CabinProAct: Visually Triggered Proactive Intervention for Intelligent Cabin Agents

Anonymous ACL submission

Abstract

Traditional in-vehicle agents mainly follow an instruction-following paradigm, where actions are executed only after explicit user commands. However, real-world cabin scenarios often require agents to proactively determine whether intervention is necessary by jointly reasoning over visual and state inputs. To address this gap, we propose **CabinProAct**, a multimodal benchmark for proactive intelligent cabin agents. In this setting, agents must jointly reason over in-cabin and out-cabin visual information, vehicle states, and executable APIs to autonomously decide whether to intervene and generate corresponding actions. We further introduce an execution-verification-based synthetic data generation framework to automatically construct execution-consistent proactive interaction data. We evaluate six state-of-the-art multimodal models. The results indicate that execution-grounded synthetic supervision effectively transfers to real-world in-cabin settings, enabling more robust proactive responses in real-world cabin scenarios.

1 Introduction

Recent multimodal large language model (MLLM) agents have demonstrated strong capabilities in visual understanding, tool use, and interactive decision-making (Xi et al., 2025; Mialon et al., 2023; Qin et al., 2024; Koh et al., 2024; Xie et al., 2024). Existing vehicle agents and embodied assistants, however, largely operate under an instruction-following paradigm: the system waits until users explicitly issue commands before determining how to respond (Du et al., 2024; Chen et al., 2024; Giebisch et al., 2025; Khiabani et al., 2025). While such reactive interaction is effective for conventional software control and task execution, it remains insufficient for intelligent cabin environments, where many assistance opportunities emerge implicitly from surrounding visual contexts rather than explicit user requests.

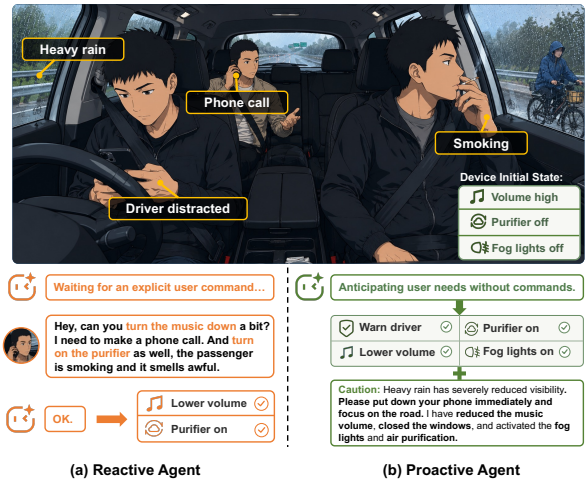


Figure 1: Task intuition. A reactive cabin agent waits for an explicit user request and therefore may remain silent even when the cabin state indicates latent assistance opportunities. A proactive cabin agent instead grounds the same visual evidence in structured vehicle state and executable APIs, then communicates and intervenes when warranted.

In real-world driving scenarios, proactive cabin assistants are expected to autonomously discover latent intervention opportunities from multimodal observations (Schilit and Theimer, 2002; Lu et al., 2025; Zhang et al., 2024; Liu et al., 2025). For example, as illustrated in Figure 1, an agent may observe a driver using a phone during heavy rain, passengers smoking inside the vehicle, or incoming phone calls occurring while loud music is playing. In these situations, the driver and passengers do not explicitly request assistance. In such cases, the agent must first perceive environmental cues, infer potential risks or comfort-related needs, and determine whether intervention is necessary.

More importantly, proactive intervention is inherently conditional on the current vehicle state. Identical visual observations may require entirely dif-

ferent actions depending on existing system configurations. For instance, smoking behavior only warrants activating air purification when the purifier is currently disabled; lowering music volume during a phone call becomes unnecessary if the audio volume is already sufficiently low. Consequently, proactive cabin interaction is fundamentally different from traditional instruction-conditioned API execution (Schick et al., 2023; Li et al., 2023; Patil et al., 2024; Trivedi et al., 2024; Yao et al., 2024). It requires jointly solving three tightly coupled challenges: ❶ discovering intervention opportunities from visual observations, ❷ reasoning whether intervention is necessary under structured vehicle states, and ❸ generating executable cabin control actions only when warranted. Meanwhile, the agent must also learn calibrated abstention to avoid unnecessary or redundant operations.

To address these challenges, we introduce **CabinProAct**, a benchmark for visually-triggered conditional proactive intervention in intelligent cabin agents. Built upon an executable vehicle simulation environment (Yang et al., 2025a), CabinProAct reformulates vehicle interaction from reactive instruction execution into proactive intervention reasoning. Given multimodal interior and exterior observations, structured vehicle states, and executable cabin APIs, the agent must determine whether intervention is necessary and generate executable control programs only when warranted.

Another major challenge lies in data scalability. Real-world proactive interaction data is inherently difficult to collect and annotate because intervention opportunities are sparse, highly conditional, and strongly coupled with dynamic vehicle states. Moreover, evaluating proactive actions requires executable consistency verification rather than surface-level textual matching (Qin et al., 2024; Guo et al., 2024; Tang et al., 2024). These limitations make purely real-world collection insufficient for training robust proactive cabin agents. To support scalable training for proactive agents, we further propose an execution-grounded synthetic data generation framework. The framework automatically constructs proactive interaction scenarios through structured state initialization, multimodal scene generation, executable action synthesis, semantic consistency consolidation, and sandbox-based execution verification. This design enables controllable generation of large-scale proactive interaction data while maintaining executable state consistency between visual observations, vehicle states, and generated actions.

Based on this framework, we construct 2169 synthetic training samples and a carefully curated 350-case real-world evaluation set. We evaluate six state-of-the-art open-source and proprietary multimodal models under zero-shot, few-shot, and supervised fine-tuning settings. Results show that execution-grounded synthetic supervision significantly improves intervention precision and abstention capability, demonstrating strong synthetic-to-real transferability for proactive cabin agents.

Our contributions are summarized as follows:

- We formalize visually-triggered conditional proactive intervention as a new executable multimodal agent task requiring visual opportunity discovery, state-conditioned reasoning, action generation, and calibrated abstention.
- We introduce CabinProAct, a synthetic-to-real benchmark for proactive cabin agents featuring multimodal observations, structured vehicle states, executable cabin APIs, and real-world evaluations.
- We propose an execution-grounded synthetic data generation framework that enables scalable construction of proactive interaction data with executable state consistency verification.
- We demonstrate that execution-grounded synthetic supervision substantially improves real-world proactive intervention capability, highlighting the effectiveness of synthetic-to-real training for future proactive agents.

2 Related Work

Proactive agents. Proactive agents aim to infer latent user needs from context rather than waiting for explicit instructions. This direction builds on context-aware computing and has recently been studied in general LLM agents, work-context agents, planning assistants, implicit-intention understanding, proactive dialogue, mobile GUI agents, and open-world context-aware agents (Schilit and Theimer, 2002; Lu et al., 2025; Tang et al., 2026; Zhang et al., 2024; Wu et al., 2024; Qian et al., 2024; Deng et al., 2023; Yang et al., 2025b; Kong et al., 2026; Yang et al., 2026; Liu et al., 2025). As summarized by the representative comparison in Table 1, these works motivate the proactive-agent setting, but they generally

Work	Domain	Vision	Cabin	State	Exec.	Proact.	Abstain	Real Eval
ProactiveAgent (Lu et al., 2025)	General agent	✗	✗	✗	✗	✓	✓	✗
ProAgentBench (Tang et al., 2026)	Work context	✗	✗	✗	✗	✓	✓	✓
FingerTip (Yang et al., 2025b)	Mobile GUI	✓	✗	✗	✓	✓	✓	✓
ProactiveMobile (Kong et al., 2026)	Mobile device	✓	✗	✓	✓	✓	✓	✗
ContextAgent (Yang et al., 2026)	Open-world context	✗	✗	✓	✗	✓	✓	✗
Proactive IVCA (Du et al., 2024)	In-car dialogue	✗	✓	✗	✗	✓	✗	✗
Cockpit-Llama (Chen et al., 2024)	Cockpit intent	✗	✓	✗	✗	✗	✗	✗
DriveLM (Sima et al.)	Autonomous driving	✓	✗	✗	✗	✗	✗	✓
Dolphins (Ma et al., 2024b)	Autonomous driving	✓	✗	✗	✗	✗	✗	✓
VehicleWorld (Yang et al., 2025a)	Cabin API	✗	✓	✓	✓	✗	✗	✗
AppWorld (Trivedi et al., 2024)	Software apps	✗	✗	✓	✓	✗	✗	✗
CabinProAct (ours)	Cabin proactive agent	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison with related benchmarks and environments. Vision denotes visual observations; Cabin denotes vehicle-cabin or occupant-centered settings; State denotes explicit structured environment or device state; Exec. denotes executable API/action evaluation; Proact. denotes proactive behavior without an explicit user command; Abstain denotes explicit no-action calibration. ✗ denote partial support.

focus on assistance suggestion, clarification, context understanding, or mobile/workplace actions rather than executable cabin intervention grounded in dense vehicle state and visual evidence.

In-vehicle assistants and driving VLMs. In-vehicle assistant research studies proactive dialogue, driver intent prediction, factual reliability, cockpit safety constraints, and user-requested function calling in intelligent vehicle settings (Du et al., 2024; Chen et al., 2024; Giebisch et al., 2025; Ma et al., 2024a; Khiabani et al., 2025). Driving VLM benchmarks instead focus on autonomous-driving perception, prediction, and planning over external road scenes (Sima et al.; Ma et al., 2024b; Tian et al., 2024). These directions are complementary to our setting. CabinProAct focuses on passenger-facing cabin intervention, where the relevant decisions involve audio, windows, purification, lighting, notifications, and localized comfort or safety controls, and where the correct action depends on the interaction between occupant behavior, external context, and current device state.

Tool-use and executable evaluation. Tool-use and agent benchmarks evaluate API calling, visual interface interaction, and execution-based outcomes in software or simulated environments (Schick et al., 2023; Li et al., 2023; Qin et al., 2024; Patil et al., 2024; Liu et al., 2024; Koh et al., 2024; Xie et al., 2024; Yao et al., 2024; Trivedi et al., 2024; Tang et al., 2024). These works show the importance of judging agents by executed outcomes rather than surface-form text. VehicleWorld provides a structured multi-device cabin API substrate (Yang et al., 2025a), but it follows a reactive function-calling setting without visual observations or proactive abstention. CabinProAct extends executable evaluation to proactive cabin agents: predicted programs are judged by sandbox state transitions, and unnecessary actions are penalized to measure calibrated abstention.

3 Execution-Grounded Proactive Data Construction

3.1 Problem Formulation

We formulate proactive cabin assistance as an execution-grounded data construction problem. Each instance must specify not only what is visible in the vehicle cabin, but also whether the situation warrants intervention under the current vehicle state and, if so, which executable cabin-control program should be produced.

Formally, an instance consists of a visual observation I, an initial vehicle state S_0, and an API document $\mathcal{A}$. The observation captures in-cabin occupant behaviors and, when available, out-cabin environmental cues. The state S_0 describes fine-grained device configurations, such as audio, windows, air purification, lighting, locks, displays, and seat-specific controls. Given these inputs, the agent predicts a proactive API program as defined in:

$$\pi = f_\theta(I, S_0, \mathcal{A}), \tag{1}$$

Here, π may be empty when no intervention is warranted, and π^* denotes the oracle program. The predicted and oracle programs are executed from the same initial state, yielding $S_p = \mathrm{Exec}(S_0, \pi)$

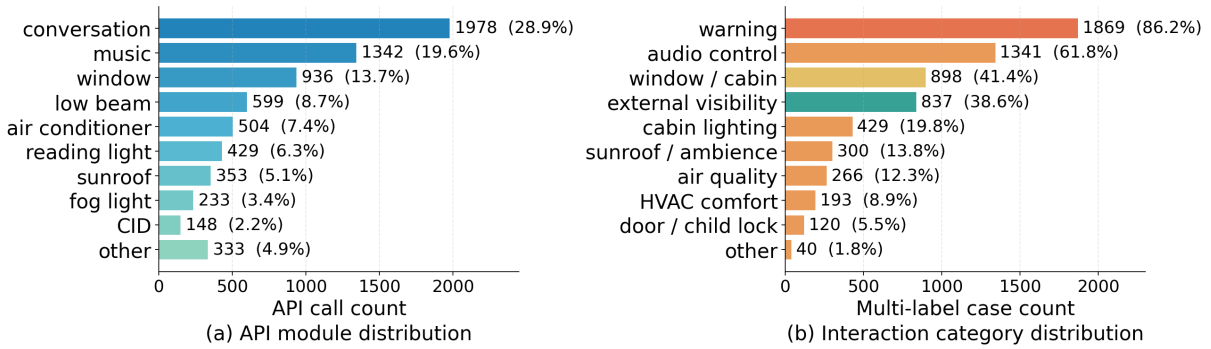


Figure 2: Distribution of the 2169-case synthetic training pool. API modules are counted by API-call frequency, while interaction categories are multi-label case counts; therefore, interaction percentages need not sum to 100%.

and $S_g = \mathrm{Exec}(S_0, \pi^*)$, respectively. The supervision target is therefore an executable state transition rather than a surface-form textual response.

This formulation makes proactive cabin assistance fundamentally conditional. The same visual evidence may require different labels depending on S_0: a smoking passenger should trigger air purification only when purification is disabled or insufficient, and a phone-call scene should reduce audio volume only when the current volume is disruptive. Thus, each valid instance couples three decisions: discovering a latent intervention opportunity, determining whether the current state requires action, and generating executable API calls only when justified. For agent-initiated physical control, the program should also include user-facing communication, such as speech or display notification, so that occupants remain aware of the intervention and the system mitigates the controllability risks introduced by proactive actions.

3.2 Necessity of Synthetic Generation

Training proactive cabin agents requires scalable execution-grounded supervision, which is difficult to obtain from real-world data alone. Unlike instruction-following data, proactive data cannot be collected simply by asking users to issue commands: the agent must learn when an implicit situation should trigger intervention, when it should abstain, and which executable actions are valid under the current vehicle state.

This difficulty comes from three properties of proactive cabin interaction. First, intervention opportunities are sparse in natural cabin recordings, while rare safety- or comfort-related triggers are exactly the cases needed for robust training. Second, labels are highly state-dependent: the same visual scene may require action, no action, or a different action depending on whether the relevant device state already satisfies the latent need. Third, labels must be executable rather than descriptive. A phrase such as "turn on air purification" is insufficient unless it is translated into valid API calls, satisfies parameter constraints, executes successfully from S_0, and produces the intended state transition. For proactive physical actions, the label should also contain appropriate user-facing communication to avoid silent agent-initiated control.

These challenges motivate an execution-grounded synthetic generation strategy. We construct controlled scenario–state pairs, generate visually grounded observations, synthesize candidate API programs, and verify the resulting labels through sandbox execution. This allows us to scale proactive supervision while preserving the task's key property: each label is grounded in the interaction between visual evidence, structured vehicle state, and executable cabin APIs.

3.3 Execution-Grounded Data Generation Pipeline

Figure 3 illustrates our synthetic data generation pipeline. The goal is to construct proactive cabin interaction data whose labels are executable programs with verified state transitions. We instantiate the pipeline on top of VehicleWorld (Yang et al., 2025a), a deterministic vehicle-control simulator that provides structured cabin states, callable device APIs, and executable state transitions. VehicleWorld serves as the execution substrate for initializing states, replaying candidate programs, and verifying state-change consistency. The pipeline contains three stages.

Scenario–State initialization. We first sample candidate cabin situations together with executable

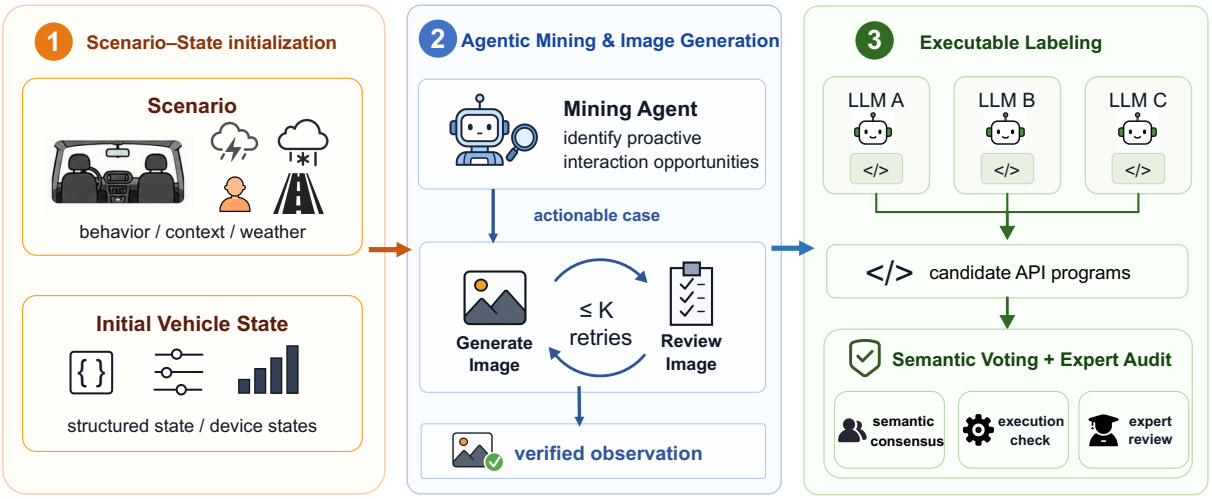


Figure 3: Overview of CabinProAct data construction. A scenario–state gym samples candidate cabin situations and executable initial states. An agentic mining and image-generation stage identifies proactive interaction opportunities and verifies visual observations. Executable labels are produced by multiple LLMs and consolidated through semantic voting, execution checking, and expert audit.

initial vehicle states. Each scenario specifies contextual factors such as occupant behavior, cabin context, and exterior conditions, while the initial state defines fine-grained device configurations including audio, windows, air purification, lighting, locks, displays, and seat-specific controls. This stage follows a random-first design: instead of predefining a target action, we first construct a diverse scenario–state pair and leave the decision of whether it is actionable to the subsequent mining stage. This reduces target-action bias and encourages coverage of both intervention and abstention cases.

Agentic mining and image generation. Given a sampled scenario–state pair, an LLM-based mining agent judges whether the situation contains a plausible proactive intervention opportunity. The judgment is conditioned on both visual context and vehicle state, since a visually salient event is not actionable if the current state already satisfies the corresponding need. For intervention-oriented data, actionable pairs are passed to image generation. For non-intervention cases, we specifically sample normal scenarios or already-satisfied vehicle states, which are incorporated into the same construction pipeline and ultimately yield abstention labels. We render the selected cases into visual observations using Gemini 3 Pro Image (Nano Banana Pro) and GPT-Image-2, with GPT-Image-2 used for most generated images. Generated images are then refined through a bounded image-review-and-editing loop for at most K revision rounds ($K = 2$ in most cases). A review agent checks for severe artifacts or missing visual evidence, and unresolved failures are rejected, ensuring that retained labels are grounded in what is actually visible to the model.

Executable labeling. For each verified observation, multiple LLMs independently generate candidate API programs using the image, the initial vehicle state, and the full API document. Candidate calls are parsed into direction-aware semantic action keys that preserve the target device, operation direction, and affected seat or scope. We perform majority voting at the semantic-action level rather than directly comparing raw API strings, since different API forms may express the same intended state change. The selected actions are mapped back to canonical VehicleWorld API calls and executed from S_0 to verify that they produce valid state transitions.

Finally, execution checking and expert audit are used to filter or correct invalid functions, illegal parameters, inconsistent transitions, missed necessary actions, and redundant operations. After the physical actions are finalized, user-facing speech and display notifications are regenerated to align with the verified device controls. For abstention cases, we specifically sample normal scenarios and vehicle states, which are then incorporated into this stage with the generated images, yielding reliable abstention labels. The resulting supervision is execution-grounded for both action and abstention cases: each retained sample contains a visual observation, an initial state, and a verified program or

Setting	Model	Device Positive F1	Strict Change F1	Change IoU	Strict Change Precision	Strict Change Recall
Zero-shot	Qwen3-VL-8B	29.70	28.85	23.21	35.49	33.27
	Qwen3-VL-32B	34.61	34.50	26.62	39.25	47.06
	Qwen3-VL-235B	27.07	26.74	19.38	30.59	40.86
	GPT-5	38.68	38.56	31.21	43.06	49.16
	Gemini 3 Pro	40.33	39.57	31.16	40.91	57.70
	Claude Sonnet 4.6	26.45	26.23	18.37	22.10	58.70
Few-shot	Qwen3-VL-8B	41.58	41.47	34.99	48.03	47.02
	Qwen3-VL-32B	42.67	42.62	35.01	45.02	56.55
	Qwen3-VL-235B	40.13	39.76	30.42	39.91	60.28
	GPT-5	52.45	52.24	44.68	52.76	63.62
	Gemini 3 Pro	46.95	46.49	37.63	42.35	**76.75**
	Claude Sonnet 4.6	33.93	33.90	24.80	29.36	68.78
SFT	**Qwen3-VL-8B**	**65.09**	**65.09**	**56.64**	**67.84**	68.05

Table 2: Main semantic-interaction evaluation on the 350-case real-world benchmark. The test set comprises 300 proactive-intervention cases and 50 no-interaction cases. The SFT model is trained on the full 2169-case synthetic pool. All metric values are percentages.

empty program, whose execution defines the target state-change set used for evaluation.

3.4 Synthetic-to-Real Dataset Composition

The resulting CabinProAct dataset follows a synthetic-to-real design: synthetic data provides scalable execution-grounded supervision, while real-world data is reserved for transfer evaluation. The synthetic training pool contains 2169 instances and is used for supervised fine-tuning and dataset-scaling analysis. As shown in Figure 2, it covers diverse proactive interaction categories and API modules, with each instance pairing visual observation, structured initial vehicle state, and executable action or abstention label.

The real-world evaluation split contains 350 held-out cases and is never used for training. It focuses on authentic single-perspective in-cabin imagery, including RGB and infrared images, paired with systematically constructed vehicle states to isolate state-conditioned reasoning. The split includes 300 proactive intervention cases and 50 no-interaction cases, allowing us to evaluate not only whether agents can identify latent assistance opportunities, but also whether they can abstain when no intervention is warranted. Accordingly, redundant device operations are penalized as false positives.

3.5 State-Change-Based Evaluation

We evaluate proactive programs by their executed state changes rather than surface-form text matching. Given the same initial state S_0, the predicted program π and oracle program π^* are executed in the sandbox to obtain $S_p = \mathrm{Exec}(S_0, \pi)$ and $S_g = \mathrm{Exec}(S_0, \pi^*)$. As defined in Equation 2, we compare the resulting state-change sets induced by the prediction and the oracle:

$$P = \Delta(S_0, S_p), \qquad G = \Delta(S_0, S_g). \quad (2)$$

This protocol makes executable validity part of the evaluation: invalid API calls, illegal parameters, or failed executions cannot produce the intended state transition.

A predicted change is counted as a true positive only when it matches the corresponding oracle change on the same state field and with the correct effect. Extra changes in P are false positives, while missing changes in G are false negatives. For no-interaction cases, the oracle change set is empty; therefore, any unnecessary device operation is penalized as a false positive. User-facing speech and display notifications are evaluated as semantic interaction events, while physical device APIs are evaluated through their executed state changes.

We report Strict Change Precision, Recall, F1, and Intersection over Union (IoU) over state changes. We additionally report Device Positive F1 to measure whether the agent selects the correct interaction fields before enforcing value-level constraints, and Redundant Rate to quantify unnecessary predicted operations, with more details provided in Appendix E.

4 Experiments and Analysis

4.1 Experimental Setup

We evaluate six MLLMs on the real-world split: proprietary frontier models GPT-5 (Singh et al., 2025), Gemini 3 Pro (Google DeepMind, 2025), and Claude Sonnet 4.6 (Anthropic, 2026), together with the open-source Qwen3-VL (Bai et al., 2025) series at 8B, 32B, and 235B scales. All models

Training size	Device Positive F1	Strict Change F1	Change IoU	Strict Change Precision	Strict Change Recall	Redundant Rate
300	57.31	57.21	49.02	69.50	57.68	22.22
600	59.25	59.25	51.74	68.60	58.04	**21.40**
1000	62.48	62.48	54.37	67.74	63.15	27.11
1500	63.87	63.87	55.03	**70.17**	64.99	26.97
2169	**64.17**	**64.17**	**55.93**	68.64	**67.08**	29.36

Table 3: Dataset-scaling analysis for Qwen3-VL-8B SFT on the 350-case real-world semantic-interaction benchmark. All metric values are percentages.

receive the same inputs: the cabin image, the structured vehicle state, the full API document, and the required JSON output schema.

We consider three evaluation settings: ❶ **Zero-shot prompting**, where models directly predict the proactive API program without demonstrations; ❷ **Compact few-shot prompting**, where we add four text-only examples from the synthetic split to clarify action granularity, executable formatting, and user-facing communication for proactive physical control; and ❸ **Supervised fine-tuning (SFT)**, where we train Qwen3-VL-8B on the full 2169-case synthetic split using the same full-input format as the prompting baselines. Detailed training hyperparameters are provided in Appendix F.

4.2 Main Results: Synthetic-to-Real Transfer

Table 2 and Figure 4 summarize the main results on the real-world evaluation split. The key finding is that execution-grounded synthetic supervision transfers effectively to authentic cabin observations. The SFT Qwen3-VL-8B model, trained *only* on the synthetic split, achieves the best overall performance with 65.09 Strict F1, 56.64 Change IoU, and 67.84 Strict Precision. It outperforms the strongest prompted baseline, few-shot GPT-5, by 12.85 absolute Strict F1 points. This result suggests that synthetic supervision teaches more than output formatting or API syntax: it helps the model acquire a state-conditioned proactive policy that generalizes to real in-cabin imagery.

This transfer is particularly meaningful because proactive interaction requires calibrated intervention rather than maximizing recall alone. An over-aggressive agent may identify many visually salient events, but still perform redundant or unnecessary operations when the current vehicle state does not require action. The SFT model achieves a balanced precision–recall profile, with 67.84 Strict Precision and 68.05 Strict Recall. In contrast, prompted frontier models often show a strong recall bias. For example, few-shot Gemini 3 Pro reaches the highest Strict Recall of 76.75, but its Strict Precision

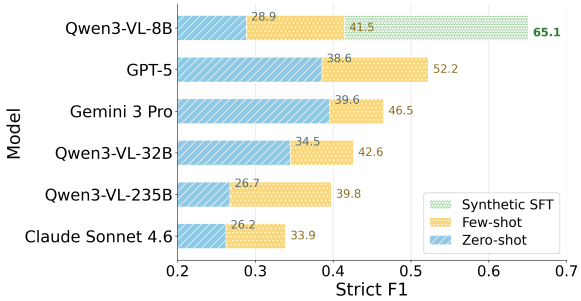


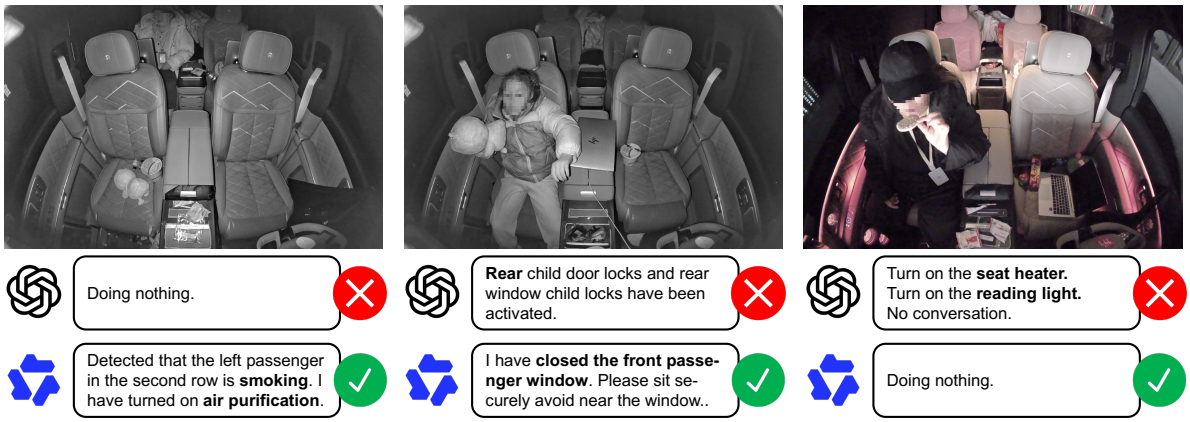
Figure 4: Strict F1 comparison across zero-shot, few-shot, and SFT settings. Compact few-shot prompting improves most models, but synthetic full-API SFT gives the largest gain.

is only 42.35; Claude Sonnet 4.6 similarly obtains 68.78 Strict Recall but only 29.36 Strict Precision. These results indicate that prompted models can often perceive potential triggers, but fail to reliably decide whether intervention is necessary under the current state.

Few-shot prompting improves performance, but mainly by aligning models with the procedural format of the task. Compared with zero-shot prompting, few-shot GPT-5 improves from 38.56 to 52.24 Strict F1, and Qwen3-VL-8B improves from 28.85 to 41.47. However, these gains remain far below SFT, suggesting that a few demonstrations can teach models how to express actions in the sandbox, but cannot sufficiently teach the conditional boundary between helpful intervention and unnecessary interruption. This gap supports our central claim: robust proactive cabin behavior requires execution-grounded synthetic supervision, not merely stronger prompting.

4.3 Scaling Synthetic Supervision

Table 3 analyzes how performance changes as the synthetic training pool expands from 300 to 2169 instances. To isolate the effect of data volume, all scaling runs are trained for a single epoch under the same setup. Strict F1 improves monotonically from 57.21 to 64.17, while Change IoU increases from 49.02 to 55.93. Strict Recall also rises from 57.68 to 67.08, suggesting that broader synthetic



(a) smoking case (b) child security case (c) eating case

Figure 5: Representative qualitative cases from the real-world benchmark. Green panels show successful state-conditioned interventions or abstention, while red panels show failures including missed intervention, generic action substitution, and unnecessary over-triggering.

supervision helps the model cover a wider range of latent intervention opportunities.

The scaling trend also reveals a calibration trade-off. The 1500-instance setting achieves the highest Strict Precision of 70.17, whereas the full 2169-instance setting obtains the best Strict Recall and Change IoU but slightly higher Redundant Rate. This indicates that larger synthetic supervision improves intervention coverage, but may also make the agent more willing to act. Overall, the consistent gains in Strict F1 and IoU support the value of scaling execution-grounded synthetic data for learning proactive cabin policies.

4.4 Qualitative Analysis

Figure 5 shows representative cases from the real-world evaluation split. These examples illustrate that the key challenge is not only recognizing visible occupant behavior, but deciding *whether and how* the vehicle state should be changed. We highlight three representative cases below.

Missing interaction. As shown in Figure 5(a), the prompted model fails to intervene, whereas the fine-tuned model correctly associates the visible smoking behavior with the need for air purification and produces a concise user-facing response. This demonstrates that execution-grounded synthetic supervision can help the model map real in-cabin visual evidence to executable proactive actions.

Incorrect action. As shown in Figure 5(b), rather than issuing a generic safety action such as activating child locks, the fine-tuned model selects the more appropriate intervention of closing the relevant window and communicating the reason to the occupant. This suggests that the model learns to ground interventions in the concrete cabin state and available APIs, instead of relying only on broad visual safety priors.

Over-triggering. As shown in Figure 5(c), although the scene contains visually salient passenger activity, no intervention is required under the current state. The prompted model nevertheless triggers comfort-related controls, producing unnecessary actions without communication. In contrast, the fine-tuned model correctly outputs no action.

Together, these cases show that proactive cabin agents must balance three core abilities: ❶ identifying latent assistance opportunities, ❷ selecting state-specific device controls, and ❸ abstaining from unnecessary interaction.

5 Conclusion

We introduced CabinProAct, an execution-grounded synthetic data generation framework and synthetic-to-real benchmark for proactive cabin agents. By coupling visual observations with structured vehicle states, executable APIs, and sandbox-verified labels, CabinProAct provides scalable supervision for learning when to intervene and when to abstain. Experiments on real-world cabin imagery show that synthetic execution-grounded supervision transfers effectively, enabling stronger state-conditioned intervention and better calibration than prompting-based frontier models. These findings highlight executable synthetic supervision as a promising path toward proactive cabin agents that can act precisely, communicate transparently, and avoid unnecessary interaction.

6 Limitations

Temporal extension. CabinProAct focuses on single-frame cabin observations paired with static initial states, which provides a controlled setting for isolating visual grounding, state-conditioned reasoning, and executable intervention. An important future direction is to extend this setup to temporally continuous observations, where agents reason over video streams and evolving vehicle states to determine the appropriate timing of intervention. Such an extension would enable richer evaluation of temporal persistence, delayed intervention, and continuous proactive assistance.

Subjective preference and personalization. CabinProAct considers proactive intervention under a shared interaction policy, without modeling personalized preferences across different users. In cabin environments, occupants may have different thresholds for when agent-initiated intervention is perceived as helpful or intrusive, and habitual preferences may also affect control parameters such as temperature, volume, lighting, or notification style. Future work could study how user-specific preference data can be collected and used responsibly to improve comfort, reduce unnecessary interruptions, and maintain safety and controllability.

7 Ethical and Safety Considerations

CabinProAct is an offline research benchmark built in a simulated VehicleWorld API environment. It should not be interpreted as validation for deploying autonomous cabin-control agents in real vehicles. Because proactive vehicle systems may affect driver attention, passenger comfort, and safety-critical behavior, any real deployment would require strict human oversight, fail-safe mechanisms, user consent, and extensive testing beyond the scope of this benchmark.

The task also involves agent-initiated physical control, which introduces risks of unwanted or silent intervention. We therefore require user-facing communication whenever the agent changes cabin device states, and our evaluation penalizes unnecessary operations and over-triggering. These design choices are intended to study proactive assistance under controlled conditions, not to authorize unrestricted autonomous control.

The real-world split contains in-cabin imagery. Data collection, annotation, and release should follow applicable privacy, consent, anonymization, and licensing requirements. When images include identifiable occupants, released artifacts should either anonymize sensitive visual information or restrict access to metadata and evaluation scripts where full image release is not appropriate.

References

Anthropic. 2026. System Card: Claude Sonnet 4.6. System card, Anthropic. February 17, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Yi Chen, Chengzhe Li, Qirui Yuan, Jinyu Li, Yuze Fan, Xiaojun Ge, Yun Li, Fei Gao, and Rui Zhao. 2024. Cockpit-llama: Driver intent prediction in intelligent cockpit via large language model. *Sensors*, 25(1):64.

Yang Deng, Wenqiang Lei, Wai Lam, and Tat-Seng Chua. 2023. A survey on proactive dialogue systems: Problems, methods, and prospects. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 6583–6591.

Huifang Du, Xuejing Feng, Jun Ma, Meng Wang, Shiyu Tao, Yijie Zhong, Yuan-Fang Li, and Haofen Wang. 2024. Towards proactive interactions for in-vehicle conversational assistants utilizing large language models. *arXiv preprint arXiv:2403.09135*.

Rafael Giebisch, Ken E Friedl, Lev Sorokin, and Andrea Stocco. 2025. Automated factual benchmarking for in-car conversational systems using large language models. In *2025 IEEE Intelligent Vehicles Symposium (IV)*, pages 1238–1243. IEEE.

Google DeepMind. 2025. Model Evaluation – Approach, Methodology & Results: Gemini 3 Pro. Technical report, Google DeepMind. November 2025.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Yahya Sowti Khiabani, Farris Atif, Chieh Hsu, Sven Stahlmann, Tobias Michels, Sebastian Kramer, Benedikt Heidrich, M Saquib Sarfraz, Julian Merten, and Faezeh Tafazzoli. 2025. Optimizing small language models for in-vehicle function-calling. *arXiv preprint arXiv:2501.02342*.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Dezhi Kong, Zhengzhao Feng, Qiliang Liang, Hao Wang, Haofei Sun, Changpeng Yang, Yang Li, Peng Zhou, Shuai Nie, Hongzhen Wang, and 1 others. 2026. Proactivemobile: A comprehensive benchmark for boosting proactive intelligence on mobile devices. *arXiv preprint arXiv:2602.21858*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Tianjian Liu, Fanqi Wan, Jiajian Guo, and Xiaojun Quan. 2025. Proactiveeval: A unified evaluation framework for proactive dialogue agents. *arXiv preprint arXiv:2508.20973*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, and 1 others. 2025. Proactive agent: Shifting llm agents from reactive responses to active assistance. In *International Conference on Learning Representations*, volume 2025, pages 47431–47457.

Jun Ma, Meng Wang, Jinhui Pang, Haofen Wang, Xuejing Feng, Zhipeng Hu, Zhenyu Yang, Mingyang Guo, Zhenming Liu, Junwei Wang, and 1 others. 2024a. Development and evaluation study of intelligent cockpit in the age of large models. *arXiv preprint arXiv:2409.15795*.

Yingzi Ma, Yulong Cao, Jiachen Sun, Marco Pavone, and Chaowei Xiao. 2024b. Dolphins: Multimodal language model for driving. In *European Conference on Computer Vision*, pages 403–420. Springer.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, and 1 others. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and 1 others. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1088–1113.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pages 1–16. IEEE.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Bill N Schilit and Marvin M Theimer. 2002. Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32.

C Sima, K Renz, K Chitta, L Chen, H Zhang, C Xie, J Beißwenger, P Luo, A Geiger, and H Drivelm Li. Driving with graph visual question answering. In *Proceedings of the European Conference on Computer Vision (ECCV), Milan, Italy*, volume 29, pages 256–274.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, and 1 others. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.

Hao Tang, Darren Key, and Kevin Ellis. 2024. WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, volume 37, pages 70148–70212.

Yuanbo Tang, Huaze Tang, Tingyu Cao, Lam Nguyen, Anping Zhang, Xinwen Cao, Chunkang Liu, Wenbo Ding, and Yang Li. 2026. Proagentbench: Evaluating llm agents for proactive assistance with real-world data. *arXiv e-prints*, pages arXiv–2602.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. 2024. Drivevlm: The convergence of autonomous driving and large vision-language models. *arXiv preprint arXiv:2402.12289*.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*.

Cheng-Kuang Wu, Zhi Rui Tam, Chao-Chung Wu, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. I need help! evaluating llm's ability to ask for users' support: A case study on text-to-sql generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2191–2199.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. 2026. Contextagent: Context-aware proactive llm agents with open-world sensory perceptions. *Advances in Neural Information Processing Systems*, 38:167509–167543.

Jie Yang, Jiajun Chen, Zhangyue Yin, Shuo Chen, Yuxin Wang, Yiran Guo, Yuan Li, Yining Zheng, Xuanjing Huang, and Xipeng Qiu. 2025a. VehicleWorld: A highly integrated multi-device environment for intelligent vehicle interaction. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 403–442, Suzhou, China. Association for Computational Linguistics.

Qinglong Yang, Haoming Li, Haotian Zhao, Xiaokai Yan, Jingtao Ding, Fengli Xu, and Yong Li. 2025b. Fingertip 20k: A benchmark for proactive and personalized mobile llm agents. *arXiv preprint arXiv:2507.21071*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024. Ask-before-plan: Proactive language agents for real-world planning. *arXiv preprint arXiv:2406.12639*.

A Prompt and Output Format

All evaluated models receive the same task interface: a cabin observation image, the current structured vehicle state, the complete executable VehicleWorld API document, and a required JSON output format. The prompt consistently frames the model as a *proactive cabin agent*: it should intervene only when the visual observation and vehicle state jointly justify an action, and it should abstain when no intervention is needed or when the current state already satisfies the latent need. For compact few-shot prompting, we insert four text-only demonstrations from the synthetic training split. These demonstrations specify decision granularity and output style, but do not include additional images; the current test image remains the only visual evidence for the current case.

The output is a JSON object with four fields:

- `scene_analysis`: visible occupant behavior and relevant environmental evidence.
- `status_analysis`: state-grounded analysis of whether the current vehicle configuration requires intervention.
- `decision_reasoning`: a concise explanation linking visual evidence, vehicle state, and the selected action or abstention decision.
- `api_code`: a list of executable VehicleWorld API calls, or an empty list when no intervention is warranted.

The prompt explicitly requires API calls to follow the provided documentation, discourages redundant or unrelated operations, and requires user-facing communication when the model initiates physical cabin control. A compact example target is shown below:

Scene/status. A rear passenger is smoking, and air purification is currently off.
Decision. Activate air purification and notify occupants because the agent is initiating a cabin-state change.
API program. Enable air purification and issue a voice message through the conversation API.

Listing 1: Compact prompt skeleton used for training and evaluation. Few-shot examples and more prompts are provided in the following pages.

```
## System Prompt:
You are a proactive intelligent cabin interaction agent.

You will receive:
```

```
1. One cabin image.
2. Optionally, one exterior driving−scene image when
      available.
3. The full current vehicle device status.
4. The full available VehicleWorld API documentation.

Your task:
– Analyze the cabin and exterior scene.
– Compare the visual scene with the current device status.
– Decide whether proactive interaction or device control is
      needed.
– Prefer safety−critical actions over comfort actions.
– Avoid irrelevant or excessive actions.
– Output executable VehicleWorld API calls only when
      justified by the image and state.

Output valid JSON only:
{
  "scene_analysis": string,
  "status_analysis": string,
  "decision_reasoning": string,
  "api_code": string[]
}

## User Prompt:
[IMAGE] cabin observation image
[OPTIONAL IMAGE] exterior driving−scene image, if
      available

Current vehicle device status:
[CURRENT_FULL_DEVICE_STATUS_JSON]

Available VehicleWorld APIs:
[VEHICLEWORLD_API_DOCS]
```

B Scenario–State Sampling

The synthetic generator uses a random-first scenario–state sampling strategy rather than a target-action-first strategy. It first samples a cabin scene seed and a full VehicleWorld state patch, and only later asks the mining agent to determine whether the sampled pair contains a meaningful proactive interaction opportunity. This design is intended to reduce template bias: the generator does not decide in advance that a case must be, for example, a "phone-call volume-reduction" case or a "smoking purification" case. Instead, interaction opportunities emerge from the sampled combination of occupant behavior, external context, and device state.

Cabin scene seed. The scene seed specifies the occupants, their seats, actions, attention states, postures, emotions, and objects. The driver is always instantiated as an adult driver, while passenger seats are sampled from the front passenger seat and two rear seats. Passenger types include adults, elderly occupants, children, infants, pregnant passengers, and pets. The sampler applies basic plausibility constraints before image generation: pets and infants are not assigned inappropriate handheld objects; pets are restricted to actions such as resting, looking outside, discomfort, or agitation; and infant states are restricted to sleeping, resting, crying, or discomfort. For human occupants, object, attention, and posture are inferred from the sampled action when possible, such as mapping phone calls to a phone object or sleeping to closed eyes.

External context seed. The external context includes weather, road type, traffic situation, road surface, time of day, and visibility. Some variables are conditionally adjusted for physical plausibility. For example, rainy and stormy weather increase the probability of wet road surfaces, snowy weather increases the probability of snow or ice, and night weather sets the time of day to night. These variables are not labels by themselves; they provide context that may or may not become actionable depending on the vehicle state.

Vehicle state patch. For each sampled scene, the generator samples a full structured state patch covering the environment state and major VehicleWorld device modules. The environment portion includes vehicle motion, active sound channel, volume, speaker position, cabin temperature, and language. Device modules include music playback, phone-call state and hands-free mode, per-seat windows, per-seat doors and child locks, per-seat air-conditioning states, purification and circulation modes, reading lights, seat heaters, sunroof, rear sunshade, low beams, high beams, fog lights, and hazard lights. Several state variables are sampled at the seat level, enabling localized interactions such as closing a specific window, disabling a specific reading light, or enabling a specific seat heater.

State consistency checks. The sampler applies lightweight consistency constraints after random sampling. For example, if air purification is sampled as active while all air-conditioning zones are off, one air-conditioning zone is activated so that the purification state is executable in VehicleWorld. The state patch is then compiled into initialization API code and replayed in the simulator. The replayed initialized state is compared with the intended patch after canonicalizing supported aliases and value formats. Cases with unsupported paths or mismatched replayed states are filtered before entering downstream stages.

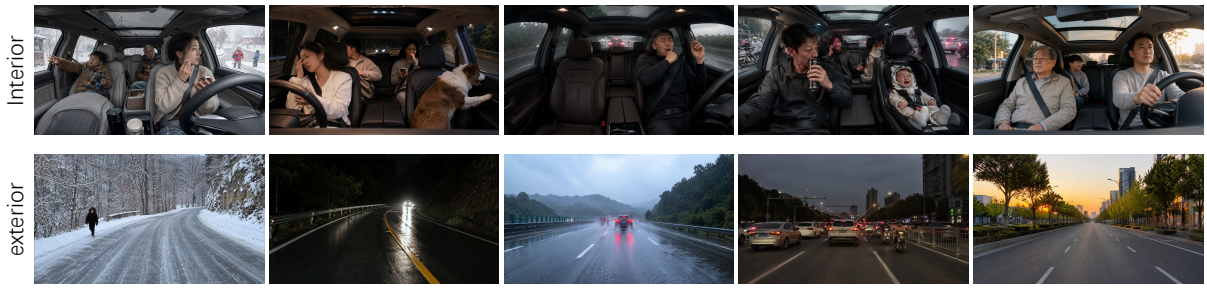


Figure 6: Partial examples from the generated dataset, illustrating diversity in weather conditions, time of day, passenger states, and other factors. The corresponding states and APIs can be found in the displayed prompt content.

C Synthetic Label Construction

Initial-state replay. Each synthetic instance contains a structured initial vehicle state. Before label generation, this state is compiled into VehicleWorld initialization code and replayed in the simulator. The replayed state is compared with the intended state after canonicalizing supported aliases, seat names, and value formats. Instances whose initial state cannot be faithfully replayed are excluded or returned for correction, because their labels would not define a reliable executable transition.

Candidate program generation. For each verified observation, three independent LLM families—GPT-5, Claude 4.6, and Gemini 3 Pro—generate candidate proactive programs using the same image, initial state, and full API document. Each candidate is parsed into a list of API calls. Candidates with invalid JSON, hallucinated functions, illegal parameter types, or execution failures are marked invalid and are not automatically admitted into the voted label. This step keeps each model output auditable before any consolidation is performed.

Semantic action canonicalization. Raw API strings are canonicalized into direction-aware semantic action keys. A key records the controlled device or module, the operation type, the direction or target value, and the affected scope when recoverable, such as a seat position or global target. For example, a relative volume decrease and setting the volume to a lower value can map to the same `music.volume.decrease` key, while volume increase maps to a different key. Similarly, opening and closing a window are separate keys, and a local rear-seat light action is distinguished from a global lighting action. This normalization allows semantically equivalent API forms to agree while preventing opposite or over-broad actions from being merged.

Semantic voting and execution verification. Voting is performed at the semantic-key level rather than the raw-string level. A semantic action enters the automatic consensus core when it is selected by at least two of the three LLMs. Singleton actions are not accepted automatically; they are either discarded or sent to expert audit when omitting them would make the label incomplete. The selected semantic actions are then mapped back to canonical VehicleWorld API calls and executed from the initial state. Only programs that produce valid state transitions are retained. After the physical actions are finalized, speech and display-notification content is aligned with the final API program to avoid mismatches between the natural-language explanation and the executable controls.

Expert audit protocol. Expert audit checks semantic appropriateness, not only API syntax. The audit follows a checklist: whether the image provides sufficient evidence for intervention; whether the initial vehicle state makes the intervention necessary; whether the action set is minimal and device-specific; whether required actions are missing; whether any action is redundant or unsafe; whether speech or notification content accurately explains the physical controls; and whether abstention cases are genuinely no-action cases rather than missed interventions. Labels that fail the checklist are corrected when the required fix is unambiguous, otherwise they are removed from the high-confidence pool or sent back to the construction pipeline.

D Real-World Label Construction

The real-world split follows a different construction route from the synthetic split because the visual observations are collected before the interaction scenario is defined. For each real-world cabin image, we ask GPT-5.4 and Gemini3pro to inspect the im-

age together with the VehicleWorld state tree and propose how different initial vehicle states could make the observed situation actionable. The constructed states are not intended to reproduce CAN-bus logs from the recording vehicle; instead, they are used to isolate state-conditioned reasoning under controlled executable conditions. Each model generates three candidate interaction scenarios with priority annotations. Each candidate specifies the visible evidence, the constructed initial-state condition, the intended proactive interaction, and why the state makes the interaction necessary.

Human experts then review the candidate scenarios for each image. The selection criteria are: whether the interaction is grounded in the image; whether the constructed state creates a realistic and non-trivial proactive need; whether the interaction is minimal and device-specific; whether the case contributes diversity relative to other cases; and whether the scenario can be represented with executable VehicleWorld APIs. Experts choose the most appropriate candidate or revise it when a small correction is sufficient. Candidates that rely on weak visual evidence, implausible states, or overly generic actions are rejected.

After expert selection, the real-world instance is passed to the same executable labeling stage used in the main pipeline. The input to this stage contains the real-world image, the selected constructed initial state, the interaction rationale, and the full API document. The final label is then generated through multi-model candidate programs, semantic action canonicalization, semantic voting, execution verification, and expert audit as described in Appendix C. Thus, the real-world split uses authentic cabin imagery while maintaining the same executable label standard as the synthetic data.

No-interaction real-world cases follow the same procedure, except that the candidate construction objective is inverted. Instead of mining a latent intervention opportunity, the models propose initial states that remove or already satisfy potential intervention needs visible in the image. Experts then select cases where abstention is justified by both the visual evidence and the constructed vehicle state. These cases are labeled with empty API programs and are used to evaluate calibrated abstention and over-triggering.

E Evaluation Protocol

State-change extraction. Given the same initial state S_0, the evaluator executes the reference program π^* and predicted program π to obtain the oracle and predicted final states:

$$S_g = \mathrm{Exec}(S_0, \pi^*), \qquad S_p = \mathrm{Exec}(S_0, \pi). \quad (3)$$

Nested vehicle states are flattened into field paths. As defined in Equation 4, the reference change set G and predicted change set P contain the field-level changes induced by the oracle and predicted programs:

$$G = \Delta(S_0, S_g), \qquad P = \Delta(S_0, S_p). \quad (4)$$

Each element in a change set records both the changed field path and its executed effect, such as the final Boolean/categorical value or the direction of a numeric adjustment. Invalid API calls receive no credit because they do not induce the required state transition.

Strict change matching. Strict matching evaluates whether the predicted program achieves the correct executed state changes. A predicted change $p \in P$ is a strict true positive if there exists a reference change $g \in G$ on the same field path and the executed effect matches. For Boolean and categorical fields, the predicted final value must match the reference final value. For numeric fields such as volume, temperature, brightness, or opening degree, relative changes are matched by direction when appropriate. A prediction that changes the correct field but in the wrong direction or to the wrong value is treated as a wrong-effect change. A prediction that changes a field outside G is an unnecessary change, while a field in G that is not correctly achieved by the prediction is a missing change.

Let $\mathrm{TP}_{\mathrm{strict}}$ be the number of strictly correct predicted changes. The strict false positives and false negatives are defined in Equation 5:

$$\begin{aligned} \mathrm{FP}_{\mathrm{strict}} &= |P| - \mathrm{TP}_{\mathrm{strict}}, \\ \mathrm{FN}_{\mathrm{strict}} &= |G| - \mathrm{TP}_{\mathrm{strict}}. \end{aligned} \quad (5)$$

Thus, wrong-effect changes and unnecessary changes are counted as false positives, while required changes that are not correctly achieved are counted as false negatives. We do not count true negatives, since unchanged vehicle fields are numerous and would dominate the metric without reflecting proactive interaction quality.

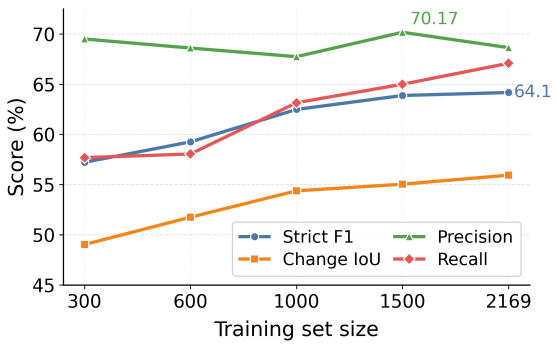


Figure 7: Scaling curve for Qwen3-VL-8B SFT. Expanding the synthetic training pool yields monotonic improvements in both strict F1 and recall, while the intermediate 1500-case scale maximizes strict precision.

Configuration	Value
Base model	Qwen3-VL-8B
Training data	2169 synthetic full-API examples
Fine-tuning method	LoRA (Hu et al., 2022)
LoRA rank	16
LoRA alpha	32
LoRA dropout	0.05
Target modules	Linear projections in attention and MLP blocks
Training epochs	3
Global batch size	8
Peak learning rate	2×10^{-4}
Warmup ratio	3%
Image resolution	Dynamic resolution up to 1,048,576 pixels
Distributed training	8 A100 GPUs with DeepSpeed ZeRO-3 (Rajbhandari et al., 2020)

Table 4: Full-API supervised fine-tuning configuration.

Strict metrics. Strict Change Precision and Strict Change Recall are computed as:

$$\begin{aligned} \text{Precision}_{\text{strict}} &= \frac{\text{TP}_{\text{strict}}}{\text{TP}_{\text{strict}} + \text{FP}_{\text{strict}}}, \\ \text{Recall}_{\text{strict}} &= \frac{\text{TP}_{\text{strict}}}{\text{TP}_{\text{strict}} + \text{FN}_{\text{strict}}}. \end{aligned} \tag{6}$$

Strict Change F1 is then computed as:

$$\text{F1}_{\text{strict}} = \frac{2 \cdot \text{Precision}_{\text{strict}} \cdot \text{Recall}_{\text{strict}}}{\text{Precision}_{\text{strict}} + \text{Recall}_{\text{strict}}}. \tag{7}$$

Change IoU is computed over strict state changes:

$$\text{IoU} = \frac{\text{TP}_{\text{strict}}}{\text{TP}_{\text{strict}} + \text{FP}_{\text{strict}} + \text{FN}_{\text{strict}}}. \tag{8}$$

When both P and G are empty, the prediction exactly matches the abstention target, and the per-case Strict Change F1 and IoU are set to 1. When G is empty but P is non-empty, the case is treated as over-triggering and receives zero Strict Change F1.

Device Positive F1. Device Positive F1 evaluates whether the model selects the correct interaction fields before enforcing value-level or direction-level constraints. Let $\mathcal{F}(P)$ and $\mathcal{F}(G)$ denote the deduplicated field paths changed by the prediction and reference program, respectively. The device-level counts are defined as:

$$\begin{aligned} \text{TP}_{\text{dev}} &= |\mathcal{F}(P) \cap \mathcal{F}(G)|, \\ \text{FP}_{\text{dev}} &= |\mathcal{F}(P) \setminus \mathcal{F}(G)|, \\ \text{FN}_{\text{dev}} &= |\mathcal{F}(G) \setminus \mathcal{F}(P)|. \end{aligned} \tag{9}$$

Device Positive Precision and Recall are computed as:

$$\begin{aligned} \text{Precision}_{\text{dev}} &= \frac{\text{TP}_{\text{dev}}}{\text{TP}_{\text{dev}} + \text{FP}_{\text{dev}}}, \\ \text{Recall}_{\text{dev}} &= \frac{\text{TP}_{\text{dev}}}{\text{TP}_{\text{dev}} + \text{FN}_{\text{dev}}}. \end{aligned} \tag{10}$$

Device Positive F1 is computed as:

$$\text{F1}_{\text{dev}} = \frac{2 \cdot \text{Precision}_{\text{dev}} \cdot \text{Recall}_{\text{dev}}}{\text{Precision}_{\text{dev}} + \text{Recall}_{\text{dev}}}. \tag{11}$$

This metric gives credit when the model identifies the correct device field even if the final value or adjustment direction is not strictly correct.

Redundant Rate. Redundant Rate measures unnecessary or wrong-effect predicted operations:

$$\text{Redundant Rate} = \frac{\text{FP}_{\text{strict}}}{|P|}. \tag{12}$$

It is defined as zero when P is empty. This metric explicitly captures over-triggering, including redundant device operations, wrong-direction changes, and unnecessary communication events.

Abstention and communication events. For no-interaction cases, G is empty, so the correct prediction is an empty API program. Any device operation or unnecessary communication in such cases is counted as over-triggering. For proactive intervention cases, user-facing speech and display notifications are represented as semantic events, such as `semantic_interaction.speak_triggered`. The evaluation checks whether the required communication event is triggered, but does not require exact string matching of the generated utterance. This design penalizes silent physical control while allowing natural wording variation.

Aggregation. All reported metrics are macro-averaged across cases. In particular, Strict Change F1 and Device Positive F1 are computed per case and then averaged; they are not recomputed from averaged precision and averaged recall.

F Training Details

We fine-tune Qwen3-VL-8B using full-input supervised fine-tuning. Each training example uses the same format as evaluation: image input, full structured vehicle state, complete API documentation, and the JSON target containing analysis fields and executable API code. The synthetic split is used only for training and scaling analysis; the 350-case real-world split is held out for evaluation. The full training run takes approximately 40 A100 GPU-hours in total.

Figure 7 visualizes the dataset-scaling results reported in Table 3 in the main text, showing how performance changes as the synthetic training pool expands. Table 4 summarizes the full-API supervised fine-tuning configuration used for the final Qwen3-VL-8B SFT model.

Listing 2: Extended prompt used for few-shot prompting, with four text-only demonstrations.

```
Compact few−shot demonstrations:
These examples demonstrate decision granularity and JSON format only.
Do not copy example actions unless the CURRENT image and CURRENT device status justify them.
The examples omit full API docs and full device status because those are already provided for the current case.

Action notification rule:
− If api_code contains any proactive control action, the assistant should also notify the user with a relevant voice message
      using vw.conversation.carcontrol_conversation_speak(...).
− The voice message must briefly explain what was done and why it was done.
− Use priority="high" for safety−critical warnings and priority="normal" for comfort or convenience actions.
− If no proactive action is needed and api_code is empty, do not add a voice message.
− Do not add unrelated visual notifications unless the situation is safety−critical or genuinely needs a persistent visual alert.

Example 1:
{
 "example_input": {
  "visual_summary": "......",
  "relevant_state": "......",
  "decision_focus": "......"
 },
 "example_output": {
  "scene_analysis": "......",
  "status_analysis": "......",
  "decision_reasoning": "......",
  "api_code": [
   "vw.music.music_soundVolume_decrease(degree=\"large\")",
   "vw.conversation.conversation_call_handsFree_switch(True)",
   "vw.window.carcontrol_window_switch(position=[\"passenger seat\"], switch=False)",
   "vw.conversation.carcontrol_conversation_speak(content=\"I have lowered the music volume, switched the call to hands−
      free mode, and closed the front−passenger window. Driver, please avoid holding the phone and pay attention to
      passenger safety.\", priority='high')"
  ]
 }
}

Example 2:
{
 "......",
}

Example 3:
{
 "......",
}

Example 4:
{
 "example_input": {
  "visual_summary": "......",
  "relevant_state": "......",
  "decision_focus": "......",
 },
 "example_output": {
  "scene_analysis": "......",
  "status_analysis": "......",
  "decision_reasoning": "......"
  "api_code": []
 }
}

Now solve the CURRENT case.
Only the attached current image and the current vehicle device status are ground truth for this case.
Use the full API docs above. Return valid JSON only.
```

Listing 3: Prompt used for Mining Agent.

```
## System Prompt:
You are an interaction−worthiness miner for proactive in−cabin agents.
Your task is NOT to generate final GT and NOT to write API code.
Given a randomly sampled cabin scene and vehicle full−state patch, decide whether this pair contains a meaningful
    opportunity for proactive interaction.

Important principles:
1. Random samples can be positive or negative. Do not force every case to be positive.
2. A positive case should have concrete evidence from BOTH scene semantics and device/vehicle state.
3. Do not use target templates. Discover interaction opportunities from the random combination.
4. Mild comfort opportunities can be positive only when the device state makes the intervention useful.
5. Keep negative cases if there is no clear need, no state conflict, or the response would be unnecessary disturbance.
6. The next stage will generate images only for worthy cases. Later GT must be based on generated images, not blindly on
    this structured seed.

Output strictly valid JSON with this schema:
{
 "interaction_worthy": boolean,
 "confidence": "low|medium|high",
 "priority": "low|medium|high|critical",
 "scene_evidence": [string],
 "status_evidence": [string],
 "interaction_opportunities": [string],
 "likely_response_direction": [string],
 "why_not_random_noise": string,
 "negative_reason_if_not_worthy": string|null,
 "image_generation_should_continue": boolean
}

## User Prompt
{
 "id": "<case_id>",
 "environment": "<randomly sampled environment seed>",
 "occupants": "<randomly sampled occupants seed>",
 "state_patch": {
  "environment": "<environment state patch>",
  "music": "<music patch>",
  "conversation": "<conversation patch>",
  "window": "<window patch>",
  "door": "<door patch>",
  "airConditioner": "<AC patch>",
  "readingLight": "<reading−light patch>",
  "seat": "<seat patch>",
  "sunroof": "<sunroof patch>",
  "sunshade": "<sunshade patch>",
  "lowBeamHeadlight": "<low−beam patch>",
  "highBeamHeadlight": "<high−beam patch>",
  "fogLight": "<fog−light patch>",
  "hazardLight": "<hazard−light patch>"
   ......
 },
 "method_note": "This is a random scene−state sample. Do not assume it must be interaction−worthy. If later images diverge
    from this structured seed, final GT should be grounded in images."
}
```

Listing 4: Prompt used for Labeling.

```
You are generating the Ground Truth for a synthetic proactive cabin interaction sample.

You will be given:
- a generated in-cabin image;
- Optionally, one exterior driving-scene image when available.
- a randomly constructed structured scene and a verified initial vehicle device state;
- the interaction opportunity hint produced by the previous mining stage.

You must follow these rules:
1. The final GT must be based on the two generated images. If the structured seed conflicts with the images, the images take
   precedence. If a detail cannot be confirmed from the images, do not force uncertain details from the seed into the
   analysis as facts.
2. The initial vehicle state has already been set by `init_state_code`; do not repeat state initialization in the response `
   api_code`.
3. Only generate actions describing how the proactive cabin system should respond at the current moment.
4. The direct-execution version should contain only low-risk, reasonable, and necessary proactive actions, and should use
   speech when appropriate to explain what has already been done.
5. Do not control irrelevant devices merely to add more actions. The action set should be minimal and sufficient.
6. All code must be executable Python single-line strings, and may only use calls/assignments related to `vw.` or `
   Environment.`.
7. Do not invent APIs. You may only use APIs that exist in the API docs below.
8. If the API docs do not provide a direct API for pausing music, prefer legal actions such as lowering the volume.
9. The output must be strict JSON. Do not use markdown or provide explanations.

The output JSON must contain exactly the following top-level keys:
- scene_analysis
- status_analysis
- direct_execution_variant

`direct_execution_variant` must contain:
- decision_reasoning
- api_code


[IMAGE] cabin observation image
[OPTIONAL IMAGE] exterior driving-scene image, if available

Current vehicle device status:
[CURRENT_FULL_DEVICE_STATUS_JSON]

Available VehicleWorld APIs:
[VEHICLEWORLD_API_DOCS]
```

Listing 5: Prompt used for Text Repair After Voting.

```
You are the GT text repairer for a proactive cabin interaction dataset.

You will receive a sample that has already gone through multi−model voting. The voted API action set has already been
    determined, and the action set must not be changed.

Your task is limited to only three things:
1. Rewrite `direct_execution_variant.decision_reasoning` so that it explains the voted direct `api_code`, rather than
    describing the voting process.
2. Rewrite `confirmation_variant.decision_reasoning` so that it explains the confirmation−based workflow, rather than
    describing the voting process.
3. Rewrite the Chinese text for voice announcements and center−console pop−up messages so that it accurately covers the
    API actions actually executed, without claiming that nonexistent actions were performed.

Strict constraints:
− Do not add, delete, or replace any control action.
− Do not change any API function name, parameter structure, `priority`, `level`, `position`, `switch`, `action`, or other
    non−text parameters.
− Only provide new `content` for the `text_items` I list.
− The text must be concise, natural, in Chinese, and suitable for proactive in−vehicle interaction.
− The direct text may state what has already been done for you.
− The confirmation pre text must ask for consent in the form of Do you need me to execute ... for you?
− The confirmation post text must state which actions have been completed after consent.
− Do not mention GPT, Claude, Gemini, voting, multi−model, consensus, or semantic key.

Output strict JSON, containing only these keys:
{
  "direct_decision_reasoning": string,
  "confirmation_decision_reasoning": string,
  "text_rewrites": [
    {"path": string, "content": string}
  ]
}
```