FalconCopilot: Empowering LLMs Towards Integrated Human-Machine Systems for Aviation Autonomy

Jingyuan Yan, Qingchen Liu*, Qichao Ma
University of Science and Technology of China
jy_yan@mail.ustc.edu.cn, {qingchen_liu, qcma}@ustc.edu.cn

Abstract

Complex flight tasks demand both intricate, long-horizon decision-making and precise operations, which imposes immense cognitive, knowledge and experience demands for pilots and highlights the need for advanced copilot systems. While Large Language Models (LLMs) bring powerful potential to this area, a comprehensive LLM-based copilot system—one that addresses deficiencies in task-level adaptability and fine-grained decision support while integrating with a high-fidelity environment—is critically lacking. To address this gap, we present FalconCopilot, pioneering the first such comprehensive system, composed of two parts: 1) Textual DCS, an interface built upon Digital Combat Simulator (DCS) World that unifies multi-modal cockpit data and piloting knowledge into a stable semantic interface for LLMs. Building on this interface, we introduce 2) FalconAgent, an LLM-powered copilot agent that performs optimized task planning, incorporating capabilities for multi-crew task allocation and procedural pruning. Our built-in human-AI interaction is grounded by a bidirectional feedback loop of runtime verification and human correction. In human-in-the-loop experiment, FalconCopilot shortens task completion time while attaining a level of performance approaching that of a human instructor.

1 Introduction

The escalating complexity of modern cockpits imposes substantial demands on pilots, requiring not only proficiency in intricate aircraft operations (Socha et al., 2020) but also deep domain expertise for complex, long-horizon task planning (Casner and Schooler, 2014). Pilots must simultaneously manage high-level mission objectives while executing fine-grained control actions, a dual challenge that frequently saturates their cognitive capacity. Sustained exposure to such a high cognitive load can lead to mental fatigue and degraded performance, thereby increasing the risk of operational errors and potential incidents (Barter et al., 1993; Xu et al., 2024a). While the proliferation of automated systems has alleviated some operational pressures, they are fundamentally ill-equipped to handle knowledge-intensive task planning and can even contribute to a loss of situational awareness (Kantowitz and Campbell, 2018; Sellen and Horvitz, 2024). Crucially, they lack the deep semantic understanding and contextual reasoning required for complex, nuanced decision support, necessitating a paradigm shift towards modern artificial intelligence.

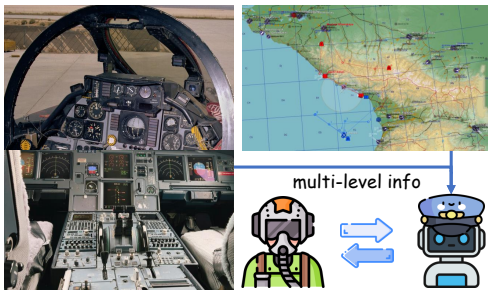


Figure 1: Flight tasks illustrate the complex interaction of cockpits and mission demands, where copilots assist pilots in managing such operational complexities.

Recent progress in Large Language Models (LLMs) has unlocked new possibilities for human-centered assistance (Wang et al., 2024b). In the automotive domain, for instance, current AI systems demonstrate capabilities ranging from foundational path planning to navigating complex open-road environments (Cui et al., 2024; Sima et al., 2024). However, the aviation domain presents a fundamentally different challenge. The complexity of internal operations and mission planning in aviation is an order of magnitude greater than in automotive, and both are deeply reliant on specialized expert knowledge(Goodrich et al., 2016). Furthermore, the data-driven paradigms prevalent in auto-

* Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 30001–30026

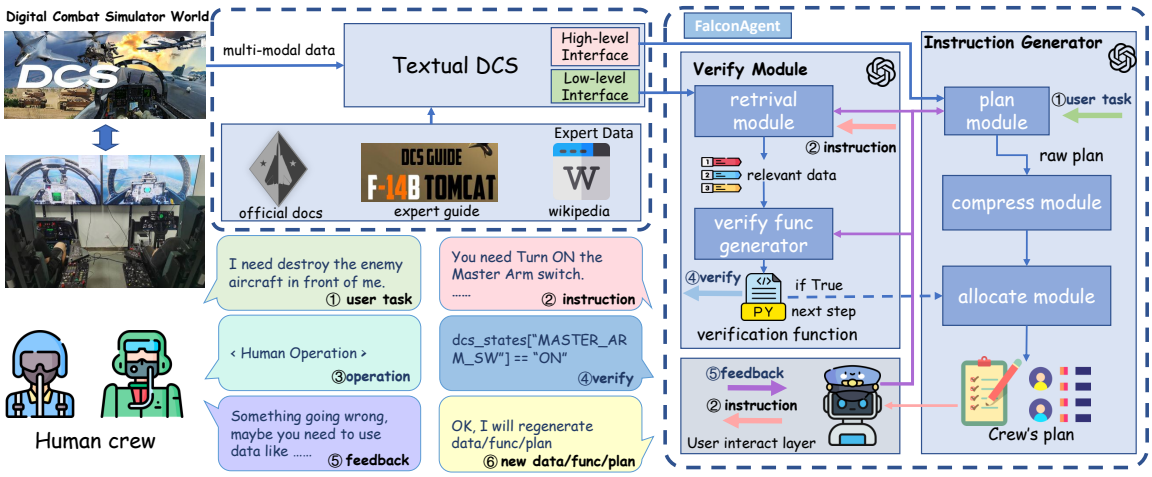


Figure 2: Overview of the FalconCopilot framework. The framework first interacts with Digital Combat Simulator (DCS) World via Texual DCS to provide the core agent with a dual-level semantic interface. The FalconAgent receives the ① user task from the human pilot and utilizes high-level information to generate correct ② operational instructions through its instruction generator, while the pilot ③ executes the final instruction. Simultaneously, it uses a verify module to generate a ④ verification function from low-level information. This function allows the FalconAgent to confirm whether the human has completed the current instruction before providing the next step. The human can also provide semantic ⑤ feedback at multiple stages to correct the FalconAgent's behavior and generate ⑥ new data/verification function/plan, forming a bidirectional feedback structure.

motive research are impractical in the data-scarce, knowledge-intensive aviation context, where the risk of model hallucination is critical. This necessitates a shift towards a knowledge-driven approach (Martino et al., 2023). While initial efforts have applied LLMs to aviation contexts (Wen et al., 2025; Zhang et al., 2025; Li et al., 2024), they are often limited on two fundamental levels. At the high-level (task) layer, they often lack end-to-end planning capabilities, typically functioning as information retrieval tools or advanced alarm systems. At the low-level (interaction) layer, even when high-fidelity simulators are used, they generally lack the fine-grained, button-level interface necessary to establish deep human-AI collaboration.

To address the aforementioned challenges, we introduce **FalconCopilot**, a pioneering LLM-powered copilot system designed for high-fidelity flight environments. Our system is composed of two core components: 1) **Textual DCS**, a novel middleware framework that builds a robust bridge for interaction between the LLM and the Digital Combat Simulator (DCS) World, a high-fidelity simulator whose intricate action-state space far surpasses those of typical agent environments (Fan et al., 2022; Ma et al., 2024). While we adopt DCS for its acclaimed operational realism and system complexity, it is a closed environment that lacks the structured interfaces essential for LLM reasoning, such as symbolic state representations, structured cockpit events, or accessible pilot action traces. Textual DCS bridges this semantic gap by grounding low-level, multi-modal simulator signals into a unified semantic representation, while defining a high-level, knowledge-driven action space for the agent. **2) FalconAgent**, an LLM-powered agent that serves as the core of the human-AI interaction and decision support process. It facilitates this interaction through end-to-end task planning, translating high-level mission objectives into a detailed sequence of cockpit operations for the pilot. The reliability of this collaborative process is ensured by a bidirectional closed-loop feedback mechanism that integrates runtime verification and human correction. Furthermore, the agent is designed to support complex multi-crew workflows through specialized modules for task allocation and procedural pruning. Our contributions are manifold:

Bidirectional Human-AI Closed-loop Paradigm: We introduce a novel interaction framework that enhances decision-support in LLM-Powered human-AI systems by establishing a closed feedback loop. This loop integrates AI-generated runtime verification with human-led semantic correction.

High-Fidelity Environment for LLMs' Research: We present and release Textual DCS,[1] a

[1] https://github.com/Yilgrimage/FalconCopilot.git

middleware that bridges the semantic gap between LLMs and high-fidelity simulators. It provides a robust foundation for future research on agents and aviation system within highly realistic settings.

A Novel LLM-Powered CoPilot System: We design and implement FalconCopilot, the first assistive copilot system that enables deep human-AI collaboration across an end-to-end workflow, spanning from complex aviation task planning to fine-grained cockpit operations in a high-fidelity flight simulator. Through an extensive human-in-the-loop study, we demonstrate the system's efficacy in significantly reducing cognitive workload and achieving near-expert performance.

2 Related Works

2.1 LLM Based Agent

Large Language Models (LLMs) have demonstrated remarkable capabilities across numerous domains (Achiam et al., 2023; Yang et al., 2025). Their capacity for complex planning and task decomposition, in particular, has established them as a powerful foundation for intelligent agents (Wang et al., 2023). This has led to successful applications in a wide range of interactive environments, including open-world games where agents plan and act (Wang et al., 2023, 2024a,e; Park et al., 2025) through learned low-level controllers (Baker et al., 2022; Lifshitz et al., 2023), web navigation and social simulation (Gur et al., 2024; He et al., 2024), and embodied AI where LLMs guide physical robots (Brohan et al., 2023; Singh et al., 2023; Wang et al., 2024f). Beyond fully autonomous operation, LLMs have also shown immense potential as collaborative partners, functioning as effective copilots in tasks that require close cooperation with or direct assistance to humans (Liu et al., 2024; Wang et al., 2024d). However, there remains a critical gap in assistive systems for environments where both the high-level planning and low-level execution are characterized by high complexity and require deep expert knowledge.

2.2 Copilot System

Copilot systems are a major research area, with automotive Advanced Driver-Assistance Systems (ADAS) being the most prominent field (Bengler et al., 2014; Cui et al., 2024). The recent advent of LLMs and VLMs (Yang et al., 2025; Bai et al., 2025) has further advanced this domain, enabling end-to-end driving models and sophisticated

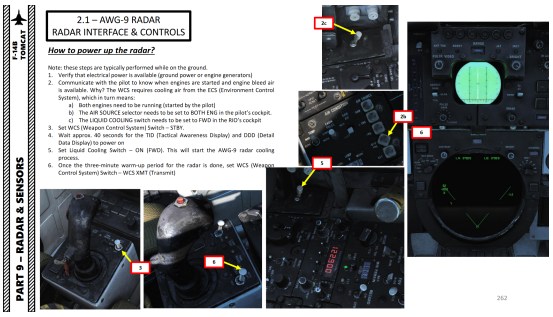


Figure 3: Example of Chuck's Guide

scene understanding with natural language reasoning (Sima et al., 2024; Xu et al., 2024b). In contrast, the aviation domain presents a different set of core challenges, focusing more on managing internal procedural complexity and complex task planning than reacting to external traffic (Oster Jr et al., 2013). Early explorations into traditional aviation copilots targeted specific capabilities, such as enhancing pilot perception or adapting interfaces to cognitive load (Şenol, 2020). More recent work has begun to apply LLMs to this area, though often for offline data analysis or as in-cockpit knowledge bases for information retrieval (Li et al., 2024; Wen et al., 2025), yet they remain deficient in both high-level planning and low-level interaction.

3 Method

Our proposed FalconCopilot features a hierarchical architecture with two core components. The foundational Textual DCS is an innovative middleware framework bridging the semantic gap between the Large Language Model (LLM) and the high-fidelity simulator, providing a structured, two-level interface. Built atop this, FalconAgent serves as an LLM-powered copilot collaborating with human pilots via instruction generation and bidirectional feedback, enabling robust, corrective interaction. Figure 2 illustrates the pipeline.

3.1 Textual DCS

To bridge the gap between an LLM's symbolic reasoning and the complex, multi-modal environment of a high-fidelity flight simulator, we introduce Textual DCS. Its core function is to translate the entire high-fidelity operational environment—encompassing the field environment and mission objectives, as well as the aircraft's own state—into a structured, symbolic language that the LLM can comprehend and act upon. Its architecture is explicitly designed to address the dual

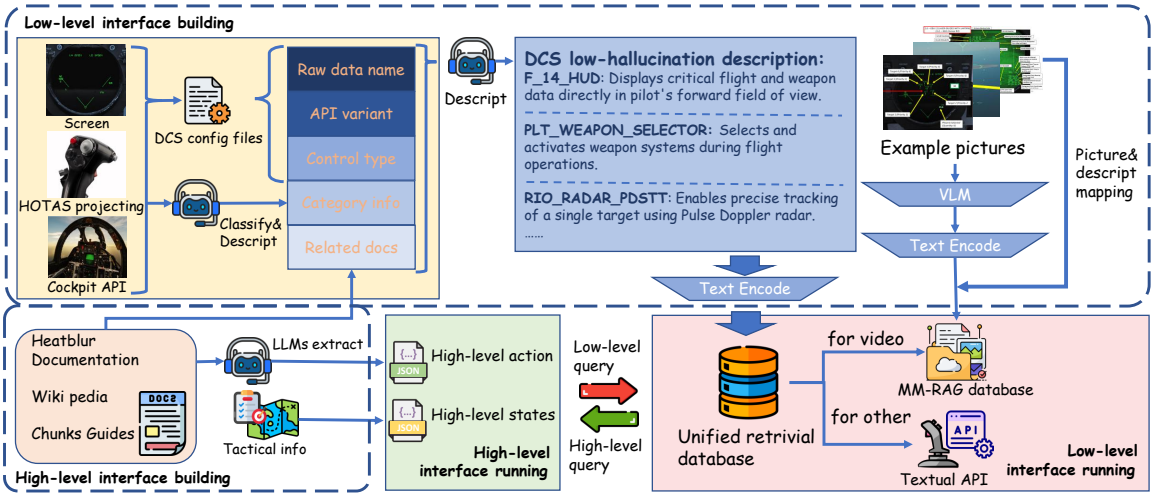


Figure 4: Overview of Textual DCS, The framework is centered around a dual-level interface structure. During its construction, the High-level interface utilizes LLMs to extract and encapsulate operational guidance from expert knowledge to form the action space. Simultaneously, it receives mission information from within the simulator to serve as the state space, and organizes this into JSON format for convenient external access. The Low-level interface is built by integrating multimodal data sources and synthesizing low-hallucination text descriptions for them, creating a Unified Retrieval Database, utilizing 'Example pictures' as relevant visual context. For external calls, based on the modality of the retrieved data, the system invokes either multimodal RAG or textual API to obtain the final data.

challenges at both high and low levels: enabling complex task planning while providing the fine-grained, button-level semantic grounding that previous systems have lacked. Figure 4 illustrates the overall structure of the Textual DCS.

3.1.1 High-Level Interface for Task Planning

This layer provides the LLM an abstracted, symbolic interface for strategic reasoning and action within the simulation, comprising a state space for perception and an action space for planning.

Inspired by how human pilots perform strategic-level planning, we construct a high-level state space capturing mission information. This includes mission objectives, critical positions, and ownship mount states. These elements are abstracted into a structured state description and fed into the LLM to enable context-aware planning under uncertainty.

To act upon this state, we construct a corresponding high-level, task-oriented action space. To mitigate LLM hallucinations and reduce planning ambiguity, this action space is derived from community-authored flight tutorials (Chuck's Guides, as illustrated in 3) within the DCS World ecosystem. Specifically, we employ an LLM-based pipeline that processes these tutorials segmented by chapter, distilling raw step-by-step guidance into formalized, reusable high-level actions. These sequences are manually verified and assigned style-unified descriptions by the LLM to ensure compatibility with downstream planning. Each action corresponds to a goal-driven and parameterized procedure, which enables the system to adapt to diverse and complex mission scenarios. Concurrently, we consolidate official aircraft manuals and aviation wikis into a textual knowledge base. This repository equips the agent with essential knowledge retrieval capabilities and serves as a semantic reference for subsequent data optimization.

3.1.2 Low-Level Grounding for Real-time Perception

This layer grounds the low-level interface in raw, multi-modal data streams, aligning the LLM's symbolic representations with the simulation state.

Accurate retrieval from a high-dimensional state space (>1,000 items) poses a significant challenge. While some methods (Qin et al., 2024; Du et al., 2024) manage scale via complex multi-stage reasoning, they incur high inference overhead. Instead of complicating the retrieval logic, we adhere to the standard generation-retrieval paradigm but optimize the underlying data representation. We construct a unified, text-centered database that projects all heterogeneous data—switches, displays, and HOTAS (hands on throttle and stick)—into a shared semantic space through explicit semantic mapping (details in **Appx A.1**), enabling efficient flat re-

trieval.

As shown in Figure 4 (upper), we employ a multi-source context fusion pipeline to synthesize hallucination-free descriptions. This process anchors generation in DCS configuration files and official flight manuals, jointly leveraging metadata, category contexts, and RAG-retrieved documents to constrain the semantic space. At runtime, this database powers a unified retrieval interface where incoming observations are grounded via vector matching against the semantic knowledge base. 'High-level query' refers to task instructions provided by the pilot, while a 'Low-level query' represents the internal requests issued by the agent to retrieve underlying simulator data. For non-visual queries, the system maps the retrieved symbolic key to the simulator's internal state to fetch real-time values directly. For visual queries, a multimodal RAG module retrieves relevant visual context to support reliable VLM-based answering.

By standardizing these distinct streams, Textual DCS effectively abstracts underlying modality heterogeneity, providing the LLM with a streamlined interaction layer. Implementation details—including the data generation pipeline, runtime processing workflows, and quantitative evaluation of data enhancement effects—are provided in **Appx. A.2 A.3**.

3.2 FalconAgent

To enable human–AI interaction in complex flight operations and planning tasks, we introduce FalconAgent. Its architecture consists of two core components: a multi-stage Instruction Generator that transforms high-level goals into optimized step-by-step instructions for the aircrew, and a Bidirectional Closed-Loop Feedback mechanism that supports reliable and adaptive execution.

3.2.1 Instruction Generator

The Instruction Generator is the planning core of FalconAgent, implemented as a multi-stage pipeline that transforms high-level task intentions into precise, context-aware, and collaborative step-by-step instructions for the aircrew.

Stage 1: Parameterized High-Level Planning: As outlined previously, we model the problem as a planning task with a state space (S) and an action space (A). Each high-level action in A is parameterized (e.g., `Waypoint_Entry(waypoint="WP1")` or `Set_Radar_Mode(mode="TWS")`), enabling the LLM to generate more expressive and adaptive plans. Given the current state $s \in S$, the LLM planner outputs an ordered sequence of parameterized actions, $\pi = (a_1, a_2, \ldots, a_n)$, forming the macroscopic mission strategy.

Stage 2: Decomposition and Optimization: This stage compiles the macroscopic strategy produced by the LLM into human-executable instructions. 1) **Step Decomposition:** First, the pipeline decomposes each high-level action a_i in the sequence into a set of atomic, executable steps(e.g., Press Launch Button), based on an expert knowledge base which is offered by Textual DCS. 2) **Procedural Pruning:** Naive concatenation of decomposed steps often results in redundant operations across adjacent sub-tasks (e.g., repeatedly configuring identical power switches). To enhance efficiency, we introduce a two-stage LLM prompting scheme: the module first explicitly identifies similar or overlapping operations within the full sequence, and subsequently performs logical reasoning to safely discard unnecessary steps while generating a rationale. Detailed examples can be found in **Appx. B.2**. 3) **Multi-Crew Scheduling:** For dual-seat aircraft like the F-14A, purely serial execution underutilizes the crew. To enable concurrent operations without prerequisite conflicts, the LLM conducts a step-by-step dependency analysis on the pruned sequence. It partitions the workflow into parallel instruction streams for the Pilot and Radar Intercept Officer (RIO), annotating specific steps with cross-crew dependencies. Instructions are dynamically delivered only when their prerequisite actions are fulfilled by the other crew member. Any hard-to-handle corner cases can be flexibly resolved by manually appending dependency rules to the high-level actions in the knowledge base (see **Appx. B.3** for scheduling visualization).

Stage 3: Incremental Instruction Delivery Finally, FalconAgent incrementally delivers the optimized parallel plan, dynamically aligning instruction flow with pilot execution pace and real-time system feedback. This dependency-aware mechanism enhances operational clarity and coordination under realistic conditions. Further details of each modules in three stages can be found in **Appx. B**.

3.2.2 Bidirectional Closed-Loop Feedback

Establishing an effective feedback loop is critical for the reliability and safety of Large Language Model (LLM) applications (Madaan et al., 2023; Liu et al., 2023), especially in complex human-in-the-loop settings such as flight cockpits. In flight

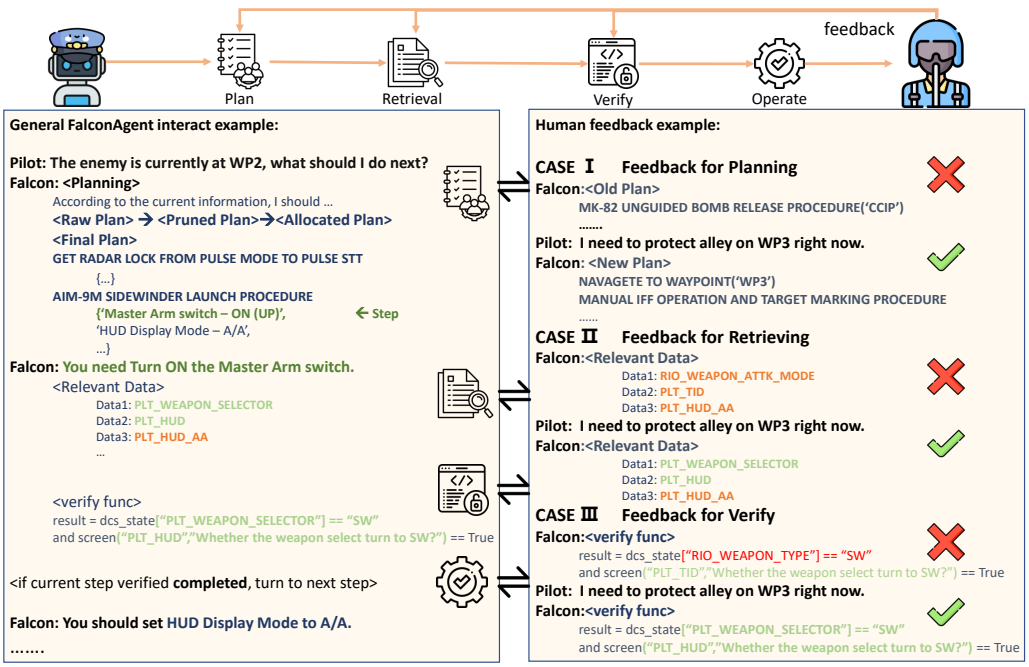


Figure 5: FalconAgent running example with human feedback in planning, retrieval, and verification stages

scenario, AI-to-Human feedback serves as a real-time monitor to verify that procedural steps undertaken by the pilot are correctly performed, while Human-to-AI feedback enables human intervention to correct AI behavior. However, implementing such a loop introduces two key challenges: the inefficiency and latency of using the LLM for real-time state verification, and the need for a human supervisory channel to correct the LLM's probabilistic errors. To address these challenges, FalconAgent employs a bidirectional closed-loop feedback mechanism consisting of an AI-to-Human verification channel and a Human-to-AI correction channel.

AI-to-Human: Verification Function Generator To overcome the limitations of using the LLM as a real-time monitor, we introduce a verification mechanism based on dynamic check function generation, inspired by prior work on LLM-generated executable code (Liang et al., 2023; Wang et al., 2024c). Instead of having the LLM reason over raw cockpit states step-by-step, verification is delegated to lightweight code snippets. For each instruction, we employ a retrieval-augmented code generation approach grounded in a unified database: the LLM retrieves relevant cross-modal representations and synthesizes a task-specific verification function. For example, to verify a "lock and fire" command, the generated function may check both hardware states (e.g., `states["PLT_WEAPON_TRIGGER"] == "Pressed"`) and semantic visual states (e.g., `screen["HUD", "Target locked?"] == True`). These snippets then run asynchronously, checking instrument states and cockpit variables to confirm the pilot's execution with near-zero delay. Once a step is verified, the system provides an explicit text notification on the UI to inform the pilot before transitioning to the next instruction. This paradigm decouples verification from LLM inference, offloading real-time validation to low-latency modules while freeing the LLM to focus on high-level decision-making.

Human-to-AI: Multi-Level Feedback Complementing the AI-to-Human verification channel, this human correction channel ensures adaptability while preserving the pilot's ultimate authority. To maintain human control and fault tolerance, we design a multi-level feedback mechanism that allows pilot intervention at both the planning and execution stages. During planning, pilots can use natural language to contest suboptimal high-level plans (e.g., waypoint order), triggering replanning. During execution, when a verification error occurs—due to either faulty data retrieval or flawed function generation—pilots can either override the alert or correct the agent's logic. In the latter case, pilots provide a natural language description of their perception, prompting the LLM to revise its data retrieval or regenerate the verification function. A detailed example is shown in Figure 5.

This bidirectional design creates a robust, fault-

Figure 6: High-fidelity cockpit integrated with DCS

tolerant collaborative loop. The AI-to-Human channel provides concrete, low-latency validation of physical actions, while the Human-to-AI channel ensures the high-level plan remains corrigible and responsive to expert guidance.

4 Experiment

This section evaluates our system from foundational accuracy to real-world operational impact. We structure the evaluation into: 1) **Foundational Capability**: measuring Textual DCS's core semantic grounding efficacy and quantifying performance gains from the bidirectional feedback loop; and 2) **Task-Level Performance**: a human-in-the-loop study evaluating FalconCopilot's end-to-end effectiveness in reducing task completion time and cognitive workload during realistic missions.

4.1 Foundational Capability Experiment

4.1.1 Setting

We evaluate Textual DCS on low-level grounding and feedback adaptability using a custom benchmark of 9 representative F-14A actions (112 annotated steps). At each step, the LLM retrieves required cockpit data and generates a Python-based verification function. Performance is measured via standard retrieval metrics (Precision, Recall, F1) and Verification Success (VS)—success rate of the generated verification function.

Stage I: Standalone Verification. Various foundational LLMs interact directly with Textual DCS to establish baseline grounding capabilities.

Stage II: Human-Guided Refinement. To quantify the gains from bidirectional feedback while ensuring reproducibility, we simulate the human-AI loop using a GPT-4o-based "virtual instructor." For each failure in Stage I, this simulator generates a corrective hint based on ground-truth states, which is injected into the agent's context to measure performance uplift. This approach provides controlled yet realistic evaluation of feedback adaptability.

Full benchmark composition and experiment implementation details are provided in **Appx. E F**.

Model	Standalone Verification				Add Feedback	
	Recall	Prec.	F1	VS	Recall	VS
GPT-4	0.79	0.74	0.76	0.74	0.98	0.93
Claude-3.5	0.89	0.72	0.79	0.74	0.96	0.94
Gemini-2.5	0.84	0.73	0.78	0.76	0.95	0.90
Qwen-Max	0.85	0.76	**0.81**	0.74	0.93	0.89
DeepSeek-V3	**0.94**	0.41	0.57	**0.81**	**0.99**	0.90

Table 1: Performance of Foundational Capability. The left section shows the standalone performance of Textual DCS; the right section demonstrates the significant uplift from the bidirectional feedback loop.

4.1.2 Results

Table 1 summarizes the performance. In the foundational stage, Textual DCS supports robust grounding across diverse models. Notably, while DeepSeek-V3 achieves the highest Recall (0.94) and VS (0.81) by generating extensive candidates (sacrificing Precision), other models like Qwen-Max maintain a better balance (F1=0.81). Crucially, the alignment stage validates our feedback design. With a single corrective hint from the simulator, Retrieval Recall surges to near-perfect levels ($> 93\%$) across all models, and VS sees absolute improvements of up to 20% (e.g., GPT-4: $0.74 \rightarrow 0.93$). This confirms that FalconCopilot allows even non-technical user feedback to effectively rectify complex logic errors.

4.2 Task-Level Performance Experiment

4.2.1 Setting

Since no prior work provides end-to-end task evaluation in this domain, we designed a controlled comparison. Existing methods (Li et al., 2024; Schlichting et al., 2025) are UI-wrapped RAG systems; others (Wen et al., 2025; Xiang et al., 2025) add specialized sensors to fixed task procedures. We implemented their core functionality as a baseline, which retrieves documents based on the task and provides them to the pilot via a UI. Given that their assumptions differ from our focus on multi-level task complexity, we utilize the standalone instruction generation module of FalconAgent as an Agentic Manual to represent the performance of agent systems directly applied to pilot assistance.

Each participant completed tasks under four counterbalanced conditions: 1) **Baseline:** The

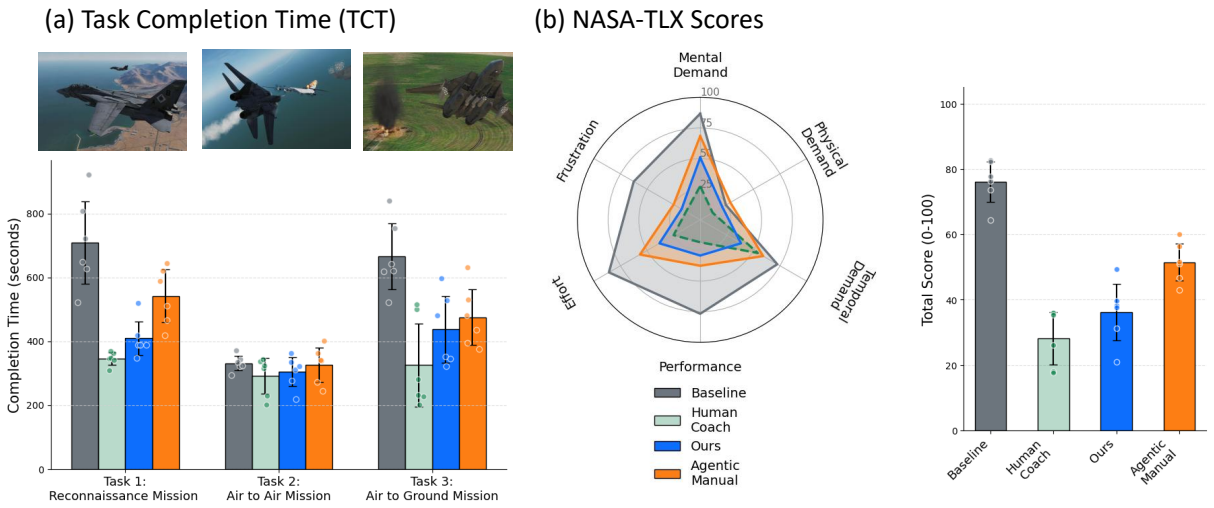


Figure 7: **(a)** Average task completion time (TCT) across three mission tasks under all four conditions. Bars show mean completion time, with error bars indicating standard deviation across participants. **(b)** NASA-TLX results comparing perceived cognitive workload. The radar plot shows subscale scores, while the bar chart summarizes weighted total scores.

state-of-the-art solution in this domain, which we implemented on our dataset; 2) **Agentic Manual:** Serving as an agentic, task-adaptive guide; 3) **Human Coach:** Remote expert guidance; 4) **Ours:** The full FalconCopilot system.

We adopted a within-subjects design with participants (**N=6**) spanning novice to expert levels. Each participant completed three distinct tasks under all four conditions, with both task and condition order counterbalanced to mitigate learning and order effects. The study was conducted in a high-fidelity F-14A cockpit replica using DCS World with customized, reproducible missions (Figure 6), preserving the aircraft's full operational complexity and ensuring ecological validity. Additional experiment details are provided in the **Appx. G**.

4.2.2 Metrics

We evaluated performance using both **objective** and **subjective** metrics. Objective performance was measured by **Task Completion Time (TCT)**, capturing the efficiency of procedural execution under different assistance conditions.

Subjective cognitive workload was assessed using the **NASA-TLX** questionnaire (Hart and Staveland, 1988), which measures six dimensions: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. Participants completed the questionnaire after each task. We report all dimensions' scores and weighted NASA-TLX scores aggregated across participants, where lower values indicate reduced perceived workload.

4.2.3 Quantitative Results

Given the limited within-subject sample ($N = 6$), we assessed statistical significance using Wilcoxon signed-rank tests (see full analysis table in **Appx. G.6**) and report rank-biserial correlation (r) as the effect size to characterize the magnitude of improvement.

Task Completion Time: Figure 7(a) illustrates TCT results. FalconCopilot substantially outperformed the baseline in procedure-intensive missions, achieving statistically significant gains in reconnaissance and air-to-ground tasks (Task 1 & 3, $p = 0.03$, $r = -1.0$). In contrast, for the skill-dominant air-to-air task (Task 2), no significant performance differentiation was observed. This outcome aligns with our framework's design scope, which aims to provide assistance for procedural planning and operation rather than assisting with low-level motor control. Notably, in the complex Task 3, the system achieved statistical parity with the Human Coach ($p = 0.09$), demonstrating expert-level efficiency. The comparison with Agentic Manual further confirms that closed-loop feedback is critical for procedural tasks, as its removal significantly degraded performance in Task 1 ($p = 0.03$).

Cognitive Load Evaluation: Weighted NASA-TLX scores (Figure 7b) show that FalconCopilot consistently reduces perceived workload relative to baseline ($p = 0.03$, $r = -1.0$). The same advantage is maintained over the ablated variant ($p = 0.03$, $r = -1.0$), identifying closed-loop as-

sistance as the primary driver of cognitive relief. Crucially, no statistically significant difference was observed between FalconCopilot and the Human Coach ($p = 0.22$), indicating that our system effectively lowers cognitive burden to a level comparable to human expert guidance .

4.2.4 Qualitative Analysis

We analyzed interview transcripts to contextualize the quantitative gains. This reveals both expertise-dependent interaction strategies and shared psychological benefits enabled by the feedback loop.

Divergent Usage Strategies. Interaction patterns varied by expertise. Novices utilized the system as *instructional scaffolding* for attentional guidance, reducing the cognitive cost of panel searching. Familiar players treated it as a *contextual bridge*, accelerating skill transfer from prior aircraft knowledge. Experts employed the system for *redundancy management*, delegating procedural monitoring to the AI to prevent muscle-memory-induced complacency while maintaining tactical focus.

Value of the Feedback Loop. The closed-loop mechanism provided distinct benefits across psychological and operational dimensions. Psychologically, it served as a safety net for novices, who reported that the confirmation signal eliminated the "anxiety" of uncertain inputs, allowing them to proceed with confidence. Operationally, it proved critical for experts in intercepting "fatal" omission errors. As one expert noted, the system successfully flagged a missed "Ground Mode" switch—a subtle state error easily overlooked during rapid muscle-memory execution—thereby salvaging a mission opportunity that would have been lost in a real combat scenario (full content in **Appx. G.7**).

5 Conclusion

In this work, we present FalconCopilot, the first comprehensive assistive flight co-pilot system bridging high-level tasks and low-level operational instructions. This is built on our foundational framework, Textual DCS, which builds a dual-layer semantic link between LLMs and the high-fidelity flight environment. Leveraging this interface, FalconAgent supports pilots with task planning and operational decisions via its instruction generator and bidirectional feedback loop. We validate the system's effectiveness through multi-dimensional experiments, showing human-in-the-loop performance approaching that of a human instructor.

Limitations

Our framework aims to provide cognitive offloading for procedural planning rather than replacing low-level motor control; therefore, it does not offer assistance for aircraft maneuvering. Furthermore, constrained by the inherent inference latency of current LLMs, the system prioritizes long-horizon decision-making, offering limited support in scenarios requiring rapid strategic adjustments.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants U2541214, 62573396 and 62373341, and in part by the Anhui Provincial Natural Science Foundation under Grant 2508085J039.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

SA Barter, JQ Clayton, and G Clark. 1993. Aspects of fatigue affecting the design and maintenance of modern military aircraft. *International journal of fatigue*, 15(4):325–332.

Klaus Bengler, Klaus Dietmayer, Berthold Farber, Markus Maurer, Christoph Stiller, and Hermann Winner. 2014. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent transportation systems magazine*, 6(4):6–22.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, and 1 others. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR.

Stephen M Casner and Jonathan W Schooler. 2014. Thoughts in flight: Automation use and pilots' task-related and task-unrelated thought. *Human factors*, 56(3):433–442.

Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, and 1 others. 2024. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 958–979.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11812–11829, Vienna, Austria. PMLR.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

Kenneth H Goodrich, Jim Nickolaou, and Mark D Moore. 2016. Transformational autonomy and personal transportation: Synergies and differences between cars and planes. In *16th AIAA Aviation Technology, Integration, and Operations Conference*, page 3604.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*.

Sandra G Hart and Lowell E Staveland. 1988. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.

Barry H Kantowitz and John L Campbell. 2018. Pilot workload and flightdeck automation. In *Automation and human performance*, pages 117–136. CRC Press.

Fan Li, Shanshan Feng, Yuqi Yan, Ching-Hung Lee, and Yew Soon Ong. 2024. Virtual co-pilot: Multimodal large language model-enabled quick-access procedures for single pilot operations. In *2024 IEEE Conference on Artificial Intelligence (CAI)*, pages 1501–1506. IEEE.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. 2023. Steve-1: A generative model for text-to-behavior in minecraft. *Advances in Neural Information Processing Systems*, 36:69900–69929.

Jijia Liu, Chao Yu, Jiaxuan Gao, Yuqing Xie, Qingmin Liao, Yi Wu, and Yu Wang. 2024. Llm-powered hierarchical language agent for real-time human-ai coordination. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pages 1219–1228.

Zeyi Liu, Arpit Bahety, and Shuran Song. 2023. Reflect: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning*, pages 3468–3484. PMLR.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Runji Lin, Yuqiao Wu, Jun Wang, and Haifeng Zhang. 2024. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Advances in Neural Information Processing Systems*, 37:133386–133442.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Ariana Martino, Michael Iannelli, and Coleen Truong. 2023. Knowledge injection to counter large language model (llm) hallucination. In *European Semantic Web Conference*, pages 182–185. Springer.

Clinton V Oster Jr, John S Strong, and C Kurt Zorn. 2013. Analyzing aviation safety: Problems, challenges, opportunities. *Research in transportation economics*, 43(1):148–164.

Junyeong Park, Junmo Cho, and Sungjin Ahn. 2025. Mrsteve: Instruction-following agents in minecraft with what-where-when memory. In *The Thirteenth International Conference on Learning Representations*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Marc R Schlichting, Vale Rasmussen, Heba Alazzeh, Houjun Liu, Kiana Jafari, Amelia F Hardy, Dylan M Asmar, and Mykel J Kochenderfer. 2025. Leraat: Llm-enabled real-time aviation advisory tool. *arXiv preprint arXiv:2503.16477*.

Abigail Sellen and Eric Horvitz. 2024. The rise of the ai co-pilot: Lessons for design from aviation and beyond. *Commun. ACM*, 67(7):18–23.

Mehmet Burak Şenol. 2020. A new optimization model for design of traditional cockpit interfaces. *Aircraft Engineering and Aerospace Technology*, 92(3):404–417.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. 2024. Drivelm: Driving with graph visual question answering. In *European conference on computer vision*, pages 256–274. Springer.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Vladimir Socha, Lubos Socha, Lenka Hanakova, Viktor Valenta, Stanislav Kusmirek, and Andrej Lalis. 2020. Pilots' performance and workload assessment: Transition from analogue to glass-cockpit. *Applied Sciences*, 10(15):5211.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Jing Yi Wang, Nicholas Sukiennik, Tong Li, Weikang Su, Qianyue Hao, Jingbo Xu, Zihan Huang, Fengli Xu, and Yong Li. 2024b. A survey on human-centric llms. *arXiv preprint arXiv:2411.14491*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024c. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Xinru Wang, Hannah Kim, Sajjadur Rahman, Kushan Mitra, and Zhengjie Miao. 2024d. Human-llm collaborative annotation through effective verification of llm labels. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–21.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 34153–34189.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, and 1 others. 2024e. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Ziming Wang, Qingchen Liu, Jiahu Qin, and Man Li. 2024f. Ensuring safety in llm-driven robotics: A cross-layer sequence supervision mechanism. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9620–9627. IEEE.

Shaoyue Wen, Michael Middleton, Songming Ping, Nayan N Chawla, Guande Wu, Bradley S Feest, Chihab Nadri, Yunmei Liu, David Kaber, Maryam Zahabi, and 1 others. 2025. Adaptivecopilot: Design and testing of a neuroadaptive llm cockpit guidance system in both novice and expert pilots. In *2025 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 656–666. IEEE.

Wei Xiang, Ziyue Lei, Haoyuan Che, Fangyuan Ye, Xueting Wu, and Lingyun Sun. 2025. Hand by hand: Llm driving ems assistant for operational skill learning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 10334–10342.

Rongbing Xu, Shi Cao, Suzanne K Kearns, Ewa Niechwiej-Szwedo, and Elizabeth Irving. 2024a. Computational cognitive modeling of pilot performance in pre-flight and take-off procedures. *Journal of Aviation/Aerospace Education & Research*, 33(4).

Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K Wong, Zhenguo Li, and Hengshuang Zhao. 2024b. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *IEEE Robotics and Automation Letters*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhuorui Zhang, Shanshan Feng, Yang Tiance, Ruobing Huang, Hao Wang, Fu Wang, and Fan Li. 2025. Aviationcopilot: Building a reliable llm-based aviation copilot inspired by human pilot training. *Available at SSRN 5270094*.

A Textual DCS Implement Details

This section will introduce more implementation details for Textual DCS, supplementing the main text and discussing its generalization capabilities.

A.1 Low-Level Semantic Mapping

While DCS World provides a data export API for developers via Lua scripting, manually configuring it to read all necessary data is prohibitively difficult. To circumvent this challenge, we utilize DCS-BIOS, a community-developed tool that directly reads all in-game aircraft-related data, providing us with data packets at a rate of 30 Hz. We parse and save the data from each packet. This step is

also directly adaptable to other aircraft, as the data parsing logic relies only on the aircraft's in-game configuration files.

The raw data provided by DCS is often non-semantic. For example, the pilot's HUD mode is received as "PLT_HUD_MODE" == "1". For this data to be readable by humans and LLMs, the value "1" must be mapped to a semantic label like "A/G" (Air-to-Ground). Since the game does not provide this mapping and no open-source solution exists, we manually annotated the corresponding mappings for all controls of the F-14A model. During this annotation process, we also labeled the cockpit location of each control.

Furthermore, the HOTAS and video data require annotation following the DCS-BIOS style, as the default DCS environment does not support direct interaction with these signals and therefore lacks the corresponding data labels. While extending our work to other aircraft would require a similar one-time annotation effort, it is a manageable and controllable process.

A.2 Data Enhancement

As mentioned in the main text, to improve retrieval accuracy, we synthesize a low-hallucination description for each low-level data point. This section provides further details on this process.

We begin with the base metadata available for each control in DCS, which typically includes a very short, often uninformative text description, its physical type (e.g., knob, switch, or indicator), and its data format (e.g., a transient signal like a button press, or a multi-state continuous signal). To incorporate visual data, we also add a 'screen' type. We then inject this structured metadata, augmented with relevant information retrieved from our document knowledge base via RAG, into an LLM to synthesize a comprehensive description for each individual data point.

We further observed that the LLM could still hallucinate or misinterpret abbreviations for which no relevant documents were retrieved. To mitigate this, we leverage the in-game data taxonomy provided in the game files. For a given category, we provide the LLM with all the data points within that category along with their previously synthesized descriptions as few-shot examples, and prompt it to generate a description for the category itself. This generated category-level description is then added as additional context to the individual description synthesis process. This hierarchical approach further

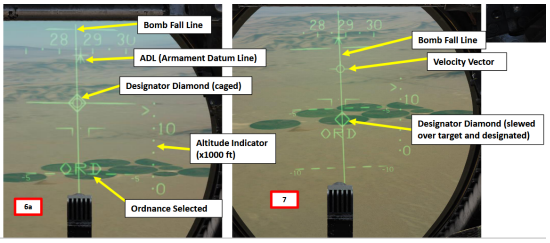


Figure 8: Example of annotated reference images

reduces hallucinations for data-description pairs, resulting in a highly accurate retrieval database.

The code for this data processing and synthesis pipeline is available in the supplementary source code to ensure reproducibility and extensibility. We conducted an ablation study comparing the retrieval database generated by our full synthesis method against a baseline database created using only the minimal in-game descriptions augmented with RAG. The results (Table 2) demonstrate that our method significantly improves description accuracy for data points that originally have minimal semantic information.

Model	Naive RAG	Ours
Gemini-2.5-flash	0.76	0.84
Deepseek-V3	0.90	0.94

Table 2: Effect of Data Enhancement in Recall.

A.3 Multi-Modal Retrival

For visual queries—comprising a target screen name and a specific question—it invokes a multi-modal RAG module: annotated reference images 8 (containing explicit textual markers) are retrieved and provided as few-shot visual prompts. This context enables the VLM to answer the question by mapping the current raw video feed to the correct symbolic state.

Compared to purely textual queries, this visual grounding process inevitably introduces additional latency due to perception and retrieval overhead. To mitigate this issue, we deploy a lightweight VLM (Qwen2.5-VL-7B) locally and process visual inputs at a frequency of 1 Hz in our experiments, which provides a practical balance between responsiveness and accuracy.

B Instruction Generator

This section provides a detailed breakdown of the Instruction Generator within FalconAgent. Unlike autonomous agent systems that can directly act on

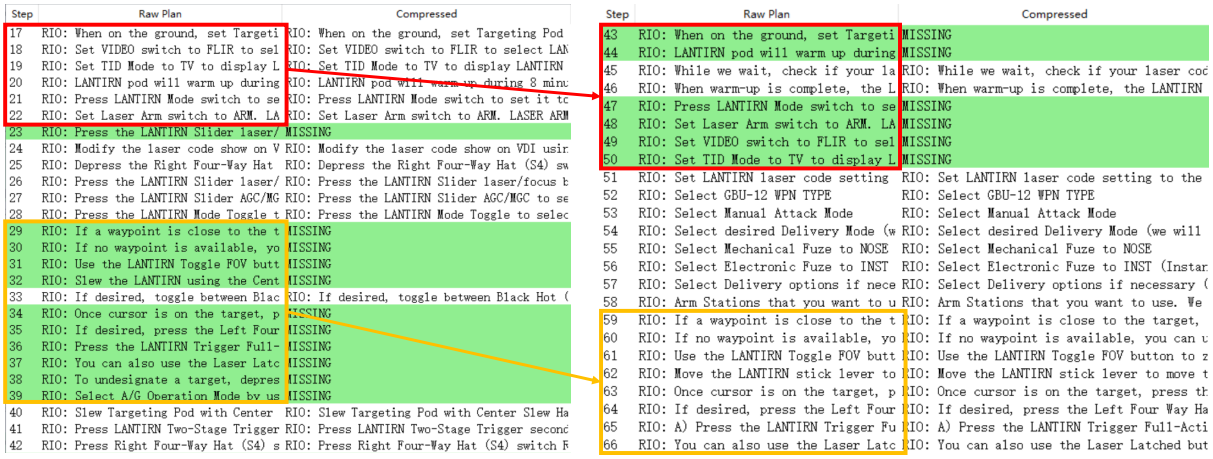


Figure 9: Illustration of the Plan Pruning mechanism. We present an example involving procedural redundancy between two task sequences: LANTIRN Start-Up Procedure (left) and Ground Attack with LANTIRN (right). When both procedures appear in the same execution plan, Plan Pruning identifies and eliminates overlapping operations to streamline execution. Operations highlighted in green indicate steps that have been pruned due to redundancy. Steps marked in red represent operations that must be performed before encountering a target, leading to pruning of similar operations in the second half. Conversely, orange highlights denote operations typically executed when approaching and engaging a target, which causes related steps in the first half to be pruned.operations.

an environment to be quickly validated with quantitative metrics, evaluating the quality of a co-pilot's generated plan is more nuanced. Therefore, we will demonstrate the functionality of our designed modules through concrete examples.

B.1 Raw Plan Generator

Similar to common agent planning systems, we provide the LLM with a predefined list of encapsulated, high-level actions to mitigate hallucinations and constrain the planning space. We experimented with several planning strategies, including Tree-of-Thought with both Depth-First and Breadth-First Search (ToT-DFTS and ToT-BFTS), as well as a simpler Chain-of-Thought (CoT) approach. While ToT strategies have shown excellent performance in many domains, in our specific context, they did not demonstrate a clear advantage over a direct CoT-based planning approach. We have not conducted a more in-depth investigation into this phenomenon and leave the enhancement of this component to future work.

B.2 Plan Pruning

A concrete example of Plan Pruning is shown in Figure 9.

B.3 Task Allocate

A concrete example of Task Allocate is shown in Figure 10.

C Generalization Ability

The current system is fundamentally data-driven and does not require customized training for the utilized models or modules. Therefore, migrating to a new aircraft model can be achieved by simply replacing the data sources within Textual DCS and appropriately adjusting the few-shot examples in the prompts. (If migrating to other environments, such as Microsoft Flight Simulator, the underlying interaction interface would also need to be replaced, but the overall pipeline remains unchanged.) As mentioned in the 3.1.1 and **Appx A.1**, the data extraction process is highly automated. The primary workload lies at the low level, where we need to manually supplement some annotated data not found in the DCS configuration files to address the issue of non-semantic data. For the dual-seat F-14A used in this paper—which features the most sophisticated clickable cockpit and complex internal operations among all high-fidelity simulator modules, far exceeding the complexity of other aircraft types —two members familiar with the aircraft (covering both the pilot and RIO positions) spent approximately 10 hours referencing manuals to annotate hundreds of data points. When migrating to simpler aircraft models or annotating only key operational controls based on specific mission requirements, this duration is expected to be reduced to just a few hours. The adaptive labeling for HOTAS and video data required only a few hours

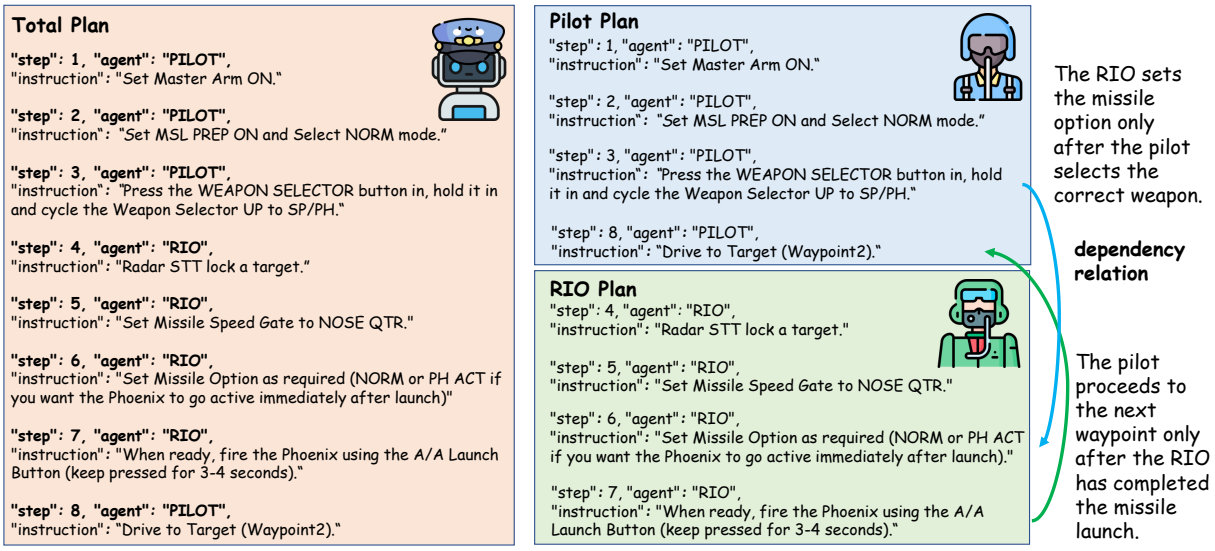


Figure 10: The Multi-Crew Task Allocation process. The initial pruned sequence (left) is analyzed for cross-pilot dependencies. The system then partitions the plan into two parallel instruction streams (right), one for the Pilot and one for the RIO, with synchronization points clearly marked.

by a single person.

D Real-time Performance Optimization

For a copilot system operating in real-time, particularly within the aviation domain, a discussion on system responsiveness is essential. While agent-based systems offer enhanced capabilities, they inherently introduce computational latency. We mitigate this overhead through specific engineering optimizations: **First**, we decouple the functional modules from the central interaction agent. This architecture requires only a lightweight model to perform accurate and rapid function calls, ensuring that the latency for each interaction step remains under one second (< 1s). Although we did not specifically fine-tune a small model for this study, such optimization could foreseeably further enhance the user experience. **Second**, to address computationally intensive processes, we employ a pre-computation strategy. Upon completion of the task planning phase, we immediately execute post-processing for all planned steps in parallel—specifically, data retrieval and verification function generation—and cache the results. During actual flight operations, the agent's tool invocations access these cached results directly. The system only re-processes data for the current step when human feedback intervenes; if such feedback triggers task replanning, the cache is regenerated immediately following the production of the new task sequence.

We further address the real-time performance of individual step verification. For textual queries, verification is effectively instantaneous. For visual queries, under our design, the VLM is only required to produce a binary True/False response, further reducing both computational cost and response latency. While this design effectively minimizes perceived latency during standard operations, ensuring real-time performance remains challenging in extreme scenarios that require frequent task replanning. We attribute this to the inherent inference latency of LLMs rather than a flaw in the system architecture.

E Benchmark Dataset Construction

This section introduces the custom benchmark we created to analyze the retrieval accuracy of Textual DCS and the capabilities of our bidirectional feedback mechanism. The benchmark was constructed by selecting a subset of the extracted high-level actions. For each high-level action, every constituent step was manually annotated with two pieces of ground-truth information: 1) the set of relevant data that must be retrieved for verification, and 2) the corresponding ground-truth verification function. The selected sub-tasks cover the major functionalities of the F-14A and include a diverse range of operational procedures. The specific sub-task names are as follows:

- GET RADAR LOCK FROM PULSE MODE TO PULSE STT

- LANTIRN TARGETING POD START-UP AND LASING PROCEDURE
- SETTING UP TAC/LINK4A DATALINK
- WAYPOINT ENTRY PROCEDURE
- MANUAL IFF OPERATION AND TARGET MARKING PROCEDURE
- ENGINE START AND GROUND POWER DISCONNECTION PROCEDURE
- AIM-9M SIDEWINDER LAUNCH PROCEDURE WITH RADAR TRACKING
- MK-82 BOMB RELEASE PROCEDURE IN COMPUTER TARGET/CCRP MODE
- AIM-54 PHOENIX MISSILE MULTI-TARGET STRIKE PROCEDURE

A brief example of our annotation format is shown in List E. While the full annotated dataset is not provided directly, the complete process for data acquisition and annotation has been made available, allowing the benchmark to be fully reproduced.

```
{
"name":"LANTIRN TARGETING POD START-UP AND
    LASING PROCEDURE",
"content":[
{
"instruction": "When on the ground, set
    Targeting Pod Power Switch to POD.",
"data":["RIO_LANTIRN_PW"],
"func": "result = dcs_states['RIO_LANTIRN_PW']
    == \"POD\""
},
{
"instruction": "Set VIDEO switch to FLIR to
    select LANTIRN video feed in lieu of TCS
    feed for the TID.",
"data":["RIO_LANTIRN_1_FLIR"],
"func": "result = dcs_states['RIO_LANTIRN_1_FLIR
    '] == \"ON\""
},
{
"instruction": "Set TID Mode to TV to display
    LANTIRN feed.",
"data":["RIO_TID_MODE"],
"func": "result = dcs_states['RIO_TID_MODE'] ==
    \"TV\""
},
...]}
```

Example of annotation format.

F LLMs as human-feedback

In our main experiment, we employed an LLM to simulate human feedback to enable a large-scale and reproducible evaluation. However, the validity of this simulation as a reliable proxy for genuine human interaction is a critical assumption that requires verification. This section details the experiment we conducted to validate this approach.

We established two experimental conditions using our benchmark dataset of failure cases: one with feedback sourced from real human participants, who can see the answer like LLMs but are prohibited from revealing any direct or answer-related information in their feedback, and another with feedback generated by our LLM simulator. To compare them, we measured the downstream task performance—the success rate of our main agent in correcting its error after receiving feedback from each source.

	RS	**RS+FB**	**VS**	**VS+FB**
GPT-4o	0.94	0.99	0.81	0.90
Human 1	0.94	0.98	0.81	0.85
Human 2	0.94	0.98	0.81	0.86
Human 3	0.94	0.99	0.81	0.86

Table 3: Comparison between feedback from three human annotators and simulated feedback by GPT-4o. Evaluations are based on the failure set generated from DeepSeek-V3 rollouts, with identical RS and VS applied across all conditions.

As shown in Table 3, we observe that real human feedback and LLM-generated feedback exhibit comparable effectiveness in correcting retrieval data. However, for feedback on the verification function, LLMs provide significantly stronger guidance. This discrepancy arises from the fact that LLMs can "cheat" by implicitly including more answer-revealing signals in their simulated feedback. Therefore, while LLM-simulated feedback is useful for large-scale, reproducible experiments, it cannot be solely relied upon to accurately quantify the performance gains achievable through real human feedback.

Nevertheless, these results still reflect the underlying trend: feedback, whether simulated or real, plays a critical role in improving agent performance. More importantly, the goal of incorporating feedback is not merely performance optimization but establishing an interactive communication channel between humans and AI agents. Even though the impact of human feedback on verification function generation may not be immediately significant, its functional necessity in enabling interpretable, interactive agent behavior remains essential.

G Human Experiment

This section provides supplementary details regarding our human-in-the-loop experiment, including the experimental platform, participants, and procedures, to offer interested readers sufficient detail for reproducibility. It is important to emphasize that the goal of our user study is not to achieve statistical validation through a large-scale experiment, but rather to conduct a formative evaluation of the system's feasibility. Due to the significant challenge of recruiting a cohort of participants large enough to produce statistically significant results, we instead focus on a detailed, case-by-case analysis of each experimental session. This approach allows us to provide a high-quality qualitative analysis to address potential concerns regarding the reliability and practical performance of our work.

G.1 About Digital Combat Simulator World

Digital Combat Simulator (DCS) World is a high-fidelity air combat simulation platform (Figure 11) and is currently the highest-fidelity platform of its kind that is publicly available. It features one-to-one, realistic replications of numerous modern aircraft, with most models designed with fully clickable cockpits, providing an operational experience that closely mirrors that of a real aircraft.

Figure 11: Screenshot from DCS World.

Its built-in Mission Editor provides a rich set of components, including ground units, buildings, and a wide variety of aircraft models. The editor's event system allows for flexible scripting of in-game AI behavior, enabling highly customizable mission design to meet diverse scenario requirements (Figure 12). All of our test mission scenarios were custom-built using this editor, with official missions serving as a reference. Furthermore, the platform's Campaign Editor supports multi-mission, long-range planning, and the various official task and campaign DLCs provide valuable references for mission design.

Figure 12: Screenshot from DCS World Mission Editor.

G.2 About Devices

We constructed a custom-built, high-fidelity F-14A dual-seat cockpit to serve as the platform for our human-AI interaction experiments. This physical cockpit features a full-button replication of both the front and rear seats of the F-14A; even a few buttons that are non-functional in DCS were created as physical controls with signal circuits. All available instrument readouts are outputted to custom display screens, providing a near-real operational experience for the participants.

For the HOTAS (Hands On Throttle-And-Stick) setup, we used the Thrustmaster Warthog (A-10C) throttle and stick set. This is a widely used piece of simulation hardware whose design is well-suited for modern fighter aircraft. The community tutorials from which we extracted expert data also provide detailed mapping and setup configurations for this specific HOTAS across various aircraft models.

Our computational setup consists of two computers, each configured with an Intel i5-13400K CPU, 32GB of RAM, and an NVIDIA GeForce RTX 2070 Super GPU. These machines run the front and rear cockpit stations, respectively, and are connected via a self-hosted DCS server. The primary visual display for the main game window is a Xiaomi TV.

G.3 Mission Detailed

We organize our tasks into three categories: (1) Reconnaissance Mission, (2) Air-to-Air Mission, and (3) Air-to-Ground Mission. Below, we present a detailed description of each task in Figure 14.

G.4 Train Details

The participants without prior F-14A rear cockpit experience underwent a 30-minute acclimatization and training session. First, they were briefed on

Figure 13: The cockpit.

the functions of the various modules in the F-14A's rear cockpit. Next, we specifically highlighted the key buttons and instruments that would be used in the subsequent experimental tasks. Our physical cockpit features labels for most controls, similar to a real aircraft, to mitigate the challenge of unfamiliarity. For the HOTAS device, which has many unlabeled buttons, we provided participants with a mapping diagram that was available for reference throughout the experiment.

The goal of the training was to familiarize participants with the various interaction methods within the cockpit and the use of the HOTAS equipment. We provided simple demonstrations of key operations, such as how to input data via the CAP keyboard, to facilitate a quick onboarding process. Before each task, participants were given a pre-task briefing informing them of the general category of operations and the overall mission objective.

G.5 Excute Details

This section details the specific protocol for each of the three counterbalanced conditions: RAG-Only (Baseline), FalconCopilot (Ours), and the Human Coach.

Baseline: In this condition, the system uses a standard RAG technique to retrieve the most relevant sections from the flight manual based on the current task and displays them to the participants via the UI interface. This condition is representative of many current intelligent cockpit concepts. Participants were instructed to follow the procedures as written in the retrieved manual sections. To ensure a consistent baseline, even the expert participant was required to read the manual step-by-step rather than relying on memory.

FalconCopilot (Ours): In this condition, participants were instructed to follow the step-by-step instructions provided by the co-pilot. As the AI can make errors, participants were allowed to provide corrective feedback if they believed an instruction was incorrect. If a participant was confident that a step was completed correctly despite a system alert, they could use the "Override" function, which was implemented as a dedicated button in the UI to simplify input. When a participant chose to provide corrective natural language feedback, the mission timer was paused until the AI processed the feedback and provided a new response.

Agentic Manual (Open-Loop Guide): In this condition, participants were similarly instructed to follow the step-by-step instructions provided by the co-pilot. However, the bidirectional interaction channel was disabled; participants were prohibited from offering corrective feedback to the AI, even if they suspected an anomaly in the current operation. Correspondingly, the AI ceased to provide runtime verification of the pilot's actions. The completion of each step relied entirely on the pilot's self-judgment, requiring a manual input (clicking "Skip") to advance to the subsequent stage. While this configuration structurally mirrors the open-loop paradigm prevalent in mainstream works, it is methodologically enhanced by the integration of our dynamic task planning capability.

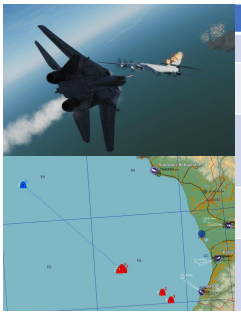
Human Coach: For this condition, an experienced F-14A RIO player, serving as coach with 300+ hours of DCS flight time (Figure 15) specializing in the F-14A module, provided remote, verbal guidance to the participant. The coach observed the participant's game screen via screen sharing and provided step-by-step instructions. A special case was when the expert participant was in this condition; they were instructed to perform the task themselves without external guidance, acting as their own expert coach to achieve their maximum possible speed.

To mitigate the inherent latency of the agent paradigm, we implement a pre-computation and caching strategy. Before each mission begins, FalconCopilot generates the full plan and then preemptively retrieves the necessary data and generates the corresponding verification function for every instruction in the plan in parallel. At runtime, the system directly fetches this cached information, providing near-instantaneous instructions to the user. When a user provides corrective feedback that requires a change, only the affected data and functions are regenerated and overwritten in the cache. This method significantly reduces runtime response latency.

(a) Reconnaissance Mission

Task	Reconnaissance Mission
Location	42°05’36" North 40°28’38" East
Weapon Mounts	8A:AIM-9M, 8B:AIM-7M, 6:MK-83*3, 5:MK-83, 4:MK-83, 3:MK-83*3, 1B:AIM-7M, 1A:AIM-9M
Waypoint	home base:None, hostile area:None, waypoint1:None, waypoint2:None, waypoint3:None
Available Mileage	650nm
Info	You have successfully taken off. Proceed to recon the battle conditions at coordinates 42°09'33" North 41°52'22" East and 42°11'00" North 42°12'17" East. Identify hostile and friendly units at both locations.

(b) Air-to-Air Mission



Task	Air to Air Mission
Location	42°49’49" North 39°20’14" East
Weapon Mounts	8A:AIM-9M, 8B:AIM-54A,6:AIM-54A, 5:AIM-54A, 4:AIM-54A, 3:AIM-54A, 1B:AIM-54A, 1A:AIM-9M
Waypoint	home base:None, hostile area:41°54’44" North 40°33’33" East, waypoint1:None, waypoint2:None, waypoint3:None
Available Mileage	650nm
Info	Our allied forces have detected numerous enemy targets at coordinates 41°54'44" North, 40°33'33" East. Your mission is to intercept and destroy them.

(c) Air-to-Ground Mission

Task	Air to Ground Mission
Location	42°19'45" North 41°24'20" East
Weapon Mounts	8A:AIM-9M, 8B:LANTIRN, 6:GBU-12, 5:GBU-12, 4:GBU-12, 3:GBU-12, 1B:AIM-9M, 1A:AIM-9M
Waypoint	home base:None, hostile area:42°24'23" N 41°34'43‘’, waypoint1:None, waypoint2:None, waypoint3:None
Available Mileage	650nm
Info	Night has fallen, and you are airborne. The enemy ground unit cluster is positioned at 42°24'23" North, 41°34'43" East. Your mission is to eliminate them.

Figure 14: Illustration of the three task categories: (a) Reconnaissance, (b) Air-to-Air, and (c) Air-to-Ground. The central map depicts the mission scenario as configured in the DCS mission editor, while the right panel shows the corresponding state inputs used by FalconCopilot for decision-making.

Figure 15: The coach’s logged flight hours in DCS World via Steam.

G.6 Full Data Analysis

In this section, we present the statistical evaluation of the Human-in-the-Loop experiment. Given the limited within-subject sample size ($N = 6$) and the potential violation of normality assumptions, we adopted a conservative statistical approach. We utilized the non-parametric Wilcoxon signed-rank test for all paired comparisons across both Task Completion Time (TCT) and NASA-TLX metrics. To interpret the magnitude of the results independent of sample size, we report the rank-biserial correlation (r) as the effect size. Values of $|r|$ closer to 1.0 indicate a stronger effect, with $r = 1.0$ representing a case where all participants showed a difference in the same direction. Statistical significance was defined as $p < 0.05$. Table 4 summarizes the complete test statistics.

G.7 Participants and Interviews

As introduced in the main paper, we recruited three participants representing a spectrum of expertise to participate in our study:

- **Novice:** A participant with no prior flight simulation experience.
- **Familiar Player:** A participant experienced with other flight simulators but unfamiliar with the F-14 rear cockpit.
- **Expert:** A proficient F-14 RIO (Radar Intercept Officer) player.

To isolate the variable of procedural task performance, each participant performed their tasks

Metric	Comparison Pair	Mean Diff.	Test Type	p-value	Effect Size (r)
Task 1: Reconnaissance (TCT)					
	Falcon vs. Baseline	-298.66s	Wilcoxon	**0.0312***	-1.000
	Falcon vs. Human	+63.00s	Wilcoxon	**0.0312***	1.000
	Falcon vs. Agentic Manual	-132.50s	Wilcoxon	**0.0312***	-1.000
Task 2: Air-to-Air (TCT)					
	Falcon vs. Baseline	-26.66s	Wilcoxon	**0.0312***	-1.000
	Falcon vs. Human	+12.67s	Wilcoxon	1.0000	0.048
	Falcon vs. Agentic Manual	-22.33s	Wilcoxon	0.0938	-0.857
Task 3: Air-to-Ground (TCT)					
	Falcon vs. Baseline	-229.00s	Wilcoxon	**0.0312***	-1.000
	Falcon vs. Human	+111.66s	Wilcoxon	0.0938	0.810
	Falcon vs. Agentic Manual	-37.84s	Wilcoxon	0.1562	-0.714
Cognitive Load (Weighted NASA-TLX)					
	Falcon vs. Baseline	-39.95	Wilcoxon	**0.0312***	-1.000
	Falcon vs. Human	+8.05	Wilcoxon	0.2188	0.619
	Falcon vs. Agentic Manual	-15.22	Wilcoxon	**0.0312***	-1.000

Table 4: Statistical analysis of Task Completion Time (TCT) and NASA-TLX scores using non-parametric tests. Given the small sample size ($N = 6$), the **Wilcoxon signed-rank test** was used for all comparisons. **Bold** p-values indicate significance at $p < 0.05$.

from the rear seat of the F-14A (the RIO station). A constant, proficient F-14 pilot occupied the front seat for all trials. This setup was chosen for two reasons. First, the majority of the F-14A's complex procedural operations are the responsibility of the RIO, while the front-seat pilot focuses on aircraft control. Second, recruiting multiple participants with proficient piloting skills is challenging and would introduce piloting ability as a significant confounding variable. Using a fixed front-seat pilot mitigates these issues.

Following the experiment, in addition to the NASA-TLX questionnaire (Figure 16), we conducted semi-structured interviews with each participant to gather qualitative feedback on the system:

Novice: "Following the manual was just too difficult for me. For the reconnaissance task, I even misunderstood the objective at first. With the AI, I just had to follow the instructions step-by-step. The best part was the feedback after I completed an action; I didn't have to worry if I had pressed the wrong button, whereas with the manual, I never knew if I was doing it right. Also, just remembering where all the buttons are is hard, but the AI would tell me which panel to look at. Even though it couldn't point to the exact button, it saved me a lot of time searching."

Familiar Player: "I mostly fly the F-18, and the cockpit differences between aircraft are massive. Using a manual is fine for studying on your own, but in a live mission, it's incredibly clumsy to look things up even if it's right in front of you. This copilot system would be a massive help for getting familiar with a new aircraft. Plus, I could ask it about things I didn't understand. I feel like if I flew a few more missions with the AI, I could become an F-14 RIO master too."

Expert: "For me, operating on my own is definitely the fastest; the procedures are mostly muscle memory. However, the Copilot provides one crucial benefit: it prevents me from omitting a step. The F-14 has so many functions, and even I forget things sometimes. For example, during the ground strike mission in the experiment, I forgot to switch to the ground targeting mode. I didn't realize until I was about to release the bomb and nothing happened. In a real combat situation, that mistake would be fatal, as you often only get one chance to strike. Wasting that precious opportunity over something like that is incredibly frustrating."

The qualitative feedback from these different user archetypes suggests that our copilot system provides distinct value to operators irrespective of their initial skill level. Despite the small sample size, the analysis of their experiences points to a generalizable conclusion: our system offers signif-

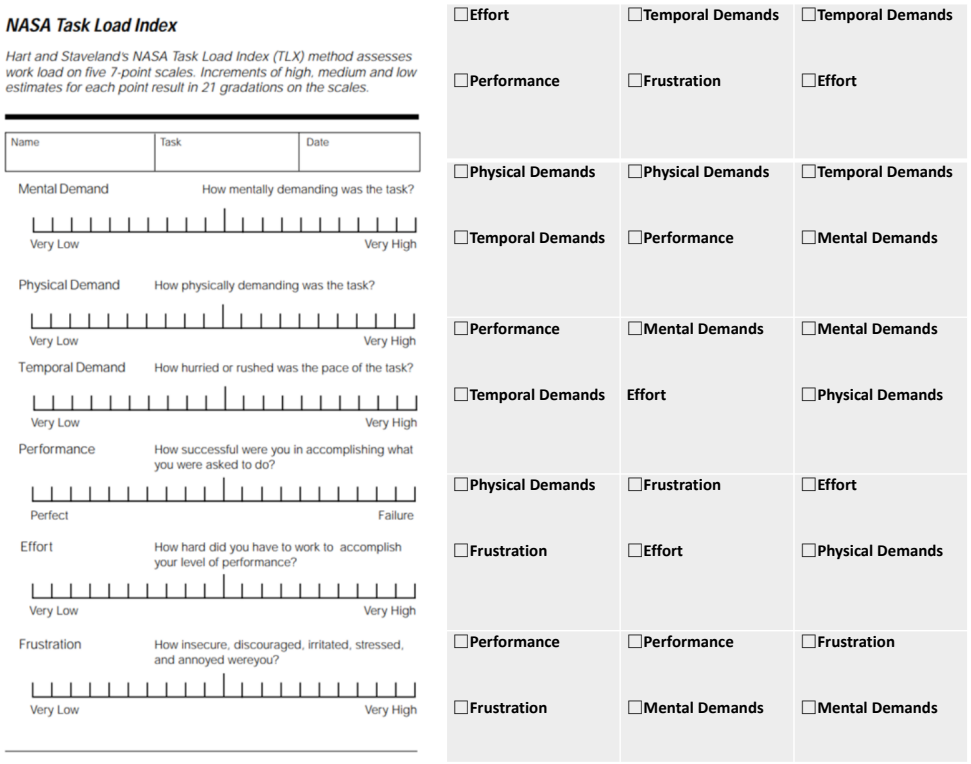


Figure 16: NASA Task Load Index (NASA-TLX) assessment form. The left side displays the six primary subjective workload dimensions: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. Each dimension is rated on a 0–100 scale using a sliding marker. The right side presents the pairwise comparison section, where participants select the more significant factor in each of 15 dimension pairs. These selections are used to compute weighted scores, allowing for a more individualized workload profile.

icant decision-making support on both an operational and a psychological level.

H Prompt

This section will introduce the prompts for each module involved in the FalconCopilot system, divided into: 1) FalconAgent System Prompt; 2)Verification Function Generator System/Query Prompt; 3) Instruction Generator System/Query Prompt; 4)Allocate Module Prompt; 5)Prune Module Prompt; 6)Retrieval Module Prompt; 7)Image Descript Prompt; 8)LLMs As Human-Feedback Simulator Prompt.

```
You are an F-14 aircraft assistance system, and
    your task is to help the pilot complete
    missions. The current task is {subtask_name
    }, and you need to use various tools to help
     the pilot complete the task, including but
    not limited to: providing guidance
    information, checking task execution status,
     analyzing screen information, etc.

You need to call generate_task_plan() once to
    generate a task plan for the current mission
     (Note: you only need to use
    generate_task_plan() once in this
    conversation)

After that, you should call the next_step
    function to get the next step instruction
    first.
Then you should call the get_relevant_data
    function to get the relevant data.
finally, call the
    generate_instruction_and_check_func function
     to generate the instruction and check
    function to detect whether the user finished
     action.
After that, if you are reminded that the current
     action has been completed by the user, call
     next_step() to continue the loop.

Please strictly follow the above steps,and
    refrain from invoking any additional
    functions unless the user explicitly
    requests or requires them.

If there is external input information from the
    user, please call tools or respond to the
    user's questions.

Please ensure your output is concise and does
    not include any irrelevant information. when
     you need to use tools, only output tool
    calls, do not output any other information.

<NOTICE: you can only use one tool at a time.>
```

FalconAgent System Prompt

```
You are the F-14 fighter jet assistance system. Please help the pilot complete the mission according to the following rules:

Next, you need to guide the pilot step by step based on the information above. Please generate guidance information for the pilot based on the previous information (example action sequence, relevant data names, current data).
Note that you need to determine whether the current step needs to be appropriately modified based on the current information. Please output the step you consider appropriate.

And since the pilot may not be aware of the location of the required switches or instruments, you need to provide their positions based on the data.

In addition, when generating specific operations, you also need to generate a judgment function to determine whether the current step is completed, which will be used for subsequent step planning.
This function should be in the form of Python code, and the data involved in the function should be selected from the previously retrieved actual data. Be careful not to arbitrarily generate data that may not exist.
The format reference is as follows:
example:

## normal case
input:
<step info>
Set the MSL PREP switch ON. This commands the WCS to start missile preparation for the AIM-7 and AIM-54. When the individual missiles are tuned and ready the corresponding missile status windows turns white to indicate a ready missile. This should take approx. 2 minutes.
</step info>

<specific data>
name: PLT_MISSLE_PREP
description: Initiates missile preparation sequence for AIM-7 and AIM-54 weapon systems.
api_variant: momentary_last_position
value_scale: [\'0\':\'OFF\', \'1\':\'ON\']
position: ACM Panel

name: PLT_MSL_PREP_ON
description: Indicates missile preparation status for pilot awareness.
api_variant: None
value_scale: [\'0\':\'OFF\', \'1\':\'ON\']
position: ACM Panel
</specific data>

output:
<think>
this step need to Set the MSL PREP switch ON, it located on ACM Panel so I should info pilot to operate it and tell the position.
</think>

<text>
Set the MSL PREP switch ON, it located in ACM Panel.
</text>

<check func>
result = dcs_states['PLT_MSL_PREP_ON'] == "ON"
</check func>

## multi-options case
for some multi-options step, you can use more complex func to detect the status, for example:


## screen type case
When processing api_variant data of screen type, follow these guidelines:
1. For screen-type information (visual interface data), generate string-based queries that will be co-processed with the screen capture by a Vision-Language Model (VLM).

2. Format requirements:
   - Use natural language questions in quotation marks
   - Ensure questions target specific visual elements
   - Maintain aircraft system terminology consistency

NOTICE: When no specific screen type data is provided, absolutely do not attempt to access screens that may not exist without proper evidence. In such cases, prioritize using non-screen data whenever possible.

for example:
the instruction is:
"Line up target with ADL (within 20 deg) in HUD"
and the related data is not able to represent ADL, so you must analyze the HUD screen, so your check function is:

result = self.analysis_screen(screen_name="F_14_HUD", prompt="Does the target have already been line up with ADL(within 20 deg)?")

## format note:
Note: Please output the thinking, the step instructions and check functions strictly in the following format, and do not output anything else:
- Thinking should be contained in <think>...</think> tags.
  you may need choose some data you want use in check function generating, and think what pilot need to know, but your thinking will not show to pilot.

- Step instructions should be contained in <text>...</text> tags.
For step instructions, Please omit some descriptive statements and keep it concise,
```

```
    using the fastest speed to make the user
    understand the current step.

- Check functions should be contained in <check
    func>...</check func> tags.
For check function:
- dcs_states is a dictionary, ensure don't use
    dcs_states['xx']['xx'] in the check function
    , instead use dcs_states['xx'] directly.
- self.analysis_screen is a function in class,
    you can directly use it in the check
    function in correct format.
- If the value scale is given(inform you the
    number's meaning), please use the number's
    meaning instead of using the number;
for example:
name: "PLT_HUD_MODE_AWL"
value_scale:{"1": "ILS","0": "ACL"}
if the value is 1, it means the value is ILS,
    check function should be:
result = dcs_states['PLT_HUD_MODE_AWL'] == "ILS"

- If there is no value scale given, you can use
    0 or 1 (type:int) as the judgment value.
```

Verification Function Generator System Prompt.

```
<step info>
{step_num}:{step_ins}
</step info>

<common data>
{common_data}
</common data>

<specific data>
{specific_data}
</specific data>
Based on the above information, please provide
    the user with a language prompt and a check
    function.

Please remember that some unnecessary data may
    be provided to you, such as redundant
    information, irrelevant information, etc.

Please combine the context and actual scenario
    and use only the required data in the check
    function.

If the api_variant of a data is not None, please
     pay special attention to it.

For example, momentary_last_position means that
    this key is a sequential key, which is
    suitable for detecting whether the key is
    pressed and should not be relied on to
    determine the system status.

if feedback is not empty, please be sure to pay
    attention to the help provided in the
    feedback
<feedback>
{feedback}
</feedback>
```

Verification Function Generator Query Prompt.

```
You are an auxiliary pilot of an F14-A two-seat
    fighter. To solve the given task, you need
    to plan appropriate actions for the two
    pilots.
When user need to change their plan, you should
    re-planning for new query for task.
```

Instruction Generator System Prompt.

```
<TASK>
{task}
</TASK>
This is the information you currently get:
<OBSERVATION>
{obs}
</OBSERVATION>
This is a list of actions you can choose when
    making a decision. Note that if there are
    parentheses, the information in the
    parentheses represents the parameter name.
    Please note that when you use this type of
    action, you need to output the output in the
     form of FUNCTIONNAME(parm1='xxx',...), for
    example, using WAYPOINT NAVIGATION PROCEDURE
    (waypoint='hostile area')
<ACTION_LIST>
{action_list}
</ACTION_LIST>

Please give a complete task planning plan for
    the current TASK. Please refer to the
    following example:
<EXAMPLES>
## Example 1
#INPUT:
<TASK1>
Cold start already finished,We need takeoff from
     Ground Airfield, Destroy Enemy Aircraft
    near 42'04''33''' North 42'20''31''' East,
    then return to homebase.
</TASK1>
<OBSERVATION>
'location': '43'41''13''' North 43'05''35'''
    East',
'weapon mounts': "8A:AIM-9M,8B:LANTIRN,6:GBU
    -12,5:AIM-54,4:AIM-54,3:GBU-12,1B:AIM-9M,1A:
    AIM-9M,MG:M61A1",
'waypoints': 'home base:42'14''33''' North
    42'25''31''' East, hostile area:None,
    waypoint1:None,waypoint2:None,waypoint3:None
    '
'LADAR search mode': 'None',
'available mileage': '500 n mile'
</OBSERVATION>

#OUTPUT:
<thinking>
## Weapon
I am equipped with GBU-12 laser-guided bombs and
     a LANTIRN targeting pod, which allows me to
     conduct precision strikes on targets. I
    also have an M61A1 cannon for close-range
    air combat or strafing ground targets.
    Additionally, I am carrying four AIM-54
    missiles and two AIM-9 missiles, which can
    be used for precise air-to-air engagements.

## Navigation
The navigation waypoint for the mission
    objective has not been entered yet, so I
```

```
    need to input the target location as a
    waypoint and navigate to that position.
    After completing the mission, I need to
    return to base, and since the home base
    waypoint is already provided, I can navigate
     back upon mission completion.

## Task Analysis
The current mission objective is to destroy
    enemy airborne units at a specific location.
     First, I should navigate and fly to the
    vicinity of the target point. Based on my
    available weapons, I should prioritize using
     the AIM-54 or AIM-9 missiles to conduct
    long-range precision strikes against enemy
    aircraft, utilizing radar to lock onto the
    target. After completing the engagement, I
    will return to base and land.
</thinking>

<PLAN>
WAYPOINT ENTRY PROCEDURE(waypoint_type='hostile
    area',loc="42'04''33''' North 42'20''31'''
    East")
SHORE TAKEOFF PROCEDURE
WAYPOINT NAVIGATION PROCEDURE(waypoint='hostile
    area')
DRIVE TO TARGET PROCEDURE(waypoint='hostile area
    ')
TWS MANUAL MODE - TARGET HOOK WITH TID CURSOR
AIM-54 PHOENIX MISSILE SINGLE-TARGET STRIKE
    PROCEDURE BY PILOT(AIM-54 mounting position
    ='5,4')
WAYPOINT NAVIGATION PROCEDURE(waypoint='home
    base')
DRIVE TO TARGET PROCEDURE(waypoint='home base')
SHORE LANDING PROCEDURE
</PLAN>

## Example 2
#INPUT:
<TASK2>
fly to 42'04''33''' North 42'20''31''' East and
    destroy the enemy.
</TASK2>
<OBSERVATION>
'location': '43'41''13''' North 43'05''35'''
    East',
'weapon mounts': "8A:AIM-9M,8B:LANTIRN,6:GBU
    -12,5:GBU-12,4:GBU-12,3:GBU-12,1B:AIM-9M,1A:
    AIM-9M,MG:M61A1",
'waypoints': 'home base:None, hostile area
    :42'04''33''' North 42'20''31''' East,
    waypoint1:None,waypoint2:None,waypoint3:None
    '
'LADAR search mode': 'None',
'available mileage': '500 n mile'
</OBSERVATION>

#OUTPUT:
<thinking>
## Weapon
I am equipped with GBU-12 laser-guided bombs and
     a LANTIRN targeting pod, which enables me
    to conduct precise ground strikes. I also
    have an M61A1 cannon for close-range air
    combat or strafing ground targets, and four
    AIM-54 missiles for effective air-to-air
    engagements.

## Navigation
The navigation waypoint for the mission
    objective has already been set to "hostile
    area." Since there is no mention of
    returning to base, there is no need to input
     additional waypoints. I can proceed
    directly to the designated target waypoint.

## Task Analysis
The current mission objective is to destroy
    enemy ground units. Given my available
    weapons, I should use the GBU-12 bombs in
    conjunction with the LANTIRN pod to carry
    out a precision strike on the enemy armored
    vehicles. As the aircraft has already taken
    off, there is no need for cold start or
    takeoff procedures. I will navigate to the
    target area and execute the ground attack
    upon arrival, and wait for next task.
</thinking>

<PLAN>
WAYPOINT NAVIGATION PROCEDURE(waypoint='hostile
    area')
DRIVE TO TARGET PROCEDURE(waypoint='hostile area
    ')
GBU-12 LASER-GUIDED BOMB DEPLOYMENT PROCEDURE
    WITH LANTIRN
</PLAN>

</EXAMPLES>
Please note that your output only needs to
    include the thinking and the action sequence
     you planned, and do not give any additional
     content, include labels like <PLAN> and </
    PLAN> in your output.
you need use "<thinking>...</thinking>" to
    include your thinking, and use "<PLAN>...</
    PLAN>" to include the action sequence you
    planned.
```

Instruction Generator Query Prompt.

```
This is an operation sequence from the F-14
    cockpit, where certain actions can be
    performed concurrently by the two crew
    members (the Pilot and the RIO). However,
    some steps require the other crew member to
    complete a prior action before proceeding.

Based on the provided sequence, identify the
    steps that depend on the other crew member's
     actions.
Only focus on the dependencies between crew
    members - there is no need to identify
    dependencies between steps performed by the
    same person.
You should output in the following format:
Because of the reason explained, <step X> must
    be executed after <step Y>.
...

##Example
task input:
[['step': 1, 'agent': 'RIO', 'instruction': '
    Verify that the Liquid Cooling Switch has
    been set to ON (FWD). This controls the
    liquid cooling system for the AWG-9 and AIM
    -54 missile'], ['step': 2, 'agent': 'PILOT',
     'instruction': 'Set Master Arm switch - ON
```

```
    (UP)'], ['step': 3, 'agent': 'PILOT', '
    instruction': 'Set the MSL PREP switch ON.
    This commands the WCS to start missile
    preparation for the AIM-7 and AIM-54. When
    the individual missiles are tuned and ready
    the corresponding missile status windows
    turns white to indicate a ready missile.
    This should take approx. 2 minutes.'], ['
    step': 4, 'agent': 'PILOT', 'instruction': '
    Press the WEAPON SELECTOR button in, hold it
     in and cycle the Weapon Selector UP to SP/
    PH.'], ['step': 5, 'agent': 'PILOT', '
    instruction': 'Press the WEAPON SELECTOR
    button again to toggle between Sparrows (SP)
     and Phoenixes (PH).'], ['step': 6, 'agent':
     'PILOT', 'instruction': 'Select NORM
    Missile Mode'], ['step': 7, 'agent': 'RIO',
    'instruction': 'Set Missile Speed Gate to
    NOSE QTR.'], ['step': 8, 'agent': 'RIO', '
    instruction': 'Set Missile Option as
    required (NORM or PH ACT if you want the
    Phoenix to go active immediately after
    launch)'], ['step': 9, 'agent': 'RIO', '
    instruction': 'Set Radar WCS Mode to TWS
    AUTO (Track While Scan)'], ['step': 10, '
    agent': 'RIO', 'instruction': 'As targets
    are scanned they will automatically be
    numbered in terms of priority (1 = highest
    priority).'], ['step': 11, 'agent': 'RIO', '
    instruction': 'You do not need to radar lock
     a target; you just have to launch weapons
    and the TWS mode will automatically pick
    which target is the highest priority for you
     and launch a Phoenix at it.'], ['step': 12,
     'agent': 'RIO', 'instruction': 'When ready,
     fire the Phoenix using the A/A Launch
    Button (keep pressed for 3-4 seconds).
    Missile will track by itself the target with
     the highest priority (1). TWS priority
    numbers are to the right of contact symbols,
     while target altitude is displayed to the
    left in tens of thousands of feet.'], ['step
    ': 13, 'agent': 'RIO', 'instruction': 'On
    the TID, a timer in seconds (TTI, Time to
    Impact in seconds) will appear next to the
    selected target.'], ['step': 14, 'agent': '
    RIO', 'instruction': 'If you use the A/A
    Launch Button again, TWS will automatically
    switch to the target with the next highest
    priority (2). And fire the missile on this
    target. Keep using the A/A Launch Button
    until all missiles are launched.'], ['step':
     15, 'agent': 'RIO', 'instruction': 'And
    that's it! You have now performed a "Six
    Shooter" (engaged six targets almost
    simultaneously).']]

output:
Because of the cooling system's relationship to
    missile preparation, <step 3> must be
    executed after <step 1> Verify Liquid
    Cooling Switch is ON.
Because weapon configuration by the front seat
    crew is required before firing, <step 12>
    must be executed after <step 6> Select NORM
    Missile Mode.

Note: Your output should only include the
    identified dependency relationships and
    their reasons, without any additional
    content.

Here is the current task:
{task}
```

Allocate Module Prompt.

```
You are an expert F-14 Tomcat checklist analyst.
     Your task is to review a sequence of
    operational steps and identify redundant
    actions. A redundant action is a step that
    repeats a previous action or is made
    unnecessary by a previous step.

Analyze the provided task list in TWO STEPS:

## STEP 1: IDENTIFICATION
First, identify and categorize the following
    types of items:
1. **DUPLICATE ITEMS**: Steps that repeat the
    same action as a previous step
2. **MULTI-CONDITION ITEMS**: Steps that have
    multiple conditions or contexts that might
    make them redundant

For each identified item, list it in this format:

- DUPLICATE: Step X duplicates Step Y (same
    action: [describe the action])
- MULTI-CONDITION: Step X has multiple
    conditions [list the conditions]

## STEP 2: ANALYSIS AND SELECTION
For each item identified in Step 1, analyze
    which step should be kept and which should
    be removed. Consider:
- Timing and sequence logic
- Context and conditions
- Operational efficiency
- Safety considerations

Output your final recommendations in this format:

**KEEP <Step X>, REMOVE <Step Y>**: [explain the
     reasoning]

Warning: Format like <Step A&B> cannot be parsed
     correctly, please avoid using this format."

## Example
task input:
[
 {'step': 1, 'agent': 'PILOT', 'instruction': '
    Set Master Arm switch to ON (UP)'},
 {'step': 2, 'agent': 'RIO', 'instruction': '
    When on the ground, set Targeting Pod Power
    Switch to POD.'}
 {'step': 3, 'agent': 'RIO', 'instruction': 'Set
     Radar mode to RWS'},
 {'step': 4, 'agent': 'RIO', 'instruction': 'Use
     the LANTIRN Toggle FOV button to zoom in or
     out as required.'},
 {'step': 5, 'agent': 'PILOT', 'instruction': '
    Verify Master Arm switch is in the ON
    position'},
 {'step': 6, 'agent': 'PILOT', 'instruction': '
    Select weapon type to AIM-54'},
 {'step': 7, 'agent': 'PILOT', 'instruction': '
    Arm the Master Arm switch'},
 ...
```

```
 {'step': 15, 'agent': 'RIO', 'instruction': '
    When on the ground, set Targeting Pod Power
    Switch to POD.'}
 {'step': 16, 'agent': 'RIO', 'instruction': '
    Use the LANTIRN Toggle FOV button to zoom in
     or out as required.'},
]

output:
## STEP 1: IDENTIFICATION
- DUPLICATE: Step 5 duplicates Step 1 (same
    action: Master Arm switch verification)
- DUPLICATE: Step 7 duplicates Step 1 (same
    action: Master Arm switch operation)
- DUPLICATE: Step 15 duplicates Step 2 (same
    action: Targeting Pod Power Switch setting)
- DUPLICATE: Step 16 duplicates Step 4 (same
    action: LANTIRN FOV adjustment)

## STEP 2: ANALYSIS AND SELECTION
**KEEP <Step 1>, REMOVE <Step 5>**: Step 1 sets
    the Master Arm switch, and Step 5 only
    verifies it. The verification is redundant
    since the setting action is already
    performed.
**KEEP <Step 1>, REMOVE <Step 7>**: Step 7
    repeats the same Master Arm operation as
    Step 1 without any intermediate weapon
    changes, making it redundant.
**KEEP <Step 2>, REMOVE <Step 15>**: Step 2 sets
     the Targeting Pod Power Switch on the
    ground, which is the correct timing. Step 15
     repeats this action later unnecessarily.
**KEEP <Step 16>, REMOVE <Step 4>**: FOV
    adjustment should be done later when closer
    to the target, so Step 16 is more
    appropriate than Step 4.

Note: Your output should follow this exact two-
    step format with clear identification and
    analysis sections.

Here is the current task list to analyze for
    redundancy:
{task}
```

Prune Module Prompt.

```
#### Instruction ####
You are a pilot of F-14 fighter, now you need to
     execute an operation, please generate the
    related instrument data and give a
    description of the instrument.

#### Example ####
For example:
command:
Press the TRIGGER SECOND STAGE (SPACE) on the
    stick to fire missile once seeker is
    tracking a good infrared source.
output:
PLT_WEAPON_SELECTOR: Selects and activates
    weapon systems during flight operations.

Here are some other output format you can use:

RIO_RADAR_PDSTT: Enables precise tracking of a
    single target using Pulse Doppler radar.
PLT_HUD_MODE_LAND: Activates HUD landing mode
    for approach and touchdown guidance.
RIO_LANTIRN_Right_S4_Hat: Controls LANTIRN pod
    functions including QADL/QHUD, QDES, QSNO
    modes and declutter cycling via four-way hat
    .

You can also get the image of a specified screen,
     likes:

F_14_HUD: Displays critical flight and weapon
    data directly in pilot's forward field of
    view.
Please note: When requesting a screen image,
    only provide the monitor name and its
    concise description. Do not include any
    extra adjectives or embellishments, as such
    information typically does not exist.
For example, Data like "PLT_HUD_ADL: Displays
    the aircraft's Aspect Display Line for
    target alignment." doesn't exsist, to get
    data ADL on HUD, just use "F_14_HUD"


remember there are many kinds of indicator to
    use, likes:

RIO_SAM_LIGHT: Indicates SAM system status and
    warnings for the RIO.
PLT_SW_COOL_ON: Indicates Sidewinder missile
    cooling system activation for the pilot.

Please classify two kind of keyboard like:
RIO_CAP_BTN_6:  Selects function 6 in the Combat
     Air Patrol (CAP) menun, not used for
    specific number input.
RIO_CAP_LONG_6: Selects Combat Air Patrol
    longitude or number '6' for specific number
    input.

Both of these buttons belong to the "button x"
    category, but when you need to input a
    specific number, use "RIO_CAP_XXXX_x". "
    RIO_CAP_BTN_x" is more for CAP menu
    hierarchy operations. Please distinguish
    between the two.
if you need to input a specific number, use
    RIO_CAP_XXXX_x, if you need to operate the
    CAP menu hierarchy, use RIO_CAP_BTN_x.

#### Format Rules ####
- Mandatory pattern: <EXACT_DEVICE_NAME>:
    Concise description
- Forbidden elements:
Introductory phrases (e.g., "used for", "
    function is")
Bullet points/paragraph breaks
Never use "PLT_TRIGGER_SECOND_STAGE" in the
    output, it unexsist.

#### Command ####
Here is the command you need to generate the
    related instrument data:
{command}

#### Feedback ####
If you have feedback, it will be given in below,
     you should use the feedback to generate the
     related instrument data:
{feedback}
```

Retrieval Module Prompt.

```
You are a professional image analyst. Please
    describe the image content concisely and
    clearly, focusing on key details.
Based on the information provided by the
    filename and folder name, accurately
    describe the image content. Do not directly
    mention the filename in your description.
The description should be objective and accurate,
     without adding content that is not in the
    image. The description should be concise and
     not exceed 100 words.
```

Image Descript Prompt.

```
You are an experienced F-14 fighter jet Radar
    Intercept Officer (RIO) and flight
    instructor. Your mission is to evaluate an
    AI copilot.
The AI copilot has made a mistake while
    executing a task. You must act like a real
    instructor and provide a helpful hint to
    guide it toward the correct solution.

Your feedback must adhere to the following
    golden rules:
1.  **Absolutely do not provide the complete,
    correct answer!** This is the most important
     rule.
2.  Your feedback should be a single, concise
    hint, not a lengthy explanation.
3.  For a "retrieval_failure", you should hint
    at a line of reasoning it might have missed.
     For example, say "You seem to have
    overlooked checking the radar's operational
    mode," instead of "You should retrieve the
    key RIO_RADAR_MODE."
4.  For a "check_func_failure", you should point
     out the flaw in its logic, using the
    provided data context (like value scales).
    For example, say "The condition for
    confirming missile readiness seems
    incomplete," instead of giving the correct
    code `result = dcs_states['...'] == "RDY"`.
5.  Maintain a professional and calm instructor'
    s tone.
```

LLMs As Human-Feedback Simulator Prompt.