How Electric Era builds trustworthy AI

A technical whitepaper on response correctness, customer data security, and abuse prevention for HaloAI, the digital assistant built into Electric Era EV charging stations

By Hasitha Dharmasiri, VP Software Engineering, Electric Era

Electric Era

Executive summary

Public-facing AI systems carry brand risk for the operator hosting them. A confidently wrong answer, a leaked piece of data, or a successful jailbreak doesn't just affect the AI vendor; it affects the storefront where the kiosk sits. We've architected HaloAI, a part of the Electric Era RetailEdge Platform, with that asymmetry in mind.

Our approach rests on three pillars:

1. Response correctness through scoped context, a shared event bus that supplies all factual grounding, and a strict policy that Halo does not draw on its parametric knowledge for facts.
2. Customer data security through approved-use boundaries, ephemeral session context, a cache-only on-device data posture, and a hard isolation boundary between Halo and cardholder data.
3. Abuse prevention through a parallel response validator that corrects the conversation or terminates it, plus a deliberately narrow capability surface that excludes high-risk tool use.

Underpinning all three pillars is a model evaluation pipeline that gates every change Halo ships, with golden-set regression testing, adversarial evaluation, skill-scoped evaluation, and canary deployments. The remainder of this paper walks through each component and cites industry benchmarks for the techniques in use.

1. Background: Where AI lives in the Electric Era stack

The RetailEdge Platform is Electric Era's DC fast charging platform designed from the ground up for retailers. In addition to charging EVs at speeds up to 400 kW per stall, every charging kiosk includes an interactive touchscreen and two-way audio to deliver a rich driver experience in the forecourt environment. HaloAI is the embedded assistant layer; it operates as a voice-to-voice model, taking driver speech as input and producing spoken output.

Halo serves three operational roles:

1. Multilingual digital station attendant for driver guidance and real-time troubleshooting.
2. Retail concierge for loyalty signups, membership validation, and retailer promotions.
3. Reliability layer that surfaces telemetry-driven explanations of session behavior to drivers and feeds operational signals back to our predictive maintenance system.

The architectural component most relevant to this paper is the **event bus**. The event bus is a structured, real-time feed of authoritative technical and operational state, including charger state, session telemetry, vehicle handshake data, plug-in behavior, fault conditions, payment state, and retailer-provided context. It is exposed to Halo and to every other application running on the charger.

Halo does not generate facts from its parametric knowledge. It reads them from the event bus and reasons about them within the bounds of the active skill. Every correctness, data security, and abuse-prevention property described in Figure 1 depends on that distinction.

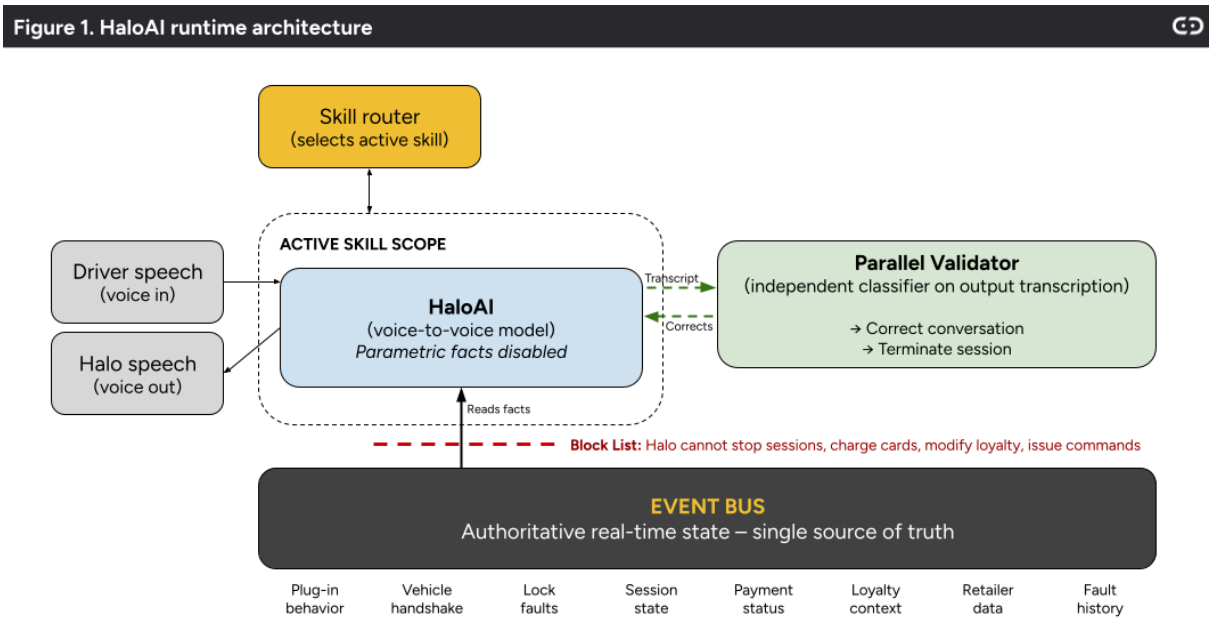

Figure 1. HaloAI runtime architecture. Skill router selects the active skill scope, Halo reads from the event bus, the parallel validator runs on the output transcription.

2. Ensuring correctness of AI responses

Hallucination, defined here as a model output that is not supported by the available context, is a failure mode with one of the highest brand costs for retail partners. Unoptimized retrieval-augmented systems—those that pass retrieved documents to a model without reranking, grounding constraints, or output verification—consistently plateau around 80% factual accuracy on domain-specific QA tasks, even when given relevant and accurate context.

The architecture described in this whitepaper pushes well beyond that baseline by constraining what Halo is allowed to reason about in the first place and by validating outputs against the same ground truth Halo had access to.

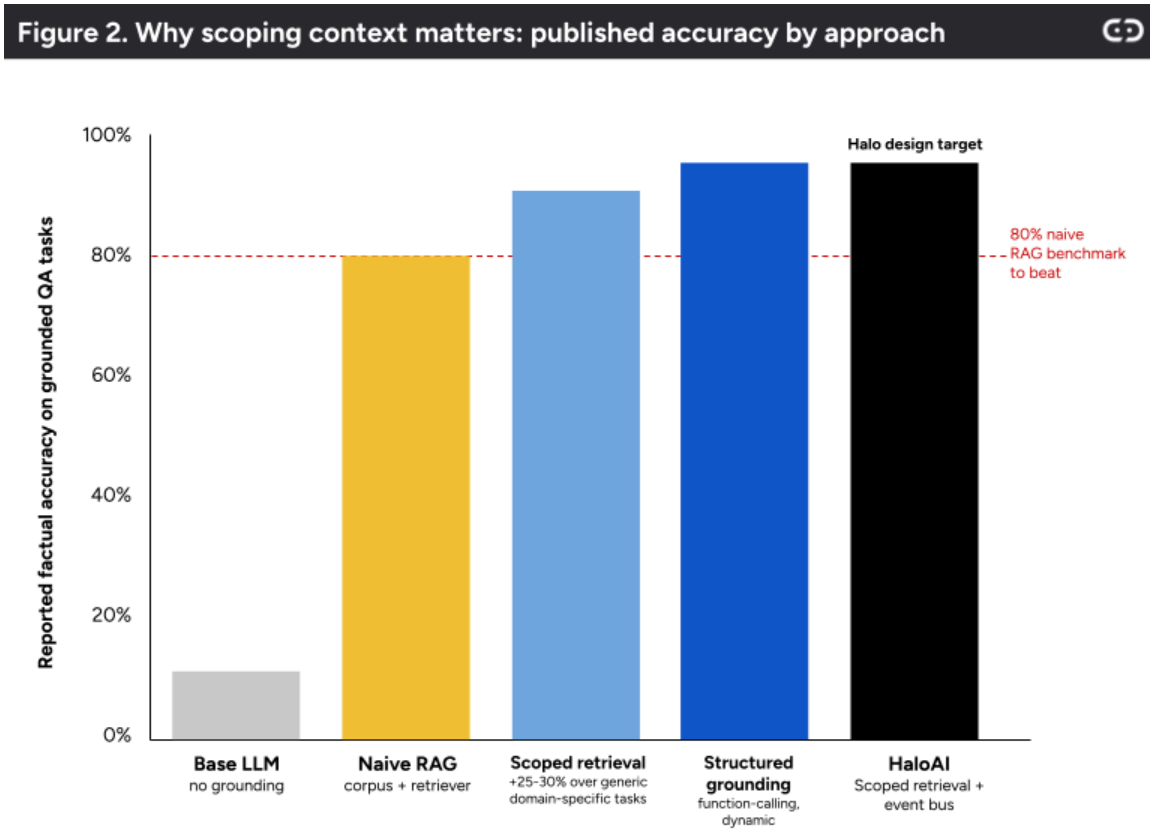

Figure 2. Why scoping context matters: accuracy by approach. Naive RAG systems set an ~80% benchmark to beat; scoped retrieval and structured grounding shift the floor up. Figures shown are representative, not individual benchmark results.

2.1 Scoped context over open-ended generation

Halo operates within tightly scoped contexts driven by the active work function, which we call a *skill*. Example skills include:

- First-time driver plug-in guidance
- Active session troubleshooting
- Charging speed and time-to-target explanations
- Loyalty signup and membership validation
- Retailer-configured promotion delivery

Each skill receives only the slice of event-bus state it needs to perform its function. The troubleshooting skill is granted access to session telemetry and recent fault history; the loyalty skill is granted access to the retailer's loyalty integration; the two contexts do not overlap. When a query falls outside the active skill's scope, Halo is configured to refuse or redirect rather than improvise.

This is functionally equivalent to RAG in that both approaches ground generation in an external state, but scoped context is stricter at the boundary. Rather than giving the model a corpus and a retriever, we predefine the operating envelope per skill. Published research on long-context LLMs shows that as active context length grows, reasoning accuracy degrades. Restricting context to the minimum required by the active skill keeps the model within the regime where it performs well.

Independent benchmarks of scoped retrieval and tool-use approaches report accuracy gains of roughly 25 to 30% over generic prompting on domain-specific tasks, and dynamic retrieval methods that condition on the active task report function-calling success rate improvements ranging from 23% to 104% over static retrieval baselines. These figures illustrate the gains available when the context window is narrowed to what the active task actually needs.

2.2 The event bus as the sole source of facts

When Halo needs to answer a factual question about the session, the charger, or the vehicle, it queries the event bus. It is explicitly configured *not* to draw factual claims from its parametric knowledge (the LLM weights produced during pre-training). Pre-trained model weights cannot have learned facts about a specific charging session at a specific stall on a specific date, and they cannot have learned the retailer's current loyalty terms; any answer the model produced from its weights about those topics would be a hallucination by definition. Confining factual grounding to the event bus removes that failure mode at the source.

The event bus exposes telemetry-derived insight into the categories that drive the majority of driver questions:

- **Plug-in behavior:** Whether the connector is seated correctly, whether the handshake between the vehicle and charger completed, and whether the cable needs to be re-seated.

- **Vehicle lock faults:** Whether the vehicle's charge port lock engaged, disengaged, or failed to release, and the recommended driver action.
- **Vehicle compatibility:** Whether the connecting vehicle is electrically and protocol-compatible with the dispenser. A common case is a Tesla without a CCS retrofit attempting to use a CCS connector. Halo can identify the mismatch and direct the driver to a compatible stall or adapter.
- **Human-induced charging errors:** This is the largest category in our operational data. Common causes include the cable not being fully inserted, the vehicle not being in the correct charge-enable state, payment authorization not yet completing, and the driver unlocking the vehicle during an active session, which terminates charging on most vehicles. These are conditions a driver can resolve in seconds if accurately diagnosed.
- **Session state:** Current power delivery, elapsed time, projected time to target state of charge.
- **Payment and loyalty state:** Authorization status, applied discounts, loyalty enrollment state.

To ensure accuracy, the relevant property is that Halo's answers to "Why isn't my car charging?" are grounded in the actual state of the specific session, on the specific stall, with the specific vehicle handshake in question. This is structurally similar to a function-calling architecture in which the model selects a tool and is then constrained to reason about its return value. Industry literature on function-calling and tool use consistently identifies grounding in structured data as a primary mechanism for reducing hallucination, with observed reductions in hallucination rates of approximately 40% relative to non-grounded baselines depending on the technique and benchmark.

Because the event bus is shared infrastructure, every application on the charger reads from a single source of truth. This eliminates a class of inconsistency bugs in which the on-screen UI, Halo, and a retailer-built experience could disagree about the state of the session.

2.3 The model evaluation pipeline

Scoped context and the event bus reduce the surface area for errors. The evaluation pipeline is how we hold that surface area shrinking over time and gate every change before it reaches production.

Golden-set regression testing
We maintain a curated corpus of real driver questions, edge cases, multilingual variants, and prior failure modes drawn from production. Every model change, prompt change, or skill change is run against this corpus and scored along four axes: 1) factual correctness against the simulated event bus state, 2) scope adherence, 3) tone and brand fit, and 4) refusal behavior on out-of-scope inputs. Changes that regress on any axis are blocked from progression.

Adversarial and red-team evaluation
A separate suite tests Halo against jailbreak attempts, prompt injection patterns, off-topic baiting, and offensive-content elicitation. Public benchmarks for guardrail evaluation, including AdvBench,

JailbreakBench, and the deepset prompt-injection corpus, inform suite construction. Every release is gated on this suite. New attack patterns observed in production are added before the next release.

Skill-scoped evaluation
Each skill carries its own evaluation suite, allowing independent iteration on individual skills without requiring unrelated functionality to be revalidated. This reduces release latency and reduces the likelihood that a regression in one skill blocks improvements to another.

Automated scoring with human review on subjective dimensions
Objective dimensions, including factual correctness, schema conformance, scope adherence, and refusal, are scored programmatically. Subjective dimensions, including helpfulness, naturalness, and brand fit, are reviewed by humans. Disagreements between automated and human scores are added to the evaluation corpus, which monotonically increases coverage over time.

Canary deployments
Changes that pass offline evaluation are deployed to a small fraction of stations first. Live signals on the canary fleet, including validator-flagged response rates, session-completion rates, and driver feedback, are monitored before fleet-wide rollout. A canary that breaches predefined thresholds is held and investigated before rollout proceeds.

Continuous improvement loop
Conversations flagged in production by the parallel validator (described in Section 5) feed back into the evaluation corpus as new test cases. The evaluation bar therefore rises monotonically, and the same failure cannot recur across releases.

The aggregate property is that every version of Halo deployed to an active charging station has cleared a quantitative gate on a battery of automated and human-reviewed tests representative of real-world conversations and adversarial inputs.

Figure 3. Model evaluation pipeline

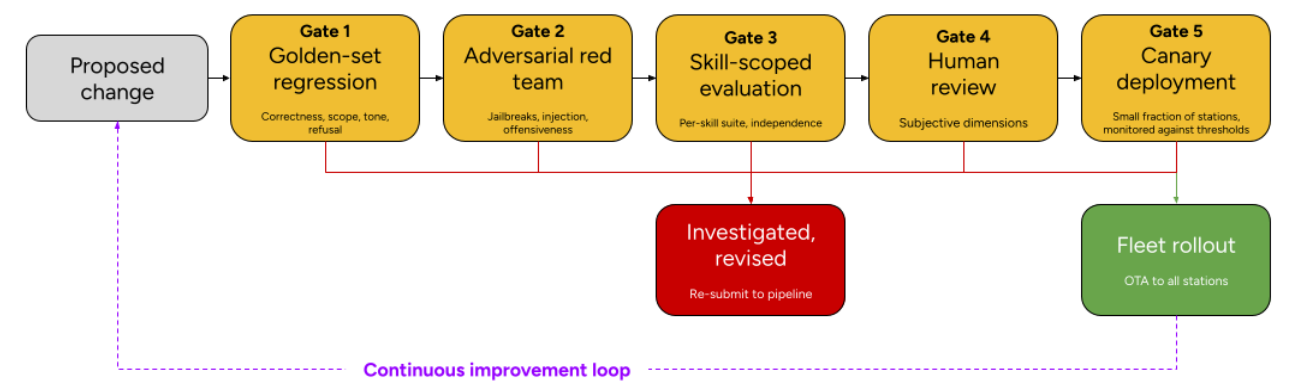

Figure 3. Model evaluation pipeline. Every model, prompt, or skill change passes through five gates (golden-set regression, adversarial red-team, skill-scoped eval, human review, canary deployment) before fleet rollout. Production validator findings feed back into the evaluation corpus.

3. Security of customer data

The kiosk is a public, payment-accepting device that handles driver interactions. Standard data protection controls apply across the platform: TLS for data in transit, encryption for data at rest, and centralized key management. Beyond those baseline controls, our data security architecture has four layers specific to how Halo interacts with customer data: 1) approved-use boundaries, 2) a cache-only on-device posture, 3) context wiping on deauthorization, and 4) a hard isolation boundary around cardholder data.

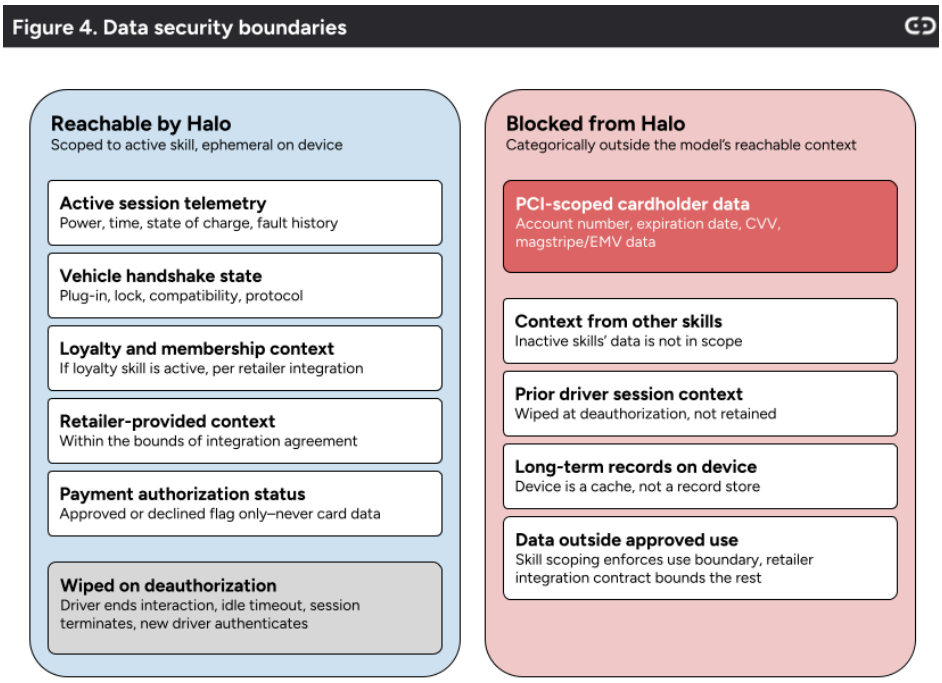

Figure 4. Data security boundaries. The blue column shows what is reachable by Halo through approved use; the red column shows what is categorically blocked.

3.1 Approved data use

Halo acts on data only within the approved use defined by the active skill and the integration contract with the retailer. Session telemetry is used for session assistance. Loyalty data is used for loyalty operations. Retailer-provided context is used within the bounds of the integration agreement.

Repurposing of data across skills or beyond the agreed integration is not architecturally available to the model.

This is enforced through the skill-scoping mechanism in Section 3.1. A skill that has no defined use for a data class does not receive that data class in its context. The enforcement is structural rather than purely policy-based.

3.2 Cache-only on-device posture

The charging kiosks have substantial local storage capacity, which is necessary to support media playback for retailer experiences, OTA staging, telemetry buffering during connectivity loss, and on-device inference. That storage is operated as a cache, not as a record store. Driver interaction data, session context, and any personally identifying information that transits the device are written only to ephemeral cache locations and are evicted when no longer needed for the active operational purpose.

Two properties motivate this design:

1. **Staleness elimination:** Halo cannot reason about a current driver using residual state from a prior session. Eviction at session boundaries removes a class of context-leak bugs that affect persistent caches.
2. **Reduced data-at-rest attack surface:** Attacks against data at rest target persistent storage on a device, typically through physical access or supply-chain compromise. A device that does not retain customer data across sessions presents a structurally smaller target. Combined with the encryption-at-rest controls noted previously, an attacker who obtains the storage media still finds neither persistent customer records nor decryption keys for what little ephemeral data may be present.

The kiosk is not a long-term record store and is not operated as one. Records that need to persist (anonymized session metrics, retailer reporting data, operational telemetry) are forwarded to our cloud backend over TLS and managed under our cloud data retention policy.

3.3 Context wiping on deauthorization

Halo's working context for an active driver interaction is wiped when a deauthorizing event occurs. Deauthorizing events include:

- The driver explicitly ending the interaction.
- An idle timeout with no further driver input or presence.
- Session termination at the charger level (session complete, session aborted, or session faulted).

- A new driver authenticating to the same kiosk (new payment detected, new vehicle ID entered).

After any of these events, the prompt history, the populated event-bus context, and any transcribed driver speech for that interaction are evicted from memory. The next driver encounters a fresh context. This eliminates cross-session leakage at the model layer in addition to the storage layer.

3.4 The cardholder data boundary

Tap-to-pay payment data is the most sensitive class of data the kiosk handles. It is fully isolated from every application on the kiosk, including Halo, by a PCI-scoped firewall.

Cardholder data flows through a payment path that is segregated at the application boundary. Halo cannot read it, cannot query it, and cannot infer it. It does not appear in:

- Halo's prompt or context window
- The event bus
- Application logs
- Telemetry sent to the cloud
- Any artifact accessible to any non-PCI-scoped application on the charger

The relevant property is categorical: there is no failure mode by which Halo can leak cardholder data, because Halo does not have access to it. No prompt injection, no jailbreak, no log misconfiguration, and no model-internal state can expose data that is categorically outside the model's reachable context.

4. Preventing abuse

Public voice AI systems encounter a predictable set of adversarial behaviors, and our architecture is designed on the assumption that do occur. Our primary defenses are a parallel response validator running independently of the primary model and a deliberately narrow capability surface that excludes high-risk tool use entirely.

4.1 The abuse patterns we plan for

The categories below are well-documented across the public-facing voice AI literature. We address each through a combination of architectural scoping, capability restriction, and validator coverage:

- **Jailbreak and prompt injection attempts:** Inputs designed to override Halo's instructions or cause it to act outside its skill scope. Because Halo's capabilities are scoped to the active skill,

the surface area available to a successful jailbreak is small; the validator catches attempts that do influence output.

- **Off-scope use:** Requests outside the active skill, such as general trivia, homework help, or extended unrelated conversation. Halo redirects these per its scope policy, which also preserves kiosk availability for the next driver.
- **Inappropriate language or harassment directed at the assistant:** Detected by the validator, which can correct the conversation or terminate it.
- **Impersonation and social engineering:** Claims to be retailer staff or Electric Era support to extract privileged behavior. Privileged behavior is gated by authenticated context, not by user-asserted identity.
- **Resource exhaustion:** Extended monologues, repeated triggering, or other patterns that tie up the kiosk. Per-session and per-kiosk rate limits bound this.

4.2 Capability surface restriction

The strongest abuse-prevention measure in our architecture is what Halo categorically cannot do. Halo does not have tool-use access to consequential actions on the kiosk or in the retailer's systems. In particular, Halo cannot:

- Initiate or authorize a payment, complete a purchase, or apply a discount outside the predefined retailer integration.
- Stop, pause, or alter an active charging session.
- Modify driver account state, loyalty balances, or membership status.
- Issue commands to the charger hardware or to retailer point-of-sale systems.

This is enforced at the system boundary, not in the prompt. A successfully jailbroken Halo still cannot stop a session or charge a card, because the action surface to do so is not exposed to the model. This forecloses the most consequential attack vectors—not reputational embarrassment, but manipulation into actions that affect the driver, the retailer, or the charger. Where Halo needs to assist with such actions, it does so by guiding the driver through the standard authenticated UI flow rather than performing the action itself.

As Halo's capability set expands toward agentic commerce, each new exposed action will be added behind explicit authentication, explicit driver consent, and a separate abuse-prevention review.

4.3 Parallel response validators

Halo's output is monitored by an independent validation layer that runs in parallel with the primary model. Because Halo is a voice-to-voice model, the validator operates on a transcription of Halo's spoken output rather than on a text token stream emitted directly by the model.

To preserve the low-latency interaction expected of a voice assistant, the validator does not gate output. Output is rendered to the driver as the model speaks. The validator runs alongside in real time and acts on what it observes, with two intervention modes:

1. **Conversation correction:** When the validator detects drift toward an off-scope, hallucinatory, or off-rails response, it injects a correction signal that steers Halo's next turn back into scope. The cost paid is a brief stretch of suboptimal conversation rather than a frozen kiosk or a halted interaction.
2. **Conversation termination:** When the validator detects abuse signals such as jailbreak attempts in progress, prompt-injection patterns, social-engineering markers, or sustained offensive content, it terminates the conversation. Halo gracefully ends the interaction and the kiosk returns to idle.

This non-gating design is a deliberate latency tradeoff. A serial gate on every spoken token would introduce noticeable lag in conversational turns and would degrade the user experience disproportionately to the marginal safety benefit, given that the upstream defenses (scoped context, capability restriction, and the absence of consequential tool use) already prevent the most severe outcomes at the architectural level. The validator is designed to identify vulnerabilities that bypass prior security layers. By operating without processing every individual statement in sequence, it maintains a high level of oversight while preserving the natural flow of the interaction.

The architectural decision to run the validator independently of the primary model rather than as a self-check is also deliberate. A model that has been successfully manipulated cannot be relied on to flag its own manipulation; this is documented in the OpenAI guardrails literature and elsewhere. An independent validator with separate logic and separate thresholds does not share the failure mode of the primary model.

The effectiveness of this class of defense is well-established in the literature. Anthropic's published work on many-shot jailbreaking reports that adding an input classifier in front of the primary model dropped the attack success rate from 61% to 2% in a best-case configuration. Layered defenses of this kind, in which an independent classifier intercepts adversarial patterns the primary model would otherwise comply with, are the standard industry response to jailbreak and prompt-injection attacks.

We also note that no guardrail is perfect. Detection accuracy on classifiers trained against known attack patterns has been observed to drop substantially when those classifiers encounter novel out-of-distribution attacks. We address this through the continuous improvement loop in Section 3.3: novel attack patterns observed in production are added to the evaluation corpus and to the validator's training distribution before the next release.

Flagged conversations also feed into the evaluation pipeline, which means each abuse attempt structurally improves the next release.

5. Closing

The engineering principle behind the RetailEdge Platform, drawn from our team's aerospace background, is that errors should be detected and corrected before the customer sees them. We've applied that principle to the AI layer as well as the hardware. Scoped context and the no-parametric-facts policy constrain what Halo reasons about. The event bus constrains what it claims as fact. The evaluation pipeline constrains what we ship. Approved-use boundaries, the cache-only on-device posture, context wiping, and the cardholder firewall constrain what data is reachable. The capability surface restriction and the parallel validator constrain what Halo can do and what reaches the driver.

As Halo's capability set expands toward agentic commerce from the kiosk and the vehicle, each new capability passes through the same gates. The system grows, but the bar for shipping does not move.

Learn more at electricera.tech.