Sari Sandbox: A Virtual Retail Store Environment for Embodied AI Agents

Supplementary Material

S1. Texture fidelity

As discussed in Sec. 4.2, we evaluated texture fidelity using PaddleOCR [37]. While it performs well on structured product labels, it exhibits limitations in more challenging scenarios. Figure S1 and Figure S2 provide qualitative examples where OCR accuracy declines. Figure S1 presents a case involving mixed horizontal and vertical text orientations, which often result in incorrect segmentation or recognition. Figure S2 highlights the challenges posed by stylized fonts and logo-like text, where decorative design elements interfere with accurate character detection. These examples encourages OCR-aware scene design and the potential need for active viewpoint control in embodied agent pipelines.

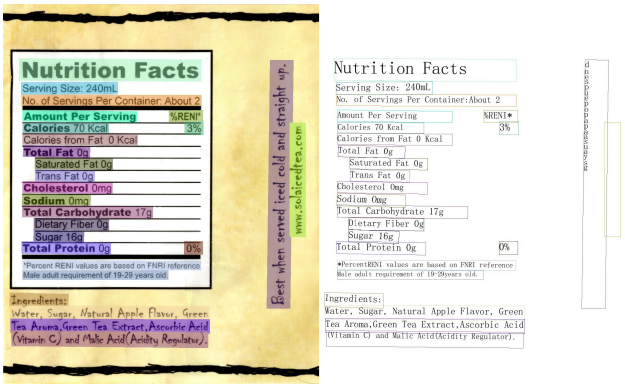

Figure S1. PaddleOCR failed in identifying the rotated text because majority is horizontal text.

S2. Embodied agent

An agent is an entity that perceives its environment through sensors and acts upon it using actuators [40]. As illustrated in Figure S3, the agent, a modular system, operates within its environment (sandbox) through a continuous loop of perception, cognition, and action to achieve a set goal. Upon goal completion, it halts until a human provides a new directive. This paradigm is powerful as it shifts from static programming to a dynamic, interactive model where systems are both observers and actors.

For our agent's cognitive engine, we required a VLM instead of an LLM, incorporating visual sensory inputs. We based our selection on the Massive Multi-discipline Multimodal Understanding and Reasoning (MMMU) [41]. At the time of evaluation, Gemini 2.5 Pro was state-of-the-art, achieving a score of 84.0 on the MMMU validation set (using its experimental Deep Think mode), nearing the human expert benchmark of 88.6 [42].

Figure S2. PaddleOCR failed in identifying stylized text of the Moby Brand despite being able to identify texts in the same image.

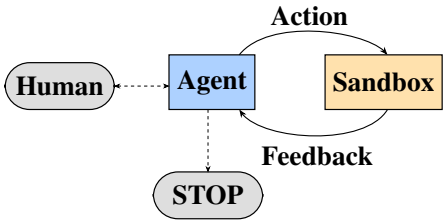

Figure S3. The elementary concept of an autonomous agent.

In our sandbox, the agent faces two core challenges: autonomous navigation and object manipulation, both vital for product search and retrieval. We adapted the ReAct framework for our needs. To focus on core capabilities, we simplified the embodied agent's navigation and manipulation; complex behaviors like multi-item handling or checkout were excluded. We forego A* planning [43] as our embodied agent lacks access to a pre-built grid-map, relying solely on visual input during navigation. Our primary objective is not to develop a state-of-the-art agent, but rather to construct a functional one that rigorously tests the practicality and usability of the APIs designed in Section 3.6 through basic item search and retrieval tasks.

Our embodied agent operates using the pattern in Figure S4, which is designed for both efficiency and control when processing a grocery task (e.g., *"Find a healthy snack"*). Our key approach here is that the embodied

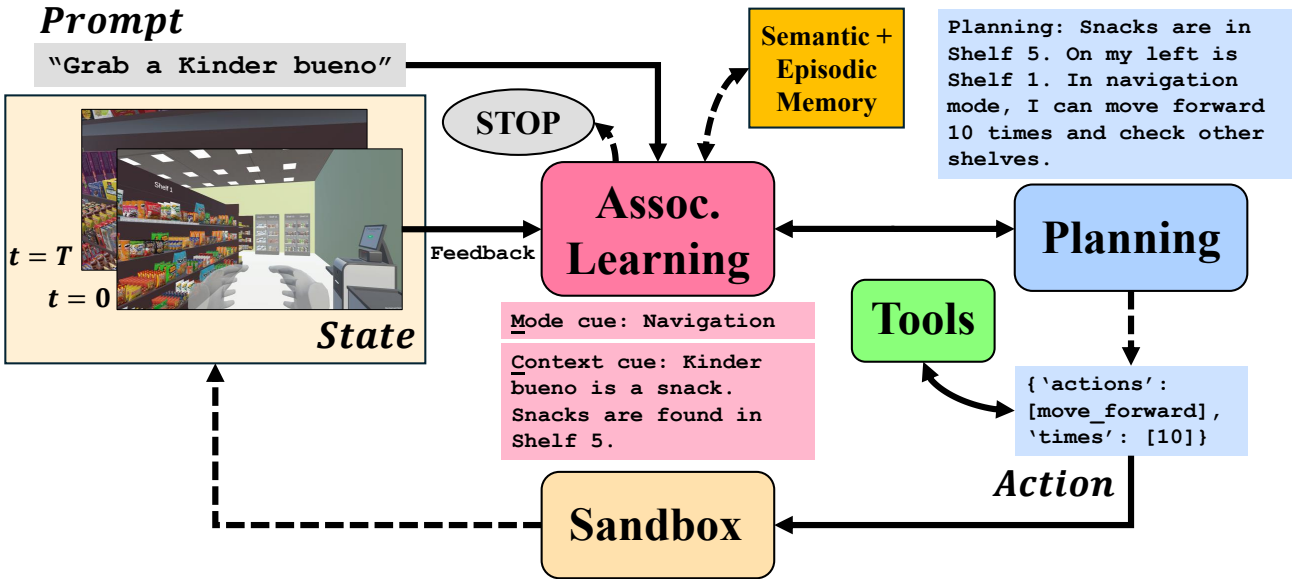

Figure S4. An overview of our agentic pattern. The process begins with a prompt, initiating a continuous loop. In the initial **associative learning** step, the embodied agent consults its semantic and episodic memory, and synthesizes its current state and the given task into mode and context (M+C) cues. These cues are included as inputs for the **planning** step, where the VLM generates an action sequence. This sequence, comprising elementary actions and potential tool calls, is executed within the sandbox. The loop then repeats with the new state, continuing until the associative learning step issues a `STOP` command.

agent generates an action sequence—a chunk of related commands—rather than a single action per cycle. This approach is highly effective for repetitive tasks such as grocery shopping (e.g., *"navigate to Shelf 1, search; navigate to Shelf 2, search; navigate to Shelf 3, grasp"*). This pattern effectively manages such decomposed, sequential steps. Furthermore, this design substantially offloads the cognitive load from the VLM. The associative learning step pre-processes and provides mode cues (e.g., indicating navigation or manipulation) and context cues. This means the planning step does not need to deliberate on these operational states, streamlining its decision-making. Consequently, this pattern makes debugging easier and provides more robust control over the outputs.

The agentic loop operates through three key steps. First, the **associative learning step** queries the sandbox for the embodied agent's current state (coordinates, visual input) at each step. It then processes this state against the prompt to generate two cues for the next stage: a **mode cue** and a **context cue**. The mode cue switches the embodied agent's operational state between navigation and manipulation, constraining the VLM to specific symbolic actions for improved reliability. The context cue provides recall information from the embodied agent's semantic and episodic memory. Second, **planning step** takes this state information, along with the mode cue, context cue, and available tools (function-calling APIs), and passes them to the VLM, which generates a structured action sequence. Finally, during the **execution step**, an external parser translates this sequence into executable API calls dispatched to the sandbox, yielding an observable result. The embodied agent perceives this new state, and the loop repeats until the associative learning step issues a `STOP` command.

To overcome the stateless nature inherent in VLM-powered embodied agents, our embodied agent explicitly models memory, leveraging the four-part cognitive architecture. This framework, managed within our agentic pattern, comprises the following components: **procedural memory** which contains the embodied agent's core rules and skills, implemented as the system instructions provided to the VLM; **working memory** which contains the immediate interaction history, managed by caching and injecting context within the VLM's context window; **semantic memory** which contains the factual knowledge about the environment; and **episodic memory** which contains the distilled takeaways from the embodied agent's own experiences. To implement the semantic memory, we first created a base semantic memory which is a text file containing the store layout, product locations (e.g., *"Shelf 1 contains cereals"*), and rudimentary directions from the embodied agent's spawn point to any shelf (mimicking natural instructions). Similarly, the episodic memory is implemented as a text file, which starts blank at the beginning of each task.

The semantic and episodic memories are managed by a memory writing operation that occurs during the associative learning step. At each timestep, this module consults

both memory files. It uses the semantic memory to ground the embodied agent in its environment. After action execution, the associative learning step updates the episodic memory by synthesizing a three-point reflection on the action that had just been executed: a dense summary of what occurred, what actions worked, and what to avoid in the future. The key information recalled from both memories is then encoded into the context and mode cues, and passed to the planning step to reduce the cognitive load and enhance context. It is important to note that while the associative learning and planning steps use the same Gemini model release version, they function as distinct modules with different system instructions.

S3. Actions and tools

Our embodied agent interacts with the Sari Sandbox environment through a defined set of actions, categorized into distinct operational modes: navigation and manipulation (Table S1). These actions enable precise control over the embodied agent's movement and interaction with objects within the simulated grocery store.

The navigation mode allows the agent to control its body's position and orientation. This includes fundamental actions such as `move_forward`, which advances the agent by 0.1 units, and `pan_left` and `pan_right`, which rotate the agent's view horizontally by 2.5-degree increments. While these appear as single, high-level API calls, their underlying implementation involves iterative, atomic calls to the simulator's core API functions. For instance, `move_forward` directly invokes `TransformAgent((0, 0, 0.1), (0, 0, 0))`, allowing for controlled, fine-grained movement up to a predefined unit limit. Similarly, `pan_left` and `pan_right` are built upon `TransformAgent((0, 0, 0), (0, -2.5, 0))` and `TransformAgent((0, 0, 0), (0, +2.5, 0))`, respectively, to control the agent's yaw rotation.

Table S1. Different modes of operation and their associated actions. Action descriptions: `move_forward` to move the embodied agent forward by 0.1 units in the sandbox. `pan_left` and `pan_right` to pan left and right by 2.5 degrees, respectively. `center_object_on_screen` to center the embodied agent's body on the target object in the frame. `retrieve_item` to approach the target object, grab it with the embodied agent's hand, and inspect it. Navigation and manipulation invokves `TransformAgent` and `TransformHands`, respectively.

Mode	Actions
Navigation	`move_forward`, `pan_left`, `pan_right`
Manipulation	`center_object_on_screen`, `retrieve_item`

The manipulation mode enables the agent to interact directly with items in the environment. This includes actions like `center_object_on_screen` and `retrieve_item`.

The `center_object_on_screen` action leverages Gemini 2.5 Pro for object detection. It uses visual inputs (obtained via first-person point-of-view screenshot within the sandbox) to calculate the target object's bounding box. The VLM's perception output after calling `loc_object` tool, providing `ymin`, `xmin`, `ymax`, `xmax` coordinates, is then translated into pixel coordinates. Based on the object's horizontal and vertical deviation from the screen center, the agent directly invokes `TransformAgent` to perform precise yaw and pitch rotations, aligning its perspective with the object.

The `retrieve_item` action is a more complex, compound behavior that orchestrates several steps:

- **Depth estimation**. The agent first moves generally towards the detected target using visual input and estimated depth via `est_depth` tool. This often involves `move_forward` actions, which, as described, translate to repeated `TransformAgent((0, 0, 0.1), (0, 0, 0))` calls.
- **Fine-tuning orientation**. It then adjusts its orientation to face a cardinal direction for consistent alignment, again utilizing `TransformAgent` for precise yaw control.
- **Horizontal centering**. The embodied agent performs a `strafe_to_center` operation to precisely align itself with the object. This action calculates the horizontal offset of the target object's bounding box from the image center. It then converts this pixel offset into a required linear movement in world units. Subsequently, `strafe_to_center` executes a series of granular calls which correspond to `TransformAgent((+0.1, 0, 0), (0, 0, 0))` or `TransformAgent((-0.1, 0, 0), (0, 0, 0))` to incrementally shift the embodied agent's body sideways until the object is horizontally centered in its view.
- **Final approach and interaction**. The embodied agent moves to the item's immediate vicinity and executes the physical `grab_and_read` operation. This critical step involves a choreographed series of atomic hand movements directly implemented via the `TransformHands` API. Specifically, the agent can extend its hands forward by adjusting their Z-axis position, pull them backward for Z-axis retraction, or raise and lower them to control their Y-axis position. Rotational actions, also achieved by `TransformHands`, allow for precise manipulation of the hand's yaw rotation. Once positioned, the agent can grasp the item using `ToggleLeftGrip` API calls to simulate a grip. Following the grab, the `grab_and_read_item` operation initiates a new screenshot and then processes this image using an Opti-

cal Character Recognition (OCR) tool via `ocr_object` to extract any text present on the item, thereby simulating visual inspection. These primitive hand and vision-based interactions are crucial for the embodied agent to accurately reach, grasp, and inspect the target item, even though they are not exposed as high-level API actions in the table.

The VLM serves as the cognitive engine for these actions. It processes visual information and textual prompts to determine the appropriate sequence of actions and their parameters. Tools found in Table S2 are integral to the embodied agent's perception and decision-making loop, particularly for tasks requiring object identification and precise interaction. The structured nature of these higher-level actions, built upon the fundamental `TransformAgent` and `TransformHands` APIs, ensures the embodied agent can perform complex grocery tasks by decomposing them into manageable, executable steps.

Table S2. Tools that are part of an elementary action and their corresponding purposes.

Tool	**Purpose**
`loc_object`	Uses Gemini 2.5 Pro's object localization capability to output bounding box coordinates of a specified item or item of interest. Format: `[ymin, xmin, ymax, xmax]`. Part of: `center_object_on_screen`.
`ocr_object`	Uses PaddleOCR [37] for item inspection via optical character recognition (OCR). Part of: `retrieve_item`.
`est_depth`	Uses Depth-Anything-V2 [44] Small for computing the distance between target object and embodied agent before item inspection. Part of: `retrieve_item`.