ShopGym: An Integrated Framework for Realistic Simulation and Scalable Benchmarking of E-Commerce Web Agents

Chinmay Savadikar[1,2,†,*], **Mingyu Zhao**[2,†], **Yuanzheng Zhu**[2,†], **Han Li**[2], **Shuang Xie**[2], **Alberto Castelo**[2], **Tianfu Wu**[1], **Lingyun Wang**[2,‡]

[1]North Carolina State University [2]Shopify

Website: https://shopgym-research.github.io/

Abstract

Developing and evaluating e-commerce web agents requires environments that preserve meaningful task structure while enabling controllable, reproducible, and scalable scientific comparison. Existing methodologies force a tradeoff: live storefronts provide realism but are non-stationary, difficult to inspect, and irreproducible, while hand-built sandbox benchmarks provide control but cover only a narrow range of layouts, catalogs, policies, and interaction patterns. We argue that the core bottleneck is methodological: the field lacks a scalable way to construct evaluation settings that are simultaneously realistic, diverse, controllable, inspectable, and reproducible. We introduce ShopGym, an integrated framework for realistic simulation and scalable benchmarking of e-commerce web agents. ShopGym is a framework for constructing e-commerce simulation environments and grounded benchmark tasks. Its simulation layer, ShopArena, converts live seed storefronts into self-contained sandbox shops through anonymized shop specifications and a staged, validated generation process. On top of these simulated storefronts, ShopGuru synthesizes benchmark tasks across seven skill categories, grounding each task in the shop's catalog, navigation structure, policies, and interaction affordances. Together, ShopArena and ShopGuru produce self-contained, resettable, inspectable, and stable evaluation artifacts that preserve structural properties and agent-evaluation signals relevant to shopping tasks. We validate the framework through graph-based structural analysis and agent-based behavioral evaluation with 224 generated tasks across six sandbox shops: three constructed with synthetic data and three with real data. Our results show that the synthetic shops preserve key structural properties of live storefronts, with agent performance on synthetic shops positively correlated with performance on live storefronts.

1 Introduction

As web agents become more capable, the field increasingly needs stable environments that support long-horizon interaction in realistic, user-facing settings. E-commerce is a particularly demanding domain for this purpose: shopping tasks require agents to navigate multi-page interfaces, interpret product information, satisfy user constraints, compare alternatives, and execute actions such as selecting variants or adding items to cart. These tasks combine perception, reasoning, and sequential decision-making in a grounded setting, making storefronts a compelling testbed for web-agent

*This work was done as part of internship at Shopify

†Equal contribution, order decided randomly

‡Corresponding author: lingyun.wang@shopify.com

RLxF: Reinforcement Learning from World Feedback at ICML 2026

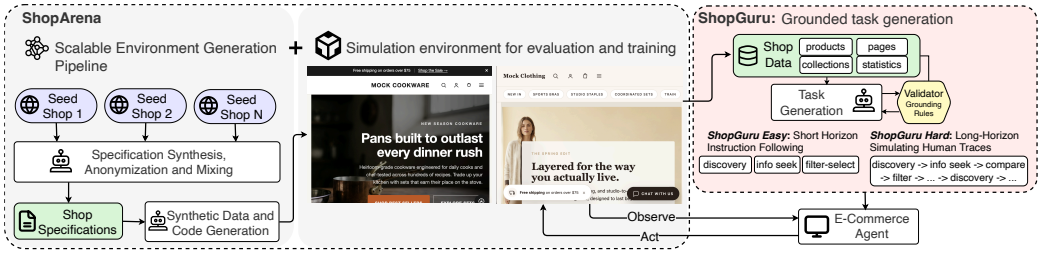


Figure 1: **ShopGym** comprises two components. **ShopArena** provides a simulation environment populated with synthetic sandbox shops, along with a scalable pipeline that generates new sandbox shops from one or more live seed storefronts through specification synthesis followed by data and code generation. **ShopGuru** then consumes the resulting catalog, collections, pages, and shop statistics to generate both short-horizon tasks covering primitive skills and long-horizon shopping journeys that combine these skills.

evaluation. The importance of this domain is reflected in growing interest in shopping assistants [23, 25, 17], offline A/B testing [2, 15], and user behavior simulation for e-commerce applications [10, 23, 26, 20].

Existing environments for e-commerce agents face a fundamental tradeoff between realism and experimental control. Live-websites [21, 7, 11] offer strong realism: agents are run and evaluated against realistic layouts, and naturally occurring variation and interaction patterns. But live storefronts are inherently non-stationary. Catalogs, layouts, and webpage design on live storefronts can change, collections can be reorganized, and storefronts participating in live A/B testing can serve separate variants for different requests. This non-stationary behavior makes reproducible training and evaluation of web agents challenging. The evaluation performance can reflect not only agent capability but also incidental variation from website drift, location-specific behavior, and operational noise, making results difficult to reproduce and agents difficult to compare fairly [21], and training runs difficult to reproduce.

Sandbox environments [28, 8, 22, 16, 18, 12], in contrast, are controllable and repeatable by design, but at the cost of realism and diversity of live websites. Manually constructing realistic storefront environments is expensive and covers only a narrow range of layouts, catalog structures, and interaction patterns.

Existing work proposing environments for e-commerce agents has largely explored the two methodologies, with works like DeepShop [11] using live websites, and WebShop [22], ShoppingBench [16], ShopSimulator [18], and WebMall [12] manually building sandbox environments. *We argue that the deeper bottleneck is methodological*: the field lacks a scalable way to ***construct*** *reliable simulated environments that are realistic, controllable, inspectable, and reproducible*.

To address this gap, we propose **ShopGym**, a framework for constructing realistic e-commerce simulation environments and grounded web-agent benchmarks. ShopGym comprises two complementary components in a synergistic workflow: (a) the simulation environment layer, **ShopArena**, transforms one or more live seed storefronts into a self-contained sandbox shop; and (b) **ShopGuru** synthesizes benchmark tasks grounded in the sandbox shops.

ShopArena: validated simulation environment. To generate the simulation environment grounded in real storefronts, ShopArena separates storefront understanding (exploration) from sandbox synthesis (generation) through an intermediate specification document that serves as the sole interface between exploration and generation. Given one or more seed storefronts, ShopArena autonomously explores the sites and writes an anonymized specification capturing their navigation structure, user-facing affordances, high-level page organization, and catalog statistics. When multiple seeds are provided, ShopArena composes their structural and behavioral signals into a single specification, so one sandbox shop can span diversity that no individual source storefront would cover. A staged generation pipeline then synthesizes a runnable sandbox shop from the specification, iteratively refining the implementation through build checks, type checking, and multimodal visual validation. ShopArena does not aim to clone a particular storefront; it produces *behaviorally aligned approximations* – synthetic shops grounded in real storefront structure and interaction patterns, but with synthetic brand identity, products, descriptions, and imagery. The goal is not visual or brand-level fidelity but structural, behavioral, and statistical alignment sufficient for controlled evaluation. *Because*

the specification is human-readable, benchmark creators can also edit generated environments in controlled ways without re-exploring the source storefronts.

ShopGuru: grounded task generation. ShopGuru consumes the sandbox shop's catalog, navigation graph, and store policies and synthesizes tasks that are valid against the environment they will be executed in. This design separates live-web exploration from benchmark execution. The generated shops are stable and resettable, while the intermediate specification provides an inspectable control surface that benchmark creators can edit without re-exploring the source storefronts.

We evaluate ShopGym by generating synthetic sandbox shops and anonymized twin shops, then validating them using structural metrics and agent-based behavioral comparisons. Across paired tasks defined over source and generated shops, agent success rates on sandbox environments positively correlate with success rates on live storefronts, indicating that generated storefronts retain meaningful evaluative signal from live-web settings while substantially improving control and reproducibility. While it does not completely replace live-web evaluation, ShopGym provides a stronger methodology for standardized benchmarking and analysis. To summarize, our contributions are:

- We identify the realism-control tradeoff in e-commerce environments and agent evaluation as a methodological bottleneck, and argue that the missing capability is scalable construction of realistic, controllable, and reproducible environments and evaluation benchmarks.
- We bridge this gap with **ShopGym**, a framework for constructing realistic, controlable, and reproducible e-commerce agent evaluation environments.
- We present **ShopArena**, a pipeline that converts live seed storefronts into anonymized, self-contained sandbox shops through specification synthesis and staged code generation.
- We present **ShopGuru**, a grounded task-generation pipeline that created short-horizon and long-horizon shopping tasks tied to each generated shop's catalog, navigation structure, filters, and policies.
- We validate that the simulated environments preserve meaningful signal from live storefront evaluation, supporting their use as controlled, behaviorally aligned benchmarking environments.

2 ShopGym

ShopGym combines two complementary frameworks: **ShopArena**, a simulation environment composed of interactable sandbox shops, and **ShopGuru**, an evaluation framework with a synthetic task generation pipeline grounded in the sandbox shops. ShopArena consists of a simulation environment containing multiple sandbox shops. It also provides a scalable pipeline to extend the environment by generating new synthetic sandbox shops: given one or more seed storefronts $\{S\}_N$ (real e-commerce websites), the ShopArena generation pipeline can generate a stable, self-contained, and anonymized sandbox shop E using a multi-agent shop exploration and code synthesis pipeline. The ShopGuru evaluation framework generates a set of instruction following tasks T defined over the sandbox shop E. Following standard practice [28, 1], we define a task $T = (E, s_0, p, V)$ as a sandbox shop E with initial state s_0, a natural-language intent p, and a verification function V that maps the agent's trajectory to a binary success score. Figure 1 illustrates the full pipeline.

2.1 ShopArena

To generate each sandbox shop in the ShopArena environment, we propose a scalable pipeline that converts one or more live seed storefronts $\{S\}_N$ into a realistic, self-contained, inspectable and reproducible sandbox shop. *Importantly, the pipeline can be extended to any live storefront, making ShopArena an evolving environment that can be updated with new sandbox stores*.

The generation pipeline. Doing this end-to-end in one pass is an extremely long-horizon task: it combines storefront exploration and understanding, synthetic catalog construction, and source-code generation in a single trajectory that is hard to control and debug, and prone to incomplete outputs as agentic LLMs accumulate context [9, 14]. We instead split synthesis into two phases connected by a single intermediate artifact: an anonymized design specification M. The *exploration* phase (§2.1.1) browses $\{S\}_N$ and writes M; the *generation* phase (§2.1.2) reads M and writes the source code for a runnable sandbox shop. This split confines all live-web dependence to exploration, lets either phase

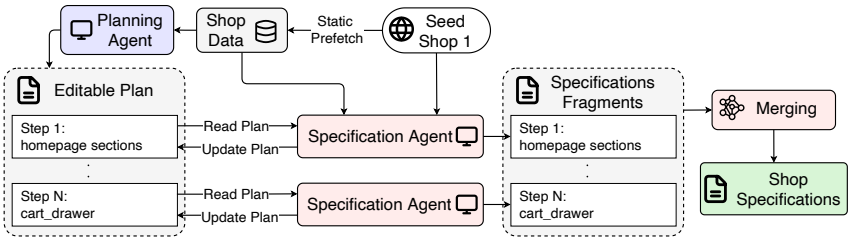


Figure 2: The exploration phase fetches key pages, and the public product catalogue. A planner agent then writes a plan using a list of predefined features to implement. Finally, Specification agents write anonymized specifications for each sub-task, which are merged to form the shop specifications.

be re-run independently, and turns the human-readable specification into a control surface for editing the generated environment without re-exploring the source.

Multi-agent framework. Both phases are organized as small sets of agents communicating through the file system. Each agent is an instance of a coding agent (Claude Code, with Claude Opus 4.6 as the underlying model) differentiated by (a) the tools it is given access to and (b) the objective described by its system prompt. Agents share a workspace directory but do not share a conversational context.

2.1.1 Exploration: From Live Storefront to Specifications

The exploration phase browses one or more seed storefronts $\{S\}_N$, extracts the properties of a live storefront that matter for downstream environment generation, and generates a specification document M. The specification consists of three facets: (i) a design manual, which is a natural language description of the store and its visual style; (ii) a structured list of its attributes that are typically present on storefronts such as filter attributes, navigation menu design, pagination information, etc. (iii) statistics such as number of products, price range and distribution, etc. Figure 2 shows the full exploration pipeline.

Static prefetch. The phase begins with a deterministic prefetch of canonical surfaces (homepage HTML, sitemap, search and cart endpoints, public catalog JSON, policy pages).

Planner agent ($\textbf{AGENT}_{\textbf{plan}}$). A single planner consumes the prefetched files and, with a small live-browsing budget, writes a plan that decomposes exploration of S_i into focused subtasks. The planner is shown an example plan but is required to adapt it to S_i, so coverage is grounded in what the storefront actually exposes rather than in a fixed template. An example plan is shown in Fig G.1.

Specification agent ($\textbf{AGENT}_{\textbf{spec}}$). For each planned subtask, the harness spawns a fresh specification agent equipped with browser-automation tools (Playwright). The agent performs targeted browsing, captures evidence (screenshots, snapshots, optional XHR traces), and writes an anonymized fragment of M for its subtask. $\text{AGENT}_{\text{spec}}$ is instructed to ignore brand names, product names, named individuals, and absolute URLs at write time, so everything downstream of exploration is anonymous by construction — no separate redaction pass is required.

Consolidation and multi-seed composition. A non-agentic LLM merges the per-subtask fragments into one specification per feature, yielding M. For a sandbox shop calibrated on multiple seeds, exploration runs independently on each storefront, producing one $M(S_i)$ per seed; a second LLM merge composes them into a single composite specification. Multi-seed composition is how ShopArena scales *within* a shop: a single sandbox can span structural and behavioral diversity that no individual source storefront would cover.

At the end of the exploration phase, the pipeline produces the anonymized shop specifications M for the sandbox shop, with the design document divided into multiple predefined *features* (website skeleton, homepage, collections, etc.) A full list of features and an example of the full specifications is provided in Appendix F

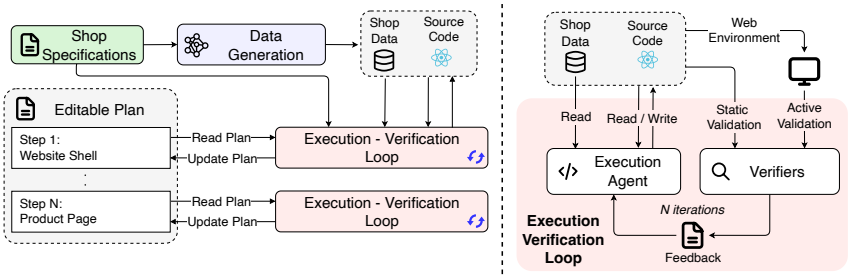


Figure 3: The generation phase. A staged sequence of feature-scoped steps is each driven by an execution–verification loop in which a fresh Execution Agent edits the codebase and a separate Verification Agent writes natural-language feedback.

2.1.2 Generation: From Specifications to Sandbox Shop

The generation phase synthesizes a runnable storefront from M. Because it sees only M and never the prefetched evidence, generation is anonymous by construction. The generation phase has two parts: a catalog synthesis pipeline and a source-code synthesis loop.

Synthetic catalog generation. The generation step first synthesizes the collection details (description, associated products), product names, product descriptions, and product images using the design manual and statistics from M. The data synthesis is structured as simple LLM calls that generate the data for one collection at a time. The details of the models used for the data generation are provided in Appendix B.1.

Stepwise source-code synthesis. ShopArena generates the storefront source code in a fixed sequence of *steps*, each scoped to one feature: site shell and homepage, collection pages, product detail page, cart, search and filtering, policy pages, and a final integration pass. Each step is given only the slice of M relevant to its feature, the catalog, and the current state of the codebase, and is executed by a dedicated execution–verification loop. Steps run in a fixed order over a shared workspace, so later steps build on a stable surface produced by earlier ones, and a failure in one step can be repaired without regenerating the rest of the shop.

Execution–verification loop. Within each step, code generation is structured in the spirit of the Ralph technique[4]: a fresh agent process is spawned at each iteration, reads the current state of the workspace from disk, edits the codebase, and exits; the harness then runs verification, writes feedback to disk, and spawns the next iteration. Two specialised agents alternate within the loop:

- **Execution Agent ($\text{AGENT}_{\text{exec}}$).** At each iteration, a coding agent with the codebase and terminal access receives the feature to implement, the slice of M, the catalog, and (if present) the verifier's most recent feedback. It edits the source code and runs lightweight checks (type checking, build).
- **Verifiers.** The code generated by $\text{AGENT}_{\text{exec}}$ is checked by both rule-based verifiers and a multimodal verification agent. The rule-based verifiers ensure the site builds successfully, required pages return valid HTTP success codes, and outputs are properly structured. The verification agent then hosts the shop locally, uses browser automation to inspect the rendered site for visual and functional issues, and produces natural-language feedback for the next $\text{AGENT}_{\text{exec}}$ iteration.

Long-context management. Each step could in principle be driven by one long-running agent that holds the codebase, the verifier's reports, and the slice of M in a single context window. We avoid this because long-horizon code generation degrades as context grows [9, 14], and we observed the same failure mode in our own development: partially broken outputs the agent itself reports as complete. Spawning a fresh process at each iteration externalises three concerns that would otherwise pile up inside one prompt: the *state context* (the current source code, read directly from disk), the *implementation context* (the agent's prior tool-use traces, discarded), and the *evaluation context* (the verifier's most recent feedback, at a known path). Each fresh $\text{AGENT}_{\text{exec}}$ iteration sees only what the workspace and the latest verifier note convey; the prompt does not grow unboundedly, and the agent

[4] `https://ghuntley.com/ralph/`

re-establishes context from the artifacts that matter. The loop runs for a fixed iteration budget (30 by default) or terminates early when the agent reports no further action is required.

Final consolidation. After all feature-specific steps complete, ShopArena runs a final consolidation pass: a global agent navigates a tour of canonical routes, and fixes the remaining defects raised by the verifiers in prior steps.

2.2 ShopGuru: Grounded Task Generation

ShopGuru uses the collections, product details, and statistics from ShopArena's generation phase (§2.1.2) and generates synthetic tasks grounded in the sandbox shop. The pipeline uses a small set of *deterministic generators*, which construct short horizon tasks meant for evaluating primitive skills, and a single *LLM-authored* generator for long-horizon shopping journeys. The pipeline is structured as an execution-verification loop: the generated tasks (using rule-based or LLM-based generators) are verified using multiple rule-based verifiers. The verifiers validate that the collections, products, filter types and options, and pages referenced by the task are present in the shop's data. A detailed description of the verifiers can be found in Table 2. Figure 4 illustrates the full pipeline.

2.2.1 Skill Catalog

We organize ShopGuru's seven skill categories into three short-horizon groups and one long-horizon group, summarized in Table 1 in Appendix H. The three short-horizon groups are themselves the underlying primitives that the long-horizon journeys re-combine.

2.2.2 Short-Horizon Task Generation

Short-Horizon tasks are meant to evaluate the basic instruction following capabilities of agents. The short-horizon tasks are generated using rule-based generators to evaluate three primitive skills:

Product Discovery (`search-exact`, `search-substitute`). The discovery generators sample products from the catalog and emit two task variants per product: one asking the agent to find the product by its exact title, and one asking for a semantically similar alternative. To keep the dataset diverse, the sampler accepts at most one product per `product_type`, and excludes inactive products, gift cards, and products with no available variant (so an agent is never asked to add an out-of-stock item).

Filter-Selection (`browse`, `filter`). Browse tasks pick a populated, non-generic collection (at least three products; generic catchall handles like `all`, `sale`, `featured`, or `best-sellers` excluded) and ask the agent to navigate to the collection page and add any product to cart. The filter generator additionally pairs the collection with a realistic facet $(dim, value)$. The non-trivial design decision here is feasibility: shop-wide option statistics are a poor source for per-collection filter facets because a dimension can be common in the catalog overall yet absent from a specific collection. We therefore build a *per-collection* option index from the collection's own products and only emit a task when at least one product in the collection actually has the chosen $(dim, value)$ pair. As a fallback, when no priority variant-option dimension is feasible in a collection, we use the universal product fields `Vendor` (storefront-rendered as "Brand") and `ProductType` (rendered as "Type"), which every storefront filter UI exposes regardless of variant-option coverage.

Information-Seeking (`shipping`, `returns`). The policy generator searches the shop's extracted `pages.json` for pages whose title or handle matches a small keyword list (*shipping*/*delivery* for shipping; *return*/*refund*/*exchange* for returns). Both the `/pages/`⟨`handle`⟩ route and the platform-default `/policies/`⟨`name`⟩`-policy` route are emitted as soft URL hints (`url_contains` plus a `url_contains_alt` list), since some storefronts surface the policy under a custom page handle and others under the platform default.

2.2.3 LLM-Authored Long-Horizon Journeys

Long-horizon journeys are the workhorse of the benchmark's `hard_long_horizon` bundle. They mimic the kind of multi-step behavior we observe in real human shopper traces — filter → sort → inspect → detour to a policy page → add to cart → edit quantity — which the primitive-task generators by themselves do not cover. Authoring such journeys by hand is prohibitively expensive at scale, so ShopGuru delegates the long-horizon authoring to an LLM (e.g. GPT-5 or Gemini 3 Pro)

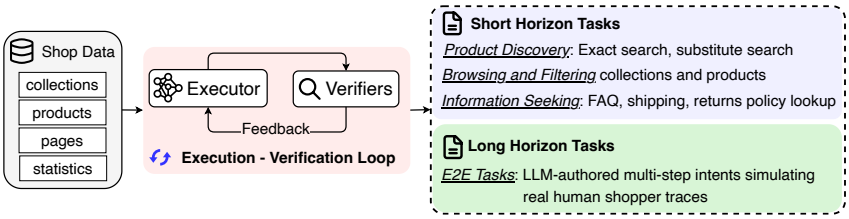


Figure 4: The ShopGuru task generation pipeline. Deterministic generators emit short-horizon tasks (`search-exact`, `search-substitute`, `browse`, `filter`, `shipping`, `returns`); an LLM-authored generator emits long-horizon shopping journeys reconciled against the shop data via a validator-driven polish loop. All emitted tasks are checked by rule-based verifiers (Table 2).

with two structural guards: (i) a few-shot prompt grounded in the shop's actual catalog, navigation, and policy pages, and (ii) a polish loop that uses the same validator from §2.2.4 as a feedback signal.

Listing 3 shows a representative end-to-end (E2E) task that chains collection navigation, a Color filter, a price sort option, and a final cart action. The few-shot examples in the user prompt template (eight in total; the full prompt and three representative examples are in Appendix L) are constructed to reflect commonly observed click-stream patterns in human shopper sessions on production storefronts, drawing on prior characterizations of online shopping behavior [19, 10]: multi-pet household shopping, returns-cautious browse-without-purchase, free-shipping-threshold calculation, cross-brand comparison, filter-recover, first-time-visitor info-page detour, homepage-callout brand drilldown, and so on.

The polish loop (Appendix J) follows from a known weakness of single-pass LLM authoring: the model occasionally hallucinates a product into the wrong collection, asks for a variant option that a product does not expose (e.g. "select a Color" on a gift card), or names a filter dimension that has no realizable values in the target collection. We close this gap by reusing the validator as a critic. After each generation pass, the validator returns a list of per-task issues; ShopGuru filters to the actionable subset (every error-severity finding plus the hallucination-class warnings) and constructs a polish prompt that lists *only the flagged tasks* alongside their specific issues and the shop context, and asks the LLM to regenerate *only those tasks* while preserving their IDs. The polish loop runs for at most two rounds. In our experiments, this is sufficient to drive the residual error rate to zero on most shops without human review. When tasks still fail validation after two rounds, the build halts with a non-zero exit code rather than silently shipping the flawed tasks; a human curator then either re-runs generation with a different seed or hand-authors replacements under `data_sources/`, which override the auto-generated outputs in the next build.

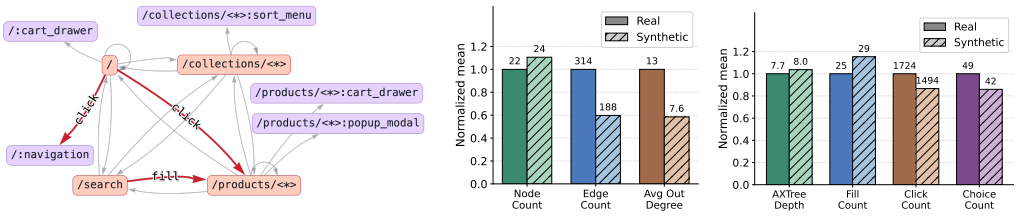
2.2.4 Post-Generation Validation

A small dependency-free validator runs over every emitted benchmark file as the final gate. Its seven core rules (Table 2)—four error-severity and three warning-severity—target the failure modes we observed during multi-shop benchmark regeneration; collectively they catch every class of grounding mismatch we encountered. An actionable subset of the validator's findings (every error plus the two hallucination-class warnings) is also reused as the polish signal for the LLM-authored journey generator (§2.2.3), so the same audit logic both gates the output and steers the LLM toward feasible tasks; the remaining `unknown-page` warning is advisory only.

3 Evaluating Sandbox Shop Quality

For the synthetic shops in the ShopArena environment to be a reasonable substitute for live storefronts, we want the sandbox shops to be *structurally and behaviorally* similar to real websites. Structural validation aims to evaluate the static structure of the synthetic shop against the real shops, whereas behavioral validation evaluates shopping workflows.

To demonstrate and validate the ShopArena as a structurally and behaviorally aligned environment, we generate three synthetic sandbox shops spanning clothing, cookware and electronics hardware domains. We refer to these shops as `sandbox shops`. The screenshots of the sandbox shops can be found in Section E, along with the links to the live hosted sandbox shops. We further generate



(a) (b) (c)

Figure 5: (a) An example directed graph visualization of the website; (b) the State Transition statistics; (c) the Observation/Interaction statistics

three synthetic shops using real product data instead of synthetic data, which we refer to as `twin shops`. When generating the twin shops, we use an additional visual verification agent that matches the visual theme and structure of the real storefronts.

3.1 Structural Validity

To validate the structure of the `sandbox shops`, we compare them with real shops along two axes: *Observation and Interaction Complexity*, and *State Transition Complexity*. We use 7 real shops and 3 synthetic shops for the comparisons. We note that this is an unpaired validation.

Observation/Interaction Complexity. To compare the observation space, we use the depth of the simplified accessibility tree as a proxy for evaluating the nesting complexity of the websites[5]. The nesting complexity evaluates how much structured content the agents must parse. We compare the interaction complexity by counting the number of unique elements on the website that support the `fill` action, `click` action, or elements that support select style controls (`choice`). Figure 5c shows that the synthetic sandbox shops are comparable in terms of observation and interaction complexity.

State Transition Complexity. To compare the state transition complexity, we treat the website as a directed state-transition graph, where each node corresponds to a distinct UI state, as illustrated in Figure 5a. Each pink node (for example `/`) denotes individual pages, and violet nodes (`/:navigation`) denote alternate UI configurations of those pages when interactive elements are opened. Edges denote the actions that trigger transitions between the states, for instance, the red arrow shows clicking the navigation bar on `/` transitions the state to `/:navigation`.

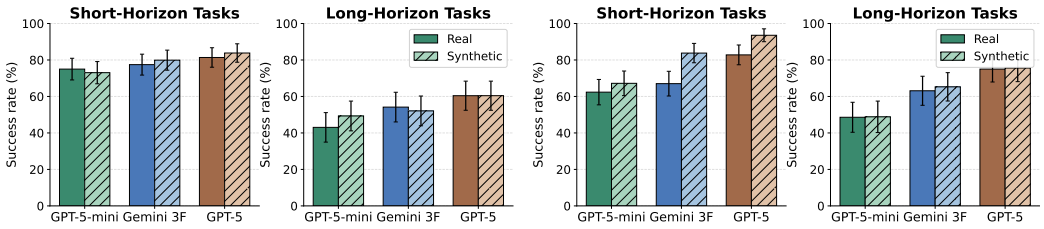
The aggregated statistics in Figure 5b show that the synthetic sandbox shops have a comparable number of nodes to the real shops, indicating similar coverage of distinct UI states. However, they exhibit fewer edges and a lower average out-degree, meaning that each state tends to have fewer outgoing transitions. This reduction is largely because real storefronts contain additional navigation paths through external links, marketing pages, and auxiliary site components that are intentionally excluded from the sandbox environments. Overall, the synthetic shops closely match real storefronts in observation complexity, action affordances, and transition-graph scale, supporting their use as realistic sandbox environments.

3.2 Behavioral Validation

We use the `twin shops` to perform behavioral validation of the ShopArena pipeline. The `twin shops` are visually and structurally similar to real shops, with the shop-specific information such as names, locations, contact details, etc. replaced with mock information. We then generate the ShopGuru evaluation suite for the real and twin shops, and evaluate multiple frontier models on both variants. We evaluate GPT-5-mini, GPT5 [13], and Gemini 3 Flash.

We evaluate the models using two harnesses: a BrowserGym [4] based implementation that uses accessibility trees (AXTrees), and a proprietary implementation that uses screenshots along with AXTrees. We use GPT-5 as a judge in both implementations, which analyzes the agent trajectories and provides a binary success rate. The implementation details are provided in Appendix B.

[5]As defined by BrowserGym

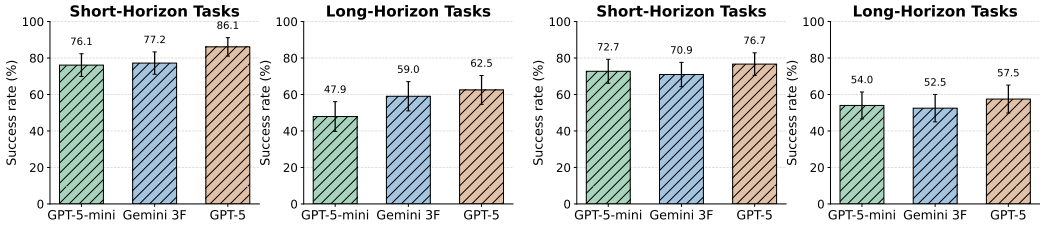


(a) BrowserGym Harness. (b) Internal Harness.

Figure 6: Success rates on real storefronts vs. ShopArena twins.

Figure 6 shows that all models achieve similar performance on the real storefront and its twin variant. Furthermore, the performance on the complex tasks improves with stronger models. This shows that from the perspective of agent behavior, the sandbox shops are behaviorally consistent with real shops.

4 Validating ShopGuru task complexity



(a) BrowserGym Harness. (b) Internal Harness.

Figure 7: Success rates on the synthetic sandbox shops.

To validate that the ShopGuru task generation pipeline can generate complex tasks, we evaluate frontier models with the ShopGuru evaluation suite on the `sandbox shops`. We use the same web agent, harness, and judge setup as §3.2.

As seen in Figure 7a GPT-5-mini with the BrowserGym harness achieves a success rate of 47.9% on the Long-Horizon complex tasks, Gemini 3 Flash achieves 59%, and GPT-5 achieves 62.5%. The results with the custom harness show similar performance across all models. A low performance of frontier models shows that the ShopGuru task generation pipeline can generate meaningfully hard tasks which are useful for training and evaluation of next generation e-commerce agents.

5 Conclusion

We presented ShopGym, an integrated framework for realistic simulation and scalable benchmarking of e-commerce web agents. ShopGym addresses a central methodological gap in current environments: live storefronts provide realism but are non-stationary and irreproducible, while sandbox environments are controllable but limited in diversity and realism. By combining ShopArena for sandbox shop generation and ShopGuru for grounded task synthesis, ShopGym produces evaluation artifacts that are self-contained, inspectable, resettable, and stable across runs.

Our results show that the generated shops are structurally comparable to real storefronts and preserve meaningful behavioral signal for agent evaluation. In particular, agent performance on synthetic twins positively tracks performance on live storefronts, while fully synthetic composite shops yield benchmarks of comparable difficulty. These findings suggest that ShopGym provides a practical middle ground between live-web realism and sandbox control, and offers a scalable foundation for more reproducible and interpretable evaluation of e-commerce web agents.

References

[1] Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026.

[2] Alberto Castelo, Zahra Zanjani Foumani, Ailin Fan, Keat Yang Koay, Vibhor Malik, Yuanzheng Zhu, Han Li, Meysam Feghhi, Ronie Uliana, Shuang Xie, Zhaoyu Zhang, Angelo Ocana Martins, Mingyu Zhao, Francis Pelland, Jonathan Faerman, Nikolas LeBlanc, Aaron Glazer, Andrew McNamara, Lingyun Wang, and Zhong Wu. Simgym: Traffic-grounded browser agents for offline A/B testing in e-commerce. *CoRR*, abs/2602.01443, 2026. doi: 10.48550/ARXIV.2602.01443. URL https://doi.org/10.48550/arXiv.2602.01443.

[3] Hyungjoo Chae, Jungsoo Park, and Alan Ritter. Safe and scalable web agent learning via recreated websites. *CoRR*, abs/2603.10505, 2026. doi: 10.48550/ARXIV.2603.10505. URL https://doi.org/10.48550/arXiv.2603.10505.

[4] Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=5298fKGmv3. Expert Certification.

[5] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

[6] Apurva Gandhi and Graham Neubig. Go-browse: Training web agents with structured exploration. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IpzRWE52yw.

[7] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 6864–6890. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.371. URL https://doi.org/10.18653/v1/2024.acl-long.371.

[8] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024*, 2024. URL https://arxiv.org/abs/2401.13649.

[9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2023. URL https://api.semanticscholar.org/CorpusID:259360665.

[10] Yuxuan Lu, Jing Huang, Yan Han, Bingsheng Yao, Sisong Bei, Jiri Gesi, Yaochen Xie, Qi He, Dakuo Wang, et al. Can llm agents simulate multi-turn human behavior? evidence from real online customer behavior data. *arXiv preprint arXiv:2503.20749*, 2025.

[11] Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuyi Chen. Deepshop: A benchmark for deep research shopping agents. *ArXiv*, abs/2506.02839, 2025. URL https://api.semanticscholar.org/CorpusID:279118560.

[12] Ralph Peeters, Aaron Steiner, Luca Schwarz, Julian Yuya Caspary, and Christian Bizer. Webmall–a multi-shop benchmark for evaluating web agents [technical report]. *arXiv preprint arXiv:2508.13024*, 2025.

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

[14] Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms. *ArXiv*, abs/2509.09677, 2025. URL https://api.semanticscholar.org/CorpusID:281252776.

[15] Dakuo Wang, Ting-Yao Hsu, Yuxuan Lu, Hansu Gu, Limeng Cui, Yaochen Xie, William Headean, Bingsheng Yao, Akash Veeragouni, Jiapeng Liu, et al. Agenta/b: Automated and scalable web a/btesting with interactive llm agents. *arXiv preprint arXiv:2504.09723*, 2025.

[16] Jiang Wang, Kejun Xiao, Qi Sun, Huaipeng Zhao, Tao Luo, Jiandong Zhang, and Xiaoyi Zeng. Shoppingbench: A real-world intent-grounded shopping benchmark for llm-based agents. In *AAAI Conference on Artificial Intelligence*, 2025. URL https://api.semanticscholar.org/CorpusID:280536823.

[17] Jiangyuan Wang, Kejun Xiao, Huaipeng Zhao, Tao Luo, and Xiaoyi Zeng. Productresearch: Training e-commerce deep research agents via multi-agent synthetic trajectory distillation. *CoRR*, abs/2602.23716, 2026. doi: 10.48550/ARXIV.2602.23716. URL https://doi.org/10.48550/arXiv.2602.23716.

[18] Pei Wang, Yanan Wu, Xiaoshuai Song, Weixun Wang, Gengru Chen, Zhongwen Li, Ke Yan, Ken Deng, Qi Liu, Shu-Man Zhao, Shaopan Xiong, Xuepeng Liu, Xuefeng Chen, Wanxi Deng, Wenbo Su, and Bo Zheng. Shopsimulator: Evaluating and exploring rl-driven llm agent for shopping assistants. *ArXiv*, abs/2601.18225, 2026. URL https://api.semanticscholar.org/CorpusID:285050373.

[19] Ziyi Wang, Yuxuan Lu, Wenbo Li, Amir A. Amini, Bo Sun, Yakov Bart, Weimin Lyu, Jiri Gesi, Tian Wang, Jing Huang, Yu Su, Upol Ehsan, Malihe Alikhani, Toby Jia-Jun Li, Lydia B. Chilton, and Dakuo Wang. Opera: A dataset of observation, persona, rationale, and action for evaluating llms on human online shopping behavior simulation. *ArXiv*, abs/2506.05606, 2025. URL https://api.semanticscholar.org/CorpusID:279244562.

[20] Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Jing Huang, and Dakuo Wang. Customer-r1: Personalized simulation of human behaviors via rl-based llm agent in online shopping. *arXiv preprint arXiv:2510.07230*, 2025.

[21] Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *CoRR*, abs/2504.01382, 2025. doi: 10.48550/ARXIV.2504.01382. URL https://doi.org/10.48550/arXiv.2504.01382.

[22] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

[23] Zijian Yu, Kejun Xiao, Huaipeng Zhao, Tao Luo, and Xiaoyi Zeng. Shopping companion: A memory-augmented LLM agent for real-world e-commerce tasks. *CoRR*, abs/2603.14864, 2026. doi: 10.48550/ARXIV.2603.14864. URL https://doi.org/10.48550/arXiv.2603.14864.

[24] Peng Yuan, Yuyang Yin, Yuxuan Cai, and Zheng Wei. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *arXiv preprint arXiv:2604.10988*, 2026.

[25] Shuo Zhang, Boci Peng, Xinping Zhao, Boren Hu, Yun Zhu, Yanjia Zeng, and Xuming Hu. Llasa: Large language and e-commerce shopping assistant. *CoRR*, abs/2408.02006, 2024. doi: 10.48550/ARXIV.2408.02006. URL https://doi.org/10.48550/arXiv.2408.02006.

[26] Yimeng Zhang, Jiri Gesi, Ran Xue, Tian Wang, Ziyi Wang, Yuxuan Lu, Sinong Zhan, Huimin Zeng, Qingjun Cui, Yufan Guo, et al. See, think, act: Online shopper behavior simulation with vlm agents. *arXiv preprint arXiv:2510.19245*, 2025.

[27] Shuyan Zhou. Webarena-infinity: Generating browser environments with verifiable tasks at scale. *shuyanzhou.com*, March 2026. URL https://webarena.dev/webarena-infinity/.

[28] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Related Work

Static web environments and benchmarks. Prior work on web-agent evaluation has pursued two goals: broad coverage of real websites and reproducible executable environments. Mind2Web [5] provides large-scale offline traces from 137 real websites across 31 domains, emphasizing breadth and cross-site generalization. OPeRA [19] similarly collects observations, personas, rationales, and actions from real human shopping sessions on Amazon. In contrast, WebArena [28] provides self-hosted, fully functional websites with executable tasks, and VisualWebArena [8] extends this setting to visually grounded tasks. Such environments support autonomous exploration and reproducible evaluation, unlike purely offline datasets. Several benchmarks focus specifically on e-commerce. WebShop [22] offers a synthetic storefront with 1.25M products. ShoppingBench [16] provides a large shopping sandbox with API-based interaction, aimed primarily at intent grounding. ShopSimulator [18] introduces a Chinese web environment for multi-turn shopping dialog and fine-grained product differentiation. WebMall [12] expands to four simulated shops with authentic product offers and comparison-shopping tasks. Although these benchmarks enable controlled evaluation, they remain tied to a fixed set of layouts, taxonomies, and policies, and therefore capture only a limited slice of real-world e-commerce diversity.

Live-website evaluation. To address this limitation, Online-Mind2Web [21], WebVoyager [7], and DeepShop [11] evaluate agents directly on live websites, including real shopping domains. These settings improve diversity and realism, but depend on external sites whose layouts, inventories, flows, and access policies may change without notice. This non-stationarity makes comparisons over time difficult and limits reproducibility, especially for RL training.

Automated environment and task generation. A recent line of work uses LLMs to automate environment construction. Gym-Anything [1] treats setup as a multi-agent pipeline that installs and configures existing software, producing CUA-World with 10K+ tasks. WebArena-Infinity [27] generates synthetic web apps from user manuals. WebForge [24] uses a Plan/Generate/Refine/Validate pipeline to create self-contained web environments at scale, and VeriEnv [3] constructs executable synthetic versions of real websites with deterministic rewards exposed through a Python SDK. These approaches scale well, but often sacrifice fidelity: their environments are derived from manuals or ungrounded LLM design choices and may miss the layouts, integrations, and edge cases of production storefronts. In contrast, ShopArena grounds generation in observed real shops: it first explores a live storefront to produce an anonymised Shop Manual capturing taxonomy, navigation, capabilities, and statistics, then synthesises a calibrated sandbox shop from that manual. This preserves much more of the diversity of real storefronts while retaining determinism and reset-ability. Go-Browse [6] proposes a multi-agent pipeline for generating tasks and trajectories by navigating websites and maintaining a graph of visited URLs for efficient exploration.

B Implementation Details

B.1 Models used for agent implementations

- Planning Agent: Claude Code with Claude Opus 4.6
- Specification Agent: Claude Code with Claude Opus 4.6
- Collections generator: Claude Opus 4.6
- Product name and description generator: Claude Sonnet 4.6
- Product Image Generator: Gemini 3 Pro
- Execution Agent: Claude Code with Claude Opus 4.6
- Visual Verification Agent: Claude Code with Claude Opus 4.6, enabled with Playwright for browser automation

B.2 Harness A: BrowserGym Agent

Observation. The BrowserGym harness uses the accessibility tree as the observation. We do not use screenshots for simplicity.

Action Space. The agent is allowed to use 20 actions:

- `noop`: Do nothing, optionally waiting for the given number of milliseconds.
- `send_msg_to_user`: Send a message to the user (terminal action that ends the episode).
- `scroll`: Scroll the page horizontally and vertically by the given pixel deltas.
- `fill`: Type a value into an `<input>`, `<textarea>`, or `[contenteditable]` element.
- `select_option`: Select one or multiple options in a `<select>` element by value or label.
- `click`: Click an element with a given mouse button and optional modifier keys.
- `dblclick`: Double-click an element with a given mouse button and optional modifier keys.
- `hover`: Move the mouse over an element to hover it.
- `press`: Focus an element and press a key combination (e.g. `Control+Enter`, `ArrowDown`).
- `focus`: Move keyboard focus to the matching element.
- `clear`: Clear the contents of an input field.
- `drag_and_drop`: Drag one element and drop it onto another.
- `upload_file`: Click an element to open a file chooser and upload one or more files.
- `go_back`: Navigate back to the previous page in browser history.
- `go_forward`: Navigate forward to the next page in browser history.
- `goto`: Navigate the current tab to the given URL.
- `tab_close`: Close the currently active tab.
- `tab_focus`: Activate the tab at the given index, bringing it to the front.
- `new_tab`: Open a new tab and make it the active one.
- `report_infeasible`: Declare the task impossible with a short reason (terminal action that ends the episode).

Browser. We use a headless chromium browser for all experiments with BrowserGym, with Playwright for browser interaction.

Trajectory details. The BrowserGym agent uses a think - action - memory loop, where only the observation from the current step is used as context. The agent is instructed to reason first using `<think>...</think>` tags, output a single action in `<action>...</action>` tags, and write whatever it thinks is necessary to solve the task in `<memory>...</memory>` tags. The think, action and memory tags from all the steps are used as context in the future steps, along with any execution errors. The agent is allowed a maximum of 30 steps, and the episode terminates when the agent sends a message to the user using the `send_msg_to_user` action, reports the task as infeasible using the `report_infeasible` action, or the step threshold is reached. Episodes that reach the step budget are graded as failed.

B.3 Harness B: Internal Harness

The internal harness implements a single-call-per-step planner–executor agent that runs over a textual page projection and emits one Playwright-grounded action per step. Each task is evaluated across several independent browser sessions to absorb agent and storefront stochasticity.

Observation. At each step, the current page is rendered into a tree-shaped textual projection that fuses the rendered DOM and the accessibility tree into a single representation. Every interactive element (clickable control, text input, combobox, menu item, etc.) is tagged with a stable identifier of the form `[elem-N]` that the agent uses to reference the element in its action. For all experiments in this paper the agent additionally receives a viewport screenshot of the current page at each step, so the observation is multimodal (page projection plus rendered pixels). The agent prompt also includes the current URL and a running *memory*: a per-step textual log of prior decisions, action outcomes (success / error strings), and the agent's own analysis. The full LLM transcript is discarded at the end of each step — only the memory summary, the fresh page projection, and (optionally) the latest screenshot survive into the next prompt — so context length grows linearly in *step count*, not in cumulative LLM output.

Action space. Each step's LLM call is constrained by a fixed structured-output schema with the following fields:

- `analysis` — free-form chain-of-thought over the current state, prior memory, and the task goal.
- `shouldEnd` — boolean flag terminating the rollout.
- `instruction` — a short natural-language description of the intended next action (e.g. "click on the filter menu titled *Price*", "type *lunch box for kids* in the header search field").
- `elementId` — the `elem-N` target of the action, referencing one of the IDs in the current page projection.
- `method` — a Playwright interaction method (`click`, `fill`, `selectOption`, `press`, `check`, etc.).
- `arguments` — an optional list of method arguments (e.g. the value to type into a text field).

The harness translates the (`elementId`, `method`, `arguments`) tuple into a Playwright call against the live browser session. Failures (selector misses, navigation errors, timeouts) are surfaced back to the agent as an error string in the next step's prompt and recorded verbatim in memory.

Termination and budgets. A rollout terminates on (i) `shouldEnd=true` emitted by the agent, (ii) hitting the per-task step budget of 40 steps, or (iii) hitting the per-task wall-clock budget of 850 s. Rollouts that exit via the step or time budget are scored as failures regardless of the cart state at termination (§B.4), since the agent itself did not declare goal completion.

Browser. The agent runs against a remote, Playwright-compatible browser session provisioned per task. Each session is initialised at the task's starting URL (`*_real` for the live source storefront or `*_twin` for the ShopArena sandbox), accepts a *cart-only* terminal-state convention (every shopping task in ShopGuru ends with the agent ending the session once the cart contains the requested items, never by clicking a Checkout button), and is torn down at the end of the rollout.

LLM client and models. All LLM calls use an OpenAI-compatible chat-completion API with structured-output (JSON) decoding. We evaluate three models spanning compact and flagship capability tiers: GPT-5 mini, Gemini 3 Flash, and GPT-5. Where the underlying model exposes it, reasoning effort is set to `medium`. Each LLM call is retried up to three times on transient API errors before the rollout is marked failed.

Repeats and aggregation. We run three independent rollouts per (task, model, environment) cell. Per-task verdicts are averaged within each cell, and the error bars in Figs. 6b and 7b span the per-cell standard error of the mean across the three repeats. Per-bundle pass rates aggregate across all tasks in the bundle (`easy_short_horizon` or `hard_long_horizon`) and all repeats.

B.4 LLM-as-Judge

Both harnesses score every rollout with the same LLM-as-judge model but slightly different protocols. After each rollout, the judge LLM (GPT-5) receives:

- the original task intent and the task's `success_criteria` JSON (URL hint plus any cart/response constraints);
- the agent's per-step memory log (textual summary of decisions and action outcomes);
- the agent's URL trajectory (every URL visited, in order);
- the rollout's `terminationType` (`goal_reached`, `steps_limit_reached`, or `time_limit_reached`).

The judge prompt instructs the model to weigh task satisfaction first, followed by output quality, tool-use effectiveness, and recovery from transient errors; it returns a structured JSON object with a binary `success` field and a free-form `reasoning` explanation. The verdict is then post-processed by a hard rule: rollouts that did not terminate via the agent's own `shouldEnd` are forced to `success=false` regardless of the judge's opinion. This guards against a class of spurious successes where the agent times out on a page whose final state happens to look correct (e.g. the requested product is on the page but was never added to cart) but never emits a deliberate end-of-session signal.

With the BrowserGym harness, `url_contains` field is treated as a hard gate, where the trajectory is graded as failed if the expected URL is not present in the trajectory. The judge LLM receives the entire cart history, the agent's reasoning, action and memory traces, as well as the screenshots of the webpages in the trajectory and outputs a binary pass/fail decision along with it's reasoning.

B.5 Infrastructure Requirements

We perform all evaluations using the official APIs from the model providers. To parallelize the evaluation suite, the agent harnesses are orchestrated on a standard cloud virtual machine.

C Limitations

ShopGym improves control, reproducibility, and scalability, but it is not a full substitute for live-web evaluation. By construction, ShopArena removes many sources of real-world non-stationarity, including inventory churn, personalization, A/B tests, third-party integrations, latency variation, and other operational noise. This is desirable for controlled benchmarking, but it also means that strong performance in ShopGym should be interpreted as evidence of capability under stabilized conditions, not as a guarantee of robustness on live storefronts. More broadly, the generated shops are behaviorally aligned approximations rather than exact reconstructions, so some fine-grained brand-specific, visual, or platform-specific cues are intentionally not preserved.

Our validation metrics are proxies for realism: the structural analysis uses accessibility-tree depth, interaction counts, and transition-graph statistics, which capture meaningful aspects of interface and navigation complexity, but do not fully measure semantic ambiguity, visual fidelity, content quality, or all sources of user-facing behavioral variation. Likewise, the behavioral validation shows that model performance on twin shops positively tracks performance on live storefronts, but correlation in aggregate success rate does not imply that the same latent capabilities or failure modes are being exercised in exactly the same way.

Finally, the current task suite covers only part of the e-commerce agent problem. ShopGuru focuses on product discovery, browsing, filtering, information seeking, and multi-step in-store shopping journeys. It does not yet cover several important settings, including login-protected flows, checkout and payment, recommendation modules and sponsored content, customer-service interactions, cross-store comparison, post-purchase workflows, multi-session memory, multilingual settings, or safety- and policy-critical edge cases. Extending the framework to these settings is an important direction for future work.

D Broader Impact and Safeguards

ShopGym is intended to improve the reproducibility, controllability, and inspectability of e-commerce web-agent evaluation. By converting live storefronts into sandboxed evaluation environments, the framework reduces reliance on non-stationary live websites and enables more reliable comparison across agents.

Potential positive impact. ShopGym is a framework for constructing e-commerce simulation environments and grounded benchmark tasks. The simulation layer, ShopArena, provides a framework for creating diverse, stable and realistic simulation environments from live storefronts. Beyond evaluations, the environments can be used for any agent simulations, such as training of e-commerce agents using Reinforcement Learning.

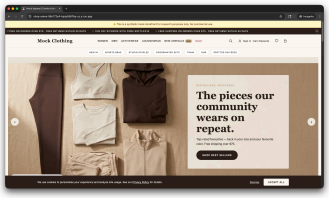
Potential negative impact. We nevertheless identify several potential negative broader impacts. First, progress on web-agent evaluation may indirectly contribute to stronger agents that could be misused for unauthorized browsing automation, large-scale scraping, spammy shopping behavior, inventory manipulation, or deceptive commercial activity. Second, because the pipeline uses live storefronts during the exploration phase, careless use could raise privacy, copyright, fair-use, or terms-of-service concerns if raw source-store content, identifiable commercial information, or original assets were copied or redistributed. Third, generated benchmarks may not be representative of all e-commerce websites, merchant categories, languages, geographic regions, accessibility patterns, or user populations. Claims about benchmark diversity should therefore be interpreted relative to the seed storefronts and task families used to instantiate a particular ShopGym benchmark.

Mitigations. We mitigate these risks through several safeguards. ShopGym is designed as a sandboxed research environment rather than a tool for acting on live commercial websites. Generated shops are self-contained, locally hostable, resettable, and inspectable. Login is disabled and checkout is mocked, so the benchmark does not support real purchases, account access, or interaction with real customers. During exploration, the pipeline extracts public, user-facing structural information such as navigation affordances, catalog statistics, page organization, and policy-page structure. Downstream generation uses anonymized specifications and does not preserve brand names, product names, named individuals, contact details, or absolute URLs. Released benchmark artifacts use synthetic brand identities, products, descriptions, and imagery rather than direct copies of source-store assets.

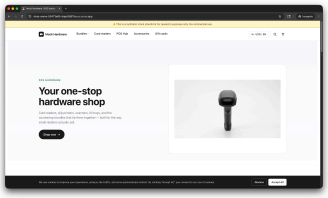
We will document the intended use, limitations, task schema, environment setup, dependency versions, and licenses of released artifacts. We do not release raw source-store data or original commercial assets. Users who instantiate ShopGym on additional storefronts should respect applicable website terms, robots policies where relevant, copyright restrictions, and regional legal requirements. These safeguards are intended to support responsible research use while reducing the risk that the framework is used for unauthorized live-web automation or redistribution of commercial content.

E ShopArena Generation Artifact Example

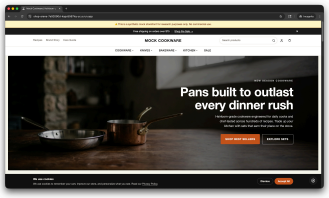
This appendix demonstrates the sandbox shops generated by the ShopArena shop generation pipeline. These sandbox shops are built from exploration artifacts generated from ShopArena exploration tasks.



(a) Mock Clothing



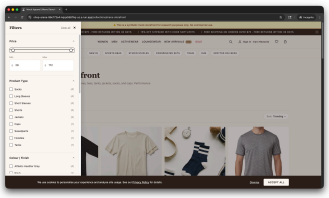
(b) Mock Hardware



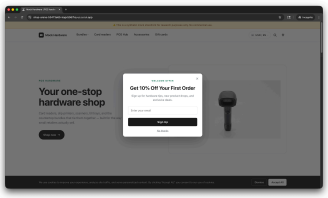
(c) Mock Cookware

Figure 8: Screenshots of Sandbox Shop Homepages

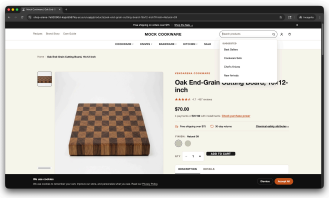
Figure 9 shows more detailed interaction surfaces within the generated sandbox shops. Built from specifications, the examples include a collection page with faceted filters, a homepage with a promotional popup, and a product detail page with search suggestions and purchasing controls. These screenshots highlight that the generated shops contain not only realistic visual layouts, but also functional e-commerce elements that support browsing, filtering, promotion handling, search, and product-level decision making.



(a) Collection page with filters



(b) Homepage with promotion popup



(c) Product detail page

Figure 9: Screenshots of Sandbox Shop Details

Live versions of the sandbox shops can be accessed at:

- Mock Clothing: `https://shop-arena-96c172e4-kqqx5567kq-uc.a.run.app/`
- Mock Hardware: `https://shop-arena-53473e65-kqqx5567kq-uc.a.run.app/`
- Mock Cookware: `https://shop-arena-7e92090d-kqqx5567kq-uc.a.run.app/`

F ShopArena Exploration Artifact Example

This appendix reproduces the three top-level outputs that `shop_arena.explore` writes for a single live storefront exploration run. The example was produced from a premium hybrid cookware storefront; all store, brand, designer, product, and URL strings were anonymized at write time per the explorer's constitution, so only structural and behavioral facts remain.

F.1 Shop Manual (`manual.md`)

The manual instructs the generation pipeline to implement the cookware shop by dividing it into 7 features: `Site Shell`, `Homepage`, `Collections & Navigation`, `Cart`, `Search`, `Floating UX`, `Policy & info pages`.

```
# Shop Manual -- Premium Hybrid Cookware and Kitchen Accessories Shop

## Overview

This is a premium hybrid cookware and kitchen accessories storefront priced in USD. The brand voice is
      bold, chef-endorsed, and direct-to-consumer, leaning on culinary-professional credibility and a
      "buy once" durability narrative. Aggregate review counts in the 50,000+ range and a stated
      lifetime warranty signal a sizeable, established catalog rather than a niche boutique.

## Site shell

A persistent dismissible announcement bar sits above a two-row sticky header. Row 1 is a utility bar:
      a left cluster of three editorial text links plus a "Partners" dropdown button listing 2-3 named
      collaborator pages, a centered logo, and a right cluster of search, account, and cart icon
      buttons. Row 2 is the primary category navigation -- eight top-level entries rendered as inline
      buttons. Most entries open flyout panels rather than a single mega menu, so navigation depth
      tops out at two levels (top-level -> sub-link). Each flyout is a full-width two-column panel: a
      vertical sub-link list ending in "Shop All [Category]" on the left, plus 2-3 featured product
      cards on the right (image, name, price with optional compare-at, savings badge, and star
      rating). One top-level entry behaves as a direct link instead of opening a flyout.

A mobile off-canvas drawer mirrors the same taxonomy as accordions, with extra direct links for sale,
      gift cards, "Shop All", and the Partners group.

The footer occupies a `contentinfo` landmark on every page and is split into a top band and a bottom
      band. The top band has three named link groups (Shop, Help, Media) plus a fourth column of four
      social icon links opening in new tabs. The bottom band is a flex row pairing a copyright line
      with two legal links (terms of service, accessibility commitment) against a brand wordmark
      image. The footer carries no inline newsletter form, no payment icons, and no locale or currency
      switcher.

## Homepage

The homepage is a long, modular composition of sixteen sections in this order:

- Full-width hero with autoplay muted looping background video and a centered headline + single CTA
      overlay.
- "Shop by Category" chip slider -- a horizontally scrolling row of seven image-backed sub-collection
      chips with prev/next arrows and a "Shop All" link.
- Featured sets promo block -- text-left / product-cards-right, with a 3-card row showing image,
      savings badge, name, current price, and original price.
- Specialty product banner -- a reversed two-column block pairing two product cards on the left with
      descriptive copy on the right introducing a limited-time sub-line.
- Chef endorsement block -- autoplay video thumbnail with a play overlay opening fullscreen playback,
      paired with an attributed blockquote and a star + aggregate-review-count social-proof line.
- 2x2 feature card grid -- four cards covering high-heat capability, PFAS-free nonstick, dishwasher
      safety, and durability/warranty, each with a small attribute badge, an all-caps headline, and a
      one- to two-sentence descriptor.
- Reviews banner -- aggregate rating + rotating customer quote on one side, a three-icon trust strip
      (lifetime warranty, free shipping, 30-day returns) on the other.
- "Best Sellers" horizontal product carousel with prev/next arrows and a "Shop All" link.
- Knife category hero with two product cards on the left and a category value-prop + customer quote
      chip on the right.
- UGC video carousel -- six video thumbnail tiles with play overlays; tiles open a fullscreen popup
      player.
- 3-up specialty category banners -- three equal-width cards each combining a product image, headline,
      descriptor, CTA link, and a customer quote chip with star rating.
- Culinary council section introducing the brand's chef advisory group with five professional names
      (roles only) and a CTA to a dedicated page.
- "Recipes" carousel of recipe cards with image, title, yield, and time, linking into the recipes blog.
- "Articles" carousel of editorial blog cards using the same card chrome minus the recipe metadata.
- Founder message -- a short two-paragraph first-person note signed with a plain-text name.
- Inline newsletter section at the page bottom with an email input, hidden honeypot, and "Subscribe"
      button.
```

```
There is no homepage modal popup. A separate Klaviyo-driven popup is configured but did not fire
      during exploration.

## Collections & navigation

Collection pages render as a three-column grid on desktop. Each card shows a product image, an
      optional promotional overlay badge ("SALE", "New", "Bestseller"), product title, star rating
      with review count, and price (strikethrough original price plus a percentage-off badge when
      discounted). The collection title displays a live product count in parentheses.

Above the grid sits a horizontal filter bar with two collapsible controls. The "Filter" panel exposes
      two checkbox facets -- "Available" and "On Sale" -- that update URL parameters immediately,
      render server-side, and combine cleanly. The "Sort by" dropdown offers eight options: featured,
      best selling, alphabetically A-Z, alphabetically Z-A, price ascending, price descending, date
      new-to-old, and date old-to-new. Sorting updates the `sort_by` URL parameter without a full
      reload.

Pagination uses a "Load more" button. The initial render shows 24 cards; clicking the button appends
      the rest of the collection in a single batch and the button then disappears. There is no
      infinite scroll, no numbered pagination, and no `page=` URL parameter.

## Product page

The product page leads with a horizontal image carousel that has prev/next arrow controls and an
      optional left-side thumbnail strip for direct slide selection. Below the title sits the price
      block: prominent current price, optional strikethrough original price, optional percentage
      savings badge, and a buy-now-pay-later financing line that opens a modal. A trust badge row
      beneath the price summarizes free shipping, a 30-day return guarantee, and (where applicable) a
      lifetime warranty, plus an expandable claim button listing chemical-safety attributes for
      nonstick items.

Variant selectors render as radio buttons in a single horizontal row. Apparel exposes a size axis
      (XS-XXL) with a "Size Guide" link above the group. Single-variant products skip the selector
      entirely. A quantity stepper (decrement, numeric display, increment) sits immediately adjacent
      to the "Add to Cart" CTA.

Descriptions vary by category. Simpler products render an inline prose paragraph plus a bulleted
      feature list. Complex products use an accordion-tab layout (e.g. "Description" + "Hybrid
      Technology" tabs) followed by a horizontal seven-icon attribute strip (warranty, utensil safety,
      nonstick, oven-safe, dishwasher-safe, induction-ready, stay-cool handle). Some PDPs include a
      "Gift With Purchase" callout above the quantity stepper showing a bundled free item.

Reviews use an Okendo-style widget. Products without reviews show a "Be the first to leave a review"
      prompt and a "Write a review" CTA. Products with reviews show an aggregate score, a clickable
      star-distribution breakdown, an auto-generated AI summary block, search and sort/filter
      dropdowns (latest, oldest, highest, lowest, most helpful; rating 1-5 or all), and individual
      review cards with reviewer name, star rating, relative date, headline, body, and helpful-vote
      buttons.

Recommendation rails differ by template. Some PDPs render a "Customers also love" rail of product
      cards with quick-add buttons and inline color-swatch selectors. Others render two stacked rails
      -- a "Frequently Bought Together" trio with inline "Add to Cart" buttons, and a "Recently
      Viewed" carousel with prev/next arrows. No customization or personalization fields (engraving,
      monogram, gift note) are exposed on the PDP.

## Cart

The cart opens as a right-side slide-out drawer (ARIA `dialog` labeled "Your Cart") triggered by the
      header cart icon. The icon shows a numeric item-count badge once the cart has any items.

Empty state shows the "Your Cart" heading with a `0` badge, a short empty-cart sentence, a CTA link
      into a best-sellers collection, a `$0.00` subtotal line, and the same trust-badge trio (lifetime
      warranty, free shipping, 30-day returns) repeated near the footer.

Filled state replaces the empty copy with a free-shipping qualifier banner once the threshold is met.
      Each line item shows a thumbnail, name (both linked to the PDP), the price block (current price
      plus strikethrough compare-at when discounted), a "Remove Item" affordance, and a three-button
      quantity stepper. Quantity changes apply via Ajax and immediately recompute three summary rows:
      products at list price, total savings, and subtotal. A full-width "Checkout" button sits below,
      with a footnote asterisk for free-shipping eligibility and the trust badges repeated.

The drawer carries a "Frequently Bought Together" upsell rail (three product cards with inline "Add to
      Cart") in both states; carted items are swapped out of the rail for fresh suggestions. A
      "Recently Viewed" rail with prev/next arrows appears beneath. There is no inline promo-code
      field -- discount codes are collected on the native checkout -- and there is no inline shipping
      estimate calculator; free-shipping eligibility is conveyed only via the qualifier banner and
      footnote.

## Search
```

```
Search is triggered by a magnifying-glass icon button in the header utility row. Clicking it expands
      an inline dropdown directly below the icon -- the page is not dimmed and navigation remains
      visible.

Predictive search renders a static "Suggested" panel inside the dropdown: typically four links
      combining one collection shortcut and three pre-set search-query links. The list is fixed and
      does not change with keystrokes. The dropdown itself does not render product thumbnails, prices,
      or product cards -- only suggested queries and collection shortcuts.

A `/search/suggest.json` predictive endpoint is active in the background and returns product-shaped
      JSON (title, handle, price, compare-at range, availability, URL, vendor, featured image, tags,
      type) but its results are not surfaced in the header dropdown. Submitting the form navigates to
      a full-page `/search?q=<query>` results listing. A second search input appears in the mobile
      off-canvas drawer with the same form action and no predictive panel.

## Floating UX

A third-party CMP renders a full-width cookie consent banner at the bottom of the viewport on first
      visit. The banner offers a privacy-policy link, "Allow All", "Reject All", and "Confirm My
      Choices" buttons. Granular preference controls cover four consent categories (analytics,
      marketing, preferences, sale of data). Choice is stored in localStorage and integrates with
      Google Consent Mode V2 for downstream tag gating.

A Klaviyo-powered newsletter popup is configured (loaded via async script). It is time- or
      session-gated and did not appear during cold-session exploration. The inline newsletter section
      in the page footer area remains the always-on capture surface.

No chat widget, no age gate, and no homepage modal popup were observed.

## Policy & info pages

Seven informational pages are exposed. Four are native policies under `/policies/<slug>` -- shipping,
      refund, privacy, and terms of service -- all sharing a minimal single-column template: H1 title,
      then rich text paragraphs, H2 sub-sections, and bullet lists, with no images or sidebars.

Three custom pages live under `/pages/<slug>`. The contact page uses a hero split layout with a
      headline, a one- to two-sentence body, and a CTA button deep-linking into an external
      customer-support portal (no inline form), followed by a three-card informational grid covering
      warranty, product registration, and 24/7 support. The FAQ page uses a custom anchored-section
      template: an H1, a horizontal in-page navigation bar with four anchors (Products and Usage,
      Orders, Contact, Warranty), and four H2 sections of accordion-style question/answer items. The
      about page is a custom editorial layout with a celebrity-chef hero quote, a brand-story section
      pairing copy with imagery, a five-row numbered feature list, a three-link "Shop" CTA row, and an
      inline email subscribe form with a hidden honeypot.
```

F.2 Capabilities (`capabilities.json`)


```
{
  "version": "0.1",
  "shop": {
    "descriptor": "Premium hybrid cookware and kitchen accessories shop",
    "category": "Cookware / Kitchen",
    "currency": "USD",
    "tone": [
      "premium",
      "bold",
      "chef-endorsed",
      "direct-to-consumer"
    ]
  },
  "site_shell": {
    "has_announcement_bar": true,
    "header_style": "two-row sticky",
    "has_mega_menu": false,
    "nav_depth": 2,
    "footer_groups": 3
  },
  "homepage": {
    "section_types": [
      "hero_video",
      "category_chip_slider",
      "featured_product_grid",
      "specialty_product_banner",
      "video_endorsement",
      "feature_cards_grid",
      "reviews_banner",
      "product_carousel",
      "product_hero_quote",
      "ugc_video_carousel",
```

```
      "category_banners_3up",
      "culinary_council",
      "recipe_carousel",
      "article_carousel",
      "founder_message",
      "newsletter_inline"
    ],
    "section_count": 16,
    "has_popup_modal": false
  },
  "collection": {
    "layout": "grid",
    "columns_desktop": 3,
    "filters": [
      "availability",
      "on_sale"
    ],
    "sort": [
      "featured",
      "best_selling",
      "alphabetically_az",
      "alphabetically_za",
      "price_asc",
      "price_desc",
      "date_new",
      "date_old"
    ],
    "pagination": "load_more"
  },
  "product": {
    "gallery_style": "horizontal_carousel_with_prev_next",
    "variant_selectors": [
      "radio_buttons"
    ],
    "has_quantity_selector": true,
    "description_layout": "accordion_tabs",
    "has_reviews": true,
    "has_recommendations": true,
    "has_personalization": false
  },
  "cart": {
    "type": "drawer",
    "has_promo_input": false,
    "has_upsells": true,
    "has_shipping_estimate": false
  },
  "search": {
    "trigger": "icon_button",
    "has_predictive": true,
    "predictive_types": [
      "suggested_queries",
      "collection_shortcuts"
    ],
    "results_layout": "fullpage"
  },
  "intl": {
    "has_locale_switcher": null,
    "has_currency_switcher": null
  },
  "floating": {
    "has_chat_widget": false,
    "has_age_gate": false,
    "has_cookie_banner": true,
    "has_newsletter_popup": true
  },
  "info_pages_present": [
    "shipping-policy",
    "refund-policy",
    "privacy-policy",
    "terms-of-service",
    "contact",
    "faq",
    "about-us"
  ]
}
```


F.3 Statistics (`stats.json`)


```
{
```

```
  "products_total": 164,
  "collections_total": 50,
  "products_per_collection": {
    "avg": 15.42,
    "median": 8.0,
    "max": 179
  },
  "price": {
    "min": 0.98,
    "max": 5499.0,
    "median": 59.99,
    "currency": "USD"
  },
  "products_with_variants_pct": 0.07317073170731707,
  "variant_axes_observed": [
    "title",
    "size"
  ],
  "navigation_depth_max": 2,
  "homepage_section_count": 16,
  "info_pages_count": 7,
  "feature_count": 48
}
```

G ShopArena Exploration Plan Example

This appendix reproduces the `plan.md` document emitted by the planner iteration of `shop_arena.explore` for the same exploration run shown in Appendix F. The plan enumerates the per-task work items the executor loop then consumes one task at a time; checkbox state (`[x]` / `[!]`) is updated by the executor at the end of each iteration.

G.1 Plan (`plan.md`)

```
# Plan -- Premium Hybrid Cookware Shop

A direct-to-consumer shop specializing in premium hybrid cookware, knives, kitchen tools, seasonings,
     and branded apparel, endorsed by a celebrity chef. Sells via USD pricing with US-focused
     shipping. The shop uses a two-row sticky header, flyout dropdown navigation, and a cart drawer
     architecture.

## Tasks

- [x] homepage_sections -- navigate to `/`; capture hero section, each below-hero section (category
     grid, endorsement block, features section, best-sellers carousel, social-proof/testimonials,
     specialty-product banners, recipe carousel), note any newsletter popup behavior; full-page
     screenshot + per-section evidence [priority: 9]
- [x] cart_drawer -- add 1 product to cart; capture drawer in empty state, filled state, qty-change
     state; check for promo-code input, upsell rail, and shipping-estimate widget [priority: 8]
- [x] header_navigation -- snapshot main nav in closed state, then trigger each top-level flyout
     button (8 categories); capture one open flyout; record nav depth and sub-link count per category
     [priority: 7]
- [x] search_predictive -- open search modal; type 2 short prefixes ("s" and "pan"); capture
     suggestion panel showing products; capture one network request via network.jsonl [priority: 7]
- [x] collection_filters -- visit the largest cookware collection; exercise "Available" and "On Sale"
     filter checkboxes; test all sort options; scroll to bottom to confirm pagination behavior;
     screenshot grid and record column count [priority: 6]
- [x] product_variants -- visit 2 PDPs: one apparel product (has size radio buttons) and one cookware
     set (single-variant); capture gallery carousel states (prev/next), variant selector states, qty
     stepper, description layout, reviews widget (Okendo), and "Customers also love" recommendations
     rail [priority: 5]
- [x] footer -- screenshot full footer; record link groups, social icons, and bottom-bar
     copyright/legal links [priority: 4]
- [x] info_pages -- visit `/policies/shipping-policy`, `/policies/refund-policy`,
     `/policies/privacy-policy`, `/policies/terms-of-service`, `/pages/contact`, `/pages/faq`,
     `/pages/about-us`; one screenshot each; note page layout type (policy template vs custom page)
     [priority: 3]
- [x] floating_widgets -- wait for and capture Klaviyo newsletter popup; inspect Crisp chat widget
     launcher; check OneTrust cookie banner presence; one screenshot per widget if visible [priority:
     1]

## Omitted Areas

- mega_menu -- nav uses flyout dropdown panels per category (single-column sub-link lists), not a
     full-grid mega menu layout; observed in homepage snapshot
- intl_switchers -- only `en_US` locale present; no locale or currency switcher visible in header,
     footer, or homepage snapshot; `index.html` locale metadata is `en_US` only
- age_gate -- not observed on `/` homepage snapshot or in `index.html` HTML structure
```

H ShopGuru Skill Catalog

This appendix accompanies §2.2.1 and summarizes the seven skill categories generated by ShopGuru.

Table 1: ShopGuru's seven skill categories grouped by horizon. Short-horizon tasks are produced by deterministic, pure-function generators over ShopArena's extracted JSON; long-horizon tasks are LLM-authored from few-shot examples reflecting human-shopper behaviors and reconciled against the same extracted data via a validator-driven polish loop (§2.2.3).

Group	**Bundle**	**Source data**	**Skill primitives**
Discovery	short-horizon	`products.json`	exact title search, substitute search
Filter-Selection	short-horizon	`collections.json` + per-collection options	collection browse, collection filter
Information-Seeking	short-horizon	`pages.json` (+ default policy fallback)	shipping policy lookup, returns policy lookup
E2E shopping journey	long-horizon	all of the above + few-shot human-trace examples	LLM-authored multi-step intents

I Examples of ShopGuru tasks

```
{
   "id": "mock_clothing-filter-1",
   "type": "shopping",
   "intent": "Navigate to the \"Frost Season Collection\" collection on this store. Find and use the
        Color filter (e.g. Black) to select an option. If products are shown after filtering, select
        any variant of a product and add it to cart. It is ok if no products match the filter; using
        filter correctly means the navigation task is completed successfully.",
   "success_criteria": {
     "url_contains": "/collections/frost-season-collection",
     "type": "navigation"
   },
}
```

Listing 1: Example collection-filter task emitted by the deterministic generator. `url_contains` is a soft URL hint to the LLM judge, not a hard pass/fail gate.

```
{
   "id": "mock_hardware-returns-1",
   "type": "navigation",
   "intent": "Find the returns and refund policy page on this store. Look for return policy
        information in the footer, menu, or any navigation links. Read the refund policy details,
        then leave the site.",
   "success_criteria": {
     "url_contains": "/policies/refund-policy",
     "type": "page_navigation"
   },
   "url_contains_alt": [
     "/pages/return-policy"
   ],
}
```

Listing 2: Example page-navigation task to find specific policy on the shop, also emitted by the deterministic generator.

```
{
   "id": "mock_cookware-e2e-v1-3",
   "type": "shopping",
   "intent": "Shopping for a new chef's knife on a budget. Navigate to the Knives collection. Apply a
        filter for Color if available, or sort the collection by Price: Low to High. Browse the
        filtered or sorted results and pick any Chef's Knife. Open its page, choose whichever Color
        variant you like best (e.g., Black Handle or Natural Wood Handle), and add it to cart. Once
        the knife is in your cart, end the session. Do not click any Checkout button.",
   "success_criteria": {
     "url_contains": "/collections/knives",
     "type": "cart_after_filter_or_sort"
   }
}
```

Listing 3: Example LLM-authored long-horizon journey that chains a collection navigation, a Color filter, a price sort option, and a final cart action.

```
{
  "id": "mock_clothing-e2e-v1-1",
   "type": "shopping",
   "intent": "First-time visitor exploring the brand's ethos. Open the About Us page from the footer
        or main menu to read up on Mock Clothing's lifestyle positioning. Next, navigate to the Fresh
        Drops collection to see what's newly available. Pick any accessory (like a scrunchie or cap)
        that catches your eye, open its product page, select any available Color and Size variants,
        and add it to your cart. Once the item is in your cart, end the session. Do not click any
        Checkout button.",
   "success_criteria": {
     "url_contains": "/pages/about",
     "type": "cart_after_about_page_detour"
   },
}
```

Listing 4: Another example LLM-authored long-horizon task consisting of a shopping journey after a page navigation detour.

J ShopGuru Polish Loop

This appendix accompanies §2.2.3 and illustrates the LLM-authored E2E polish loop.

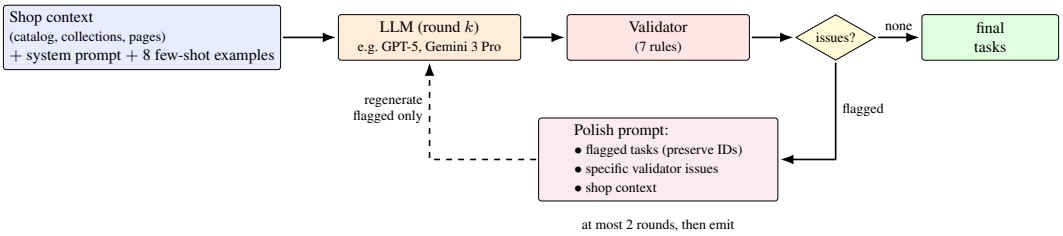


Figure 10: LLM-authored E2E polish loop. The same validator that audits deterministic generators (§2.2.4) drives an iterative refinement of LLM-authored tasks. The round-k output is audited; if any tasks are flagged, the polish prompt bundles *only the flagged tasks* with their specific validator issues and the shop context, the LLM regenerates only those tasks while preserving their IDs, and the merged result is re-audited. The loop terminates after at most two rounds; tasks that still fail validation after two rounds are surfaced as a release blocker rather than silently shipped.

K Verifiers for ShopGuru pipeline

This appendix accompanies §2.2.4 and lists the rule set used by ShopGuru's post-generation validator. The validator runs as the final gate on every emitted benchmark file: `error`-severity findings halt the build with a non-zero exit code, while `warning`-severity findings are logged and either fed back to the LLM-authored journey generator as polish-loop signals (rules marked † in Table 2) or surfaced as advisory hints for the human curator. See §2.2.4 for the narrative description and §2.2.3 for the polish-loop integration.

Table 2: ShopGuru's seven core validator rules. The validator returns a list of `Issue` records rather than raising, so callers can decide whether to error or just warn. Errors halt the build with a non-zero exit code. Rules marked with † are additionally fed back to the LLM-authored journey generator as polish-loop signals (§2.2.3); the unmarked warning is advisory.

Rule	Severity	Catches
`unknown-collection`†	error	`url_contains` references a collection handle absent from `collections.json`.
`unknown-product`†	error	Same, for `/products/`⟨`handle`⟩.
`infeasible-filter`†	error	Filter (*dim*, *value*) is not realized by any product in the targeted collection.
`intent-answer-leak`†	error	A `success_criteria.response_contains` value appears verbatim in the intent — an agent could pass by echoing the prompt without visiting the page.
`option-mismatch`†	warning	The intent asks the agent to "select a ⟨Option⟩ variant" for a product that has no such option (e.g. "select Color" on a gift card whose only option is denomination).
`product-not-in-collection`†	warning	The intent pairs a product with a collection that does not contain it.
`unknown-page`	warning	Same as `unknown-collection` but for `/pages/`⟨`handle`⟩ (softer because some storefronts surface policies via `/policies/*` fallthrough routes).

L ShopGuru E2E Prompt Templates

This appendix accompanies §2.2.3 and reproduces (i) the system prompt used by the LLM-authored long-horizon journey generator, (ii) the user prompt template skeleton (placeholders only), and (iii) three of the eight few-shot examples that ground the LLM in human-shopper behavior. The selected examples are the ones that best span the behavior categories enumerated in Table 1: *Multi-pet household shopping* (multi-product cart with a quantity edit), *Free-shipping threshold calculation* (cart-state arithmetic with a localized banner), and *Filter that returns zero results, then recover* (filter-recover). The remaining five few-shot examples are available in the released benchmark code. All prompt text is reproduced verbatim modulo ASCII-folding of a small number of non-ASCII glyphs (→, currency symbols, accented Lithuanian characters).

L.1 System Prompt

```
You are an expert at designing evaluation tasks for e-commerce web agents.
Your job is to author realistic end-to-end shopping journeys that mirror how
real human buyers actually browse and transact on online stores.

Hard rules:
- Every task must be 100% grounded in the provided shop data. Do not invent
  products, collections, page titles, or variant options that are not in
  the data dump below.
- A product mentioned in a task must exist in the catalog (Title + handle).
- A collection mentioned in a task must exist (Title + handle).
- If you ask the agent to "select a Color variant" or "select a Size",
  the named product MUST have that variant option AND values. Otherwise
  use generic phrasing like "select any variant" or simply "add it to cart".
- A product/collection pairing in the same task (e.g. "Navigate to X
  collection and add product Y") is only valid if Y is actually a member
  of X according to the Product-to-Collection Mappings section.
- Every task must end in a well-defined state the agent can reach: a
  specific set of items in the cart, a specific URL visited for a
  navigation task. Never "and then browse around".
- **Never instruct the agent to click Checkout, proceed to checkout, head
  to a checkout page, abandon checkout, or otherwise traverse the cart ->
  checkout boundary.** Checkout is intentionally disabled on the sandbox
  shop and the LLM judge grades tasks on the **final cart state only**.
  End shopping tasks with wording like "Once the product is in your cart,
  end the session. Do not click any Checkout button." For tasks involving
  cart edits (add -> remove, quantity change), describe the expected final
  cart explicitly.
- Intents should be second-person, imperative. Shopping tasks should be
  one or two short paragraphs; they may include natural detours (policy
  lookup, About page, brand comparison) before the cart-state terminal.
- Avoid tasks that require user authentication, payment entry, or any
  externally-gated flow.
- Do not produce tasks that require human judgment calls the agent cannot
  verify (e.g., "pick the most stylish product").

Quality bar (this is what separates a great task from a basic one):
- Open with a one-sentence persona or motivation that frames the journey
  ("First-time visitor.", "You're a returns-cautious shopper.",
  "Multi-pet household shopping.", "Sales hunting.").
- Reference specific storefront UI elements (homepage banner, top menu,
  footer Quick Links, brand menu) by name when they exist.
- Mix at least 2-3 distinct skills per task (search + compare + cart edit;
  policy lookup + nav drilldown + add). Single-step "search and add"
  tasks are too easy and should be the minority.
- Use exact brand/product/collection titles as written in the data --
  never paraphrase ("Fresh Drops" not "new arrivals", "Womens Run Club"
  not "running gear").

Output format: a JSON object with a single key "tasks" containing an array
of task objects, each with fields:
  id (slug-friendly string),
  type ("shopping" or "navigation"),
  intent (string),
  success_criteria (object with url_contains and a descriptive type field).
Do not include the `url` field -- that will be filled in downstream.
```

L.2 User Prompt Template Skeleton

The user prompt is built per shop by substituting the placeholders below with concrete data extracted by ShopArena. Sections marked `{placeholder}` are filled in at generation time from `products.json`, `collections.json`, `stats.json`, and `pages.json`.

```
Author {count} end-to-end evaluation tasks for the following storefront.
Follow the system prompt's schema and quality bar. Cover as many of the
behavior categories below as this store supports, and skip any that are
not applicable to this store.

### Shop profile
Name: {store_name}
Description: {store_description}
Country: {country}, Currency: {currency}, Language: {language}
Domain: {real_url}

### Top collections (title -- handle)
{collections_list}

### Top product types
{product_types_list}

### Example products (title -- handle, with REAL variant options & values)
{products_list}

### Product-to-Collection memberships (use these to ground multi-step tasks)
{mapping_str}

### Collection facets / option patterns (for filter tasks)
{option_patterns_list}

### Policy / info pages (title -- /pages/handle)
{pages_list}

### Gift card products
{gift_card_list_or_none}

### Behavior categories to cover (aim for at least 10 of 14)
1. Search & atomic add-to-cart
2. Nav drilldown (menu -> sub-menu -> collection -> product)
3. Filter + sort (e.g. Format=Hardcover then sort by price)
4. Filter that returns zero results, then recover
5. Substitute-match discovery (intended product missing -> close alternative)
6. Review / detail read on a product page
7. Size chart / fit guide lookup
8. Shipping policy lookup (with cart action after)
9. Returns / refunds lookup (with cart action after)
10. Gift card purchase (only if the shop sells gift cards)
11. Multi-product cart with edit (add A, add B, remove A, set qty 2 on B)
12. Cross-collection or cross-brand comparison (compare A and B, pick one)
13. Contact / store locator / about page
14. Free-shipping threshold or sale-discount calculation (if banner exists)

### High-quality reference tasks (from other shops in this benchmark suite)
[See Appendix sections below for three representative few-shot examples.
Five additional examples are included in the released benchmark code.]

### Anti-patterns to AVOID
- "Search for X. Pick a Color and Size variant. Add to cart." -- too thin,
  no persona, no detour, single skill.
- Mentioning "Color" or "Size" for a product whose options list above does
  NOT include that name.
- Generic UI references like "navigate to the homepage" without naming a
  specific section or CTA.
- Two products in one intent that are not co-located in any collection
  according to the membership list above.

When constructing tasks, USE EXACT product titles, collection titles, and
option names from the lists above. Use the storefront's own wording for
collection names ("Fresh Drops" not "new arrivals"). For multi-step tasks,
ensure each (collection, product) pair you mention is a real membership.

Respond with a JSON object with a "tasks" key containing the array of
{count} tasks. Each `id` should use the format
"{shop_slug}-e2e-v1-{index}" replacing {index} with numbers starting at 1.
```

L.3 Few-Shot Example A: Multi-Pet Household Shopping

A multi-product cart with a per-item quantity edit and a cross-collection nav. Exercises behavior categories 2 (Nav drilldown), 11 (Multi-product cart with edit), and the cart-state-only termination rule.

```
intent: "Multi-pet household shopping. Navigate to the Dog Kibble
collection via the Dogs menu and add any kibble product to cart.
Navigate to Cats -> Litter from the top menu and open Worlds Best Cat
Litter. Select any size variant (for example Original 28lb) and add it
to cart. Open the full cart page and increase the cat litter quantity
from 1 to 2. Once the cart contains one dog kibble and two units of
cat litter, end the session. Do not click any Checkout button."
success_criteria.type: "cart_multi_pet_with_quantity_edit"
```

L.4 Few-Shot Example B: Free-Shipping Threshold Calculation

Cart-state arithmetic over a localized homepage banner. Exercises behavior category 14 (Free-shipping threshold) and demonstrates that the prompt is robust to non-English storefronts (here, Lithuanian).

```
intent: "Hit the free-shipping threshold. Note the 'Nemokamas
pristatymas nuo 49 EUR' banner on the homepage. Navigate to Sunims ->
Skanestai and add a treat product under EUR 25 to the cart. Open the
cart page, observe how much more you need to reach EUR 49 for free
shipping. Go back to the store, navigate to the same Skanestai
category, and add a second product that pushes the total over EUR 49.
Return to cart, confirm the free-shipping banner has updated. Once
both products are in your cart with a total over EUR 49, end the
session. Do not click any Checkout button."
success_criteria.type: "cart_after_free_shipping_threshold"
```

L.5 Few-Shot Example C: Filter-Recover

A filter that yields zero or sparse results, followed by a recovery strategy. Exercises behavior category 4 (Filter that returns zero results, then recover) and demonstrates handling of edge cases that deterministic generators cannot synthesize on their own.

```
intent: "Navigate to the Christmas & Hanukkah collection via the Shop
by Holidays & Events menu. Apply a Size filter for '3-6 Months';
results will likely be sparse or empty. Observe the empty or filtered
state, then clear the Size filter and apply a different one such as
Color = Blue. If products appear, open the first one, select any
variant, and add it to cart; if still empty, remove all filters and
pick any product from the unfiltered list. Once a product is in your
cart, end the session. Do not click any Checkout button."
success_criteria.type: "cart_after_filter_recover"
```