Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

HABIT: A Self-Evolving AI-Agent Harness for GUI-Based CAE Software---Application to Aerospace

Shaoqi WU

shaoqiwu@outlook.com

COMAC https://orcid.org/0000-0002-4975-7528

Jingyi LI

COMAC

Jiayi YAO

COMAC

Xingyuan JU

COMAC

Research Article

Keywords: AI agent, aerospace digitalization, human-AI collaboration, self-evolving memory, habit learning

Posted Date: August 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10648340/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

HABIT: A Self-Evolving AI-Agent Harness for GUI-Based CAE Software—Application to Aerospace

ISIC Lab
Shanghai Aviation Industry (Group) Co., Ltd., COMAC,
Shanghai, China

***Abstract*—GUI-based CAE software sits at the center of aerospace analysis workflows, yet interacting with it remains a labor-intensive bottleneck. Embedding foundation-model agents inside such software demands two capabilities that existing work lacks. The first is reliable tool calling over complex long-horizon chains, synchronized with the real software state. The second is self-evolution: tool surfaces and user habits drift over time, and the harness must improve with use. This paper presents HABIT (*Harness with Adaptive Behavior Inferred from Trajectories*), a self-evolving, host-agnostic harness for GUI-based CAE software. It builds on seven design principles, a five-layer architecture with context engineering, and a four-mode taxonomy of human-AI collaborative GUI control. A memory system passively learns stable habits from operation trajectories and personalizes behavior across sessions. HABIT is evaluated on a 33-task post-processing suite with final-state verification, and demonstrated end to end on the CRM-WBT wing-body aerodynamic benchmark, where a habit-driven session replays mined user preferences from a single instruction.**

***Index Terms*—AI agent, aerospace digitalization, human-AI collaboration, self-evolving memory, habit learning**

I. Introduction

Complex GUI-based CAE software is used for configuring simulations, inspecting results, and managing engineering workflows. Interacting with it demands high-frequency, labor-intensive work across CAE disciplines: navigating dense menus, sequencing multi-step operations, and applying physical reasoning to validate results. In aerodynamics, engineers repeatedly inspect surface pressure fields, extract aerodynamic coefficients, and generate slices and streamlines. In structural analysis, they probe stress fields and mode shapes. In aircraft development, these inspections repeat across every design iteration. This is precisely where natural-language-driven agents can save the most effort.

AI agents offer a natural remedy: a user instructs the system in natural language, and the agent drives the visualization toolchain autonomously. However, embedding such an agent inside GUI-based engineering software differs fundamentally from general-purpose chatbots or web agents. Operations must genuinely change the software state. An agent that merely *claims* to have applied a colormap while leaving the scene unchanged is worse than no agent at all. Domain rules, such as physical units and engineering conventions, must never be dropped. GUI and rendering toolkits impose strict thread affinity: the rendering context must never be accessed from worker threads. Finally, the agent must remain observable, resumable, and aligned with the user's working habits as they evolve.

Existing work on foundation-model agents for CFD focuses on end-to-end automation of simulation setup and execution over OpenFOAM [5]–[8]. However, none addresses embedding inside GUI-based software or models the user over time. We report the constraints that recur in daily engineering use and the resulting patterns, embodied in a harness we call HABIT (*Harness with Adaptive Behavior Inferred from Trajectories*).

The main contributions of this work are as follows:

- Seven design principles for embedding AI agents in GUI-based CAE software.
- A five-layer reference architecture with context engineering: scene-aware state injection, state-fingerprint system-prompt caching, CJK (Chinese, Japanese, and Korean)-aware token budgeting, and tool-unit-atomic history truncation.
- A four-mode taxonomy of human-AI collaborative GUI control, allocating control authority by operation risk.
- A self-evolving three-layer memory system that passively mines operation trajectories into stable habits, injects them into the agent prompt, and personalizes agent behavior across sessions.
- An empirical evaluation of tool-calling success rate, stability, and error resilience on a CAE post-processing task suite, together with an end-to-end case study on the CRM-WBT wing-body configuration.

II. Related Work

Tool-calling and GUI agents. The ReAct paradigm interleaves reasoning traces with tool actions and remains the foundation of modern agent loops [1]. The Berkeley Function Calling Leaderboard [2] evaluates function-calling on generic web APIs. GUI agents operate through screenshots and low-level input. OSWorld shows that state-of-the-art models complete only a small fraction of real computer tasks [3]. UI-TARS [4] learns app-specific interaction knowledge. These works target generic web/OS environments. They do not address the state-consistency, thread-affinity, and domain-fidelity requirements of GUI-based engineering software.

Foundation-model agents for CFD. MetaOpenFOAM automates OpenFOAM case generation with a multi-agent assembly line and retrieval-augmented generation [5]. Foam-Agent extends this line with hierarchical multi-index retrieval. It exposes core functions as Model Context Protocol tools and reaches 88.2% execution success [6]. ChatCFD achieves 82.1% execution success on 315 cases and introduces physical fidelity as a metric [7]. CFDLLMBench provides a three-part benchmark for CFD knowledge, code generation, and workflow execution [8]. These systems automate simulation setup and execution over OpenFOAM. None studies the post-processing stage, the GUI-embedding constraints, or cross-session user modeling.

A parallel trend connects LLMs to scientific software through the Model Context Protocol (MCP) [9]. Besides Foam-Agent above, community and laboratory projects wrap general-purpose visualization software such as Blender and ParaView as external tool servers [10], [11]. The host application then sits behind a protocol boundary as a remote tool provider. The next section contrasts this pattern with HABIT's in-process embedding.

Human-AI collaboration, memory, and personalization. Supervisory control and mixed-initiative interaction provide the theoretical foundation for shared control authority [12], [13]. For agent memory, MemGPT introduces a hierarchical memory architecture with external context [14]. Generative Agents maintain a memory stream with retrieval and reflection [15]. Recent surveys identify personalization and habit formation as open problems [16]. The present work differs in three ways. First, habits are mined *passively* from GUI operation trajectories rather than elicited explicitly. Second, the experience file is a transparent, user-editable artifact rather than an opaque store. Third, the setting is engineering software, where personalized defaults (colormaps, viewpoints, fields) carry real consequences for result fidelity.

III. The Proposed Harness

We first formalize the setting. Let the host application's state at step t be S_t (datasets, scene graph, UI configuration). The agent observes this state only partially, through a compact snapshot $o_t = \phi(S_t)$. The agent acts through a fixed tool set $\mathcal{T}$. Each call $a_t \in \mathcal{T}$ either transitions the host state, $S_{t+1} = f(S_t, a_t)$, or returns a structured error while leaving S_t unchanged. A task is given in natural language. It is complete only when the final state satisfies a task-specific predicate $v(S)$. Verification is therefore defined on real software state, never on the model's own claims. Four constraints distinguish this setting from web and OS agents. Actions mutate persistent engineering state (C1). Domain rules such as physical units must be preserved (C2). Only the GUI thread may access rendering state (C3). Finally, the agent must remain observable, resumable, and aligned with the user's habits over time (C4). The principles and architecture below follow from these constraints.

A. Design Principles

Seven design principles must be satisfied by any agent embedded in GUI-based CAE software. The principles span five concerns: tool-call reliability (P1, P2), loop termination and context economy (P3, P4), GUI thread safety (P5), domain-rule fidelity (P6), and tool-surface design (P7). Table I summarizes the principles together with the practical issue that motivated each.

TABLE I
Seven design principles (P1–P7).

Pr.	Principle	Motivating issue
P1	*Tools never raise into the loop.* Every failure is wrapped and returned to the model as a structured error result.	An uncaught exception kills the loop and the model's chance to self-correct.
P2	*Tool layer decoupled via closures.* Tools fetch fresh state per call, never caching domain objects.	Cached references go stale; holding none keeps the loop unit-testable.
P3	*Progressive disclosure.* A compact skill index is resident; full bodies load only when invoked.	Full bodies flood the prompt and starve the decision field.
P4	*Forced termination.* Bounded step budget; the final step disables tool calls.	Without it the model can loop indefinitely while the user sees silence.
P5	*GUI operations marshal to the GUI thread.* Explicit bridge with queued delivery; silent worker-thread fallbacks are refused.	Cross-thread rendering calls corrupt the graphics context.
P6	*Domain rules in the system prompt, with anti-pattern teaching.* Wrong/right examples pair each state-changing operation; unit preservation and read-before-write are mandated.	Teaching one operation's anti-pattern made the model fake the others.
P7	*Few, composable tools beat many narrow ones.* Derivable convenience tools are removed; overlapping builders merge into one entry point.	Tool explosion degrades selection accuracy.

B. Tool Schema Robustness

Because the model is the tool layer's only caller, robustness is enforced at the boundary rather than asked of the model. Three mechanisms keep every call a_t well-typed or explicitly failed. First, arguments are validated against the tool schemas, with automatic correction of common type slips. Second, malformed or truncated argument payloads are repaired when the intended structure is unambiguous. Third, a failure marked *retryable* triggers at most one bounded re-invocation request, so recovery never degenerates into a loop.

C. System Architecture

The reference architecture (Fig. 1) comprises five layers: tool execution, loop control, knowledge, GUI integration, and memory. The tool layer exposes never-raising handlers with schema validation. The loop layer enforces iteration budgets, detects pretend actions, and nudges retries. The

knowledge layer serves skills via progressive disclosure and injects domain rules. The integration layer marshals all GUI-touching calls through a single adapter facade, the only AI-to-UI bridge. This facade delegates to per-domain controllers for scene state, curves, and animation. The memory layer is detailed in Section III-F. Two cross-cutting flows are dashed in Fig. 1. First, knowledge and memory reach the model only as system-prompt injections. Second, a compact scene-state snapshot flows back to the loop after every action, closing the observe–act cycle. A new CAE domain is onboarded through a declared profile—domain prompt, tool modules, and skill files—without modifying the base harness.

Relation to MCP-based integration. HABIT embeds the agent in-process rather than exposing the application as an MCP tool server. This brings three differences. First, an MCP client receives only what the server returns per call. HABIT instead injects the scene-state snapshot every turn and verifies the final state on the live application. Second, rendering calls must be marshaled to the GUI thread (C3), and only in-process adapters enforce this cleanly. Third, without a transport layer, failures surface as structured tool results rather than protocol errors. MCP standardizes tool access. It does not address state synchronization, thread safety, state-based verification, or memory.

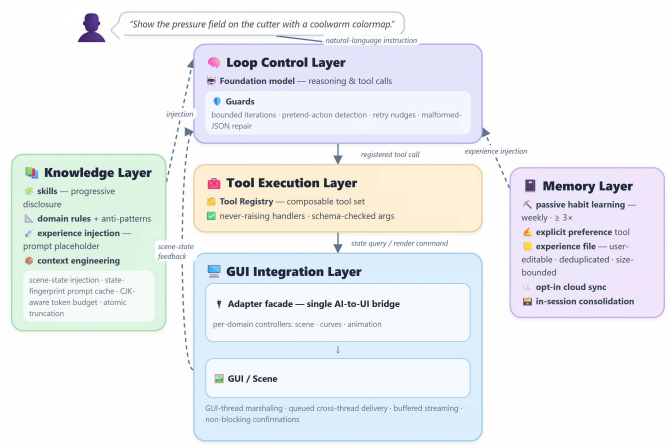


Fig. 1. Five-layer reference architecture. Knowledge (left) and memory (right) enter as system-prompt injections (dashed); scene state flows back to the loop after every action.

D. Context Engineering

The agent's context is kept accurate, compact, and valid by three mechanisms.

Scene-aware state injection. The snapshot $\phi(S_t)$ covers the scalar field, colormap, slice and probe toggles, dataset files, and available fields. It is serialized as compact JSON and appended to the system prompt every turn. The model therefore reasons over the true state, not its recollection. The compact form also cuts the block's token cost (quantified in Section IV).

State-fingerprint prompt caching. A cache key is computed from the dataset identity, a fingerprint of the scene state, the active skill set, and a short time-to-live bucket. The prompt is rebuilt only when the hash changes. The experience file's modification time is part of the key, so a just-recorded preference becomes visible immediately.

Density-aware token budgeting and atomic truncation. Token estimation distinguishes Latin from CJK text and budgets images conservatively. This distinction matters: Chinese-dense history counted as Latin undercounts tokens by nearly $3\times$. History truncation treats an assistant tool-call message and its following tool results as one atomic unit. Splitting such a unit produces an invalid message sequence that the API rejects. Orphan tool results are dropped after any rebuild.

Fig. 2 shows the runtime system prompt, whose blocks map onto the mechanisms above. Anti-pattern teaching with wrong/right pairs implements P6 (block 3). The resident skill index realizes progressive disclosure (P3, block 5). The experience file and the compact scene-state snapshot enter as injected blocks (blocks 7–8). Few-shot mappings steer recurring phrasings onto the right tool chain (block 6). The schema-validated tool surface is attached as tool definitions rather than prompt text (block 9).

E. Human-AI Collaborative GUI Control

Control authority over the GUI is allocated in four modes (Table II) under a risk-tiered principle. Reversible, low-risk operations run autonomously under system enforcement (Mode A) with GUI synchronization (Mode B). Irreversible or high-impact operations require human confirmation (Mode C). The agent may also proactively suggest actions that the user adopts with one click (Mode D). The `needs_confirm` flag supersedes an earlier `read_only` marker as its semantic inverse. A declined call becomes an error tool result, so the model can react rather than abort.

TABLE II
FOUR MODES OF HUMAN-AI COLLABORATIVE GUI CONTROL.

Mode	Control	Mechanism
A. System-enforced execution	AI executes, system verifies	Pretend-action regex detection with one bounded retry nudge
B. Shared-state synchronization	AI executes, GUI auto-syncs	`read_only` tools emit modification signals; bulk-fill tools update panels directly
C. Human-in-the-loop confirmation	Human approves first	`needs_confirm` dialog; decline becomes an error tool result
D. Proactive suggestion	AI initiates, human adopts	State-change rules $\rightarrow$ toast with an "Apply" button

All four modes rest on the thread-safe marshaling of C3. Blocking confirmations are offloaded so the event loop never freezes, and streaming output is buffered.

F. Experience-Driven Personalization: Passive Habit Learning and Layered Memory

Existing CFD agents are stateless: they neither remember the user nor change their behavior across sessions. The mem-

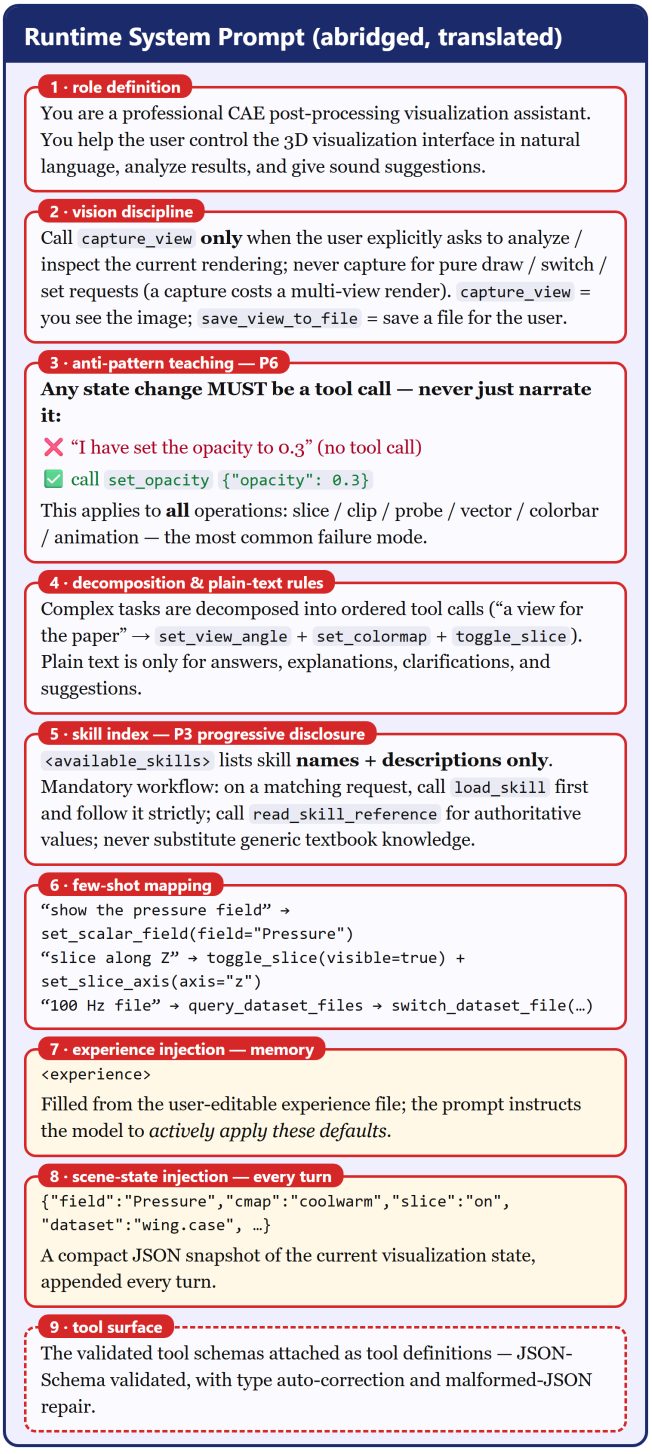


Fig. 2. The runtime system prompt (abridged, translated from Chinese). Red boxes annotate each block's role in HABIT.

ory layer addresses this with a three-layer architecture. Its centerpiece is *passive habit learning*: the user never says "remember this". Instead, the system discovers stable habits from the accumulated operation trajectory. The two memory-writing channels (passive mining and explicit preference recording) are orthogonal to the four collaboration modes above.

1) Two-channel memory writing: Every conversational turn is logged as a structured record. A record contains the user message, the assistant response, and the tool actions with parameters and success flags. These logs are the sole input to habit learning. Channel A is the passive habit learner. At application startup, at most once per week, a background pass aggregates the last seven days of logs into a compact action summary (tool frequencies, preferred parameters, repeated request patterns). It then asks the model to extract *stable habits*, defined as patterns repeated at least three times across sessions. A defensive parser recovers the habit list even behind a reasoning model's chain-of-thought preamble. Habits are appended to the experience file with deduplication.

Channel B is explicit memory: when the user says "remember this" or expresses a preference, a dedicated tool records it under a fixed set of categories (colormap, view angle, scalar field, analysis habit, dataset feature, other). An early implementation overwrote the entire file on every call, silently wiping previously recorded preferences. The append-based design stems directly from that failure.

2) Transparent storage and prompt injection: The experience file is a structured Markdown document with sections (operation preferences, project-specific knowledge, conversation highlights) and timestamped entries, as in this excerpt (translated):

```
# EXPERIENCE.md (excerpt, translated)
## Operation preferences
- [2026-03-31] Prefers the jet colormap for
scalar fields
- [2026-03-31] Checks the spl_db field when
importing
  acoustic results
## Project-specific knowledge
- [2026-08-08] Show only WALL meshes (fuselage,
tail_h,
  wing); coolwarm colormap; Y-axis slice; Cp
curves
  at η = 0.3, 0.6
## Resolved issues
- [2026-08-07] Dataset 1: field ''Wall_Shear''
not found;
  actual name is ''Wall Shear'' (with a space)
```

Entries are deduplicated and pruned oldest-first to a bounded size. Critically, they are also user-editable, which keeps the memory auditable rather than a black box. An optional cloud sync reconciles local and remote copies by modification time, so that preferences survive restarts and migrate across machines. Privacy is local-first: all data resides on the user's machine. Logs record only tool names, parameters, and success flags, never geometry or result data. Cloud sync is opt-in. At prompt assembly, the experience file is injected with an explicit instruction to *actively apply these defaults in subsequent operations*. The agent then adopts the user's habitual

colormap, viewpoint, and field choices without being asked. Together with Mode D, this forms dual-channel proactive assistance.

3) In-session layer: context compression: When a session's message history exceeds a threshold, a background task condenses it into a structured summary (topics, actions, preferences, insights). The most recent exchanges are retained verbatim, and the summary is injected as a system message. This mechanism is deliberately distinct from habit learning. Consolidation is short-term context compression within a session, while habit learning is long-term memory across sessions. The two layers complement rather than conflate.

4) The self-evolving loop: Fig. 3 closes the loop. Operation trajectories feed the weekly habit extraction, and the extracted habits are written into the experience file. The experience is then injected into the prompt, and the resulting personalized behavior produces new trajectories. The agent thus *self-evolves* toward the engineer's working style, in contrast with the stateless CFD agents surveyed in Section II. After several sessions on the CRM-WBT dataset, for example, the weekly pass mines the entry *"[2026-08-08] Show only the WALL meshes (fuselage, tail_h, wing); coolwarm colormap; Y-axis slice; C_p curves at $\eta = 0.3$ and 0.6"* into the experience file. Section IV.C shows a later session replaying exactly these defaults from a single instruction (Fig. 4(d)–(e)). The deliberately conservative settings—weekly runs, a seven-day lookback, a three-repetition threshold—protect the user from spurious habits.

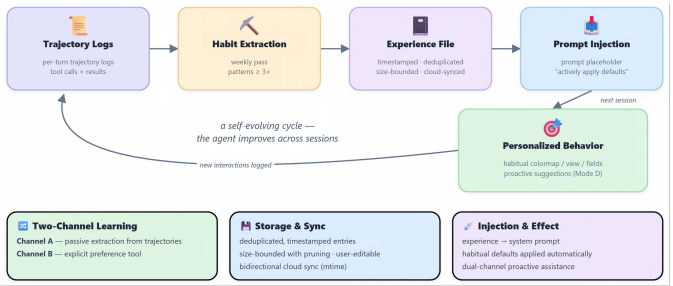


Fig. 3. The self-evolving memory loop. The bottom panels summarize the two writing channels, storage and sync, and the injection effect.

IV. Implementation and Evaluation

The evaluation is organized along two axes: task success rate and robustness of the tool-calling loop (Section IV.B), and an end-to-end demonstration on the CRM-WBT wing-body configuration (Section IV.C). Both are run on the reference implementation described below.

A. Reference Implementation

We instantiated HABIT inside a GUI-based CAE post-processing application used in daily aerodynamics work. A domain profile contributes the aerodynamics system prompt, skill files, and a composable surface of 54 registered tools. These tools cover dataset import, zone visibility, colormaps, slices and probes, curve extraction, and reporting. The compact scene-state serialization saves about 29% of prompt tokens on the state block relative to a naive JSON dump.

B. Task Success Rate and Robustness

Experimental setup. All experiments use a single fixed model: a self-hosted 35B open-weight model served through a standard chat-completions HTTP interface. Model variance is therefore excluded, and the measured quantities reflect HABIT itself. The task suite consists of 33 CAE post-processing tasks (10 single-step, 14 multi-step, 6 negative, 3 ambiguous; easy/medium/hard) over three real datasets (cylinder flow, wing pressure, cabin acoustic). Each task is scored twice. *Soft* is the tool-sequence and parameter match against the expected chain. *Hard* is the correctness of the final software state. The suite is executed in three independent rounds with fresh state per round.

Table III reports the success rates over three independent rounds (99 task executions). Final-state correctness (Hard) reaches 0.966 overall and 1.000 on single-step and ambiguous tasks. The combined score degrades gracefully with difficulty (0.91 easy / 0.86 medium / 0.69 hard). Stability is high: on 97% of the tasks the Hard score is identical across all three rounds (max per-task range 0.33). Every episode terminated within the 50-step budget, with a mean of 1.9 tool actions per task and a maximum of 8. All 261 LLM calls succeeded. Per-task wall time has median 6.6 s and P95 15.0 s.

TABLE III
Task success rates over three independent rounds (99 executions); Baseline: single-shot planning (Soft score).

	Soft	Hard	Combined	Baseline Soft	n
Overall	0.642 ± 0.966 0.434 0.175		0.837 ± 0.455 0.225		99
easy	0.767 ± 1.000 0.430 0.000		0.907 ± — 0.172		30
medium	0.657 ± 0.993 0.424 0.048		0.858 ± — 0.174		48
hard	0.433 ± 0.857 0.401 0.359		0.687 ± — 0.321		21
single-step	0.767 ± 1.000 0.430 0.000		0.907 ± 0.600 0.172		30
multi-step	0.774 ± 0.992 0.314 0.051		0.905 ± 0.573 0.133		42
negative	0.128 ± 0.833 0.320 0.383		0.551 ± 0.000 0.283		18
ambiguous	0.644 ± 1.000 0.410 0.000		0.858 ± 0.333 0.164		9

Error resilience. The negative suite exercises illegal or out-of-range requests. In 83.3% of the 18 negative executions, the final software state is left untouched (Hard pass). In 13 of them, the model attempted a tool path that the domain-rule layer rejected; it then recovered and stopped. The tool-call success rate is 96.2% (178/185). The failures are parameter-level rejections, which the retry-nudge mechanism turns into corrected or abandoned calls. The Soft–Hard gap is concentrated in benign behaviors. Three cases dominate. First,

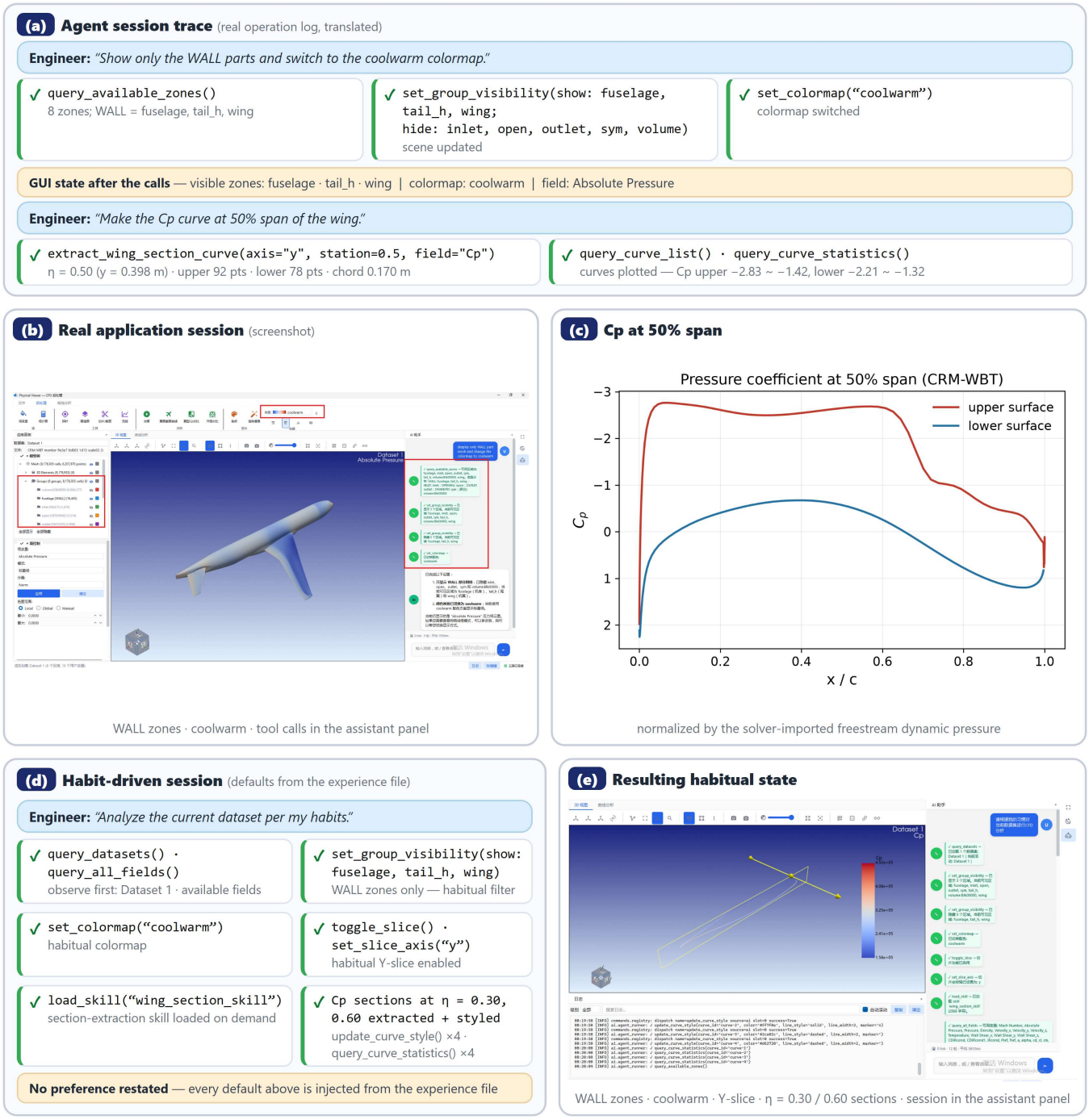


Fig. 4. CRM-WBT demonstration: (a) tool-call trace from the real operation log (translated); (b) screenshot of the resulting software state; (c) C_p at 50% span, normalized by the solver-imported freestream dynamic pressure and regenerated offline through the identical operations; (d) a later session driven by a single habit-based instruction (translated), replaying the mined defaults; (e) the resulting habitual state—WALL zones, coolwarm colormap, Y-axis slice, and C_p sections at $\eta = 0.30$ and 0.60—with the session visible in the assistant panel.

the model declines conservatively on negative and ambiguous tasks. Second, it produces sensible supersets of the expected chain on multi-step tasks (e.g., enabling the slice plane before setting its axis). Third, on the pressure-coefficient task (cfd-n05), the model queries the fields but does not complete the expression chain. This last case is a genuine capability limit of the 35B model on this task. No pretend action was detected in any of the 99 executions.

Comparison with a single-shot planning baseline. To quantify the value of the agentic loop itself, the same model plans the tool sequence for all 33 tasks in a single round, with no execution, no state feedback, and no retries. The plans

are scored with the same Soft scorer (Table III). The closed loop outperforms the baseline in every category (overall Soft 0.642 vs. 0.455). On 9 of the 33 tasks, the model produced no parseable tool sequence under single-round prompting and replied in prose instead. The loop's forced tool-call protocol and defensive parsers turn such replies into executable actions. The near-zero Soft on negative tasks in both conditions reflects correct declines, as the Hard check confirms.

C. End-to-End Demonstration on CRM-WBT

HABIT is demonstrated on the CRM-WBT wing-body configuration, a standard aerodynamic benchmark model, at cruise conditions ($Re = 3 \times 10^7$). Fig. 4(a)–(c) documents one continuous natural-language session. Every step executes through real tool calls, and final-state verification checks the resulting GUI state.

Fig. 4(a) shows the tool-call trace taken from the operation log. The first instruction, *"Show only the WALL parts and switch to the coolwarm colormap"*, is decomposed into an observe–act sequence. The agent first queries the available zones and finds that the WALL group comprises the fuselage, horizontal tail, and wing. Only then does it issue the state-changing calls that set zone visibility and the colormap. The GUI-state line recorded after the calls is the externally visible effect of the episode, and it is what final-state verification checks. The second instruction, *"Make the C_p curve at 50% span of the wing"*, triggers a domain-specific composite tool. This tool locates the $\eta = 0.5$ station, intersects the wing, and returns the upper- and lower-surface distributions. The agent then plots these distributions and summarizes them quantitatively.

Fig. 4(b) shows the real application after the session. The three WALL zones are rendered with the requested colormap, and the executed tool calls are visible in the assistant panel. The session ends in the requested state. Fig. 4(c) shows the pressure-coefficient distribution at the 50% spanwise station. The curve is normalized by the solver-imported freestream dynamic pressure. It is regenerated offline from the dataset through the identical sequence of operations. It is therefore a reproduction of the agent's actions, not a manually prepared rendering.

A second session on the same dataset demonstrates personalization in effect (Fig. 4(d)–(e)). This session was recorded after the experience file had accumulated the mined habit of Section III-F. The engineer only asks to *"analyze the current dataset per my habits"*. The agent then replays the mined defaults: WALL-only visibility, the coolwarm colormap, the Y-axis slice, and the C_p sections at $\eta = 0.30$ and 0.60. The session ends in the habitual visualization state without a single restated preference.

V. Conclusion

This paper has presented HABIT, a self-evolving harness for GUI-based CAE software, applied here to aerospace analysis. HABIT builds on seven design principles, a five-layer architecture with context engineering, a four-mode control model, and a memory loop. The memory loop mines operation trajectories into habits and personalizes behavior across sessions. On a 33-task post-processing suite, HABIT achieves a high and stable final-state success rate over three independent rounds. It outperforms a single-shot planning baseline by 0.19 in the tool-sequence score, with bounded termination, error recovery, and zero pretend actions. The CRM-WBT case study demonstrates the full workflow end to end. Future work includes extending HABIT to structural and thermal aerospace analyses and studying the privacy guarantees of passive habit learning.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023, arXiv:2210.03629.

[2] S. G. Patil, H. Mao, F. Yan, C. C.-J. Ji, V. Suresh, I. Stoica, and J. E. Gonzalez, "The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models," in *Proc. 42nd Int. Conf. Machine Learning (ICML)*, PMLR 267:48371–48392, 2025.

[3] T. Xie *et al.*, "OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2024, arXiv:2404.07972.

[4] Y. Qin *et al.*, "UI-TARS: Pioneering automated GUI interaction with native agents," arXiv:2501.12326, 2025.

[5] Y. Chen, X. Zhu, H. Zhou, and Z. Ren, "MetaOpenFOAM: An LLM-based multi-agent framework for CFD," arXiv:2407.21320, 2024.

[6] L. Yue *et al.*, "Foam-Agent: Towards automated intelligent CFD workflows," arXiv:2505.04997, 2025.

[7] E. Fan *et al.*, "ChatCFD: A large language model-driven agent for end-to-end computational fluid dynamics automation with structured knowledge and reasoning," *Advanced Intelligent Discovery*, Wiley, 2026, doi:10.1002/aidi.202500174.

[8] N. Somasekharan *et al.*, "CFDLLMBench: A benchmark suite for evaluating large language models in computational fluid dynamics," *J. Data-centric Machine Learning Research*, vol. 3, no. 13, 2025, arXiv:2509.20374.

[9] Anthropic, "Model context protocol (MCP)," 2024. [Online]. Available: https://modelcontextprotocol.io

[10] S. Ahuja, "BlenderMCP: Blender model context protocol integration," GitHub, 2024. [Online]. Available: https://github.com/ahujasid/blender-mcp

[11] Lawrence Livermore National Laboratory, "ParaView-MCP," GitHub, 2025. [Online]. Available: https://github.com/LLNL/paraview_mcp

[12] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*. Cambridge, MA, USA: MIT Press, 1992.

[13] E. Horvitz, "Principles of mixed-initiative user interfaces," in *Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI)*, 1999, pp. 159–166.

[14] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "MemGPT: Towards LLMs as operating systems," arXiv:2310.08560, 2023.

[15] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annu. ACM Symp. User Interface Software and Technology (UIST)*, 2023, pp. 1–22.

[16] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J.-R. Wen, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.