

HUBSPOT BREEZE · SHOPIFY · MAKE.COM

Rebuilding a live AI customer agent that was quietly failing

A post-launch diagnostic and rebuild of an AI customer service agent for a luxury jewellery retailer — tracing customer-facing failures to their root cause in agent instruction architecture, then re-engineering the escalation logic and the Shopify order pipeline behind it.

AI Agent Engineering

Root Cause Diagnostics

API Integration

Escalation Architecture

Security Remediation

ENGINEER

Manan Asim

HubSpot RevOps & Automations Engineer

CLIENT

Oliver Jewellery

Luxury retail · Toronto

ENGAGEMENT

Post-launch remediation

Diagnostic, rebuild & handover

01 — CLIENT OVERVIEW

Who the client is

Oliver Jewellery is a single-boutique luxury jewellery and accessories retailer in Toronto, trading in pre-owned and one-of-a-kind pieces — Birks, Tiffany, Hermès, Prada — across both an e-commerce storefront and an in-person showroom.

Their inventory model is what makes this engagement unusual. Because the pieces are pre-owned and largely unique, the catalogue is enormous, individual items sell once and disappear, and prices move. Any system that answers customer questions about products has to contend with a catalogue that is both very large and constantly changing.

PRODUCTS IN PORTAL

46,464

39,236 carrying a product URL

CONTACT DATABASE

77,282

Synced one-way from Shopify

PLATFORM TIER

Service Hub Pro

Breeze Customer Agent enabled

SUPPORT TICKETS

1,118

Across three pipelines

02 — SITUATION OVERVIEW

Where the engagement started

The client had commissioned an AI customer service agent, deployed on their live chat and email channels, to handle product questions, order status and general enquiries. The build was delivered against an agreed ten-hour scope covering knowledge setup, CRM data access, routing, escalation rules and training.

My scope on the original build was a single component: a custom action allowing the agent to answer live order-status questions by calling the Shopify Admin API through a Make.com webhook, returning fulfilment and payment status into the conversation. The remaining configuration — knowledge sources, tone, guardrails, escalation triggers and routing — was handled by other contributors.

The agent went live. Days later, the client's Customer Service Manager emailed the delivery team with two detailed complaints and a series of chat transcripts showing the agent failing in front of real customers. I was brought in to audit the full system, find the cause, and fix it.

THE COMPLICATION

The failures were not in the component I had built. Diagnosing them meant auditing configuration authored by other people, across a system already live to customers, without a specification describing what the current settings were supposed to do.

Two complaints, four failure modes

The client sent transcripts rather than descriptions — which turned out to be decisive. The agent's own words contained the diagnosis.

Complaint one: the agent could not answer product questions

Customers asking for an item's price by number received a refusal, then a request for a product link — and when the link was supplied, a request for it again. In other cases the agent confidently surfaced a *different* product: a 2.34 tcw bracelet in place of the 0.57 tcw one requested, or two unrelated handbags in place of the one asked about. Prices, where returned at all, were quoted in US dollars to customers of a Canadian boutique.

Complaint two: escalation to a human never happened

Asked whether a handbag included a particular strap, the agent replied that it could not confirm, and advised the customer to check the product page themselves and then contact the team. No handover occurred. No ticket was created. The customer was returned to the website with the question unanswered, and the team never learned the conversation had happened.

“This does not make sense — the customer *is* asking the team, and why can't she check the included items section? Why is human handoff not happening?”

CUSTOMER SERVICE MANAGER · CLIENT ESCALATION EMAIL

What the transcripts actually revealed

Four distinct failure modes sat underneath two complaints — each with a different cause and a different fix:

#	OBSERVED BEHAVIOUR	UNDERLYING CAUSE
01	Wrong product returned as if correct	Knowledge retrieval is semantic, not exact — it returns the nearest match by meaning, and always returns something. An item number carries almost no semantic signal, so adjacent SKUs resolve near-identically.
02	Repeated requests for a product link	Emergent, not configured. A style rule required every reply to end with a next step; with escalation unavailable, the only step the agent could construct was to ask for more information — including links it cannot open.
03	“Contact our team” with no handover	An instruction literally directed the agent to say this. Producing the sentence and performing a handover are separate mechanisms; only one was wired.
04	Prices shown in USD	Portal currency left at platform default, compounded by crawled page text and no governing instruction.

THE DIAGNOSTIC KEY

The agent was not uniformly broken. In one transcript it answered a product question correctly, with citations. That single success ruled out a broken retriever and isolated the real variable: **coverage**.

The coverage ceiling

Before rebuilding anything, the first question was whether product knowledge could be fixed at all by adding more of it. It could not, and proving that changed the entire shape of the engagement.

An audit of the agent's knowledge configuration found 47 sources: a site-wide crawl expanding to 2,216 URLs, a set of individually imported product pages, a small library of help articles, a handful of short answers and one file.

CATALOGUE URLS

39,236

Product pages on the live storefront

ACTUALLY INDEXED

2,216

Roughly 5.6% coverage

Three independent constraints made closing that gap impossible by adding sources:

- **Volume.** The platform caps a website knowledge source at 1,000 pages. The catalogue is nearly forty times that.
- **Freshness.** Crawled sources refresh weekly with up to a seven-day content lag. For one-of-a-kind stock, an item can sell and the agent will keep quoting its price for days.
- **Retrieval mechanics.** Even at full coverage, semantic search cannot perform exact-key lookup. Better coverage would have made failures *more* dangerous, not less — a confidently wrong price instead of an honest refusal.

FINDING

The product knowledge problem was never a configuration defect. It was an architectural ceiling that had been mistaken for one — and the contracted scope had correctly placed full catalogue coverage in a later phase. The engagement's real question became: what is the correct behaviour for an agent that *cannot* know the answer?

The answer: engineer the refusal, not the answer

Rather than chase coverage, the system was re-architected so that not knowing became a first-class, safe outcome. An agent that reliably says “I can't confirm this, let me get someone who can” is strictly more valuable to a luxury retailer than one that guesses at the price of a piece worth tens of thousands of dollars.

That reframing converted an unfixable problem into a solvable one, and kept the work inside the commercially agreed scope rather than expanding it under deadline pressure.

Rebuilding the instruction architecture

The escalation failure traced to two specific sentences across two different configuration fields. Both read as sensible in isolation. Together they made human handover structurally impossible for the most common category of customer question.

Defect one: an instruction that produced text instead of action

A rule governing stock availability directed the agent, when unsure, to point customers to the website or tell them to contact the team. The agent followed it precisely — and generalised it beyond stock to every unconfirmable product attribute. The complaint the client raised was the instruction executing correctly.

Defect two: escalation defined as keyword matching

Two separate fields declared that handover applied when a listed phrase appeared in a customer's message. The list covered refunds, repairs, sizing and complaints. It contained nothing for price, product detail, availability or item numbers — by a wide margin the highest-volume questions in retail chat. A customer asking whether a bag came with a strap matched nothing, so escalation was never evaluated.

BEFORE

- Escalation gated on literal phrase matching
- No trigger for price or product questions
- Instruction to redirect customers to the website
- Agent requests links it cannot open
- Similar products offered as substitutes
- No currency governance

AFTER

- Dual-clause escalation: phrase match *or* inability to resolve
- Six product-question categories added as triggers
- Redirect instruction removed entirely
- Explicit rule: never request a link
- Exact-match-only rule on all product facts
- Currency fixed at source and in policy

The rewritten architecture separates the two things that had been conflated: **what the agent knows** and **what the agent does when it doesn't**. Escalation now fires either on intent match or on a self-assessed inability to resolve, and the instruction set states plainly that handing off means performing the handover, not describing it.

Working inside platform constraints

Each configuration field carries its own undocumented character ceiling — 2,500 on one, 1,000 on another. The rewritten logic had to be compressed to fit without losing a single behavioural rule, and validated against the platform's pre-flight checker, which flagged contradictory precedence claims between fields and over-literal trigger phrasing. Both classes of warning were resolved by restructuring rather than by publishing through them.

Re-engineering the order pipeline

The order-status integration was the one component the client confirmed was working. Auditing it anyway surfaced three latent faults and one active security exposure.

01 Webhook Agent passes the customer's order number	02 Token exchange Client-credentials grant against Shopify	03 Order query Admin API lookup, all statuses	04 Structured reply Mapped payload returned to the agent
---	--	---	--

FAULT FOUND	RISK	REMIEDIATION
Credentials hardcoded in the integration body	Permanent app credentials embedded in an exportable file already shared across the team. Token rotation offered no protection — the keys minting those tokens never rotate.	Rotated and moved to shared platform variables, so no future export carries a secret.
No failure path	Both HTTP calls halted the scenario on error, so an API fault meant the webhook never responded and the agent received nothing.	Errors now flow to a structured response, letting the agent degrade into escalation instead of going silent.
Ambiguous empty states	A nonexistent order and a paid-but-unshipped order produced identical output. Correct behaviour depended on which field happened to be populated.	Explicit result flag and order-name echo, so match and non-match differ by design.
Expiring API version	Pinned beyond its supported window — a silent, scheduled outage nobody had visibility into.	Upgraded to a current supported version.
Data left on the table	The full order record was retrieved, then discarded in favour of two fields. Customers were told to find tracking in their own email.	Carrier, tracking number, link and ship date now surfaced directly in the conversation.

ZERO-DOWNTIME MIGRATION

The rebuilt payload deliberately preserved the original field names and reused the existing webhook identifier, so the upgrade deployed against a live, client-confirmed integration with no coordinated cutover and no window in which the agent read fields that no longer existed.

Closing the loop behind the handover

Fixing escalation triggers only matters if escalations arrive somewhere a human will see them. They did not.

The handover process created a ticket and assigned it to nobody. The portal contained over a thousand tickets, the overwhelming majority unassigned, with no service-level rules in place. Increasing escalation volume into that queue would have made the client's experience worse, not better.

Correcting it surfaced a subtler trap. An intermediate configuration assigned tickets to a named individual while also restricting assignment to available users — a combination that silently reverted every out-of-hours handover back to unassigned. A second option routed by contact owner, which would have failed just as quietly, since the vast majority of synced contacts carry no owner at all. The portal had no teams configured to route to. The final configuration guarantees an owned ticket regardless of availability or time of day.

VERIFICATION STANDARD APPLIED

No fix was reported as complete on the strength of having been configured. Each was required to demonstrate the corrected behaviour end to end — a real order number through the live integration, a real escalation producing a real owned ticket — before it was described to the client as resolved.

Controlled testing over assumption

One open question remained: whether the newly added trigger keywords were doing real work, or whether the rewritten product-facts rule was sufficient on its own. Rather than decide it by argument, it was settled by an isolated single-variable test — remove one keyword group, test against a known-uncovered item, then against a previously answerable one.

The result was informative and counter-intuitive. Behaviour was identical with and without the keywords, proving the platform interprets these lists semantically rather than literally, and that the rewritten instruction rule was the actual governing mechanism. The keywords were retained as a redundant safety layer at zero behavioural cost — a decision now supported by evidence rather than assumption.

Designed, scoped, and deliberately deferred

A live product-lookup integration — querying the Shopify catalogue directly by item number to return exact price, availability and specification — was fully designed and built as a deployable artefact. The storefront's URL structure embeds item numbers directly in product handles, meaning a single query path would resolve item numbers, product names and pasted links alike.

It was deliberately *not* deployed. Full catalogue coverage sat in a later contracted phase, and shipping it unbilled under deadline pressure would have set a precedent at the client's expense as much as the delivery team's. It was handed over as scoped, ready work with a single documented dependency: one additional API permission.

What changed

AREA	STATUS	OUTCOME
Human escalation	Fixed and verified	Customers reaching the limit of what the agent knows are transferred to a named owner, not redirected to the website.
Product and price accuracy	Resolved by design	No substituted products, no guessed prices. Anything unconfirmable escalates.
Order tracking	Upgraded and verified	Live carrier and tracking link delivered in-conversation, tested end to end against a real order.
Currency	Fixed	Corrected at portal level and governed by explicit policy.
Integration resilience	Hardened	Graceful degradation on API failure, unambiguous result states, current API version.
Credential exposure	Remediated	Live secrets rotated out of a shared exportable file into managed variables.

The engineering judgement

The highest-value decision here was choosing not to build. The obvious response to “the agent can’t answer product questions” is to make it able to. The correct response — given a hard platform ceiling, a fixed scope and a live customer-facing system — was to make its failure mode safe, honest and routed to a human, then hand the expansion over as scoped work.

The original complaint was an agent that guessed wrong and stranded customers. The delivered system errs toward caution and always reaches a person. For a retailer selling unique pieces at luxury price points, that is the correct direction to fail in.

IMPLEMENTATION TIME

Single cycle

Audit, rebuild, verification and handover

CUSTOMER-FACING DOWNTIME

None

All work deployed against a live agent

TOOLS & TECHNOLOGIES

HubSpot Service Hub

Breeze Customer Agent, Help Desk, routing

Shopify Admin API

Orders, products, OAuth client credentials

Make.com

Webhook orchestration, HTTP, error handling

GraphQL

Catalogue query design

Knowledge & RAG

Semantic retrieval, coverage analysis

Prompt Architecture

Instruction precedence, escalation logic

Manan Asim

HUBSPOT REVOPS & AUTOMATIONS ENGINEER

Diagnostics and delivery on live customer-facing AI systems — agent behaviour, escalation architecture, CRM and commerce integrations.