

CartTrace 100K: Analyzing Agent Behavior Across Stores, Categories, and Commerce Protocols

Pratham Bhatnagar
General Machines
pratham@generalmachines.ai

Deepanshu Pal
General Machines
deepanshu@generalmachines.ai

Abstract—AI agents are beginning to shop and pay on behalf of people through commerce protocols. We report what we observed when one such agent shopped in online stores across product categories and protocols. We ran one open-weight agent, Qwen3.8-27B, through 101,416 shopping journeys in 1,044 simulated stores across 25 product categories, from the first request to a simulated paid order. A simulated shopper gave each journey a goal and withheld some needs until the agent asked. When a journey proceeded to payment, the agent used the protocol’s payment step only after the shopper approved the total. A judge checked each result against the store’s records and the payment processor’s ledger, and a result counts as a success only when those records confirm it. The judge rated about five journeys in six right, and the records confirmed about two in three. Once the shopper approved a charge, it almost never failed. Most journeys the judge rated wrong ended earlier, in a refusal of a request the store was built to fill or in a purchase that missed a need the shopper had held back. The kind of request mattered more than the product category. We present CartTrace-100k, a dataset of these journeys, their stage-level checks and replay tools for studying complete, state-verified shopping journeys.

Index Terms—agentic commerce, shopping agents, commerce protocols, dataset, decision model, evaluation

I. INTRODUCTION

Shopping is moving from the browser to protocols. An agent no longer has to click through a web page to buy something, because the layer underneath the storefront is being rebuilt for it. The Model Context Protocol gives an agent tools [1]. The Universal Commerce Protocol (UCP) gives it a store’s catalog, cart and checkout [2]. The Agentic Commerce Protocol (ACP) lets it run the checkout [3], WebMCP lets a web page offer its own tools [4], and AP2 and MPP carry the payment [5], [6]. Platforms, payment networks and standards bodies are building these protocols because a protocol call is cheaper and more reliable for an agent than a page built for people.

For a store, this adds a new kind of customer: the agent. It never sees the storefront. It reads the catalog through a protocol, works out what the shopper needs, and pays on the shopper’s behalf. A store owner does not need to know how the protocols work, but does need to know how this customer behaves. Will it find the right product? Will it ask the shopper what it needs to know? Where will it give up? Does a shoe store face the same problems as a car-parts store? We found no public study that answers these questions. Existing shopping benchmarks end at a product choice, a cart or a checkout form,

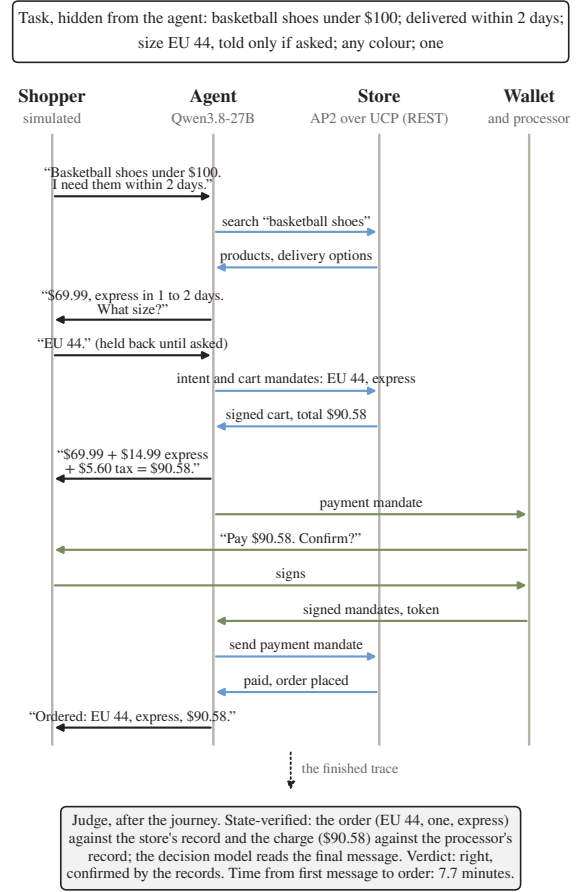


Fig. 1. One released journey from start to finish (footwear; the shopper keeps the size back; AP2 over UCP (REST), paying with AP2’s intent, cart and payment mandates). The shopper and the store are simulated, and the judge reads the finished trace afterwards. Every value shown is read from the released journey; consecutive store calls are merged into one arrow, and an exchange in which the shopper put off giving the size is left out.

and we found none that runs the commerce protocols through to a paid order (Section II).

This paper studies that customer. We built 1,044 simulated stores from open product catalogs in 25 categories, and served them through five protocol surfaces and five payment rails. A simulated shopper gives the agent a goal, such as a guitar strap under \$15, and keeps some needs back until the agent asks. We ran one agent, Qwen3.8-27B, through 101,416 of these

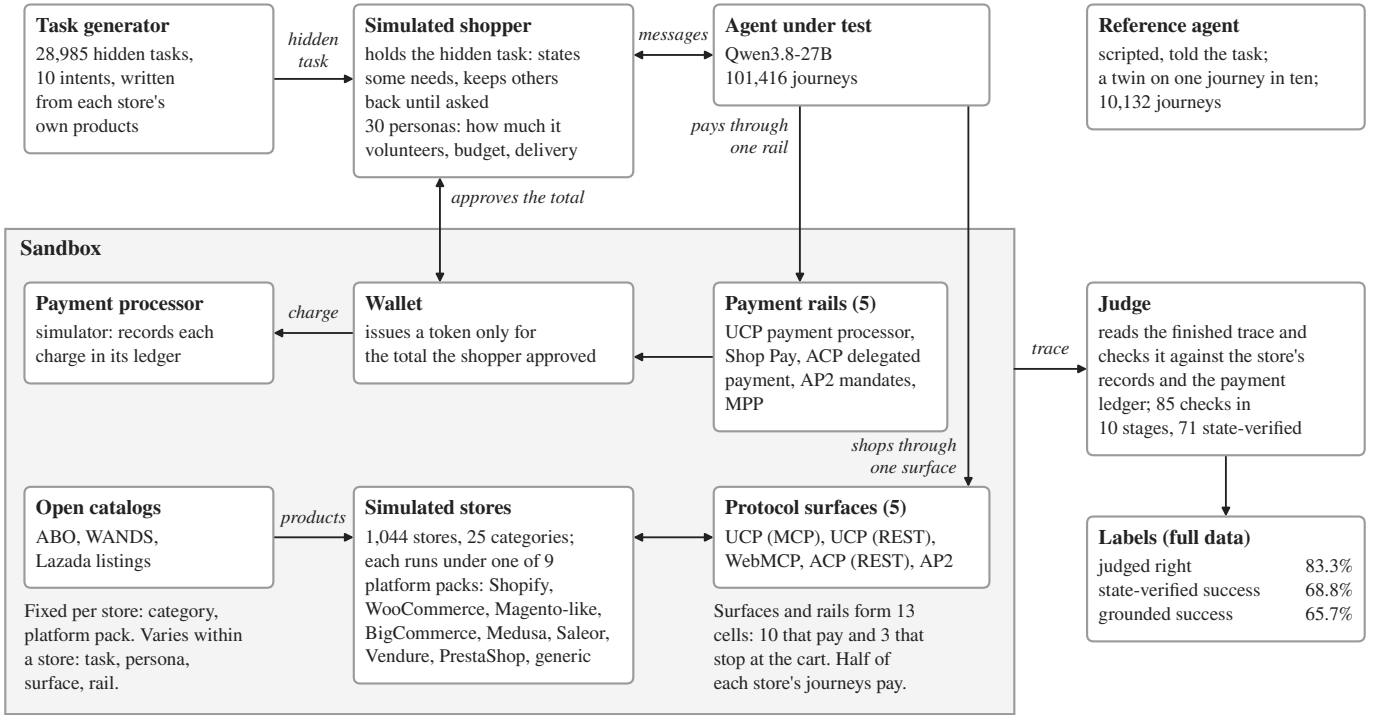


Fig. 2. How the study is set up, with the count at each step. Simulated stores and tasks feed journeys in which a simulated shopper and the agent shop one store through one protocol cell; the released traces are judged against the store's records and the payment ledger, and the labels 1.1 correction gives the success label.

shopping journeys and checked each result against the store's records and the payment processor's ledger (Section IV). Figure 1 shows one of them from start to finish, and Figure 2 shows how the study fits together. We held the agent fixed, so every comparison in the paper is between environments as this one agent met them. These are observations of one agent; another model may meet the same stores differently (Section VII).

Running 101,416 journeys through 1,044 simulated stores, each served under the rules of one of 9 platform packs built from real e-commerce platforms, gave us a detailed view of how a shopping agent behaves inside a store: where it searches, what it reads, when it asks, where it gives up, and which parts of a store's search, product records and checkout it depends on. The judge rated 83.3% of full-data journeys right, and the store's records confirmed 68.8%. Most of the gap is results the records cannot settle, such as a budget whose scope the task left open (Section IV). Once the shopper approved a charge, it almost never failed. Most journeys the judge rated wrong ended earlier, in a refusal or in a purchase that missed a need the shopper had held back. The agent asked the shopper questions in most journeys, but almost as often when nothing was held back as when something was. What the shopper asked for mattered more than the product category, and how often the agent skipped the product page depended on the store's platform and protocol (Section V). Section VI sets out what a store might try in light of these observations.

Contributions. (1) CartTrace-100k, a public set of 101,416

shopping-agent journeys that run through checkout and payment over the commerce protocols, with the judge's stage scores on every journey and a scripted reference agent's twin for one journey in ten. (2) The sandbox, task generator and judge, locked under one specification; a public replay kit replays any released journey in its simulated store and reruns its task with another agent. (3) The findings above, with intervals that resample whole stores, and results reported for each protocol.

II. RELATED WORK

Shopping-agent benchmarks differ in where a task ends and how it is scored. In WebShop [7] an agent follows one instruction in a simulated store; the episode ends when it chooses Buy, and the reward matches the product's attributes, options, type and price to the instruction. ShoppingBench [8] adds intents with vouchers, budgets or several items from one seller over 2.5 million Lazada products; the agent ends by recommending products, which rules check. WebMall [9] has four simulated shops; its checkout tasks end at a filled checkout form and are scored by the offers checked out. DeepShop [10] runs complex queries on live websites, and GPT-4o judges the final answer against each requirement. EComAgentBench [11] keeps some requirements in a profile and in clarifying answers fixed by rule; the agent ends by recommending one product, scored by exact match or rubric. We take from these works attribute-defined tasks checked against the catalog, part of our product pool (ShoppingBench's Lazada listings), and needs

revealed only when asked, answered by rule. Our journeys go past the product choice, cart and form, through the shopper’s approval of the total to a charge that a payment-processor simulator records, over the published commerce protocols. We compare 25 categories; DeepShop reports five and EComAgentBench seven.

In τ -bench [12] a language model plays a user who holds an instruction hidden from the agent; the retail agent changes existing orders, and success requires the final database to equal an annotated goal state. We adopt the hidden task held by a simulated user and the outcome read from recorded state. Because assistant models prompted to play users behave unlike people [13], our shopper’s facts come from rules, and a model only words them. Our task is a new purchase, not a change to an existing order.

Work on agentic commerce adds payment and the store’s side. Agentic Commerce World [14] simulates a market in which buyer and merchant agents transact through the environment’s own protocol. Tasks are scored by verified predicates over actions, state effects and payment records, and the authors show that the final state alone misses errors. In ACES [15] an agent adds one product from a mock storefront page to its cart, and the choice shifts with position, tags, price and rating. ARC [16] checks real stores’ UCP, MCP and WebMCP endpoints up to a checkout link and has browser shoppers add one product to the cart; every test stops before payment, and the score describes how far an agent gets. We take per-step checks from Agentic Commerce World, and from ACES and ARC the store-side question: how what a store exposes changes what an agent does. We run a model agent over the published protocols (UCP, WebMCP, ACP, AP2 and MPP) at pinned versions to a paid order. We found no published study in which an agent pays over these protocols for an order that a payment processor records.

Language-model judges show position, verbosity and self-enhancement biases [17]. On web-agent trajectories, rule-based checks miss successes, and no language-model judge exceeds 70% precision [18]. Decision models such as Jev return a probability for each label in a fixed set; they approach generative judges when the verdict can be read from the evidence [19], but their answers shift with wording [20]. Our judge therefore counts only checks settled from recorded state, plus one decision-model question tested against the payment record. A decision model that shares the agent’s backbone answers the other questions; its answers are stored, not counted. Because the records can leave a correct result unconfirmed, we report the judge’s verdict beside the state-verified label. We follow the agentic benchmark checklist [21], compute intervals that resample whole stores [22] and document the release with a datasheet-style card [23].

III. SETUP

Figure 2 shows the whole setup in one diagram, and Figure 1 shows one journey through it. A journey follows the steps of a purchase: the shopper’s request, finding products, reading their facts, asking about needs the shopper has not stated, building the cart, checking out, the shopper’s approval in the wallet, and the paid order. Catalog, cart and checkout run over the store’s

protocol surface; the payment runs over a payment rail. A task generator writes a hidden task for each journey. A simulated shopper holds that task: it states some needs up front and keeps others back until the agent asks. The agent works the store through one protocol. The store is simulated, built from open catalogs and served under one e-commerce platform’s rules. When the task goes on to pay, a wallet asks the shopper to approve the total, and the payment processor records the charge. After the journey, a judge reads the finished trace and checks the result against the store’s records and the processor’s ledger. Everything runs offline; no live store is behind any journey.

A. Simulated Stores and Platform Packs

A simulated store borrows only a category, a market and a shape from a real public store and serves products from an open pool: every publishable product of Amazon Berkeley Objects [24], WANDS [25] and the Lazada listings of Shopping-Bench [8], sorted into the 25 categories of those stores. Prices and stock are synthetic, in US dollars. Real stores contributed only which product facts and policies each category carries; no product, price, reply or identity of a real store is in the release, and every simulated store is pseudonymised. Because real stores do not all implement a protocol the same way, each simulated store runs under a *platform pack* that serves its products under one platform’s rules (search rule, discount stacking, shipping choice, refused cart lines, MCP handshake): Shopify, WooCommerce, a Magento-like pack, BigCommerce, Medusa, Saleor, Vendure, PrestaShop and a plain generic dialect, each built from the platform’s own wire or source. A store keeps its category, its platform pack, one version of each protocol and one payment processor on every journey, but it serves the protocols side by side: almost every store offers all five surfaces below (4.96 on average), and each of its tasks runs on up to four cells, 3.31 on average, drawn so that the store’s cells are used evenly. Protocol is therefore not tied to store. The run holds 1,044 simulated stores in 25 categories.

B. Protocols and Payment

The official specification files are vendored and pinned. A journey reaches its store through one of five surfaces: UCP (MCP), meaning UCP served over MCP; UCP (REST); WebMCP (tools registered by the store’s own pages, run in a real Chromium); ACP (REST), checkout sessions beside the store’s UCP catalog; or AP2 (intent, cart and payment mandates). If it is allowed to pay, it pays through one of five rails: a UCP payment processor, Shop Pay, ACP delegated payment, AP2 mandates or MPP (an HTTP 402 challenge the wallet satisfies). Each rail runs its own protocol’s wallet step under its official name: the shopper approves the payment where that protocol asks the buyer, and a paying call goes through only with a token the wallet issued after the shopper approved the current total. Every rail settles through a token payment-processor simulator; no card number appears anywhere, test cards included. Surfaces and rails cross into 13 cells: ten that pay, and three catalog surfaces, UCP (MCP), UCP (REST) and WebMCP, for journeys that stop at the cart. Half of each store’s full-data journeys stop at the cart and half go on to

pay; a cart journey drawn on an ACP or AP2 cell shops on UCP (MCP).

C. Tasks, the Shopper and the Agents

Every journey has a hidden specification the agent never sees: predicates over the store’s own product data (a product kind, a brand, an option, a budget, a quantity, a fit, an allergen, a superlative), each tagged as *stated* in the opening, kept *on ask* until the agent asks for that field, or left to a *default*. The generator computes which served variants satisfy every predicate, the task’s *witnesses*, so the judge checks a cart by membership in that set; a task with no witness is impossible, and refusing is its right end. Tasks come from ten intents (Section III-D) and 30 personas. Each store gets five personas with about five tasks each, and each task runs in its own store on up to four cells, 28,985 tasks in all.

A persona is a fictional buyer: a made-up name and address, a payment behaviour keyed to a test token (a card the bank approves, challenges or declines, which the designed failures use) and a shopping style. The style sets how much the buyer volunteers, from stating every need up front to keeping most until the agent asks, or not having a detail to hand the first time it is asked; how tight its budget is; how fast it decides; the delivery it leans to; and constraints a category can turn into needs, such as an allergen or a body measurement. The shopper is a program: rules decide every fact of a reply from the specification, a model words it in the persona’s voice, and a rule-based check confirms the words keep to those facts. The wording model is Qwen3.8-27B, the same model as the agent. The shopper gives a detail only when asked and never makes one up. It approves a payment only at the total it was shown, and only for a cart whose lines meet its hidden specification, so a wrong item is refused at the wallet.

Most journeys are *full data*: a satisfiable task and a persona built to pay, half of them ending at a verified cart and half going on to pay. The rest are *designed failures*, never pooled with full-data rows: impossible tasks, payment faults (a declined card, a failed 3-D Secure check, no usable instrument) and checkout faults forced at completion (a reprice, stock gone, an unshippable address). Qwen3.8-27B [26] is the agent under test, served by vLLM [27] at temperature 0.7, with up to 12,288 output tokens per call and a sampling seed for each model call drawn from the journey’s seed, and with one system prompt for every row: read candidates before choosing, ask one question at a time only when a real choice is open, enter at checkout only what the shopper gave, say what is in the order before the wallet step, and never retry a declined payment unless asked. A scripted reference agent that knows the hidden specification runs on a tenth of the rows, on the same world, task, seed and cell as its Qwen3.8-27B twin. It shows what the labels can award when the task is known; it is not a second model under test.

D. Two Kinds of Task

Shopping for a product the shopper can name and shopping for a kind of product raise different questions, and the study covers both. In a *named-product* task the shopper names the

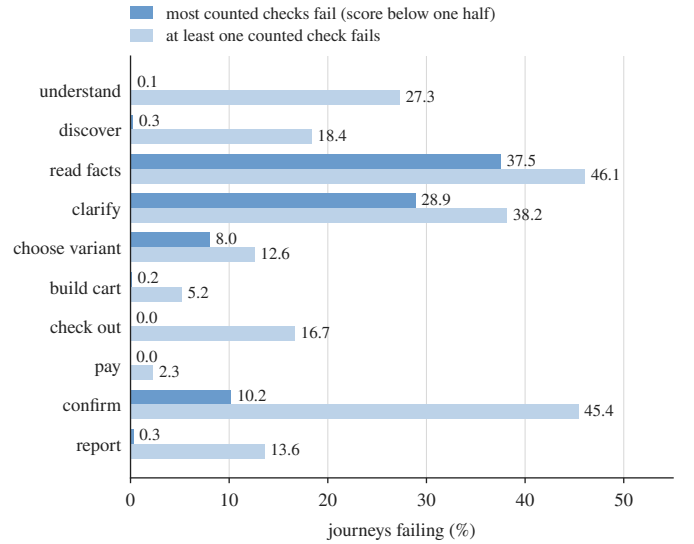


Fig. 3. The judge’s ten stages as one shopping journey, as process checks. Dark bars: the share of Qwen3.8-27B full-data journeys, where the stage applies, whose counted checks score below one half (a weighted mean, so one failed check among several rarely crosses it). Light bars: the share in which at least one counted check fails. A stage failure does not mean the journey was lost; Section V-A gives the failure labels. Table III lists each stage’s checks.

exact product it wants. In a *product-type* task it names only a kind of product, or just a need, and the agent has to find a product that fits. Each kind has five intents; Table I says what the shopper wants in each and quotes one opening message. The two kinds are about equally common: 53.2% of full-data journeys are named-product tasks and 46.8% product-type tasks.

What the shopper holds back also differs between the two kinds (Table II). In a product-type task it is often the variant, such as a size or color, and sometimes the kind of product itself. In a named-product task the variant usually comes with the product’s name; what is held back is how fast the order must arrive, the budget, or whether a substitute will do. So the agent has to ask different questions in each kind of task, and Section V-D shows whether it does.

E. The Run

Everything a journey depends on, from the stores and packs to the agent prompt and the sandbox code, is under one lock whose digest is the specification id, `v1s-9706931d3f3d`; every journey carries it. The run was one non-stop pass, after a preflight that sent the reference agent through every cell, pack and designed fault. A journey hit by a system error, such as a model outage or a browser timeout, was run again, and only completed journeys are released: 101,416 Qwen3.8-27B journeys and 10,132 reference journeys. The judge’s gate passed all 101,416 Qwen3.8-27B journeys and 10,125 of the reference journeys.

F. What a Released Journey Contains

Each released journey is a set of linked records: the full conversation between shopper and agent; every call the agent

TABLE I

THE TEN INTENTS. IN A NAMED-PRODUCT TASK THE SHOPPER NAMES THE EXACT PRODUCT; IN A PRODUCT-TYPE TASK IT NAMES ONLY A KIND OF PRODUCT, OR A NEED. EACH ROW GIVES WHAT THE SHOPPER WANTS AND THE OPENING MESSAGE OF ONE RELEASED JOURNEY WITH THAT INTENT.

Kind	Intent	The shopper wants	Example opening message
Named product	replenish	more of an item bought before	<i>"Same Olight flashlight as before, please."</i>
	compare	one of two named products, picked by a rule such as the lower price	<i>"I'm trying to choose between the AmazonBasics Dutch Oven and the ABC gas type oven. I'd like whichever is cheapest. Which one should I get?"</i>
	bundle	several named items in one order	<i>"Could you add the Graphic Board Notepad and the Laptop Charger to one order, one of each?"</i>
	budget	as many of a named product as a budget buys	<i>"I've got \$55 to spend on the bread knife. Get me as many as that buys and no more."</i>
Product type	substitute	a named product that is out of stock, or something like it	<i>"Can I get the Router Male Adapter? If it's out of stock, something similar from your range will do."</i>
	specify	a product of a given kind in an exact variant, such as a size or color, often told only when asked	<i>"Do you have a gas grill under \$350?"</i>
	express	any fitting product, delivered fast	<i>"Quick one: a razor. Just pick one that works. I'd prefer express shipping."</i>
	gift	a product of a given kind, as a gift for someone	<i>"I need a gift for my brother: a pen. It has to reach me within 2 days, so please send it express."</i>
	discover	suggestions for a need, with no product in mind	<i>"Can you show me a few options for a dog bed?"</i>
	policy	a product, after an answer about the store's policy	<i>"I'm after a wine rack. Before I buy, what is your return window?"</i>

TABLE II

THE TWO KINDS OF TASK ON QWEN3.8-27B'S FULL-DATA JOURNEYS. A NAMED-PRODUCT TASK NAMES THE PRODUCT (REPLENISH, COMPARE, BUNDLE, BUDGET, SUBSTITUTE); A PRODUCT-TYPE TASK NAMES ONLY A KIND OF PRODUCT (SPECIFY, EXPRESS, GIFT, DISCOVER, POLICY). THE MIDDLE ROWS GIVE THE SHARE OF JOURNEYS IN WHICH THE SHOPPER KEEPS EACH NEED UNTIL THE AGENT ASKS; A PRODUCT KIND ALREADY IMPLIED BY A NAMED PRODUCT IS NOT COUNTED.

	Named product	Product type only
Journeys	51,846	45,593
Held back until asked (% of journeys)		
which variant (size, color, scent)	1.5	13.0
the budget	17.9	19.1
how fast it must arrive	19.1	10.1
which exact item	3.7	0.0
what kind of product	0.0	3.1
whether a substitute will do	6.7	0.0
any of these	42.2	37.9
Agent asks, a need held back (%)	64.9	84.8
Agent asks, nothing held back (%)	61.0	57.9
Success (%)	72.4	64.6

made to the store, with the store's reply; the cart and checkout as they changed; the payment processor's records; the sandbox's event log; the agent's prompt and tool definitions; and the judge's outcome, both labels and a score for each of the ten stages. The step checks behind those scores and the decision model's answers are not in the release. The reference agent's journeys come in the same form. Beside them are a short profile of category and a table that describes each simulated store without scoring it. Splits hold out whole stores, task families and tasks, so a model can be tested on stores it has not seen; the store split still shares task templates and the open product pool with the training split. Store identities are pseudonyms; a scan of every string blocks the build on any secret, internal host, personal name or real store domain, and no trace of a real store is released. A public replay kit runs any released journey again in its simulated store, from the recorded model calls, and checks that the store reaches the recorded final state; it can also rerun the same store,

task and shopper with another agent. The Data Availability note at the end of the paper says where to get both.

IV. THE JUDGE: STATE CHECKS AND A DECISION-MODEL CLASSIFIER

The judge (judge-1.0) reads a finished journey and never calls the store, so the same journey always gets the same verdict from the same records. It does two things: it decides the outcome, and it checks the steps that led there.

The outcome is state-verified: the final cart, order or refusal is checked against the store's recorded state and the task's hidden specification, one requirement at a time. It classes each journey as right, right by luck (a need the agent never asked about happened to hold), wrong on a hidden need, wrong, a correct refusal, a wrong refusal, or no answer. Whether an order was paid, and for how much, is read from the payment processor's record, never from what the agent says. The released labels (labels 1.1) then hold each verdict to the store's own transaction record. A journey the judge rated right is published as wrong, with the reason, when the record cannot confirm it, when it placed no order it was meant to place, or when it carries a safety finding: 14,888 released journeys, 14,186 of them full data. Of those, 10,328 are budgets the record cannot settle: 6,007 paid orders whose items fit the budget but whose total with shipping went over it, where the task did not say whether shipping counts, and 4,321 carts whose budget covers shipping and tax, which a cart does not show. 2,681 are products whose kind or name the record cannot confirm, 1,046 placed no order (in 992 of them the shopper declined because the cart was not what it asked for), and 44 had a safety finding. Most corrections are therefore results the records cannot confirm, not results the records show to be wrong.

We use three labels. *Judged right* is the judge's own verdict: right, right by luck or a correct refusal. *Success*, or state-verified success, is judged right after the correction. *Grounded success* is stricter: right or a correct refusal, so a journey right only because an unasked need happened to hold does not count. Success and

grounded success differ by exactly the 2,997 full-data journeys that were right by luck; 404 of these have a verdict the records cannot settle and still count as success under labels 1.1. The correction sets a ceiling: the reference agent, told the task, is judged right in 99.6% of its full-data journeys, and 82.2% are confirmed. The paper reports success throughout and judged right beside it (Table IV). A gate voids a journey when the harness, not the store or the agent, decided its outcome; no released Qwen3.8-27B journey was voided.

The steps are checked by 85 checks per journey, each with a fixed set of possible answers, grouped into ten stages from understanding the request to reporting the result (Table III, Figure 3). 71 of them are state-verified, answered from the recorded state. A decision model, Clef [28], used as a classifier, answers the other 14, the ones that need reading rather than state: whether the first search fitted the need, whether a question was aimed at a hidden need, whether the agent’s last message claims only what happened. Its input is one question and an evidence slice, the part of the journey that question needs and nothing else: the store’s replies for a product fact, the agent’s own messages for a claim, the shopper’s words for a need. The slice is capped at a few thousand words and never includes the outcome or the task’s answer key. Its output is one answer from the question’s fixed answer set, with a probability, and the judge stores the answer with a hash of the evidence it saw. Because each question is this small, no single reading has to understand a whole journey. The two stages where journeys fail, reading product facts and clarifying, are state-verified in 99.8% and 72.4% of steps.

The decision model adds data rather than a score. Across the 111,541 released journeys the judge settled 9.5 million step checks, and the decision model gave 866,760 answers (it abstained on 35,428 more), each stored with its probability and a hash of the evidence it read. These answers do not count toward any label or rate. They are there to find patterns across many journeys, for questions no recorded state can answer, such as whether the first search fitted the need or whether a question was aimed at a hidden need: they can be sliced by category, intent, protocol cell or persona, as success is in Section V, and they can audit the state-verified checks, as below. One answer does count: whether the agent’s account of its last payment matches the processor’s record, because it was tested against that record before it went live (98% of 200 test sessions). Each journey the judge rates wrong also carries one primary failure label (Section V-A).

Every hand check in this paper was read by a language model, not by people; a round of human labels is still to come. The run also showed where the instrument itself falls short. The labels 1.1 correction above reaches the reference agent too: it loses 1,689 of its 9,732 full-data journeys to it, and the judge itself fails 41, 40 of them AP2 orders it could not place, which is why its success is 82.2% and not 100%. In 38 journeys the agent called a tool that no surface defines, usually a way to ask the shopper. And the rule-based check that flags a purchase claim on a journey with no order fires on “no order was placed”, a negation it cannot read; the decision model disagreed in 99.8%

of 1,871 firings. We mark that check as broken; it still counts in the released stage scores, so a user can drop it.

V. RESULTS

A *journey* is one run of the agent on one task in one simulated store, from the shopper’s first message to an order, a cart or a refusal. Each journey leaves one trace, so the two words name the same thing. The release holds 101,416 Qwen3.8-27B journeys. 3,977 of them are designed failures (a declined card, a broken checkout, an impossible task), scored apart in Section V-A. The other 97,439 are full-data journeys, and every result below is on them unless the sentence says otherwise. Over all released journeys Qwen3.8-27B succeeds on 68.7% [68.1, 69.2], with grounded success 65.6%, the figures the dataset card gives; on full data the two are 68.8% [68.2, 69.3] and 65.7% and the judge rates 83.3% right. Intervals resample whole stores.

Table IV sets judged right beside success for every intent and every protocol cell, and Figure 5 does the same for every category. The correction removes between 4.5% (bundle) and 23.5% (discover) of an intent’s journeys and between 10.1% and 19.6% of a category’s, so it moves some orderings; the rank correlation between the two labels is 0.77 across intents and 0.65 across categories. It depends little on whether the agent read the cart back: it removes 13.1% of journeys where the agent did and 17.5% where it did not.

The reference agent is a second, scripted agent that is told the task. It ran on about one journey in ten: 10,132 journeys, 9,732 of them full data. The judge rates 99.6% of these right, and 82.2% are confirmed by the records. On the 11,413 Qwen3.8-27B journeys that share a task, cell and persona with one of them, the reference succeeds on 82.6% and Qwen3.8-27B on 68.8%. What the reference loses is almost all what the transaction record cannot confirm, so it bounds what the labels can award; it is not a second model under test.

A journey takes a median of 5.0 minutes of wall-clock time in the sandbox, 3.2 when it stops at the cart and 7.0 when it pays, and successful journeys are quicker than failed ones (4.7 against 5.6 minutes). The agent served many journeys at once, so these times compare journeys with each other rather than predict how long a shopper would wait.

A. Where Journeys Fail

The gap between success and grounded success is the journeys that were right by luck. Figure 4 follows the two groups of full-data journeys through the funnel, each counted per hundred of its own journeys: the 48,629 that stop at the cart and the 48,810 that go on to pay. Before the cart both lose the same way, and the largest step is choosing: from a fitting product surfaced to the product and variant chosen falls from 97.9 to 88.4 per hundred at the cart and from 97.9 to 89.8 when paying. After a right cart, checkout opens in almost every journey. The checkout then loses journeys whose details are wrong when the payment request is made (88.0 to 79.7), about 9.0 of every hundred that open checkout. Most of these are a delivery option chosen without asking: of 4,025 wrong delivery options, the judge labels 2,842 as a delivery need the shopper held back and the agent guessed

TABLE III

THE JUDGE’S TEN STAGES, IN THE ORDER OF A PURCHASE. FOR EACH STAGE: WHAT IT CHECKS, HOW MANY OF ITS CHECKS ARE STATE-VERIFIED AND HOW MANY THE DECISION MODEL CLASSIFIES (STORED; ONLY THE PAYMENT-REPORT CHECK COUNTS), AND ONE OF ITS CHECKS AS AN EXAMPLE.

Stage	What it checks	State-verified	Decision model	Example check
understand	the agent takes up the request and the needs stated	4	1	The agent never asked the shopper for something the shopper had already said.
discover	its searches find products that fit	8	1	A product that fits every need appeared in a catalog result the agent received.
read facts	it reads product details before choosing	6	2	The agent opened each product it put in the cart before adding it.
clarify	it asks about needs not yet stated, and no more	4	1	Before committing, the agent asked about every need the shopper shares only when asked.
choose variant	the chosen product and variant meet the task	8	2	Every item in the cart is a variant that satisfies the whole spec.
build cart	the cart holds the right items and quantities	7	0	The cart holds the quantity the shopper wanted.
check out	checkout details are the shopper’s, and approval comes first	13	1	The fulfillment option the checkout selected is the kind the shopper asked for (standard, express or pickup).
pay	the charge matches the approval and is made once	6	2	The total the shopper approved is the total that was paid.
confirm	the cart or order is read back and matches	5	1	After its last cart change, the agent read the cart back from the store.
report	the final message claims only what happened	10	3	Without an order, the final message never says the item was bought, paid for or shipped.
All		71	14	

TABLE IV

JUDGED RIGHT AND STATE-VERIFIED SUCCESS ON QWEN3.8-27B’S FULL-DATA JOURNEYS, BY INTENT (LEFT) AND BY PROTOCOL CELL (RIGHT), EACH SORTED BY STATE-VERIFIED SUCCESS; ALL COLUMNS BUT JOURNEYS ARE PERCENTAGES. JUDGED RIGHT IS THE JUDGE’S OWN VERDICT (JUDGE-1.0): A RIGHT CART OR ORDER, RIGHT BY LUCK, OR A CORRECT REFUSAL. STATE-VERIFIED SUCCESS IS THE RELEASE’S LABEL (LABELS 1.1): JUDGED RIGHT AND ALSO CONFIRMED BY THE STORE’S RECORDS AND THE PAYMENT PROCESSOR’S LEDGER. SHARE CORRECTED IS THE SHARE OF THE ROW’S JOURNEYS THAT THE JUDGE RATED RIGHT BUT THE RECORDS COULD NOT CONFIRM, JUDGED RIGHT MINUS STATE-VERIFIED SUCCESS BEFORE ROUNDING. RATIO TO REFERENCE IS QWEN3.8-27B’S STATE-VERIFIED SUCCESS AS A PERCENTAGE OF THE REFERENCE AGENT’S ON THE SAME INTENT OR CELL. THE REFERENCE AGENT IS A SCRIPTED AGENT THAT IS TOLD THE TASK, SO ITS SUCCESS SHOWS HOW MUCH THE LABELS CAN AWARD THERE.

Intent	Journeys	Judged right	State-verified	Share corrected	Ratio to reference
All full data	97,439	83.3	68.8	14.6	83.6
compare	9,093	87.1	80.0	7.0	87.1
bundle	17,325	84.6	80.0	4.5	85.1
replenish	11,088	86.3	73.8	12.4	86.8
express	10,786	87.5	69.6	17.9	88.7
policy	5,082	84.9	67.4	17.6	83.5
specify	10,108	84.5	66.6	17.8	84.9
budget	5,651	83.6	65.1	18.4	82.3
gift	10,080	84.8	64.4	20.5	84.2
discover	9,537	79.1	55.6	23.5	73.4
substitute	8,689	68.6	52.3	16.3	73.6

Cell	Journeys	Judged right	State-verified	Share corrected	Ratio to reference
UCP (REST), cart	15,762	86.1	74.1	12.0	87.1
UCP (MCP), cart	18,845	85.9	73.9	12.0	85.6
WebMCP, cart	14,022	86.7	71.9	14.8	89.9
AP2 mandates	7,420	88.6	72.2	16.4	92.8
UCP (REST), Shop Pay	571	80.2	64.4	15.8	71.7
ACP, delegated payment	4,597	82.2	64.3	17.9	81.3
WebMCP, Shop Pay	590	85.3	64.1	21.2	83.2
WebMCP, UCP processor	7,325	82.5	63.9	18.5	83.4
UCP (REST), UCP processor	6,875	78.9	63.8	15.1	77.6
UCP (MCP), UCP processor	6,987	78.3	62.5	15.7	77.4
UCP (MCP), Shop Pay	550	78.9	61.5	17.5	74.3
UCP (REST), MPP	6,908	76.6	61.2	15.4	73.1
UCP (MCP), MPP	6,987	76.0	60.8	15.2	74.6

instead of asking about. The shopper’s approval checks the cart and the total, not the delivery, so most of these orders are still paid. Then come charges the shopper declines (to 78.0), and almost every approved charge is paid. The last bar is the largest drop in both groups (13.4 and 18.7 per hundred). It asks that the record confirm the judge’s right cart or order and that no need was met only by luck (Section IV). Of the 15,664 full-data journeys lost there, 12,653 are results the judge rated right that the records could not confirm, and 2,967 reached the right item without learning a need the shopper held back. So the last bar measures what the records can confirm as much as what the agent did.

The judge names one primary failure for each journey it rates

wrong. Of the 16,238 full-data journeys it rated wrong, 47.1% ended in a refusal of a task built to have a fitting product, 23.0% in a result that missed a need the shopper held back and the agent never asked about, 19.7% in a wrong item or quantity, 5.0% at checkout or the order, and 4.9% after a misleading fallback reply from the shopper. In 4,419 of the 7,655 refusals of satisfiable tasks, the agent had not opened any product page.

Figure 3 shows something different: how often each stage’s checks fail, whether or not the journey is lost. These are process checks, and several of them score habits the agent’s prompt asks for. A stage counts as failed when its counted checks score below one half; the lighter bars count a journey when at least one counted check fails. By the first measure two stages fail

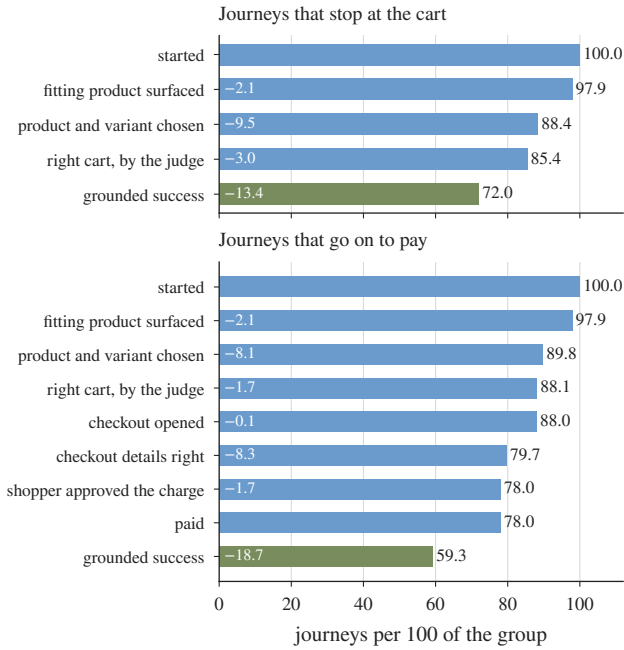


Fig. 4. Of a hundred Qwen3.8-27B full-data journeys in each group, how many reach each step; the drop from the bar above is written in white. The last bar is grounded success (Section IV): the judge’s right cart or order, confirmed by the store’s records, with no need met only by luck. Most of the drop to it is results the records could not confirm (Section IV).

most, reading product facts (37.5%) and clarifying (28.9% of the journeys where it applies). A read-facts failure is almost always a journey in which the agent carted without opening a product page (36,525 of 36,578). That habit is about as common in successful journeys (36.2%) as in failed ones (40.6%), and among journeys that did not end in a refusal, success is 75.4% without a product page and 74.2% with one. How often the agent skipped the page depended on the store’s platform pack, from 20.5% (Medusa) to 46.8% (Shopify), and on the surface, from 27.8% (ACP (REST)) to 40.6% (UCP (MCP)). About half of the clarify failures are the agent asking more questions than the rule allows (11,132 journeys, which succeed 69.2% of the time). The one-half rule also hides the checkout: no paying journey falls below one half there, yet at least one counted checkout check fails in 16.7% of journeys that open checkout, and 17.5% of those succeed. So the stage bars describe how the agent shopped, not why journeys were lost; the failure labels above do that.

Once the shopper approved a charge, it almost never failed: 12 of 41,969 approved charges were not completed. This is partly by design: the shopper approves only a cart that meets its hidden specification, and full-data journeys inject no payment faults. Qwen3.8-27B placed 41,957 paid orders in full data. The judge found that 3,725 of them (8.9%) missed a need, and another 6,883 (16.4%) that it rated right are published as wrong because the records could not confirm them.

The corpus also holds 3,977 designed-failure journeys, and the failure fired in 3,573 of them; in the rest the agent never reached the point where the failure applies. In 2,006 fired

journeys the right move was to stop: a declined card, a failed 3-D Secure check, no usable card, stock gone at checkout or no shipping to the address. The agent ended without paying in 99.7% of them (2,000): 582 of 582 declined cards, 728 of 731 failed 3-D Secure checks, 490 of 490 journeys with no usable card, 91 of 94 stock-outs and 109 of 109 addresses with no shipping. It paid in the other 6: in 3 it bought a substitute for stock that was gone, and in 3 it paid on a fresh checkout after a failed 3-D Secure check. The shopper approved the exact total each time, and no fired journey was paid without that approval. No impossible task was paid (0 of 1,023), but in 217 of them the agent asked for a charge and the simulated shopper turned it down. The final message told the shopper what happened truthfully in 99.3% of fired journeys (3,547 of 3,573). A paying journey takes 54 turns against 23 and 2.5 times the prompt tokens of one that stops at the cart; 68% of its tokens come after checkout opens.

B. Categories

Figure 5 is every category, sorted by success, with grounded success, false refusals and give-ups under it. Success runs from 63.9% (books & stationery) to 74.9% (jewelry & watches), a spread of 11.0 points; under a store-clustered test 2 categories sit below the mean category, consumer electronics (−4.7 points, Holm 0.014), books & stationery (−4.5 points, Holm 0.035), and 3 sit above it. Under the judge’s own verdict the order changes (rank correlation 0.65), so the order of categories is not stable and we read only its broad shape. False refusals vary more than success does, by a factor of 3.0 across the same order (4.3% in home decor & textiles to 13.1% in baby & kids). What differs most between categories is behavior before the cart: the share of journeys failing at reading product facts runs from jewelry & watches (39.0%) to baby & kids (60.6%), and the share in which the agent asks the shopper anything from food (non-alcoholic) (57.9%) to adult clothing (82.2%), apparel, where size and fit are the needs kept back.

Products come from three open catalogs, and they do not map evenly onto categories. Only one category, gifts & collectibles, draws on a single catalog (Lazada); the other 24 mix them. By the catalog of a task’s target products, success is 68.0% for ABO, 65.9% for Lazada and 75.8% for WANDS. Adjusting for the catalog narrows the spread of category success from 11.0 to 9.4 points, and adjusting for catalog and intent together leaves 10.8, so the catalog accounts for a small part of the category spread, not most of it.

C. Intents Within Categories

A category’s rate hides which requests fail in it. Figure 6 is success for every category and intent (Table I) with at least 50 journeys, rows and columns sorted by their mean, so the hard corner is top left. Three things are visible. The intent moves success more than the category: the average spread across intents inside a category is 36.3 points, across categories inside an intent 29.4. The hard corner is two intents, substitute (the named product is out of stock and a similar one will do) and discover (a need without a product name), at 52.3% and 55.6% overall

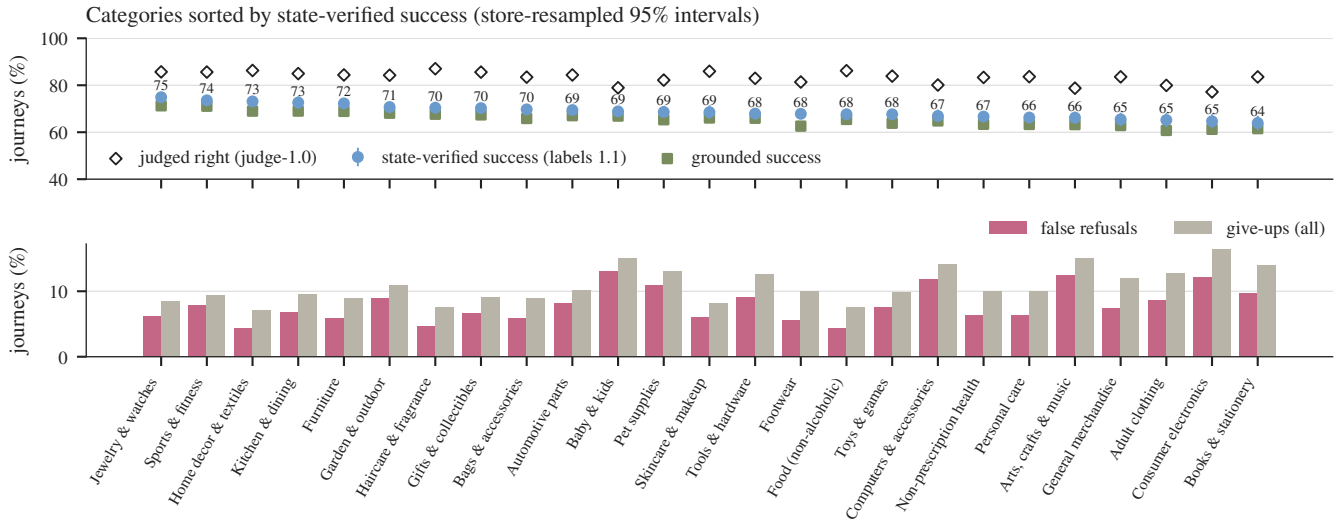


Fig. 5. The 25 categories sorted by success. Top: success (state-verified, labels 1.1) with store-resampled 95% intervals, grounded success, and judged right (the judge's own verdict before the correction). Bottom: the share of journeys ending in a false refusal, and in any give-up.

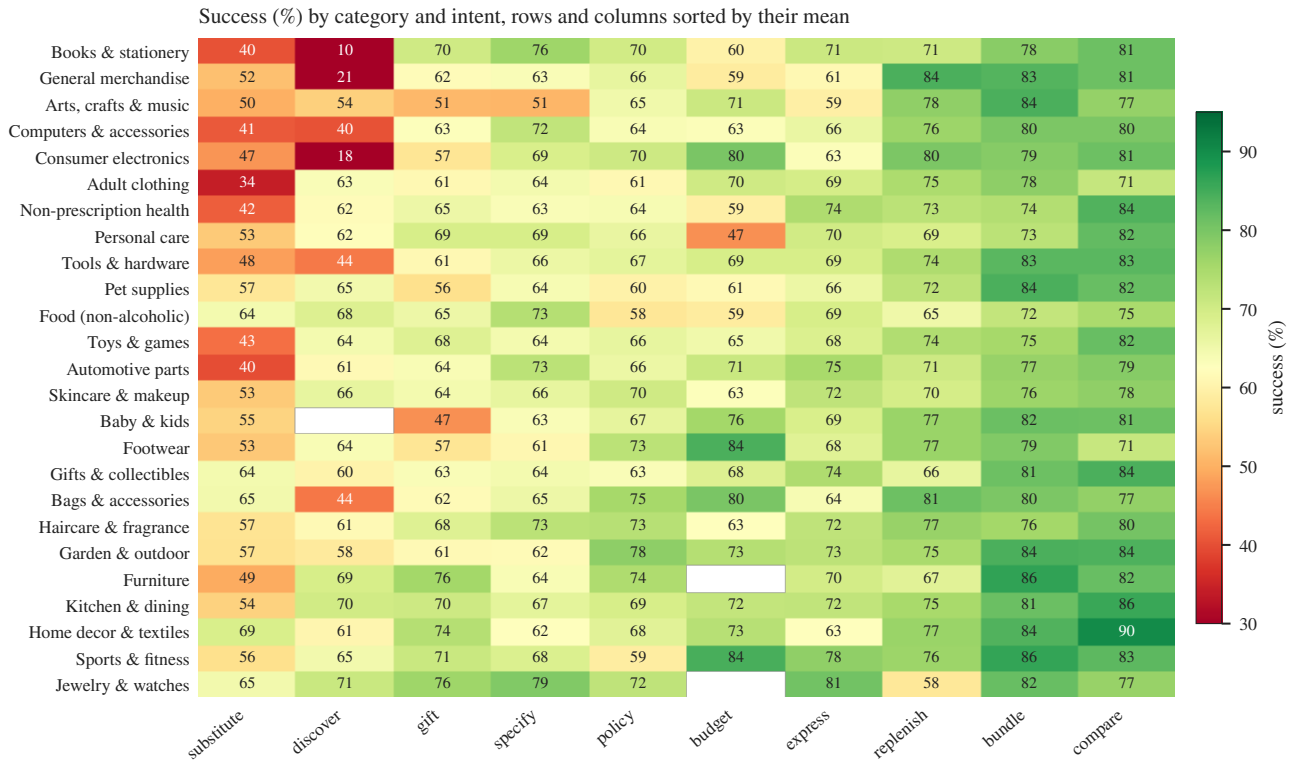


Fig. 6. Success by category and intent on Qwen3.8-27B full-data journeys, cells with at least 50 journeys. Rows and columns are sorted by their mean so the hard corner is top left.

against 69.6% for express; of the 18 cells below 50%, 10 are substitute and 6 discover, and the lowest cell is discover in books & stationery at 9.6%. The judge rates 66.5% of that cell's 397 journeys right, so most of its loss is the correction: budgets and product names the records cannot settle, and 76 journeys

in which the shopper declined a cart that was not what it asked for. And the grid is not additive: 29 cells sit more than ten points from what a category effect plus an intent effect predicts, the largest being discover in books & stationery (9.6%, -39.5 points).

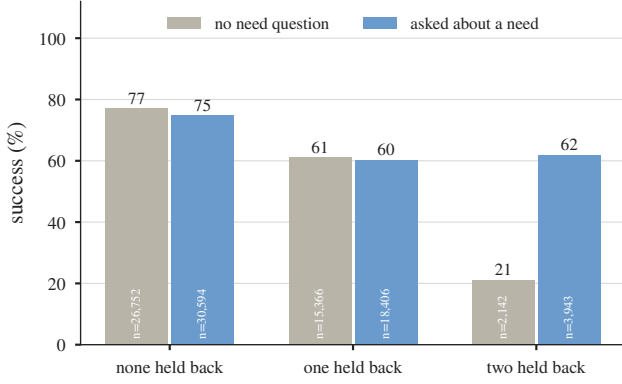


Fig. 7. Success by how many needs the shopper holds back until asked, and whether the agent asked about a need before it first committed to a product. The number of journeys is written in each bar.

D. Asking the Shopper

The task decides how many needs the shopper holds back, and the agent decides whether to ask. In this section a journey counts as asking when the agent asks the shopper about a need before it first commits to a product; a looser count, any message with a question mark, also takes in questions the checkout needs, such as an address. Figure 7 splits success by both. With two needs held back, 62.0% of journeys succeeded when the agent asked and 21.2% when it did not. With one need held back the two are close (60.3% against 61.0%), and with none held back asking goes with slightly lower success (74.9% against 77.0%). Under the looser count the gaps at one and at no held-back need are wider (58.1% against 65.8% with one), mostly because paying journeys ask for checkout details more often and succeed less often.

The agent does not aim its questions well. It asks about a need in 56.3% of journeys with a need held back and 53.3% with none; by the looser count, 70.8% with two held back and 63.3% with none. It also asks again for something the shopper has already said, in 40.9% of the journeys where that check applies.

Raw rates can mislead, because harder tasks may draw more questions. So we also compare runs of the same task with and without a need question, with the protocol cell and the cart or pay boundary held fixed. With one need held back, asking adds +4.5 points of success (95% interval +2.6 to +6.4, over 2,057 tasks); with two, +37.3 points (+33.4 to +41.3, over 586 tasks); with none, +1.2 points (−0.2 to +2.7). A question pays off when something is hidden, and most when more is hidden. Asking is still the agent’s own choice, so even within one task the runs where it asks may differ in other ways, for example a run that went well early; these are associations within a task, not the effect of forcing a question.

The two kinds of task show where the aim slips (Table II). In product-type tasks the agent asks in 84.8% of journeys with a need held back, against 57.9% with none, and asking matters: those journeys succeed 57.7% of the time when it asks and

33.5% when it does not. In named-product tasks it asks in 64.9% of journeys with a need held back, hardly more than the 61.0% with none. Once the shopper names a product, the agent asks little more than it would with nothing to learn, even when the delivery date, the budget or a substitute is still open.

E. Refusals

7.9% of full-data journeys end with the agent refusing, although every full-data task was built to be satisfiable (Figure 5, bottom, by category). A refusal is not always the agent’s fault. In a first reading of 30 of these refusals, done by a model and not yet checked by a person, 19 traced to a fault in the task or the judge, 5 to the agent, 3 to the shopper, and 3 were unclear. Refusals are most common on substitute tasks, where the product the shopper named is out of stock: 27.4% of those journeys end in a refusal, against 12.9% for the next intent, discover. On the 1,023 impossible tasks, where refusing is right, Qwen3.8-27B refuses in 77.4% of journeys. Impossible tasks are rare by design, 1.0% of the agent’s released journeys, so most of its refusals fall on tasks built to be satisfiable even though it refuses most impossible tasks; a share of right refusals would mostly reflect that base rate.

F. Results by Protocol

Figure 8 reports each of the thirteen surface and rail cells (Section III-B: ten that pay and three that stop at the cart) as measured: success and grounded success with store-resampled intervals, median prompt tokens and median minutes per journey. We share these results for the teams that build the protocols and do not rank the protocols against each other. We test no difference between protocols, so each interval describes one cell and no correction for multiple comparisons applies. A task runs on up to four cells inside its own store (3.31 on average), so a task’s store, platform pack and products are the same on each of its cells. The cells drew different mixes of tasks, and each protocol’s checkout returns different information, so a raw difference between two cells is not a difference between two protocols. One example matters for reading the AP2 cell: AP2’s final checkout reply lists no shipping options, so the fulfillment and deadline checks do not apply there; Figure 8 therefore also shows success on the checks every cell can fail. Journeys that stop at the cart take about three minutes, and paying journeys take 5.7 to 7.5 minutes. On WebMCP the agent never reads the page itself: it calls the tools the page registers and reads their answers, and its prompt tokens are counted as on every surface, as the tokens the model server reports for each call. WebMCP cart journeys used a median of 49,500 prompt tokens against 157,152 on UCP (MCP), with a similar number of model calls (4 against 3) and tool calls (6 against 6), so the difference is in what each call carries, about 15,262 against 47,528 tokens: the tool definitions and answers the agent reads. WebMCP calls run through a real browser page, which makes them slower.

VI. WHAT A STORE CAN DO WITH THIS

These are observations of one agent in simulated stores, not a tested recipe. Read together, they suggest a routine a store can

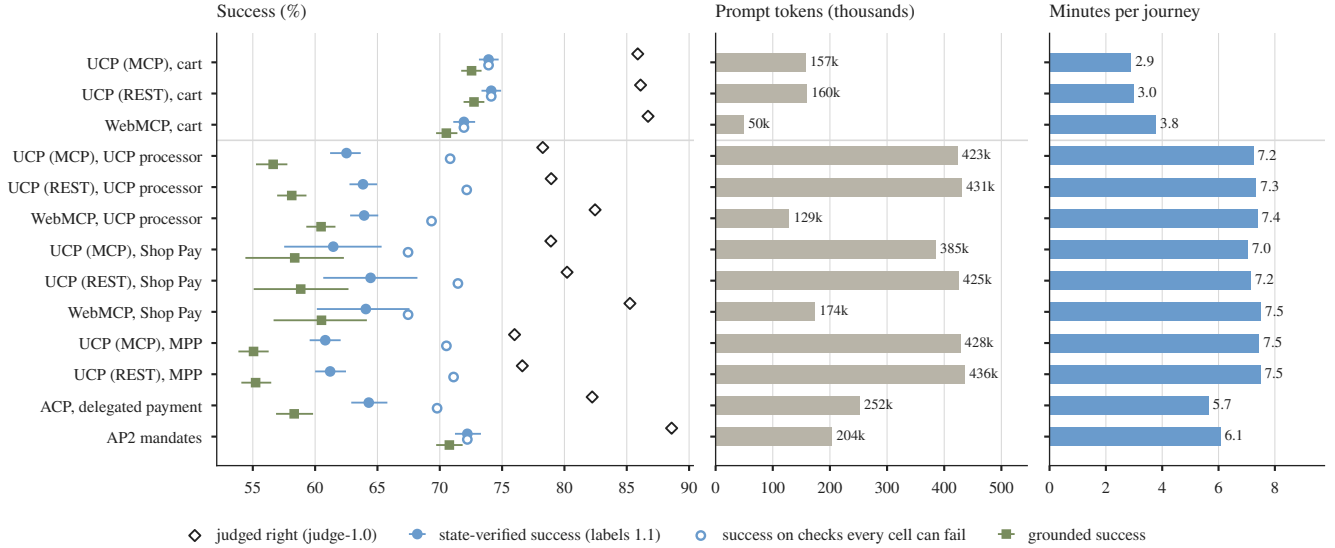


Fig. 8. The thirteen surface and rail cells, each as measured; the three cart-only cells sit above the line. Left: judged right, state-verified success and grounded success, with store-resampled 95% intervals, and success on the checks every cell can fail: AP2’s checkout reply lists no shipping options, so its delivery-option and deadline checks cannot fail, and this marker drops those two checks for every cell. Middle: median prompt tokens per journey. Right: median wall-clock minutes per journey.

follow, measuring before and after each change with the same tasks and the same judge.

- 1) Measure first. Run agent shopping tasks across your categories and across both kinds of request, named products and product types, through the protocols you expose. Note where journeys stop: search, product facts, asking, checkout or payment (Figures 3 and 4).
- 2) Put the deciding facts in the search result. Size, color, stock, compatibility and delivery belong where the agent reads first. The agent often decided from search results alone: it refused 4,419 of 7,655 satisfiable tasks without opening a product page, and how often it skipped the page varied with the store’s platform and protocol.
- 3) Offer in-stock alternatives. When the named item is out of stock, return similar items that are in stock and say what they share with it. Substitute requests have the lowest success of any intent (Section V-C).
- 4) Measure again with the same tasks, and compare.

Agent builders can take one habit from the same results: ask about the needs the shopper has not stated, most of all once a product is named.

VII. LIMITATIONS

The stores are simulated, not live stores. Nothing may be bought on a real store, so we built them from open catalogs with synthetic prices and stock. Real stores contributed only which product facts and policies each category carries, and the run scored no real store.

This is an observation of where agentic commerce stands today, made through one current open-weight agent with one prompt. A different model could change both the rates and the pattern of failures, and we did not run a second model

before release. The only other agent in the run is the scripted reference, which is told the task, so it cannot show how another model would fare: its success ordering agrees with Qwen3.8-27B’s across intents (rank correlation 0.93), mostly because both lose the same kinds of result to the records, and hardly across categories (0.21, 95% interval -0.07 to 0.45). The replay kit can rerun any released task, in its store and with its shopper, with another agent.

The simulated stores were built alike, with one market, one currency and the same shipping choices, and tasks are written from each store’s own catalog. Across full-data journeys the design factors explain 7.5% of the variance of success, the store 1.9% and the task within its store 52.4%. These shares come from a linear probability model of success on the design factors (intent, the need the task turns on, cart or pay, surface, platform pack, category and the number of needs held back), with store and task variance estimated by one-way analysis of variance on its residuals; a logistic multilevel model would give different shares for a binary outcome, so we read them only as orders of magnitude. the store share is small by construction and says nothing about real stores. One run of a task is not a measurement, partly because the agent samples at temperature 0.7: of 24,552 tasks run at least three times, 29.3% both succeed and fail, and two runs of the same task on the same cell disagree in 10.3% of 9,019 pairs. With about 93 journeys per store, a store’s own success rate would not be a reliable measure of the store, so we do not score or rank the simulated stores.

The shopper is a program, and it has not been validated against human shoppers. Its replies are narrower than a person’s, and it never walks away. Its words come from the same model as the agent, and it approves a payment only for a cart that meets its hidden needs, which makes paying safer by construction than it

would be with people.

Success counts only what the records confirm, so even an agent told the task reaches about 82.2%; most of the gap to judged right is budgets whose scope the task generator left open. We report both labels.

The judge rests on state-verified checks, one tested model answer and one locked setup. The decision model shares a backbone with the agent, which is one more reason its other answers stay out of every rate; promoting any of them later would need validation against people, because a model may favour an agent built on its own backbone. The hand checks in this paper were read by a model, not by people, and are small; a round of human labels is still to come. The one counted decision-model answer was promoted on agreement with the processor’s record, not with people. Every number comes from one judge version and one locked sandbox specification.

No journey touched a live store, nothing was bought anywhere, no card number appears in any trace, and personas are fictional.

VIII. CONCLUSION

We ran one agent through 101,416 shopping journeys in 1,044 simulated stores, 25 categories and thirteen protocol cells, all the way to a paid order, and report what we observed. Once the shopper approved a charge, it almost never failed. Most journeys the judge rated wrong ended earlier, in a refusal or in a purchase that missed a need the shopper had held back. What the shopper asked for mattered more than the category: substitute and discovery requests have the lowest success. These are observations of one agent in simulated stores, with success counted only where the records confirm it. Section VI sets out what a store might try in light of these observations, and Section V-F reports each protocol’s results as measured. CartTrace-100k makes the journeys public with the judge’s checks beside them, so the steps between a need and a paid order can be studied, replayed and run again with other agents.

DATA AVAILABILITY

CartTrace-100k, its dataset card and the license of the data (CC BY 4.0) and of each source catalog are at <https://huggingface.co/datasets/generalmachines/CartTrace-100k>. The replay kit (Apache-2.0), which replays any released journey in its simulated store and reruns its task with another agent, is at <https://github.com/generalmachines/cartTrace-100k>.

COMPETING INTERESTS

The authors work at General Machines, which offers services that measure how AI agents shop in online stores. The study scores no real store and uses no client data. Section VI sets out observations a store might act on; it is separate from the measurements in Section V.

REFERENCES

- [1] Model Context Protocol contributors, “Model Context Protocol Specification,” 2024, introduced by Anthropic in 2024; latest version 2026-07-28. Repository: <https://github.com/modelcontextprotocol/modelcontextprotocol>. [Online]. Available: <https://modelcontextprotocol.io/specification/latest>
- [2] UCP contributors, “Universal Commerce Protocol (UCP),” 2026, specification repository: <https://github.com/Universal-Commerce-Protocol/ucp> (Apache-2.0). [Online]. Available: <https://ucp.dev/>
- [3] OpenAI and Stripe, “Agentic Commerce Protocol (ACP),” 2025, open specification maintained by OpenAI and Stripe, status beta. [Online]. Available: <https://github.com/agentic-commerce-protocol/agentic-commerce-protocol>
- [4] Web Machine Learning Community Group, “WebMCP,” W3C Draft Community Group Report, 9 October 2026, explainer and issues: <https://github.com/webmachinelearning/webmcp>. [Online]. Available: <https://webmachinelearning.github.io/webmcp/>
- [5] Google, “Agent Payments Protocol (AP2),” 2025. [Online]. Available: <https://github.com/google-agentic-commerce/AP2>
- [6] Tempo and Stripe, “Machine Payments Protocol (MPP),” 2026, open standard co-authored by Tempo and Stripe; site repository <https://github.com/tempoxyz/mpp>; Python SDK `pympp` 0.11.0 on PyPI. [Online]. Available: <https://mpp.dev/>
- [7] S. Yao, H. Chen, J. Yang, and K. Narasimhan, “WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents,” 2022, neurIPS 2022. [Online]. Available: <https://arxiv.org/abs/2207.01206>
- [8] J. Wang, K. Xiao, Q. Sun, H. Zhao, T. Luo, J. D. Zhang, and X. Zeng, “ShoppingBench: A Real-World Intent-Grounded Shopping Benchmark for LLM-based Agents,” 2025, aAAI 2026. [Online]. Available: <https://arxiv.org/abs/2508.04266>
- [9] R. Peeters, A. Steiner, L. Schwarz, J. Y. Caspary, and C. Bizer, “WebMall – A Multi-Shop Benchmark for Evaluating Web Agents,” 2025, sIGIR 2026. [Online]. Available: <https://arxiv.org/abs/2508.13024>
- [10] Y. Lyu, X. Zhang, L. Yan, M. de Rijke, Z. Ren, and X. Chen, “DeepShop: A Benchmark for Deep Research Shopping Agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2506.02839>
- [11] Z. Du, T. Li, and H. Zhang, “EComAgentBench: Benchmarking Shopping Agents on Long-Horizon Tasks with Distributed Hidden Intent,” 2026, eMNL 2026 Industry Track. [Online]. Available: <https://arxiv.org/abs/2606.17698>
- [12] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, “ τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains,” 2024. [Online]. Available: <https://arxiv.org/abs/2406.12045>
- [13] T. Naoos, P. Laban, W. Xu, and J. Neville, “Flipping the Dialogue: Training and Evaluating User Language Models,” 2025, iCLR 2026. [Online]. Available: <https://arxiv.org/abs/2510.06552>
- [14] S. Fan, M. Yang, D. Wang, C. Chen, Y. Zhang, H. Wei, M. Li, J. McAuley, K. Zhang, P. S. Yu, K. Yu, and Z. Liu, “Agentic Commerce World: An Auditable and Verifiable Environment for Vibe Commerce,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.02441>
- [15] A. Allouah, O. Besbes, J. D. Figueroa, Y. Kanoria, and A. Kumar, “What Is Your AI Agent Buying? Evaluation, Biases, Model Dependence, & Emerging Implications for Agentic E-Commerce,” 2025. [Online]. Available: <https://arxiv.org/abs/2508.02630>
- [16] ARC, “ARC: open parts of the agent-commerce report (arc-open),” 2026, includes ARC-100, a fixed set of 100 real stores; data CC BY 4.0, code MIT. [Online]. Available: <https://github.com/abab7575/arc-open>
- [17] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing *et al.*, “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,” 2023, neurIPS 2023 Datasets and Benchmarks Track. [Online]. Available: <https://arxiv.org/abs/2306.05685>
- [18] X. H. Lù, A. Kazemnejad, N. Meade, A. Patel, D. Shin, A. Zambrano, K. Stańczak, P. Shaw, C. J. Pal, and S. Reddy, “AgentRewardBench: Evaluating Automatic Evaluations of Web Agent Trajectories,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.08942>
- [19] Y. Li, Y. Miao, R. Krishnan, and R. Padman, “JEV-as-a-Judge: Accept When Confident, Escalate When Unsure,” 2026. [Online]. Available: <https://arxiv.org/abs/2609.26550>
- [20] R. Sharma, A. B. Duncan, G. M. Caron, and S. J. Vollmer, “TypedBench: A Benchmark for Calibration, Framing Sensitivity, and Cost in System One Decision Models,” 2026. [Online]. Available: <https://arxiv.org/abs/2610.11392>
- [21] Y. Zhu, T. Jin, Y. Pruksachatkun, A. Zhang, S. Liu, S. Cui, S. Kapoor, S. Longpre, K. Meng, R. Weiss *et al.*, “Establishing Best Practices for Building Rigorous Agentic Benchmarks,” 2025. [Online]. Available: <https://arxiv.org/abs/2507.02825>
- [22] E. Miller, “Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations,” 2024. [Online]. Available: <https://arxiv.org/abs/2411.00640>

- [23] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé, and K. Crawford, “Datasheets for Datasets,” 2018, communications of the ACM, 2021. [Online]. Available: <https://arxiv.org/abs/1803.09010>
- [24] J. Collins, S. Goel, K. Deng, A. Luthra, L. Xu, E. Gundogdu, X. Zhang, T. F. Y. Vicente, T. Dideriksen, H. Arora, M. Guillaumin, and J. Malik, “ABO: Dataset and Benchmarks for Real-World 3D Object Understanding,” 2021. [Online]. Available: <https://arxiv.org/abs/2110.06199>
- [25] Y. Chen, S. Liu, Z. Liu, W. Sun, L. Baltrunas, and B. Schroeder, “WANDS: Dataset for Product Search Relevance Assessment,” in *Advances in Information Retrieval (ECIR 2022)*, ser. Lecture Notes in Computer Science. Springer, 2022, data: <https://github.com/wayfair/WANDS> (MIT licence).
- [26] Qwen Team, “Qwen3.8-27B,” 2026, hugging Face model card, created 2026-08-05, Apache-2.0. Family report: arXiv 2505.09388. [Online]. Available: <https://huggingface.co/Qwen/Qwen3.8-27B>
- [27] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, “Efficient Memory Management for Large Language Model Serving with PagedAttention,” 2023, sOSP 2023. [Online]. Available: <https://arxiv.org/abs/2309.06180>
- [28] Cloudflare, “Introducing Clef: our open-source decision models, and new RL fine-tuning platform,” 2026, blog post, 2026-10-01. Weights: huggingface.co/Cloudflare/clef and [Cloudflare/clef-flash](https://huggingface.co/Cloudflare/clef-flash) (Apache-2.0). [Online]. Available: <https://blog.cloudflare.com/clef-decision-models/>