed pilot node names, and the 1.04 correction factor specified in the task description. The agent surfaced these findings in a seven-step plan presented to the engineer, proposing a processing approach that addressed each deviation. The engineer approved the plan.

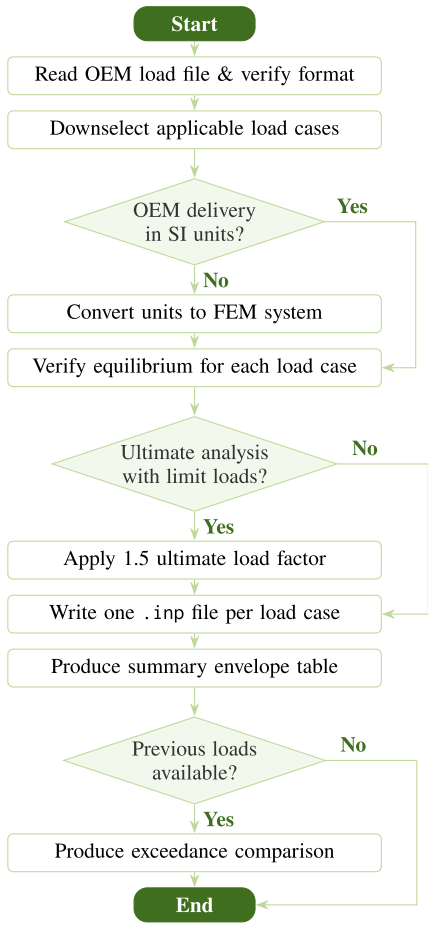

Fig. 6 Engineer-interpreted activity process flowchart of the natural language and unstructured description contained in the loads design practice document. NOTE: The agent must infer a similar workflow by reading the document; this flowchart is not available in the agent's context.

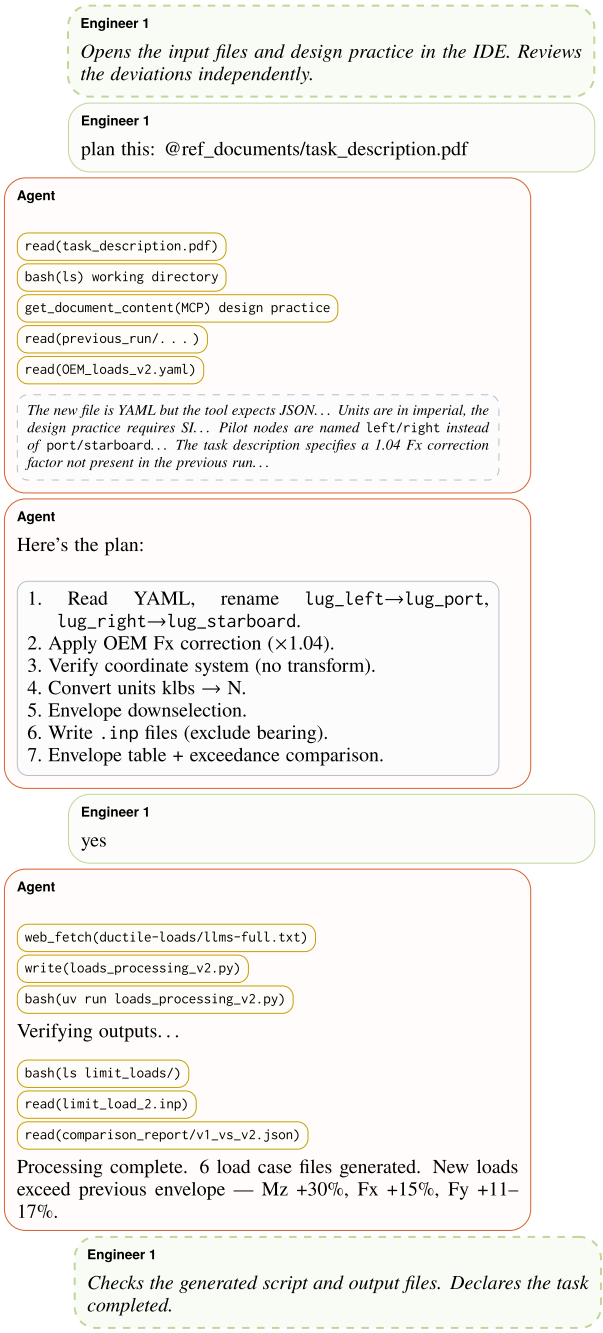

Fig. 7 Summary of engineer 1's interaction. Speech bubbles are labeled with the speaker (engineer 1 or agent). Rounded boxes within agent turns: tool calls. Dotted boxes: engineer actions outside the conversation. An expanded transcript is available in the Supplemental Material.

The agent then fetched the tool's API documentation, wrote a processing script, executed it, and verified the outputs: six `.inp` files (one per envelope-selected load case), an envelope summary table, and an exceedance comparison against the previous analysis. The engineer inspected the generated script and output files independently and declared the task completed.

5.2.2 Engineer 2: Incremental Discovery. A second engineer, unfamiliar with agentic workflows, took the opposite approach. Rather than delegating the full task, this engineer requested one step at a time: first reading the inputs, then processing a single load case, and then checking the output before proceeding. Each intermediate result was inspected before the next step was requested. Through this incremental process, the engineer discovered the task, learned the tool's behavior, and built confidence in the agent gradually. The agent adapted to this step-by-step supervision without difficulty. The final output was identical to engineer 1's result: the same load case files, envelope table, and exceedance comparison. The same four deviations were handled through a different supervision path.

5.2.3 Autonomous Environment. In this environment, a predefined user message was provided to the agent requesting to perform the three steps independently. In this case, additional documentation and tools were provided to test the performance of a longer-horizon task and wider catalog of tools, including an MCP mock connection to the high-performance computing (HPC) server where ANSYS is being run. The agent was on this occasion defined and executed using Pydantic-AI [63]. In addition, a negative test case variant was run in the same environment to verify that the agent halts on misaligned inputs when the engineer or methodology document requests it. The predefined user message was modified to instruct the agent to stop and flag the engineer if the inputs did not align with the design practice, rather than adapt to deviations as in the success path. With the same four deviations present in the OEM delivery, the agent inspected the inputs, identified all four discrepancies, and stopped before invoking the loads tool, returning a structured summary of the findings for engineer review (addresses R6).

5.3 Evaluation. Following the evaluation approach described in Sec. 4.3, the evaluation ground truth was defined by the company domain expert. The agent's performance on the interactive environment was assessed based on the agent's ability to correctly identify the input deviations in Table 3 and generate the correct load files and values. In the autonomous environment, the agent was run 10 independent times on the same scenario. The evaluation scripts and outputs are available in the Supplemental Material. This implementation is an example of the DUCTILE approach toward agentic evaluation.

The evaluation used two independent approaches. First and primarily, the full agentic log and results were manually checked by the domain expert in both the interactive and the repeated autonomous environment runs. Second, as part of an exploratory exercise, an LLM-as-a-judge [96] was used to test its suitability for these scenarios, which yielded a 100% inter-rater agreement with the expert. The negative test case for halting the agentic execution also fulfilled the pass$\hat{}$k = 10 criterion.

5.4 Failure Modes. The perfect agent performance reported in the previous section may wrongly lead to the conclusion that the implementation was a one-shot and no iteration took place. The agentic traces were manually reviewed via logs, and changes to the implementation and models were performed to arrive at the final configuration.

Three intrinsic agentic failure modes were observed during development and addressed before the final evaluation. *Implementation failures* were found when misconfigured Pydantic-AI agents or MCP servers were implemented: tools not exposed to the model, or tools whose descriptions were incorrect, causing the agent to fetch the wrong document [54,55]. *Context failures* were found when passing information that degrades performance, including irrelevant or contradictory content. For example, a top-level document specified initially one HPC flag while a nested reference specified another. Such contradictions are expected in companies whose documents are curated by many authors over decades. *Model failures* are the residual LLM-level errors that persist in the absence of the previous two. Comprehensive surveys catalog these extensively [97,98]; the present case encountered hallucination [68], cascading reasoning errors [67], and degraded long-context retrieval [99]. The solution to those failures was to increase

the size (and computational cost) of the models offered by the same provider: from Haiku to Sonnet to Opus size of model.

Other failure modes are extrinsic to the agent, including over-reliance, supervision fatigue, and skill atrophy. They are widely documented in the human–automation and HCI literature [100,101]. Of these, over-reliance was the one we observed in ourselves during development. The temptation to accept the agent's output as correct grew stronger once a second reviewer agent had also validated it, with the conclusion read before the trace was scrutinized. We do not offer a validated remedy. A candidate worth investigating is the deliberate injection of planted errors into agent responses, to force the engineer to actively look for mistakes. This is analogous to threat image projection in airport baggage screening, where fictitious threat items are inserted into the screener's image stream to counter the low-prevalence effect on detection performance [102,103].

6 Discussion

6.1 Consequences for Practice. The application of agentic support exposed the gaps in the existing design practice documentation at the case company. Experienced engineers bridge ambiguities in written procedures through tacit knowledge. Hsieh et al. [54] demonstrate that well-written tool documentation enables LLMs to use unfamiliar tools zero-shot, without fine-tuning. Yuan et al. [55] show that concise, structured descriptions outperform verbose ones. Investing in documentation quality serves both human engineers and agents. However, the development of the use case showed how those tacit activities were missing from the company design practices, and had to be added while the agentic application development and evaluations were being codeveloped. We expect agentic engineering applications to be an additional motivation for keeping updated and comprehensive engineering documentation, which could have the secondary effect of reducing the organizational vulnerability that arises when critical knowledge is concentrated in individuals.

The DUCTILE agentic approach uses in-context reasoning exclusively (no fine-tuning nor persistent memory). This is a deliberate choice because, together with evaluations, it allows the agentic application to avoid model lock-ins, and to update the model as smaller-footprint, cheaper, or more capable models are released without effort and robustly.

This contrasts with post-training approaches where tool-use strategies are embedded in model weights [74], and with rule-based systems that, despite formally explicit logic, become practical black boxes as rule chain complexity exceeds what practitioners can trace (Sec. 2.2). Both engineers in the case study confirmed that the agent's actions were easy to follow and inspect, showing promise for wider adoption in practice with minimal maintenance burden.

Despite the case for a dataset-free configuration pipeline, more complex scenarios may call for company- or task-specific models: cases where agentic performance (tested via evaluations) is not robust enough despite optimizing the context window, or where latency must be minimized and context length reduced. In such scenarios, the parameters of an open-weights model can be fine-tuned to inject company knowledge, and this is where a training dataset becomes necessary. Datasets in engineering, however, remain scarce [104]. Ironically, adopting the agentic approach itself generates the raw material. Every engineer-agent session automatically produces a conversation log capturing design intent and rationale, even when the agent performs poorly. This history can be passed to a separate LLM pipeline to generate positive examples of expected behavior, effectively producing a training dataset without dedicated engineering effort.

6.2 Risks and Unintended Consequences. The aerospace community warns that "overreliance on software tools can lead to a superficial understanding of the underlying physics" [24]. This concern applies to agents as well. A key distinction is needed, however, between dexterity and judgment. Dexterity with specific APIs, data formats, and tool interfaces is what the agent will absorb. Engineering judgment, knowing whether results make physical sense, the method is appropriate, or the output meets certification requirements, must remain with the engineer. In our experience developing aerospace products under intense project deadlines, the real risk is in the organizations stopping to recognize and allocate resources to develop the engineer's judgment. Engineer 2's step-by-step interaction pattern provides evidence that supervision and learning can coexist: this engineer built understanding of the task progressively while the agent handled the implementation details.

Gericke and Eckert [1] raise two cautions from the product development ecosystem perspective. First, a rebound effect: historically, tools that shortened development time led to more complex products requiring similar development time. Second, skill loss: designers may lose tacit skills previously developed through routine manual execution. Therefore, we call for a conscious implementation. The supervisory role must be deliberately designed to maintain engineering judgment [101]. In this high-stakes scenario where errors are rare, planted-error countermeasures from airport inspection domains may be worth borrowing [102,103].

It remains to be seen whether agentic support will enrich or exhaust the work of engineers. On one hand, many design and AI researchers, including Cross [105], argue that these systems that interactively support humans should be designed in a way that is *cognitively comfortable* for them. If implemented in a human-centered way [106], the mundane tasks could be absorbed by the agent, and the most rewarding aspects of engineering work preserved [107]. However, we are starting to observe in software engineering how these agents are not delivering to this promise. Instead, the joy of solving the little problems and the social nature of the work are reduced, being substituted by a hollow supervision role that exhausts software engineers via increasing the intensity of their work [108]. Shneiderman [109] argues that this outcome is not inevitable: systems designed as "supertools" that amplify human capability, rather than autonomous agents requiring supervision, can preserve both productivity and satisfaction. The choice of implementation, however, belongs to the engineering organizations. Ultimately, they must resist the temptation to optimize solely for short-term efficiency, and instead design an automation support that enhances productivity while improving the quality of work and developing engineering judgment.

6.3 An Engineering Automation Paradigm Shift. Engineering automation has historically required deterministic logic, but LLM-based agents violate this expectation. They are probabilistic, their reasoning is not formally verifiable, and they can produce incorrect outputs. Based on our previous work [20] and interactions at the case company, we expect their acceptance for certification activities to be contested.

This contest echoes a recurring pattern in AI where hand-crafted, ontology-driven methods are eventually displaced by statistical ones once performance is demonstrated at scale. Neumann [110] showed in 1956 that reliable systems can be built from unreliable components through redundancy. Rosenblatt himself [111] acknowledged the limitations of probabilistic methods: *the penalty that we pay for the use of statistical principles in the design of the system is a probability that we may get a wrong response in any particular case*. Nevertheless, Minsky and Papert [112] dismissed neural networks in 1969. It was not until sufficient computational power and dataset size became available that neural networks prevailed, as demonstrated by AlexNet [113]. Sutton [114] generalized this observation: methods that leverage computation consistently outperform methods that leverage human knowledge. In this framing, the probabilistic approach

that produces occasional errors and the flexibility that absorbs the design process variability are two sides of the same coin. One cannot exist without the other. We argue for the acceptance of such probabilistic agentic orchestration to be judged by the merit of the task evaluations (Sec. 4.3), and not against preconceived deterministic expectations.

6.4 Limitations. Several limitations constrain the generalizability of these findings. *Two engineers.* The experiment is illustrative; two participants' behaviors cannot be extrapolated to the engineering population. *Model dependence.* The results reflect Claude Opus 4.6 at the time of the study and may change with future versions and other models. *Document-centric approach.* Where engineering knowledge is predominantly tacit and not captured in accessible documents and tools, the agent has nothing to read and the approach does not apply. Additionally, realistic engineering knowledge repositories often include contradictory or complementary approaches in different documents. This scenario was not evaluated and is expected to decrease the agent performance. *LLM-as-a-judge validity.* At the evaluation scale proposed in Sec. 4.3 (hundreds of scenarios, repeated runs), manual qualitative review is unfeasible and the LLM-as-a-judge becomes the primary check. Its validity for engineering qualitative assessment has not been demonstrated. Using the same model family as agent and judge introduces a risk of self-enhancement bias [96]. *Text and data workflows only.* This work does not cover spatial tasks such as CAD geometry generation, finite element meshing, or graphical postprocessing. Image modality interpretation has not been explored. *Deviation coverage.* The four deviations evaluated represent frequent input format and method changes at the case company but do not cover every possible design scenario. The demonstrated task is itself a linear pipeline and does not cover iterative or branching activities. *Sensitivity to prompts.* Repeated evaluations with the same context window provide confidence on the agent's ability, but do not cover variations in the tool catalog [55] or semantic variations in documentation content or prompts [54,115], which are known to vary its performance. *Transferability and robustness of agents.* Applicability to other engineering domains, organizations, and task types is argued on structural grounds but not empirically validated beyond this case. The repeatability and performance on this use case do not imply general engineering agentic robustness. Engineering tasks and evaluation activities are very context-based. DUCTILE recommends specific evals for each activity before claiming robustness. *Generic-layer claim has limits.* The *task-agnostic prompt with runtime document retrieval* framing of DUCTILE for agentic support is valid within the task scope that current state-of-the-art models can successfully perform. However, there are design scenarios where task complexity, high throughput, latency sensitivity, or token-budget constraints require agents to depart from this ideal, at the expense of task specialization (R11). This is the spectrum from prompt preloading to dedicated subagents, at the cost of additional maintenance and change control.

6.5 Future Work. *An open dataset for agentic evaluation is necessary*. Organizations deploying agents in certified engineering need references for developing internal datasets that capture their specific tools, design practices, and quality requirements. Any change to the underlying model, prompts, tool version, or design practice can alter agent behavior and must be assessed before deployment. The research community, in turn, needs open engineering benchmarks. Efforts such as calls for curated design datasets [104] and EngiBench [116] have begun to address it for generative tasks, and similar calls have emerged for LLM evaluation in engineering [44]. Agentic engineering evaluations have additional, nuanced requirements: the framework must be executable and available to simultaneously apply domain knowledge, call tools correctly, interpret numerical outputs, and follow documented methods.

Industrial impact shall be measured. The two engineers in Sec. 5 interacted with the agent in fundamentally different ways. One delegating the full task, the other proceeding incrementally. A controlled within-subject study comparing engineer performance with and without agent support would formalize this observation and measure impact. Metrics like task accuracy, completion time, and confidence in the result across a larger sample of practicing engineers would reveal how engineering practices interact in realistic scenarios.

The trade-off between agentic performance and task specialization is not yet clear. Specialization through task-preloaded prompts, skills, dedicated subagents, or fine-tuned models can improve performance, and reshapes the balance between model size, latency, tokens consumed, context window usage, and the maintenance overhead of keeping each specialization up to date with the underlying tools and documentation. This direction moves away from the task-agnostic approach of DUCTILE, and future work will establish the point at which it becomes necessary.

7 Conclusion

The presented DUCTILE approach demonstrated the automation support separation via a constrained but realistic industrial design task at an aerospace manufacturer. The adaptive orchestration was performed by the agent, using the nondeterministic nature of the LLMs. The deterministic execution was performed by verified engineering tools. Engineers retain responsibility for supervision and judgment while the process is guided by design practices already available in companies. The approach fulfills the requirements identified in Table 1 without the need to maintain product datasets or company-specific LLMs.

In the use case, the agent handled four input deviations that would break traditional scripted automation. All four were resolved correctly across 10 independent evaluation runs on this scenario. The approach was also tested on two engineers with different supervision styles, one delegating the full task and one proceeding incrementally, and both reached the same correct result using the same agent configuration.

Traditional engineering automation is brittle because it encodes adaptation in deterministic rules bound to interfaces that, in practice, keep changing. When the ecosystem changes, or when people move, the automation support becomes a barrier and stops being an enabler. The DUCTILE agentic approach is designed to absorb that variability at the orchestration layer, and in this case study did so for a preprocessing mechanical analysis task. The same orchestration is expected to apply to postprocessing tasks and to other engineering analyses where in-house tools are combined with external tools and written procedures, with broader robustness to be established through further studies across engineering contexts. The intended outcome is that engineering time and cognitive effort are spent on the decisions that affect product performance, rather than on the data wrangling and workflow adaptation that currently consume them.

Acknowledgment

The authors thank the engineers at GKN Aerospace who participated in the experimental study, and particularly Najeem Muhammed for his valuable technical knowledge and expertise. The authors acknowledge the support from Dr. Faez Ahmed and value the fruitful discussions with all DeCoDE lab members at MIT.

Funding Data

- The research was partially funded by GKN Aerospace. The authors are grateful to the Barbro Osher Pro Suecia

Foundation, which provided financial support (Grant No. 90401181) for the first author's research period at MIT.

Conflict of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The agentic application implementation, evaluation results, and Supplemental Materials, including full session transcripts, are openly available[2] and permanently archived in Zenodo [117].

References

[1] Gericke, K., and Eckert, C., 2026, "Co-evolution of Tools and Methods in Product Development Ecosystems," *Co-evolution of Design Research and Design Practice*, K. Gericke, C. Eckert, V. Singh, and S. Venkataraman, eds., Springer Nature, Springer Nature Switzerland, pp. 6–19.

[2] Eckert, C., and Clarkson, J., 2005, "The Reality of Design," *Design Process Improvement*, P. J. Clarkson and C. Eckert, eds., Springer, London, pp. 1–29.

[3] Wynn, D. C., and Clarkson, P. J., 2021, "Improving the Engineering Design Process by Simulating Iteration Impact With ASM2.0," Res. Eng. Des., **32**(2), pp. 127–156.

[4] Eppinger, S. D., Whitney, D. E., Smith, R. P., and Gebala, D. A., 1994, "A Model-Based Method for Organizing Tasks in Product Development," Res. Eng. Des., **6**(1), pp. 1–13.

[5] Wynn, D. C., Caldwell, N. H. M., and Clarkson, P. J., 2014, "Predicting Change Propagation in Complex Design Workflows," ASME J. Mech. Des., **136**(8), p. 081009.

[6] Brahma, A., and Wynn, D. C., 2022, "Concepts of Change Propagation Analysis in Engineering Design," Res. Eng. Des., **34**(1), pp. 117–151.

[7] Liu, Y., Abulawi, Z., Garimidi, A., and Lim, D., 2025, "Automating Data-Driven Modeling and Analysis for Engineering Applications Using Large Language Model Agents," SSRN Preprint.

[8] Fernandes, A. A. A., Koehler, M., Konstantinou, N., Pankin, P., Paton, N. W., and Sakellariou, R., 2023, "Data Preparation: A Technological Perspective and Review," SN Comput. Sci., **4**(4), p. 425.

[9] Eckert, C., Isaksson, O., Hane-Hagström, M., and Eckert, C., 2022, "My Facts Are Not Your Facts: Data Wrangling as a Socially Negotiated Process, a Case Study in a Multisite Manufacturing Company," ASME J. Comput. Inf. Sci. Eng., **22**(6), p. 060906.

[10] Qin, Y., Lu, W., Qi, Q., Liu, X., Zhong, Y., Scott, P. J., and Jiang, X., 2017, "Status, Comparison, and Issues of Computer-Aided Design Model Data Exchange Methods Based on Standardized Neutral Files and Web Ontology Language File," ASME J. Comput. Inf. Sci. Eng., **17**(1), p. 010801.

[11] Cherukuri, R., and Yarram, V. K., 2024, "From Intelligent Automation to Agentic AI: Engineering the Next Generation of Enterprise Systems," Int. J. Emerg. Res. Eng. Technol., **5**(4), pp. 142–145.

[12] La Rocca, G., 2012, "Knowledge Based Engineering: Between AI and CAD, Review of a Language Based Technology to Support Engineering Design," Adv. Eng. Inform., **26**(2), pp. 159–179.

[13] Verhagen, W., Bermell-Garcia, P., Van Dijk, R., and Curran, R., 2012, "A Critical Review of Knowledge-Based Engineering: An Identification of Research Challenges," Adv. Eng. Inform., **26**(1), pp. 5–15.

[14] Vidner, O., Wehlin, C., and Wiberg, A., 2022, "Design Automation Systems for the Product Development Process: Reflections From Five Industrial Case Studies," Proc. Des. Soc., **2**, pp. 2533–2542.

[15] Quintana-Amate, S., Bermell-Garcia, P., Tiwari, A., and Turner, C., 2017, "A New Knowledge Sourcing Framework for Knowledge-Based Engineering: An Aerospace Industry Case Study," Comput. Ind. Eng., **104**, pp. 35–50.

[16] Ahmed, S., and Wallace, K. M., 2004, "Understanding the Knowledge Needs of Novice Designers in the Aerospace Industry," Des. Stud., **25**(2), pp. 155–173.

[17] EASA, 2024, "EASA Artificial Intelligence Concept Paper Issue 2—Guidance for Level 1 & 2 Machine-Learning Applications," https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-concept-paper-issue-2. Accessed February 29, 2026.

[18] ISO/IEC, 2024, "Artificial Intelligence—Functional Safety and AI Systems, International Organization for Standardization and International Electrotechnical Commission," ISO/IEC TR 5469:2024.

[19] ISO, 2024, "Road Vehicles—Safety and Artificial Intelligence, International Organization for Standardization," ISO/PAS 8800:2024.

[20] Pradas-Gómez, A., Kretzschmar, M., Paetzold-Byhain, K., and Isaksson, O., 2025, "A Team of Three: The Role of Generative AI in the Development of Design Automation Systems for Complex Products," Proc. Des. Soc., **5**, pp. 309–318.

[21] Kerley, W., Armstrong, G., Pepe, C., Moss, M., and Clarkson, P. J., 2011, "Using Simulation to Support Process Integration and Automation of the Early Stages of Aerospace Design," ICED 11—18th International Conference on Engineering Design, Vol. 1, Copenhagen, Aug. 15–18, pp. 134–146.

[22] SAE International, 2017, "AS9100: Quality Systems—Aerospace—Model for Quality Assurance in Design, Development, Production, Installation and Servicing," SAE International, 400 Commonwealth Drive, Warrendale, PA.

[23] Abollado, J. R., Shehab, E., and Bamforth, P., 2017, "Challenges and Benefits of Digital Workflow Implementation in Aerospace Manufacturing Engineering," Procedia CIRP, **60**, pp. 80–85.

[24] Krupa, G., 2025, "The Art of the Science," *Creative Stimulation in the Classroom*, AEROSPACE, Royal Aeronautical Society, London, March pp. 28–31.

[25] Rittel, H. W. J., and Webber, M. M., 1973, "Dilemmas in a General Theory of Planning," Policy Sci., **4**(2), pp. 155–169.

[26] Cross, N., 1992, *Research in Design Thinking*, Delft University Press, Delft, The Netherlands.

[27] Baskaran, S. P., Camacho Casero, J., Bagdatli, B., and Mavris, D., 2026, "MBSE-Driven MDAO in Co-architecture and Concurrent Engineering Workflows for Aircraft Development," AIAA SCITECH 2026 Forum, Orlando, FL, Jan. 6–10, American Institute of Aeronautics and Astronautics, AIAA Paper 2026-0193.

[28] Formentini, G., Bouissiere, F., Cuiller, C., Dereux, P.-E., and Favi, C., 2022, "Conceptual Design for Assembly Methodology Formalization: Systems Installation Analysis and Manufacturing Information Integration in the Design and Development of Aircraft Architectures," J. Ind. Inf. Integr., **26**, p. 100327.

[29] Gruber, T. R., 1993, "A Translation Approach to Portable Ontology Specifications," Knowl. Acquisit., **5**(2), pp. 199–220.

[30] Brimble, R., and Sellini, F., 2000, "The MOKA Modelling Language," Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW 2000), Vol. 1937 of Lecture Notes in Computer Science, Juan-les-Pins, France, Oct. 2–6, Springer, pp. 49–56.

[31] van den Berg, T., van der Laan, T., van Manen, B., Ciobotia, I., Bansal, D., and Sonneveld, J., 2023, "A Multidisciplinary Modelling System for Aircraft Structural Components," Aerospace Europe Conference 2023—Joint 10th EUCASS/9th CEAS Conference, Lausanne, July 9–13.

[32] Al Handawi, K., Brahma, A., Wynn, D. C., Kokkolaras, M., and Isaksson, O., 2024, "Design Space Exploration and Evaluation Using Margin-Based Trade-Offs," ASME J. Mech. Des., **146**(6), p. 061701.

[33] Kügler, P., Dworschak, F., Schleich, B., and Wartzack, S., 2023, "The Evolution of Knowledge-Based Engineering From a Design Research Perspective: Literature Review 2012–2021," Adv. Eng. Inform., **55**, p. 101892.

[34] Padula, S. L., and Gillian, R. E., 2006, "Multidisciplinary Environments: A History of Engineering Framework Development," 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Vol. 4, Portsmouth, VA, Sept. 6–8, pp. 2058–2068.

[35] Hiriyannaiah, S., and Mocko, G. M., 2008, "Information Management Capabilities of MDO Frameworks," Proceedings of the ASME Design Engineering Technical Conference, Vol. 3, Brooklyn, NY, Aug. 3–6, pp. 635–645.

[36] Regenwetter, L., Nobari, A. H., and Ahmed, F., 2022, "Deep Generative Models in Engineering Design: A Review," ASME J. Mech. Des., **144**(7), p. 071704.

[37] Whalen, E., and Mueller, C., 2022, "Toward Reusable Surrogate Models: Graph-Based Transfer Learning on Trusses," ASME J. Mech. Des., **144**(2), p. 021704.

[38] Yuan, C., Marion, T., and Moghaddam, M., 2022, "Leveraging End-User Data for Enhanced Design Concept Evaluation: A Multimodal Deep Regression Model," ASME J. Mech. Des., **144**(2), p. 021403.

[39] Pradas-Gómez, A., Krus, P., Panarotto, M., and Isaksson, O., 2024, "Large Language Models in Complex System Design," Proceedings of the Design Society, Vol. 4, Dubrovnik, May 27–30, pp. 2197–2206.

[40] Picard, C., Edwards, K. M., Doris, A. C., Man, B., Giannone, G., Alam, M. F., and Ahmed, F., 2025, "From Concept to Manufacturing: Evaluating Vision-Language Models for Engineering Design," Artif. Intell. Rev., **58**(9), p. 288.

[41] EASA, 2012, "Commission Regulation (EU) No 748/2012, Part 21, Subpart J—Design Organisation Approval, Section 21.A.239, Consolidated Version Revised 2025," https://eur-lex.europa.eu/eli/reg/2012/748

[42] SAE International, 2023, "ARP4754B: Guidelines for Development of Civil Aircraft and Systems."

[43] Massoudi, S., and Fuge, M., 2026, "Agentic Large Language Models for Conceptual Systems Engineering and Design," ASME J. Mech. Des., **148**(5), p. 051405.

[44] Mustapha, K. B., 2025, "A Survey of Emerging Applications of Large Language Models for Problems in Mechanics, Product Design, and Manufacturing," Adv. Eng. Inform., **64**, p. 103066.

[45] Eckert, C., Clarkson, P. J., and Zanker, W., 2004, "Change and Customisation in Complex Engineering Domains," Res. Eng. Des., **15**(1), pp. 1–21.

[46] Caputo, C., and Cardin, M.-A., 2022, "Analyzing Real Options and Flexibility in Engineering Systems Design Using Decision Rules and Deep Reinforcement Learning," ASME J. Mech. Des., **144**(2), p. 021705.

[47] Elgh, F., 2008, "Supporting Management and Maintenance of Manufacturing Knowledge in Design Automation Systems," Adv. Eng. Inform., **22**(4), pp. 445–456.

[48] Hjertberg, T., Stolt, R., and Elgh, F., 2016, "Managing Dependencies in Heterogeneous Design Automation Systems," Transdisciplinary Engineering: Crossing Boundaries, Advances in Transdisciplinary Engineering, Vol. 4, Curitiba, Oct. 3–7, 2016, IOS Press, pp. 279–288.

[49] Colvin, S., Pydantic Services Inc., 2024, "Logfire," https://github.com/pydantic/logfire, Accessed February 29, 2026.

[2] https://github.com/alex-pradas/DUCTILE

[50] Dong, L., Lu, Q., and Zhu, L., 2024, "A Taxonomy of AgentOps for Enabling Observability of Foundation Model Based Agents," arXiv.
[51] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., et al., 2024, "AgentBench: Evaluating LLMs as Agents," 12th International Conference on Learning Representations (ICLR 2024), Vienna, May 7–11, pp. 1–18.
[52] Yao, S., Shinn, N., Razavi, P., and Narasimhan, K., 2025, "τ-Bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains," 13th International Conference on Learning Representations (ICLR 2025), Singapore, Apr. 24–28, pp. 74824–74876.
[53] Battelle Memorial Institute, 2024, "Metallic Materials Properties Development and Standardization (MMPDS)," Battelle Memorial Institute, Columbus, OH, Supersedes MIL-HDBK-5.
[54] Hsieh, C.-Y., Chen, S.-A., Li, C.-L., Fujii, Y., Ratner, A., Lee, C.-Y., Krishna, R., and Pfister, T., 2023, "Tool Documentation Enables Zero-Shot Tool-Usage With Large Language Models," arXiv.
[55] Yuan, S., Song, K., Chen, J., Tan, X., Shen, Y., Kan, R., Li, D., and Yang, D., 2025, "EASYTOOL: Enhancing LLM-Based Agents With Concise Tool Instruction," Proceedings of the 2025 Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL-HLT 2025), 1, Albuquerque, NM, April 29–May 4, pp. 951–972.
[56] Kay, A., 1990, "User Interface: A Personal View," The Art of Human–Computer Interface Design, pp. 191–207.
[57] Maes, P., 1994, "Agents That Reduce Work and Information Overload," Commun. ACM, **37**(7), pp. 30–40.
[58] Wooldridge, M., and Jennings, N. R., 1995, "Intelligent Agents: Theory and Practice," Knowl. Eng. Rev., **10**(2), pp. 115–152.
[59] Pradas-Gómez, A., Panarotto, M., and Isaksson, O., 2024, "Evaluation of Different Large Language Model Agent Frameworks for Design Engineering Tasks," DS 130: Proceedings of NordDesign 2024, Design Society, Reykjavík, Aug. 12–14, pp. 693–702.
[60] Willison, S., 2025, "I Think 'Agent' May Finally Have a Widely Enough Agreed Upon Definition to Be Useful Jargon Now," https://simonwillison.net/2025/Sep/18/agents/, Accessed February 29, 2026.
[61] Etzioni, O., and Weld, D., 1994, "A Softbot-Based Interface to the Internet," Commun. ACM, **37**(7), pp. 72–76.
[62] Genesereth, M. R., and Ketchpel, S. P., 1994, "Software Agents," Commun. ACM, **37**(7), pp. 48–53.
[63] Colvin, S., Pydantic Services Inc., 2024, "Pydantic AI," https://github.com/pydantic/pydantic-ai, Accessed February 29, 2026.
[64] Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S., 2021, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event, Mar. 3–10, pp. 610–623.
[65] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I., 2017, "Attention Is All You Need," Advances in Neural Information Processing Systems, Vol. 30, Long Beach, CA, Dec. 4–9, Curran Associates, Inc., pp. 5999–6009.
[66] Labs, I., Khanna, S., Kharbanda, S., Li, S., Varma, H., Wang, E., Birnbaum, S., et al. 2025, "Mercury: Ultra-Fast Language Models Based on Diffusion," arXiv.
[67] Zhang, M., Press, O., Merrill, W., Liu, A., and Smith, N. A., 2024, "How Language Model Hallucinations Can Snowball," Proceedings of the 41st International Conference on Machine Learning, Vol. 235, Vienna, Austria, July 21–27, PMLR, pp. 59670–59684.
[68] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P., 2023, "Survey of Hallucination in Natural Language Generation," ACM Comput. Surv., **55**(12), pp. 1–38.
[69] Zelikman, E., Wu, Y., Mu, J., and Goodman, N., 2022, "Star: Bootstrapping Reasoning With Reasoning," Adv. Neural Inform. Process. Syst., **35**, pp. 15476–15488.
[70] Snell, C., Lee, J., Xu, K., and Kumar, A., 2025, "Scaling LLM Test-Time Compute Optimally Can Be More Effective Than Scaling Model Parameters," 13th International Conference on Learning Representations (ICLR 2025), Singapore, April 24–28, pp. 7595–7629.
[71] OpenAI, 2024, "OpenAI o1 System Card," arXiv:2412.16720.
[72] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D., 2022, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Advances in Neural Information Processing Systems, Vol. 35, New Orleans, LA, Nov. 28–Dec. 9, pp. 24824–24837.
[73] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T., 2023, "Toolformer: Language Models Can Teach Themselves to Use Tools," Advances in Neural Information Processing Systems, 36, New Orleans, LA, Dec. 10–16, pp. 68539–68551.
[74] Wei, T., Li, T.-W., Liu, Z., Ning, X., Yang, Z., Zou, J., Zeng, Z., et al., 2026, "Agentic Reasoning for Large Language Models," arXiv.
[75] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y., 2023, "ReAct: Synergizing Reasoning and Acting in Language Models," 11th International Conference on Learning Representations (ICLR), Kigali, Rwanda, May 1–5, pp. 1–13.
[76] Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y., 2023, "HuggingGPT: Solving AI Tasks With ChatGPT and Its Friends in Hugging Face," Advances in Neural Information Processing Systems, 36, New Orleans, LA, Dec. 10–16, pp. 38154–38180.
[77] "Work Faster and Smarter With Ansys Engineering Copilot," https://www.ansys.com/blog/work-faster-smarter-with-ansys-engineering-copilot, Accessed February 29, 2026.
[78] "Using the Chatbot Window in COMSOL Multiphysics®."
[79] "AI Agents for Mechanical Engineering," https://www.cosmon.com/, Accessed February 29, 2026.
[80] Barres, V., Dong, H., Ray, S., Si, X., and Narasimhan, K., 2025, "τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment," https://arxiv.org/abs/2506.07982
[81] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K., 2024, "SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?" https://arxiv.org/abs/2310.06770
[82] Zhang, L., Ergen, T., Logeswaran, L., Lee, M., and Jurgens, D., 2024, "SPRIG: Improving Large Language Model Performance by System Prompt Optimization," https://arxiv.org/abs/2410.14826
[83] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., et al., 2024, "DSPy: Compiling Declarative Language Model Calls Into Self-Improving Pipelines," 12th International Conference on Learning Representations (ICLR 2024), Vienna, May 7–11, https://arxiv.org/abs/2310.03714
[84] OpenAI, 2025, "OpenAI GPT-4.1 Prompting Guide," https://developers.openai.com/cookbook/examples/gpt4-1_prompting_guide/. Accessed February 29, 2026.
[85] Anthropic, 2025, "Anthropic Context Engineering Guide for AI Agents," https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents, Accessed February 29, 2026.
[86] Google, 2025, "Google Gemini Prompt Design Strategies," https://ai.google.dev/gemini-api/docs/prompting-strategies, Accessed February 29, 2026.
[87] Meta, 2025, "Meta AI Prompting Guide," https://ai.meta.com/learn/how-to-write-effective-ai-prompts-tips-and-examples/, Accessed February 29, 2026.
[88] Mistral AI, 2025, "Mistral AI Prompting Guide," https://docs.mistral.ai/capabilities/completion/prompting_capabilities, Accessed February 29, 2026.
[89] 2026, "Agent Skills Open Standard," https://github.com/agentskills/agentskills. Accessed February 27, 2026.
[90] Seepersad, C. C., Pedersen, K., Emblemsvåg, J., Bailey, R., Allen, J. K., and Mistree, F., 2006, "The Validation Square: How Does One Verify and Validate a Design Method?" *Decision Making in Engineering Design*, K. E. Lewis, W. Chen, and L. C. Schmidt, eds., ASME Press, New York, pp. 303–314.
[91] Morrison, T. M., Hariharan, P., Funkhouser, C. M., Afshari, P., Goodin, M., and Horner, M., 2019, "Assessing Computational Model Credibility Using a Risk-Based Framework: Application to Hemolysis in Centrifugal Blood Pumps," ASAIO J., **65**(4), pp. 349–360.
[92] Razi, H., Schaefer, J. D., Wanthal, S., Handler, J. J., Renieri, G. D., and Justusson, B. P., 2016, "Rapid Integration of New Analysis Methods in Production," Proceedings of the American Society for Composites—31st Technical Conference, American Society for Composites, Williamsburg, VA, Sept. 19–22.
[93] Hastie, T., Tibshirani, R., and Friedman, J., 2009, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer, New York.
[94] Clopper, C. J., and Pearson, E. S., 1934, "The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial," Biometrika, **26**(4), pp. 404–413.
[95] Anthropic, 2025, "Claude Code," https://github.com/anthropics/claude-code, Accessed February 29, 2026.
[96] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., et al., 2023, "Judging LLM-as-a-Judge With MT-Bench and Chatbot Arena," Advances in Neural Information Processing Systems, Vol. 36, New Orleans, LA, Dec. 10–16, pp. 46595–46623.
[97] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A., 2025, "A Comprehensive Overview of Large Language Models," ACM Trans. Intell. Syst. Technol., **16**, pp. 1–72.
[98] Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., and McHardy, R., 2023, "Challenges and Applications of Large Language Models," arXiv preprint, https://arxiv.org/abs/2307.10169
[99] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P., 2024, "Lost in the Middle: How Language Models Use Long Contexts," Trans. Assoc. Comput. Linguist., **12**, pp. 157–173.
[100] Lee, J. D., and See, K. A., 2004, "Trust in Automation: Designing for Appropriate Reliance," Hum. Factors, **46**(1), pp. 50–80.
[101] Bainbridge, L., 1983, "Ironies of Automation," Automatica, **19**(6), pp. 775–779.
[102] Wolfe, J. M., Horowitz, T. S., Van Wert, M. J., Kenner, N. M., Place, S. S., and Kibbi, N., 2007, "Low Target Prevalence Is a Stubborn Source of Errors in Visual Search Tasks," J. Exp. Psychol.: Gen., **136**(4), pp. 623–638.
[103] Cutler, V., and Paddock, S., 2009, "Use of Threat Image Projection (TIP) to Enhance Security Performance," Proceedings—International Carnahan Conference on Security Technology, Zurich, Switzerland, Oct. 5–8, pp. 46–51.
[104] Ahmed, F., Picard, C., Chen, W., McComb, C., Wang, P., Lee, I., Stankovic, T., Allaire, D., and Menzel, S., 2025, "Design by Data: Cultivating Datasets for Engineering Design," ASME J. Mech. Des., **147**(4), p. 040301.
[105] Cross, N., 2025, "Natural and Artificial Intelligence in Design," *Designerly Ways of Knowing and Thinking*, Springer, pp. 31–46.
[106] Li, F.-F., 2023, *The Worlds I See: Curiosity, Exploration, and Discovery at the Dawn of AI*, Flatiron Books, New York.
[107] Li, C., Zhang, R., Wong, J., Gokmen, C., Srivastava, S., Martín-Martín, R., Wang, C., et al., 2023, "BEHAVIOR-1K: A Benchmark for Embodied AI With 1,000 Everyday Activities and Realistic Simulation," Proceedings of Machine Learning Research, Vol. 205, Auckland, Dec. 14–18, 2022, pp. 80–93.
[108] Summers, L., 2026, "The Human-in-the-Loop Is Tired, Pydantic," https://pydantic.dev/articles/the-human-in-the-loop-is-tired, Accessed March 2, 2026.
[109] Shneiderman, B., 2022, *Human-Centered AI*, Oxford University Press, Oxford.

[110] Neumann, J. V., 1956, "Probabilistic Logics and the Synthesis of Reliable Organisms From Unreliable Components," *Automata Studies (AM-34)*, C. E. Shannon, and J. McCarthy, eds., Princeton University Press, Princeton, NJ, pp. 43–98.
[111] Rosenblatt, F., 1957, "The Perceptron: A Perceiving and Recognizing Automaton," Cornell Aeronautical Laboratory, Tech. Rep. 85-460-1.
[112] Minsky, M., and Papert, S., 1969, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA.
[113] Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, "ImageNet Classification With Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, Vol. 25, Lake Tahoe, NV, Dec. 3–6, pp. 1097–1105.
[114] Sutton, R., 2019, "The Bitter Lesson," http://www.incompleteideas.net/IncIdeas/BitterLesson.html, Accessed March 17, 2026.
[115] Sclar, M., Choi, Y., Tsvetkov, Y., and Suhr, A., 2024, "Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design Or: How I Learned to Start Worrying About Prompt Formatting," Proceedings of the Twelfth International Conference on Learning Representations (ICLR), Vienna, May 7–11, pp. 1–2.
[116] Felten, F., Apaza, G., Bäunlich, G., Diniz, C., Dong, X., Drake, A., Habibi, M., et al., 2025, "EngiBench: A Framework for Data-Driven Engineering Design Research," Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025), San Diego, CA, Dec. 2–7, pp. 1–15.
[117] Pradas-Gómez, A., Brahma, A., and Isaksson, O., 2026, "DUCTILE: Agentic LLM Orchestration of Engineering Analysis in Product Development Practice," https://github.com/alex-pradas/DUCTILE, doi:10.5281/zenodo.18836517