NAVGPT-3: HARNESSING CONTEXT IN A HIERARCHICAL NAVIGATION RUNTIME

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models trained with long-horizon agentic reinforcement learning can generalize knowledge through reasoning, express precise actions, and pursue goals over many steps, raising the ceiling on what an embodied agent can understand and decide. Physical interaction, however, remains the domain of action policies, which provide dense, low-latency control. We present NAVGPT-3, a harness that connects the two models, with an OS-like runtime built above it: reasoning, acting, and monitoring run as threads with their own context, tools, and permissions, while the runtime schedules them and decides which thread controls the robot's motion, so that the robot can react to sudden real-world events through interruption and thread switching. Beneath it, our action policy NavGPT VLA, trained on 19.28M examples, allocates visual tokens using codec allocation, in proportion to scene change; its 8B model alone reaches 74.51 SR on R2R-CE and leads RxR-CE with 78.19 SR. With the complete harness, NAVGPT-3 sets the state of the art on R2R-CE (81.51 SR) and, for the first time, brings an autonomous agent to human level: on RxR-CE it matches human followers in success (90.43 vs. 90.4 SR) and path fidelity (78.47 vs. 77.7 nDTW) at 1 min 22 s per episode, versus roughly 3 min for a human. We comprehensively ablate the harness design and the interaction between the two models, showing how tools and the action policy shape the path from language-model reasoning to physical control: when NavGPT VLA executes the route, the reasoning loop shortens and the system's minimum reaction time falls from 3–19 s per language-model decision to 0.5–1 s per action-policy step (1–2 Hz). These results show that designing this embodied interface is central to connecting frontier language-model intelligence with low-level physical control. We will release all models, code, and evaluation records.

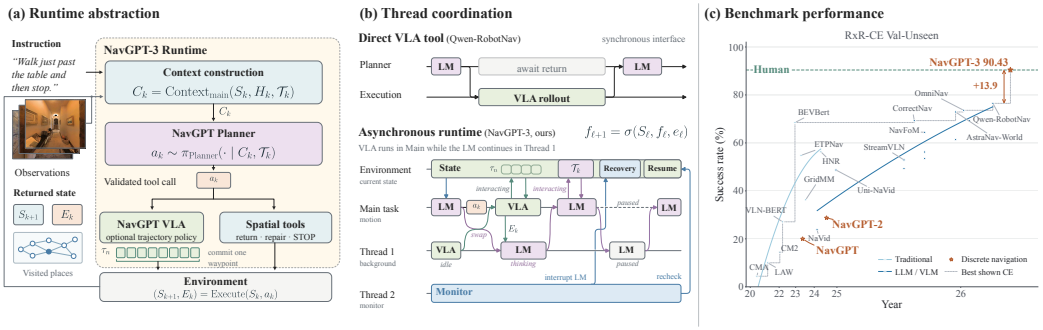


Figure 1: Overview of NAVGPT-3: (a) runtime abstraction with the planner, VLA, and spatial tools; (b) synchronous tool use versus asynchronous thread coordination; and (c) benchmark performance. **NAVGPT-3 surpasses human performance on RxR-CE**.

1 INTRODUCTION

Language models trained with long-horizon agentic reinforcement learning generalize knowledge through reasoning, express actions as tool calls and code, and pursue goals over many steps (Ope-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

nAI, 2026; Anthropic, 2026b;a; Qwen Team, 2026a). In Vision-and-Language Navigation (VLN), successive generations of language models absorb more of what navigation systems once built as separate modules: commonsense grounding, spatial description, progress estimation, and long-horizon tool use. Execution is the exception. It runs on the world's clock rather than the model's, and control must proceed at a rate that step-by-step reasoning cannot sustain. The world also does not pause while a model thinks: observations arrive during inference, motion continues past the decision that began it, and a hazard may demand a stop before any reasoning finishes. An action policy is also far smaller and weaker at following instructions than the model that plans, so it drifts over a long rollout and must report back evidence the reasoner can check. Existing systems address part of this by combining reasoning and execution hierarchically, with a planner issuing subgoals to a learned controller (Gao et al., 2025; Hu et al., 2026a; Chiang et al., 2024; Huang et al., 2026b), and recent agents add graph memory and backtracking (Liu et al., 2026) or verification and recovery skills (Wang et al., 2026).

Connecting these abilities to physical control therefore requires a complete interaction loop: grounded context for deciding, spatial tools for acting, persistent state for remembering, and execution evidence for revising a route. A *harness* shapes model behavior through this loop, defining the context models receive, the tools they can call, and the state and feedback retained across calls (Zhong & Zhu, 2026; DeepSeek-AI, 2026). Real robots add a coordination requirement: reasoning, execution, and monitoring must be able to progress asynchronously. A *runtime* above the harness manages these activities as threads and controls which thread may move the robot.

We present NAVGPT-3, which combines a complete navigation harness with an OS-like runtime. The harness brings context management, spatial tools, persistent route state, and model execution into one embodied interface. The Planner can act through those tools or delegate execution of a navigation instruction to NavGPT VLA. Returned route images, execution feedback, an observed map, and addressable visited places make the resulting rollout inspectable and revisable through return, local repair, renewed delegation, and explicit semantic termination. We developed these harness interfaces through recursive self-improvement (RSI), using automated research iterations over tool definitions and visual prompts (Section D.3). The runtime builds on these interfaces to schedule threads and control *motion authority*: review can overlap execution, and monitoring can interrupt motion without waiting for reasoning (Figure 2). Tool-capability and visual-context studies examine the harness design alongside Planner–VLA interaction (Sections 4.1 and 4.2). Beneath the harness, NavGPT VLA predicts eight waypoints from the instruction and visual history. Because the visual context is bounded, we use codec allocation from video coding: the first observation of each view is a full reference (I-frame), and later observations (P-frames), receive visual tokens in proportion to how much they change (Tang et al., 2026; An et al., 2026).

Across benchmarks, the 8B NavGPT VLA alone leads published methods on RxR-CE (78.19 SR); under the runtime, NAVGPT-3 sets the state of the art on R2R-CE (81.51 SR) and, on RxR-CE, matches human followers in both success (90.43 vs. 90.4 SR) and path fidelity (78.47 vs. 77.7 nDTW). The same runtime transfers to object-goal navigation and embodied question answering (Tables 3 to 5). Ablating the harness tools and the VLA interaction shows how this interface governs the path from reasoning to control: given good tools, a strong language model navigates well by itself but decides only once every 3–19 s, whereas letting the VLA execute the route lowers the system's minimum reaction time to 0.5–1 s per step (1–2 Hz; Table 1).

2 RELATED WORK

Agentic Vision-and-Language Navigation with LLMs. VLN requires an agent to follow natural-language instructions in unseen environments, first on discrete navigation graphs (Anderson et al., 2018; Ku et al., 2020) and then in continuous environments (Krantz et al., 2020). NavGPT initiated the line of work asking how much of navigation a LLM can carry on its own (Zhou et al., 2024b): given observations in text, the language model reasons about instruction progress and decides each move zero-shot, without any training on action data. Later navigators extend this recipe with maps, discussion, and open-vocabulary grounding (Chen et al., 2024; Pan et al., 2023; Long et al., 2023; Qiao et al., 2025), and recent agents let the model act on the embodiment directly through code (Zhou et al., 2026b). A parallel line of work targets physical execution, tuning Vision-Language Models (VLMs) into VLA policies that provide dense, low-latency control while retaining

the generalization of the VLM (Zhang et al., 2024; 2025b; Cheng et al., 2025; Wei et al., 2025b;a; InternVLA-N1 Team, 2025; Zhang et al., 2025a; AMAP CV Lab, 2026; Qwen Team, 2026b; Mistral AI, 2026). NavGPT-2 pursues both at once, emphasizing the role of reasoning in language models while training a policy that acts faster and more accurately, with the language and action heads decoupled over one shared representation (Zhou et al., 2024a). However, such training cannot perfectly align the reasoning and the actions. NavGPT-3 combines the two at the system level, connecting a reasoning model and a trained VLA policy through a harness with an OS-like runtime built above it.

Language models over learned policies. Hierarchical systems pair a planning model with a learned controller in navigation (Chiang et al., 2024; Huang et al., 2026b; Liu et al., 2026) and manipulation (Gao et al., 2025; Hu et al., 2026a; Guo et al., 2026; Li et al., 2026; Yang et al., 2026; Wang et al., 2026), and harness engineering treats context, tools, and state as the unit of design (Zhong & Zhu, 2026; DeepSeek-AI, 2026). These systems already cover graph memory, backtracking, verification, and recovery. The closest system, NavMCP, retains trajectory evidence across calls to a navigation foundation model in order to answer questions (Lei et al., 2026). NavGPT-3 instead uses returned evidence to revise an instruction-following route: after a delegated VLA rollout, the Planner can inspect the route, return to any visited place, repair locally, and decide completion.

Visual context in navigation policies. Navigation policies must fit long, multi-view observation histories into a bounded visual budget. Existing policies compress history by merging or pruning visual tokens (Zhang et al., 2025b; Wei et al., 2025b) or distribute a fixed budget across time and views (Qwen Team, 2026b). NavGPT VLA allocates tokens in proportion to scene change, as codec-aligned video encoders do (Tang et al., 2026; An et al., 2026).

3 Method: Harness, Runtime, and VLA Co-Design

NavGPT-3 co-designs a complete harness between a reasoning model and a navigation policy: context and state management, spatial tools, validated execution, structured returns, route repair, and semantic termination. A runtime built above the harness controls its threads. Part I describes these two layers; Part II describes the VLA beneath them.

3.1 Part I: The NavGPT-3 Harness and Runtime

3.1.1 Harness and Runtime Abstraction

The harness shapes model behavior through context, state, and operation contracts connecting the Planner, VLA, and Environment. It constructs model-visible observations and history, maintains spatial references and annotations, validates tool requests, dispatches authorized operations, and returns evidence for the next decision. The Planner controls semantic sequencing and task completion; the VLA supplies optional high-frequency execution. Built above the harness, the OS-like runtime controls logical threads, their lifecycle and permissions, and motion authority. Figure 2 illustrates this coordination through route review and a protective monitor.

3.1.2 Context, Spatial Tools, and Execution Returns

The harness turns observations and interaction history into decision context, and exposes spatial tools that make this context actionable. Because invoking the Planner at every sensorimotor update is impractical, the harness additionally exposes a VLA for extended navigation through a higher-frequency perception–action loop. Invoking the VLA is optional: the Planner decides whether to use it or make all navigation decisions itself through the available tools. At invocation k, it selects

$$a_k \sim \pi_{\text{Planner}}(\cdot \mid C_k, \mathcal{T}_k), \tag{1}$$

where C_k is the harness-provided context, $\mathcal{T}_k$ the permitted tool set, and a_k a call including its arguments. The harness validates and dispatches the selected interaction subject to runtime authorization; the Planner determines the navigation strategy within these boundaries.

The interface groups operations into observation, instruction delegation, spatial revision, and semantic completion. Observation returns current views and observed geometry; delegation returns route keyframes and execution status; revision resolves visited-place references or local corrections into motion; annotation and termination preserve meaning and make completion explicit. These tools

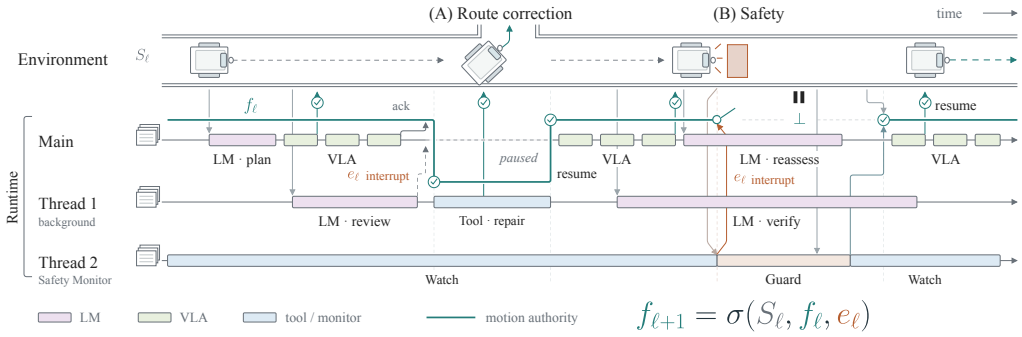


Figure 2: Illustrative foreground-control patterns in the NAVGPT-3 Runtime. Thin bars mark concurrent activity; the teal path denotes motion authority. (A) A route-review interrupt pauses main execution for correction, followed by resumption. (B) A safety interrupt revokes authority while inference continues; resumption requires fresh state and runtime authorization.

and their visual returns were iterated through RSI (Section D.3). Their common contract makes extended execution inspectable and revisable: a VLA stop ends one operation, while the Planner retains final task-stop authority (Section D).

3.1.3 EXECUTION COORDINATION AND MOTION AUTHORITY

Threads perform inference and receive observations at their own cadences, subject to their permissions, so activity on different rows of Figure 2 overlaps in time. The runtime controls which thread holds *motion authority*, the permission to issue task motion (teal path). Let $f_\ell \in \mathcal{J}_\ell \cup \{\perp\}$ denote the foreground thread at event index ℓ, where $\mathcal{J}_\ell$ is the thread set and $\perp$ denotes no holder. Control transfer follows

$$f_{\ell+1} = \sigma(S_\ell, f_\ell, e_\ell), \tag{2}$$

where S_ℓ is shared system state, including the environment (top row), e_ℓ an incoming event such as an interrupt (orange), and σ the runtime's control-transfer rule, which uses preassigned event priorities to retain, transfer, or revoke authority. Protective events have highest priority; terminal limits precede routine review, and stale events cannot restore an obsolete operation (Section A.6). A handoff preserves thread contexts and takes effect only after runtime authorization and execution acknowledgement (checkmarks). Event order ℓ does not impose a common inference clock.

For a task-motion operation a proposed by thread j, foreground eligibility is

$$\text{eligible}_\ell(j, a) = \mathbf{1}[j = f_\ell \ \wedge \ a \in \mathcal{P}_j]\,, \tag{3}$$

where $\mathcal{P}_j$ is the thread's permitted operation set. Eligibility is necessary, not sufficient: dispatch also validates the current state and authorization (Section B.6). Motion authority does not grant exclusive computation or sensing, and calling the Planner or VLA within a thread requires no switch. This mechanism supports both concurrent progress and synchronization. Planning during motion, awaiting a decision before acting, and responding to an external event are coordination patterns, not fixed runtime components. For example, in route correction (Figure 2A), the VLA holds authority while the background thread reviews its progress; the review raises e_ℓ, and after the VLA acknowledges, authority passes to a tool repair while the main thread pauses, then returns so the VLA resumes. In the safety case (B), the monitor's guard revokes authority, $f_{\ell+1} = \perp$, without awaiting LLM inference or the routine acknowledgement, while reasoning continues; the main thread resumes only with fresh state and authorization.

3.2 PART II: NAVGPT VLA

3.2.1 TRAJECTORY POLICY AND VISUAL-CONTEXT ALLOCATION

NavGPT VLA starts from Qwen3-VL-Instruct and adds an MLP action head. At VLA step n, it uses the provided instruction and up to 16 retained observations, each with front, right, back, and



Figure 3: **Codec-weighted visual-context allocation.** (a) Change, recency, and view jointly determine whole-image resolution for a four-view history, with an initial I-frame and subsequent P-frames. (b) Optical-flow diagnostics and resulting token grids for the four views at t_{16}. (c) Allocation change relative to positional weighting versus RGB change $100\delta_{h,v}$ for 41,216 replayed image/context pairs from 13 recorded histories at $B_{\text{vis}} = 3072$. Curve: local regression; numbers match (b).

left views, to predict an ordered trajectory $\tau_n = ((x_{n,r}, y_{n,r}, \theta_{n,r}))_{r=1}^{8}$. Waypoint index r runs over eight cumulative targets in the current agent frame: x points forward and y points left (both in metres), and θ is relative heading in radians. Both 4B and 8B variants expose this contract; execution commits only the first waypoint before the next prediction. Training combines waypoint regression with next-token supervision as detailed in Section B.8.

The VLA sees up to N_{hist} retained observations of N_{view} views (F, R, B, L in Figure 3a) under a visual-token budget B_{vis}; image (h, v), view v of observation h, receives $b_{h,v}$ tokens. Our reimplementation of Qwen-RobotNav's task-adaptive allocation (Qwen Team, 2026b) uses position alone: a weight $w_{h,v}^{\text{pos}}$ reflects recency and view direction, and an allocator converts the weights into integer counts with $b_{\min} \le b_{h,v} \le b_{\max}$ (Section B.8). Position records when and where an image was taken, but not whether it shows anything new.

Codec allocation adds a change score, inspired by I/P-frame video coding (Figure 3a). Within each view, we compare 64×64 RGB thumbnails of consecutive retained images: the first image (t_0) is an I-frame with full score, and later P-frames are scored by their mean absolute pixel difference $\delta_{h,v}$,

$$\delta_{h,v} = \frac{\|\text{thumb}(I_{h,v}) - \text{thumb}(I_{h-1,v})\|_1}{255 \cdot 3 \cdot 64^2}, \qquad s_{h,v}^{\text{change}} = \begin{cases} 1, & \text{I-frame,} \\ \text{clip}(5\delta_{h,v}, 0.05, 1), & \text{P-frame,} \end{cases} \tag{4}$$

The codec weight $w_{h,v}^{\text{codec}} = w_{h,v}^{\text{pos}} s_{h,v}^{\text{change}}$ thus combines change, recency, and view (bottom row of Figure 3a); the allocator and token total are unchanged, and a constant score recovers the positional rule. Whether the score can act depends on the budget-to-capacity ratio $\rho_{\text{vis}} = B_{\text{vis}}/(N_{\text{hist}} N_{\text{view}} b_{\max})$: at $\rho_{\text{vis}} \ge 1$ every image can reach $b_{\max}$ and the two rules coincide, while below one tokens shift toward changing views. Training randomizes $B_{\text{vis}} \in [1024, 4096]$ and the allocation limits so that non-trivial histories enter this active regime (Table 8).

At t_{16} (Figure 3b), F, R, B, and L receive 80, 48, 24, and 35 tokens; the flow fields and change masks only visualize inter-frame differences, as allocation uses the thumbnail score in Eq. (4). Across recorded histories (Figure 3c), codec allocation takes tokens from low-change images and moves them to images that change.

3.2.2 Data Processing and Training

Our Codec recipe jointly trains scene-change allocation and the vision encoder. Navigation sources use the instruction, four-view history, and waypoint contract above; navigation QA and grounding omit waypoint supervision. Both sizes share preprocessing and action normalization (Section B.8). The mixture is cumulative over four scopes, VLN demonstrations, embodied navigation QA, cross-embodiment navigation, and visual grounding, and totals 19.28M effective examples, of which 65.6% supervise the action prediction and 18.0% are general vision-language data that supply

Table 1: **Tool and VLA ablation on the R2R-CE subset.** Resource use is averaged per episode. Opus costs are SDK-reported; Astra costs are estimated from recorded usage at list prices. Inference excludes communication and harness overhead. Reaction is the mean interval between environment actions: Planner time per turn without the VLA, VLA inference per step with it.

Planner	Tools			Navigation				Resource use				
	VLA	Map	Backtrack	NE↓	OSR↑	SR↑	SPL↑	Tokens (k)↓	Turns↓	Cost ($)↓	Infer. (s)↓	React. (s)↓
VLA only	✓	–	–	4.23	78	71	66.14	–	–	–	**44.1**	**0.78**
GPT-6 Astra	–	–	–	4.67	74	72	62.26	325.0	32.14	0.593	105.7	3.3
GPT-6 Astra	–	✓	✓	**2.55**	**87**	**83**	69.04	190.8	10.71	0.469	66.8	6.2
GPT-6 Astra	✓	–	–	4.58	79	75	67.83	227.9	**3.37**	0.469	84.4	**0.78**
GPT-6 Astra	✓	✓	–	4.01	79	76	67.35	201.4	3.42	0.443	80.4	**0.78**
GPT-6 Astra	✓	✓	✓	3.34	84	82	69.12	**155.0**	3.88	0.391	79.1	**0.78**
Opus 5	–	–	–	5.43	70	66	55.24	721.0	47.93	0.69	425.9	8.9
Opus 5	–	✓	✓	3.45	85	73	44.82	483.4	16.75	0.990	323.4	19.3
Opus 5	✓	–	–	3.78	80	76	61.85	199.8	11.19	**0.353**	201.5	**0.78**
Opus 5	✓	✓	–	4.04	85	81	**69.29**	211.8	8.96	0.393	199.4	**0.78**
Opus 5	✓	✓	✓	3.69	83	79	66.70	185.6	8.50	0.397	191.8	**0.78**

grounding and commonsense knowledge. Both model sizes train for one epoch from their Qwen3-VL-Instruct initialization with the same global batch size, schedule, trainable vision encoder, codec allocation, and action contract; model and action-head learning rates are 5×10^{-6} and 10^{-3}, respectively (Table 9).

4 Experiments

We examine the complete harness through tool capabilities, visual-context organization, and Planner–VLA interaction, followed by system-level navigation and transfer. We evaluate on the full Val-Unseen splits of R2R-CE and RxR-CE (Krantz et al., 2020; Anderson et al., 2018; Ku et al., 2020), on object-goal navigation (Batra et al., 2020) in MP3D (Chang et al., 2017) and HM3D v2 (Ramakrishnan et al., 2021), and on embodied question answering (Ren et al., 2024; Zhai et al., 2025; Jiang et al., 2025); ablation studies use a fixed R2R subset of 100 episodes across ten MP3D scans (Qiao et al., 2025). We report success rate (SR), success weighted by path length (SPL), normalized dynamic time warping (nDTW), oracle success (OSR), and navigation error (NE), with accuracy and normalized steps for question answering. Unless varied, harness studies use the 8B NavGPT VLA. Each Planner runs inside a provider scaffold, Codex for GPT-6 Astra and Claude Code for Claude Opus 5, which handles message assembly and conversation management, including provider context compaction. The navigation briefing, registered tools, observations, spatial state, and STOP contract are replaced in NavGPT-3, differences between scaffolds are recorded in Section B. VLA training ablations are in Section C.

4.1 Harnessing and VLA Ablation

Table 1 ablates the harness tools and the VLA available to two language model Planners, GPT-6 Astra and Claude Opus 5, on the fixed R2R subset. Every Planner can observe its current view and take primitive actions (0.25 m forward, 15° turns, stop); three tool groups can be added: VLA, which delegates the instruction to NavGPT VLA and returns its execution; Map, observation tools that supply an observed top-down map and panorama beyond the current view; and Backtrack, movement tools that return to recorded places on the resulting topological map (Section D). We developed these tools, together with the visual prompts in Section 4.2, through recursive self-improvement (RSI) over executable harness versions (Section D.3).

The harness sets how well the language model navigates. With primitive actions alone and without specific harnesses, both Planners perform poorly and spend heavily: GPT-6 Astra reaches 72 SR at 325.0k tokens per episode and Claude Opus 5 reaches 66 SR at 721.0k (Table 1). A

more robust, higher-quality harness changes this. Mapping tools lets the Planner assemble a much stronger spatial context than the current view provides, and Backtrack lets it act on that context far more efficiently. Together they raise success to 83 and 73 SR while cutting tokens by 41% and 33%, and Astra's tools-only row is the most accurate in the table.

Whether the VLA helps depends on the language model. The VLA always shortens the reasoning loop, but it does not always improve without a good harness. For Opus it does: turns fall from 16.75 to 8.50–11.19, success rises from 73 to 76–81 SR, and cost falls from \$0.99 to \$0.35–0.40 per episode. For Astra it does not: turns fall from 10.71 to 3.37, yet success drops to 75 SR and cost stays at \$0.469, as the tokens saved on navigation are spent supervising and correcting the VLA's execution. A stronger language model thus absorbs more of the benefit the VLA would otherwise provide, and the upper bound on success is set by the language model itself: once it reads the environment well enough to decide accurately on its own, success is limited by its own navigation ability rather than by the interface. Once the VLA is given its own harness, with Mapping and Backtracking over its rollouts, token use reaches its minimum for both models (155.0k for Astra and 185.6k for Opus) with success close to each model's best.

The VLA sets the minimum reaction time. Without the VLA, the system acts only as often as the language model decides, once every 3–19 s on average depending on the model and tools. With the VLA, the control loop runs at VLA inference rate instead, 0.78 s per step on average for the 8B model, which gives the whole system a basic guarantee of real-time response (Table 1). Section 4.3 examines how much of this higher interaction rate the Planner can exploit by checking VLA execution at increasing frequencies.

4.2 Visual Prompt Organization

The harness also decides how tool returns are shown to the Planner: as a visual prompt of camera views and an observed map, annotated with view labels, node IDs, heading, and turn cues. Figure 4 varies one presentation factor at a time against this grounded reference while keeping tools fixed (Section B.5): caption labels move the annotations into adjacent text, compact glyphs use a smaller digit font, heading-up rotates the map with the agent, and absolute bearings replace relative turn cues with fixed-frame directions. Both language models are robust to how the visual prompt is organized: SR stays within 76–78% for Astra and 73–77% for Opus 5 across all five versions.

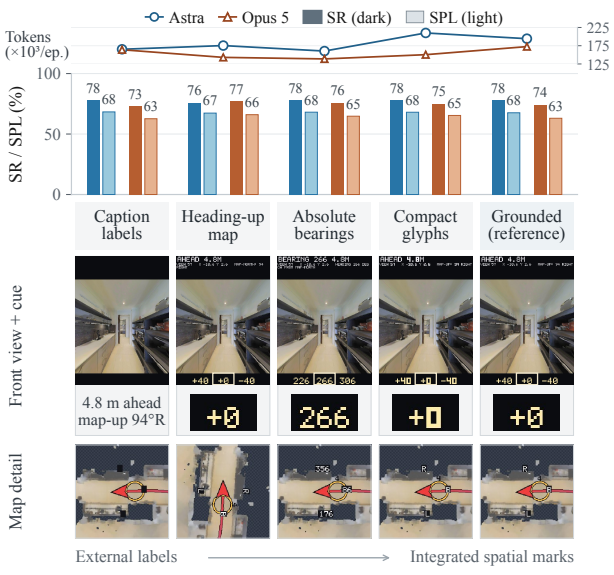


Figure 4: **Visual prompt organization.** SR/SPL and token usage for Astra and Opus 5.

What changes is how much reasoning a version demands, and that demand does not follow human readability. Absolute bearings, the hardest version for a person to read, use the fewest tokens for both models, while the most expensive version differs between them: compact glyphs for Astra and the grounded reference for Opus, within 160–210k and 139–173k mean cumulative tokens per episode, respectively. Strong language models are therefore robust to visual context, and what they read easily does not follow human readability.

Table 2: **Check frequency.** VLA steps per check.

Frequency	NE↓	OSR↑	nDTW↑	SR↑	SPL↑	Tokens (k)↓
None	3.82	81	16.74	77	67.51	**179.2**
32	3.56	81	16.66	78	68.88	1,529.9
16	**2.01**	**85**	**18.72**	**83**	**77.36**	2,319.5

Table 3: **Vision-and-language navigation.** R2R-CE and RxR-CE Val-Unseen splits. †RxR Val-Unseen English human followers (Ku et al., 2020), measured in DE. Both NAVGPT-3 systems use the 8B VLA.

Method	**Model**	**R2R-CE Val-Unseen**				**RxR-CE Val-Unseen**			
		NE↓	OSR↑	SR↑	SPL↑	NE↓	nDTW↑	SR↑	SPL↑
Human† (Ku et al., 2020)	–	–	–	–	–	1.32	77.7	90.4	–
NavFoM (Zhang et al., 2025a)	7B	4.61	72.1	61.7	55.3	4.74	65.8	64.4	56.2
ABot-N0 (AMAP CV Lab, 2026)	4B	3.78	70.8	66.4	63.9	3.83	–	69.3	60.0
AstraNav-World (Hu et al., 2026b)	3B+5B	3.86	73.9	67.9	65.4	3.82	–	72.9	61.5
OmniNav (Hu et al., 2025)	3B	3.74	74.6	69.5	66.1	3.77	–	73.6	62.0
Qwen-RobotNav (Qwen Team, 2026b)	4B	3.80	77.2	69.5	63.6	3.80	71.9	75.2	65.0
Qwen-RobotNav (Qwen Team, 2026b)	8B	3.53	78.5	72.1	66.6	3.58	72.5	76.5	65.7
Robostral Navigate (Mistral AI, 2026)	8B	3.20	81.3	77.4	**74.2**	3.47	–	75.1	68.7
Ours: executable navigation policy									
NavGPT VLA	4B	3.41	79.91	72.54	67.19	3.35	73.33	76.77	67.77
NavGPT VLA	8B	3.29	80.19	74.51	68.54	3.05	74.85	78.19	68.98
Ours: complete navigation system									
NAVGPT-3	Opus 5	2.77	84.07	79.01	70.16	2.27	75.09	84.80	69.85
NAVGPT-3	GPT-6 Astra	**2.18**	**86.99**	**81.51**	70.42	**1.56**	**78.47**	**90.43**	**74.31**

4.3 COMMUNICATION FREQUENCY AND COST

Table 2 varies how often the Planner checks VLA execution, measured in VLA forward passes per check; each pass predicts eight waypoints and executes the first (Section 3.2.1). Without periodic checks, the Planner regains control only when the VLA stops or reaches its delegation limit, after a median of 51 steps (90th percentile 123). At each check, the Planner reviews the route so far while the VLA keeps executing, and interrupts it only to revise or terminate the route. We test GPT-6 Astra with shared tools and instructions.

More frequent checks improve the route, but only at a steep cost. Checking every 32 steps gives most episodes at most one check and barely changes results (78 vs. 77 SR). Checking every 16 steps, about three times per episode, raises SR to 83 and SPL from 67.51 to 77.36, and cuts NE from 3.82 to 2.01 m. Tokens grow much faster than accuracy, because each check re-reads the accumulated route context: they rise from 179.2k to 1,529.9k per episode at 32 steps, and to 2,319.5k at 16. Periodic checks thus buy accuracy with costly supervision, which motivates event-triggered review for real-world deployment.

4.4 OVERALL NAVIGATION PERFORMANCE

Table 3 separates what the execution policy reaches alone from what the runtime adds. Without any harness, NavGPT VLA is already at the state of the art on RxR-CE. The runtime then raises R2R-CE success from 74.51 to 81.51 SR with GPT-6 Astra, and gains more on RxR-CE, where success reaches 84.80 and 90.43 SR against 78.19 for the specialist alone and nDTW improves to 78.47: RxR instructions are longer and specify the route in far more detail, which gives the returned route evidence, returns to recorded places, and local corrections more to act on. NAVGPT-3 matches RxR human followers in success (90.43 vs. 90.4 SR) and path fidelity (78.47 vs. 77.7 nDTW), the first autonomous agent to reach human-level performance on this benchmark (Ku et al., 2020).

Table 4: **Object-goal navigation.** Prior methods use HM3D v1, others HM3D v2. †: 500-episode subsets, scored with a stricter stop criterion. ‡: depth or odometry.

Model	MP3D		HM3D	
	SR↑	SPL↑	SR↑	SPL↑
Prior methods: HM3D v1				
VLFM‡	36.4	17.5	52.5	30.4
SG-Nav‡	40.2	16.0	54.0	24.9
WMNav‡	45.4	17.2	58.1	31.2
CogNav‡	46.6	16.1	72.5	26.2
Uni-NaVid	–	–	73.7	37.1
Qwen-RobotNav 4B	52.2	16.0	75.6	30.6
Qwen-RobotNav 8B	48.8	17.7	71.2	33.0
NAVGPT-3†	59.2	25.2	75.3	40.8

Table 5: **Embodied question answering.** Accuracy and normalized steps (Zhai et al., 2025) on HM-EQA (Ren et al., 2024) and MT-HM3D (Zhai et al., 2025); LLM score and path efficiency on EXPRESS-Bench (Jiang et al., 2025). *: from prior work. NAVGPT-3 uses a GPT-6 Astra judge.

System	HM-EQA		MT-HM3D		EXPRESS	
	Acc.↑	Steps↓	Acc.↑	Steps↓	Score↑	Path eff.↑
Prior systems						
Explore-EQA (Ren et al., 2024)	58.4	0.52	36.2*	0.64	–	–
Graph-EQA (Saxena et al., 2024)	63.5	0.20	45.6*	0.45	–	–
Memory-EQA (Zhai et al., 2025)	61.4	0.40	43.1	0.41	–	–
Fine-EQA (Jiang et al., 2025)	56.0	0.54	–	–	63.95	25.58
3D-Mem (Yang et al., 2025)	50.4	0.63	–	–	–	–
FAST-EQA (Zhang et al., 2026)	69.2	0.65	50.5	0.52	68.7	29.25
Qwen-RobotNav 8B executor						
NavMCP (Lei et al., 2026)	76.7	0.15	54.4	0.19	79.27	33.96
NAVGPT-3	**77.38**	0.16	**55.27**	0.18	80.43	34.26

Object navigation and embodied question answering. Table 4 evaluates NAVGPT-3 on 500-episode subsets of the MP3D and HM3D v2 validation splits against published full-split results, scoring success only when the agent stops within the distance threshold while facing the target. Both cross-task studies use the 8B NavGPT VLA under a Claude Opus 5 Planner. The two benchmarks locate the runtime's contribution differently: MP3D spans 21 goal categories, many of them small or fine-grained (cushion, towel, picture), and the gain appears in success, whereas HM3D v2 uses six large, visually distinctive categories on which published systems already succeed in roughly three of four episodes, and the gain appears in path efficiency. On embodied question answering (Table 5), NAVGPT-3 is level with the strongest prior system, Qwen-RobotNav (Lei et al., 2026): accuracy is ahead on both datasets at comparable normalized steps (Section B.3).

Qualitative navigation behavior. Four selected successful episodes in Section G illustrate how returned observations guide the Planner after extended VLA execution. Claude Opus 5 recognizes that the VLA has passed the hallway table, retraces 8.54 m to node 1, and verifies a 1.2 m endpoint correction (Figure G.1). On a longer route, it returns from a dead-end nook and directly completes the omitted cabinet and doorway clauses (Figure G.2). GPT-6 Astra recovers from stalled execution: after a repeated VLA call and a blocked local move, it retreats to node 3 and takes the other aisle (Figure G.3). Finally, Opus delegates again when local stair attempts leave the instruction unfinished; the next rollout reaches the upstairs bathroom (Figure G.4). These cases illustrate successful backtracking, local repair, and renewed delegation, not their frequency or average benefit.

Real-world deployment. We deploy NAVGPT-3 on a Unitree Go2 with RGB input, remote or onboard VLA inference, and a runtime LiDAR hazard monitor (Section F). In a matched comparison (Section F), overlapping reasoning with priority monitoring raises SR from 73.3 to 83.3 over sequential queued handling and cuts p95 halt latency from 1840 to 180 ms.

5 CONCLUSION

We presented NAVGPT-3, a complete navigation harness with a runtime layer above it. The harness organizes context and navigation state, exposes spatial tools, and returns execution evidence for route repair and semantic termination; the runtime coordinates these capabilities through thread control and motion authority. The standalone VLA and complete system attain strong full-split navigation results, with the latter matching human followers on RxR-CE under continuous rather than graph-based control. The harness studies show how spatial tools support Planner navigation, visual presentation affects processing cost, and VLA delegation provides frequent action updates with fewer reasoning turns. Together, these components establish a complete embodied interface through which reasoning can inspect, revise, and control extended physical execution. The harness provides the structure for integrating stronger reasoning and action models as they become available.

AI USE STATEMENT

Large language models are components of the studied system: GPT-6 Astra and Claude Opus 5 act as Planners, and a GPT-6 Astra judge scores EXPRESS-Bench answers, as described in Section 4 and Section B. Separately, we used generative AI coding assistants (Claude Code and Codex) to help implement the experiment and evaluation infrastructure, to create and revise scientific figures, and to draft, edit, and format parts of this paper. We have reviewed all AI-assisted work: AI-assisted code was tested before use, and all text, figures, and reported numbers were checked by the authors against the recorded experiment outputs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

AMAP CV Lab. ABot-N0: Technical report on the vla foundation model for versatile embodied navigation. *arXiv preprint arXiv:2602.11598*, 2026.

Xiang An, Yin Xie, Feilong Tang, Yunyao Yan, Huajie Tan, Didi Zhu, Changrui Chen, Xiuwei Zhao, Bin Qin, Kaicheng Yang, Yifei Shen, Yuanhan Zhang, Kaichen Zhang, Wenkang Zhang, Zheng Cheng, Nansen Zhang, Chunsheng Wu, Chunjiang Ge, Zimin Ran, Dehua Song, Chunyuan Li, Shikun Feng, Ming Hu, Zhangquan Chen, Junbo Niu, Bo Li, Ziyong Feng, Ziwei Liu, Zongyuan Ge, and Jiankang Deng. LLaVA-OneVision-2: Towards next-generation perceptual intelligence. *arXiv preprint arXiv:2605.25979*, 2026.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, pp. 3674–3683, 2018.

Anthropic. Claude Fable 5.1 and Claude Mythos 5.1 system card. `https://www.anthropic.com/claude-fable-and-mythos-5-1`, 2026a. System card released 2026-09-01.

Anthropic. Claude Opus 5 system card. `https://www-cdn.anthropic.com/b514064af1408018e64b1ad24e7d5e75850b4ffd/Claude%20Opus%205%20System%20Card.pdf`, 2026b. System card released 2026-07-24.

Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects. In *arXiv:2006.13171*, 2020.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*, pp. 667–676. IEEE, 2017.

Jiaqi Chen, Bingqian Lin, Ran Xu, Zhenhua Chai, Xiaodan Liang, and Kwan-Yee K Wong. Mapgpt: Map-guided prompting for unified vision-and-language navigation. *arXiv preprint arXiv:2401.07314*, 2024.

An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Xueyan Zou, Jan Kautz, Erdem Biyik, Hongxu Yin, Sifei Liu, and Xiaolong Wang. NaVILA: Legged robot vision-language-action model for navigation. In *Robotics: Science and Systems (RSS)*, 2025.

Hao-Tien Lewis Chiang, Zhuo Xu, Zipeng Fu, Mithun George Jacob, Tingnan Zhang, Tsang-Wei Edward Lee, Wenhao Yu, Connor Schenck, David Rendleman, Dhruv Shah, Fei Xia, Jasmine Hsu, Jonathan Hoech, Pete Florence, Sean Kirmani, Sumeet Singh, Vikas Sindhwani, Carolina Parada, Chelsea Finn, Peng Xu, Sergey Levine, and Jie Tan. Mobility VLA: Multimodal instruction navigation with long-context VLMs and topological graphs. *arXiv preprint arXiv:2407.07775*, 2024.

DeepSeek-AI. DeepSeek Harness: Everything is a plugin. `https://github.com/deepseek-ai/deepseek-harness`, 2026. Accessed 2026-08-29.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue. *arXiv preprint arXiv:2410.00037*, 2024. URL `https://arxiv.org/abs/2410.00037`.

Chongkai Gao, Zixuan Liu, Zhenghao Chi, Junshan Huang, Xin Fei, Yiwen Hou, Yuxuan Zhang, Yudi Lin, Zhirui Fang, Zeyu Jiang, and Lin Shao. VLA-OS: Structuring and dissecting planning representations and paradigms in vision-language-action models. *arXiv preprint arXiv:2506.17561*, 2025.

Pengyuan Guo, Zhonghao Mai, Zhengtong Xu, Kaidi Zhang, Quan Khanh Luu, Heng Zhang, Zichen Miao, Arash Ajoudani, Zachary Kingston, Qiang Qiu, and Yu She. PLanAR: Planning-language-grounded agentic reasoning for robot manipulation. *arXiv preprint arXiv:2602.01662*, 2026.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical VLA agents. *arXiv preprint arXiv:2606.10267*, 2026a.

Weijing Hu, Jun Wang, Teng Hu, Jiteng Chen, Siwen Xue, Yufeng Yue, Haoran Xie, Weixun Zhang, Huchuan Lu, Zongqing Lu, Haibin He, and Bolei Wang. OmniNav: A unified framework for prospective exploration and visual-language navigation. *arXiv preprint arXiv:2509.25687*, 2025.

Weijing Hu, Jun Wang, Teng Hu, Jiteng Chen, Siwen Xue, Yufeng Yue, Yanyun Wu, Haibin He, Bolei Wang, Huchuan Lu, and Zongqing Lu. AstraNav-World: World model for foresight control and consistency. *arXiv preprint arXiv:2512.21714*, 2026b.

Muye Huang, Lingling Zhang, Xingyu Yu, Lei Shi, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Jun Liu. DuplexOmni: Real-time listening, seeing, thinking, and speaking for full-duplex interaction. *arXiv preprint arXiv:2606.09186*, 2026a. URL https://arxiv.org/abs/2606.09186.

Xiangyun Huang, Xiangchen Wang, Runfeng Lin, Yihao Xu, Kangyu Huang, Hengchen Jiang, Xiwang Dong, and Jiarong Lin. From routes to steps: Separating semantic progress from local execution in vision-and-language navigation. *arXiv preprint arXiv:2608.03143*, 2026b.

InternVLA-N1 Team. InternVLA-N1: An open dual-system navigation foundation model with learned latent plans. Technical report, Shanghai AI Laboratory, 2025. URL https://internrobotics.github.io/internvla-n1.github.io/static/pdfs/InternVLA_N1.pdf.

Kaixuan Jiang, Yang Liu, Weixing Chen, Jingzhou Luo, Ziliang Chen, Ling Pan, Guanbin Li, and Liang Lin. Beyond the destination: A novel benchmark for exploration-aware embodied question answering. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9091–9101, 2025.

Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *European Conference on Computer Vision (ECCV)*, pp. 104–120. Springer, 2020.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4392–4412, 2020.

Zixing Lei, Gengze Zhou, Xiong-Hui Chen, Jiazhao Zhang, Yiyang Huang, Hang Yin, Haoqi Yuan, Qi Wu, Weixin Li, and Siheng Chen. Scaffolding foundation models into physical-world agents pushes the frontier of long-horizon navigation. *arXiv preprint arXiv:2608.30396*, 2026.

Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026.

An Liu, Bingxi Liu, Hongyu Ding, Yixuan Jiang, Yaran Chen, Fulin Tang, Cong Leng, Hong Zhang, and Jian Cheng. HAM-VLN: Harnessing hierarchical agentic memory for zero-shot vision-and-language navigation. *arXiv preprint arXiv:2607.29600*, 2026.

Yuxing Long, Xiaoqi Li, Wenzhe Cai, and Hao Dong. Discuss before moving: Visual language navigation via multi-expert discussions. *arXiv preprint arXiv:2309.11382*, 2023.

Mistral AI. Robostral Navigate. *arXiv preprint arXiv:2607.20785*, 2026.

OpenAI. GPT-6 Astra system card. https://deploymentsafety.openai.com/gpt-6-astra, 2026. Released 2026-09-03.

Bowen Pan, Rameswar Panda, SouYoung Jin, Rogerio Feris, Aude Oliva, Phillip Isola, and Yoon Kim. Langnav: Language as a perceptual representation for navigation. *arXiv preprint arXiv:2310.07889*, 2023.

Yanyuan Qiao, Wenqi Lyu, Hui Wang, Zixu Wang, Zerui Li, Yuan Zhang, Mingkui Tan, and Qi Wu. Open-Nav: Exploring zero-shot vision-and-language navigation in continuous environment with open-source LLMs. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. URL https://arxiv.org/abs/2409.18794.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork. https://qwen.ai/blog?id=qwen3.8, 2026a. Released 2026-08-03.

Qwen Team. Qwen-RobotNav technical report: A scalable navigation model designed for an agentic navigation system. *arXiv preprint arXiv:2606.18112*, 2026b.

Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Allen Z. Ren, Jaden Clark, Anushri Dixit, Masha Itkina, Anirudha Majumdar, and Dorsa Sadigh. Explore until confident: Efficient exploration for embodied question answering. In *Robotics: Science and Systems (RSS)*, 2024.

Saumya Saxena, Blake Buchanan, Chris Paxton, Bingqing Chen, Narunas Vaskevicius, Luigi Palmieri, Jonathan Francis, and Oliver Kroemer. GraphEQA: Using 3D semantic scene graphs for real-time embodied question answering. *arXiv preprint arXiv:2412.14480*, 2024.

Feilong Tang, Xiang An, Yunyao Yan, Yin Xie, Bin Qin, Kaicheng Yang, Yifei Shen, Yuanhan Zhang, Bo Li, Shikun Feng, Zhangquan Chen, Huajie Tan, Ming Hu, Kaichen Zhang, Chunyuan Li, Ziyong Feng, Ziwei Liu, Zongyuan Ge, and Jiankang Deng. OneVision-Encoder: Codec-aligned sparsity as a foundational principle for multimodal intelligence. *arXiv preprint arXiv:2602.08683*, 2026.

Wei Wang, Wenqiao Zhang, Yutong Lin, Yuqian Yuan, Tianwei Lin, Jinhao Mao, Zhenxuan Fan, Mingjian Gao, Yang Dai, Wentong Li, Zheqi Lv, Zheng Dong, Yingjie Niu, Jiaqi Zhu, Jun Xiao, Chao Li, and Yueting Zhuang. EmbodiedSkills: A unified framework for orchestrating, training, and deploying VLA agents. *arXiv preprint arXiv:2609.01281*, 2026.

Meng Wei, Chenyang Wan, Jiaqi Peng, Xiqian Yu, Yuqiang Yang, Delin Feng, Wenzhe Cai, Chenming Zhu, Tai Wang, Jiangmiao Pang, et al. Ground slow, move fast: A dual-system foundation model for generalizable vision-and-language navigation. *arXiv preprint arXiv:2512.08186*, 2025a.

Meng Wei, Chenyang Wan, Xiqian Yu, Tai Wang, Yuqiang Yang, Xiaohan Mao, Chenming Zhu, Wenzhe Cai, Hanqing Wang, Yilun Chen, et al. Streamvln: Streaming vision-and-language navigation via slowfast context modeling. *arXiv preprint arXiv:2507.05240*, 2025b.

Tianshuo Yang, Guanyu Chen, Yutian Chen, Zhixuan Liang, Yitian Liu, Zanxin Chen, Chunpu Xu, Haotian Liang, Jiangmiao Pang, Yao Mu, and Ping Luo. HiVLA: A visual-grounded-centric hierarchical embodied manipulation system. *arXiv preprint arXiv:2604.14125*, 2026.

Yuncong Yang, Han Yang, Jiachen Zhou, Peihao Chen, Hongxin Zhang, Yilun Du, and Chuang Gan. 3D-Mem: 3D scene memory for embodied exploration and reasoning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17294–17303, 2025.

Mingliang Zhai, Zhi Gao, Yuwei Wu, and Yunde Jia. Memory-centric embodied question answering. *arXiv preprint arXiv:2505.13948*, 2025.

Haochen Zhang, Nirav Savaliya, Faizan Siddiqui, and Enna Sachdeva. FAST-EQA: Efficient embodied question answering with global and local region relevancy. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 1664–1673, 2026.

Jiazhao Zhang, Kunyu Wang, Rongtao Xu, Gengze Zhou, Yicong Hong, Xiaomeng Fang, Qi Wu, Zhizheng Zhang, and He Wang. NaVid: Video-based VLM plans the next step for vision-and-language navigation. *arXiv preprint arXiv:2402.15852*, 2024.

Jiazhao Zhang, Anqi Li, Yunpeng Qi, Minghan Li, Jiahang Liu, Shaoan Wang, Haoran Liu, Gengze Zhou, Yuze Wu, Xingxing Li, et al. Embodied navigation foundation model. *arXiv preprint arXiv:2509.12129*, 2025a.

Jiazhao Zhang, Kunyu Wang, Shaoan Wang, Minghan Li, Haoran Liu, Songlin Wei, Zhongyuan Wang, Zhizheng Zhang, and He Wang. Uni-NaVid: A video-based vision-language-action model for unifying embodied navigation tasks. *Robotics: Science and Systems*, 2025b.

Hailin Zhong and Shengxin Zhu. AI Harness Engineering: A runtime substrate for foundation-model software agents. *arXiv preprint arXiv:2605.13357*, 2026.

Gengze Zhou, Yicong Hong, Zun Wang, Xin Eric Wang, and Qi Wu. NavGPT-2: Unleashing navigational reasoning capability for large vision-language models. In *European Conference on Computer Vision (ECCV)*, pp. 260–278. Springer, 2024a.

Gengze Zhou, Yicong Hong, and Qi Wu. NavGPT: Explicit reasoning in vision-and-language navigation with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 7641–7649, 2024b. doi: 10.1609/aaai.v38i7.28597.

Jian Zhou, Sihao Lin, Jin Li, Shuai Fu, Gengze Zhou, and Qi Wu. Automating the design of embodied agent architectures. *arXiv preprint arXiv:2606.30111*, 2026a.

Jian Zhou, Xunyi Zhao, Gengze Zhou, Zerui Li, Sihao Lin, Jiajun Liu, and Qi Wu. Embodied agents take control: Minimal-interface zero-shot agents rival industrial-scale policies in vision-and-language navigation. *arXiv preprint arXiv:2607.26148*, 2026b.

A Runtime State and Timing Contracts

This appendix expands Eqs. (1) to (3), separating model-facing harness contracts from thread control in the runtime built above them.

A.1 Logical Threads and Shared State

We use j for a logical thread, k for an invocation, and t for wall-clock time. Shared system state $S(t)$ combines harness-maintained instructions and spatial records, Environment observations, and runtime thread and operation metadata. Each thread carries

$$S_j^{\text{thread}}(t) = \big(\text{role}_j(t), H_j(t), \text{status}_j(t), \mathcal{R}_j(t), \mathcal{W}_j(t)\big), \tag{5}$$

where role_j is its responsibility, H_j its working history, status_j its lifecycle state, and $\mathcal{R}_j, \mathcal{W}_j$ its permitted shared-state reads and proposed writes. Shared spatial history records the route and places, whereas working history records a thread's own interactions. The runtime tracks tool-operation status separately, so the main thread can run Planner reassessment while its VLA operation continues or is suspended; tool completion does not imply task completion.

The foreground selector σ retains, transfers, or revokes motion authority in response to events. A routine transfer withdraws the previous operation's grant and acknowledges its committed action boundary before replacement motion is authorized. Thread contexts survive a handoff, but stale commands do not regain authorization on resumption.

A.2 Synchronous Realization

At synchronous handoff time t_k, with $S_k = S(t_k)$, main-thread history H_k, and registered tools $\mathcal{T}_k$, the implemented tool-call cycle is

$$\begin{aligned} C_k &= \text{Context}_{\text{main}}(S_k, H_k, \mathcal{T}_k), \\ a_k &\sim \pi_{\text{Planner}}(\cdot \mid C_k, \mathcal{T}_k), \qquad (S_{k+1}, E_k) = \text{Execute}(S_k, a_k). \end{aligned} \tag{6}$$

The harness constructs context C_k, validates call a_k, invokes the authorized operation, and returns updated state and evidence E_k. The runtime controls thread execution and motion authority; the Environment commits physical transitions. An extended operation such as a VLA rollout may contain many observations and physical updates between two call boundaries. VLA execution and direct Planner-selected motion share this interface.

A.3 Context and Update Admission

For thread j at its kth invocation, the general context and decision interface is

$$C_{j,k} = \text{Context}_j\big(S|_{\mathcal{R}_j}, H_j, \mathcal{E}_{j,k}, \mathcal{T}_j\big), \qquad a_{j,k} \sim \pi_{j,k}(\cdot \mid C_{j,k}, \mathcal{T}_j), \tag{7}$$

evaluated at the invocation's start time, where $\mathcal{E}_{j,k}$ contains delivered events and $S|_{\mathcal{R}_j}$ restricts visibility to permitted reads. The harness manages context acquisition, persistence, and provenance; Context_j constructs one request using the state and harness capabilities permitted to thread j. Provider-independent replay would additionally require a canonical request record, since the hosted provider retains conversation state. Harness state services maintain navigation records, while the Environment owns physical transitions. The runtime admits a proposed update only within $\mathcal{W}_j$ and the current task and delegation authorization; the responsible service validates its schema and target. Capability and argument validation are implemented; admission across independently executing activities remains proposed.

A.4 Motion Authority and Protective Control

Let $G_o(t) \in \{0, 1\}$ indicate whether physical operation o holds the motion grant, and $g(t)$ be the authorization generation advanced whenever a grant is revoked or replaced. A grant may be held only by a permitted operation of the foreground thread (Eq. (3)), and each task command u_o carries its issuing generation. Eligibility requires

$$\sum_o G_o(t) \le 1, \qquad \text{eligible}(u_o, t) = G_o(t)\,\mathbf{1}[\text{generation}(u_o) = g(t)]\,\mathbf{1}[\text{preconditions}(u_o, S(t))], \tag{8}$$

so late inference or waypoints queued before a suspension cannot regain authority. With a latched protective-stop condition $p_{\text{stop}}(t)$ and a low-latency filter Protect over fresh physical observations $O_{\text{phys}}(t)$, the command gateway applies

$$u_{\text{robot}}(t) = \begin{cases} u_{\text{stop}}, & p_{\text{stop}}(t) = 1 \text{ or no task command is eligible,} \\ \text{Protect}\big(O_{\text{phys}}(t), u_{\text{task}}(t)\big), & \text{otherwise.} \end{cases} \tag{9}$$

An obstacle trigger or control watchdog can latch p_{stop} and revoke the current grant without waiting for Planner inference, and recovery requires cleared protection, refreshed state, and a new grant. Only the Planner accepts the semantic endpoint; the protective path can inhibit motion regardless of that decision.

A.5 Scheduling and Interaction Frequency

Activities may run at different update periods and on different events, as in full-duplex and asynchronous interaction–thinking models (Défossez et al., 2024; Huang et al., 2026a); response deadlines are requirements rather than proven bounds. The runtime design permits review of a committed snapshot while VLA execution continues, with interruption only when the review requests revision or termination. One cycle predicts eight waypoints and executes the first; unused predictions and failed retries do not advance the counter. The benefit of concurrent review is measured on the Go2 (Section F). For constant step duration P_{VLA}, FP_{VLA} describes execution time between checks; total elapsed time also includes any paused review.

A.6 Priority and Handoff Policy

The runtime declares priorities when activities are created: protective stop or watchdog expiry, terminal episode limits, operation completion, then route-review requests. An event carries its operation, authorization generation, timestamp, and sequence number; equal-priority events follow arrival order. Obsolete review/completion events are discarded; repeated review requests for one operation coalesce into one pending request. A protective latch immediately revokes motion authority and advances the generation; reasoning may continue, but neither a late model response nor a queued waypoint can issue motion under the old grant. Clearing an alert alone does not resume motion: fresh observations and a new authorization are required. Terminal limits also revoke motion; otherwise natural completion takes precedence over a coincident scheduled review. A continuing review preserves VLA history, while a correction waits for a committed execution boundary before a replacement operation is authorized.

This policy distinguishes the runtime coordination contract from the boundary scheduler used in the local ablation adapter. The latter implements serialized motion, generation checks, natural-return precedence, one pending review request, and history-preserving continuation. Independent safety-monitor preemption and its physical timing are measured on the Go2 (Section F) rather than inferred from navigation success or mean VLA inference time.

B Additional Experimental Details

B.1 Evaluation Settings

Table 6: Harness evaluation settings.

Setting	Value
RGB resolution	512 pixels
Camera order	Front, right, back, left
Returned route keyframes	At most 8, uniformly sampled along the rollout
Episode / delegation limit	500 / 200 Environment steps
Planner-call limit	200 turns per episode
Episode timeout	2,400 s

Benchmark goal distances and success labels are available only to the offline evaluator.

B.2 HUMAN PERFORMANCE ON RXR-CE

RxR asks an agent to follow a route instruction to its endpoint. Human demonstrations use discrete graph edges, which abstract away low-level movement and obstacle avoidance; RxR-CE realizes navigation through continuous motion (Ku et al., 2020; Krantz et al., 2020). Our 90.43 SR and 78.47 nDTW match English human followers (90.4 SR and 77.7 nDTW) despite this additional control burden on the same instructions.

B.3 CROSS-TASK EVALUATION

The object-goal subsets are drawn in proportion to target categories and scenes, and the harness uses an object-goal delegation contract whose termination rule treats goal sighting, rather than instruction completion, as the stop condition. Reference values follow Table 4 of Qwen Team (2026b). We do not report per-category MP3D results, since proportional sampling leaves the rarest categories with about two episodes each. Question-answering metrics follow Table 7 of Qwen Team (2026b); prior EXPRESS-Bench scores used the now-deprecated GPT-4o-mini judge.

B.4 PLANNER SCAFFOLDS AND ACCOUNTING

We replace the default task prompt with the NavGPT-3 navigation briefing, including the robot, instruction, tool semantics, movement budget, and STOP rule, and register the corresponding navigation interfaces. The scaffolds' native task prompts are not used, and every reported Planner result is produced by the NAVGPT-3 harness running inside them. The provider scaffold remains responsible for assembling the conversation and managing its context window, including any provider-side compaction. The harness determines which observations, route evidence, and spatial state enter those messages, subject to runtime visibility permissions. Thus rewriting the task prompt does not make Codex and Claude Code identical inference interfaces: provider instruction blocks, generic execution facilities, and result wrappers may still differ. Within-Planner ablations freeze the scaffold and briefing; cross-Planner results describe the complete Planner–scaffold combinations rather than an isolated backbone ranking.

Tokens (k) sum all processed input tokens, including cached input, and generated output tokens across requests, divided by 1,000. Turns count Planner responses separated by environment feedback, including the initial response and final acknowledgement; text, reasoning, and parallel tool calls within a response count once. Cost is the total hosted inference charge in US dollars per attempted episode, including retries and cache charges. VLA latency is reported separately rather than priced. Opus costs are SDK-reported; Astra costs use recorded token usage at list prices. Inference timing excludes communication and harness overhead. The action-update interval in Table 1 is Planner time per turn without the VLA and VLA inference per step with it; it does not include physical braking or interrupt propagation.

B.5 VISUAL PROMPT VARIANTS

Every variant keeps the same eight tools, source frames, route keyframes, telemetry, node IDs, annotations, observed geometry, and execution semantics. The grounded reference binds labels to their images with 5×7 bitmap glyphs and shows the map with fixed map-north upward, a heading arrow, and body-frame turn cues; map-north names the map frame, not geographic north. Compact glyphs use the earlier $3{\times}5$ digit family at matched height and contrast, the heading-up map keeps the same crop and scale, and absolute bearings provide bearing and pose information sufficient to derive the same directions. Labels stay outside the scene content so that no variant changes occlusion.

B.6 INTERACTION FREQUENCY

The check schedule is defined in Section A. All frequencies share the same Planner, VLA, prompt, tools, route-evidence serializer, and history-preserving resume adapter, and a check may continue the current route without selecting a recovery action. The execution adapter also records a review request after two consecutive blocked transitions; such requests are delivered at the next scheduled or natural return rather than triggering an additional check, which keeps the frequency comparison free of event-triggered scheduling. A request cannot write navigation state or terminate the episode, and

the runtime releases the VLA's action lease before authorizing replacement motion. Cost includes all model calls and waiting time; periodic checks are counted separately from natural returns. Table 7 summarizes the authority each component holds at a handoff.

Table 7: Permissions and information at an interruption boundary.

Operation	Planner	VLA / adapter	Enforcing layer
Interrupt active VLA leg	Request when authorized	Acknowledge boundary	Runtime: revoke action lease
Request early Planner review	Receive when authorized	Continue executing	Runtime: queue or invalidate
Resume or revise route	Choose registered action	Execute valid delegation	Runtime: grant one action lease
Change physical / route state	No direct writes	Propose waypoint	Environment: motion; harness: route records
End complete episode	Explicit semantic STOP	Advisory stop only	Runtime: enforce STOP / limits
Expose cross-level context	Typed directive only	Typed status / evidence	Runtime: visibility; harness: context

B.7 Visual-Context Allocation

Baseline and codec rule. The positional baseline follows the Qwen-RobotNav observation interface (Qwen Team, 2026b): time and camera weights allocate a shared budget across retained images, and each image is resized to its assigned pixel budget. Codec adds the RGB-change multiplier defined in Section 3, retaining the same allocator and image-resizing path. The score measures visual change, including changes caused by camera motion and lighting; it is not a learned semantic-importance estimate or an optical-flow estimator.

Integer allocation. Flatten the retained history in time-major, front/right/back/left order. For $M = N_{\text{hist}} N_{\text{view}}$ images, initialize every allocation to $b_{\min}$, requiring $M b_{\min} \le B_{\text{vis}}$. Distribute the remaining budget proportionally to the positive joint weights, take the floor of each share, cap additions at $b_{\max}$, and redistribute any overflow among uncapped images. Assign the final integer remainder in descending weight order, filling each image's remaining capacity before advancing. The implementation uses `numpy.argsort(-weights)` on the flattened array, including its ordering of equal-weight ties. No tokens are assigned beyond an image's capacity: when $B_{\text{vis}} \ge M b_{\max}$ all images receive $b_{\max}$ and the excess budget is unused. Image resizing preserves aspect ratio and rounds dimensions to the encoder's patch/merge grid, so realized image-token counts can be below their allocated limits.

Evaluation configuration. The portable four-view VLA service defaults to $B_{\text{vis}} = 3072$, $\gamma = 2$, camera weights $(2, 1, 0.5, 1)$ in front/right/back/left order, $b_{\min} = 4$, and $b_{\max} = 196$. It retains at most 16 observations, always including the first and current ones, with the remaining indices sampled uniformly without replacement from intermediate history and sorted chronologically. With these settings, short histories of at most three observations saturate the per-image caps; allocation becomes budget-limited from four observations onward. These inference defaults are distinct from the randomized training budgets and the diagnostic bounds below.

Table 8 is a deterministic synthetic diagnostic of the portable allocator, using $b_{\min} = 4$, $\gamma = 2$, the camera weights above (one for a single view), and each row's budget and cap. We draw change scores from $\mathcal{U}[0.05, 1)$ in table order using NumPy 2.0.2, PCG64 seed 20260926, then set the first retained image's score to one for every view. The allocator retains NumPy's default `argsort` tie ordering. The table reports images with different allocations and the mean per-image $|\Delta b|$, where $\Delta b = b^{\text{codec}} - b^{\text{pos}}$. All integer bounds and total allocations are checked; the four saturated rows are negative controls with identical allocations. The released diagnostic includes exact score arrays, allocations, and the allocator source snapshot. These are local synthetic calculations, not new navigation results; performance is evaluated separately in Table 10.

Table 8: **Allocation saturation.** Deterministic synthetic change-score diagnostic.

B_{vis}	N_{hist}	N_{view}	$b_{\max}$	ρ_{vis}	Images differing	Mean $\lvert\Delta b\rvert$ (tokens)
3072	2	1	196	7.84	0 / 2	0.0
3072	8	1	196	1.96	0 / 8	0.0
3072	15	1	196	1.04	0 / 15	0.0
4096	4	4	256	1.00	0 / 16	0.0
3072	16	1	196	0.98	3 / 16	8.0
3072	8	4	196	0.49	30 / 32	47.3
1024	16	1	196	0.33	14 / 16	31.4
2048	8	4	196	0.33	31 / 32	18.9
1024	16	4	196	0.08	57 / 64	7.9

B.8 VLA Data Processing and Training Configuration

Navigation sources are converted to a common multimodal JSONL schema containing an instruction, ordered image paths, and an eight-waypoint target. Habitat observations are rendered in front/right/back/left order; the first and current observations are always retained and intermediate history is sampled up to 16 steps. Discrete actions are simulated as 0.25 m forward motion, 15-degree turns, or padded stops and transformed into cumulative agent-local coordinates normalized by $(1.0, 0.433, 2.094)$ for (x, y, θ). Geometry-aware resampling reduces the raw forward-action bias and increases turns and stops. Embodied navigation QA uses the same image/conversation schema but omits the trajectory field and trains only the next-token objective.

Table 9: **Matched VLA training configurations.** BS: global batch size; AH LR: action-head learning rate.

Backbone	**GPUs**	**BS**	**Steps**	**Length**	**LR**	**AH LR**	**Vision**
Qwen3-VL-4B	32	256	$\sim$75.3k	8,192	5×10^{-6}	10^{-3}	tuned
Qwen3-VL-8B	32	256	$\sim$75.3k	8,192	5×10^{-6}	10^{-3}	tuned

Supervision and masking. We follow the trajectory-regression and vision-language co-training objective of Qwen-RobotNav (Qwen Team, 2026b), using the local embodiment scales and training configuration above. The MLP action head predicts a 24-dimensional vector containing eight (x, y, θ) waypoints. For a minibatch with navigation examples $\mathcal{B}_{\text{nav}}$, its loss is

$$\mathcal{L}_{\text{nav}} = \frac{1}{24|\mathcal{B}_{\text{nav}}|} \sum_{i \in \mathcal{B}_{\text{nav}}} \sum_{r=1}^{8} \|\hat{\mathbf{a}}_{i,r} - \mathbf{a}_{i,r}\|_2^2, \qquad \mathcal{L} = \mathcal{L}_{\text{nav}} + \mathcal{L}_{\text{NTP}}. \tag{10}$$

Both losses have unit weight. Position and heading have equal weight after normalization; the coordinate scales therefore determine their relative weight in physical units. The fixed eight-waypoint target includes padded stop positions near trajectory ends, and the MLP loss applies no additional terminal-waypoint mask. Navigation examples are excluded from next-token supervision. Conversely, QA and grounding examples have no action target and contribute only cross-entropy over labeled answer tokens; prompt and padding labels are ignored using the -100 label. Their placeholder trajectory tensors are excluded by the navigation-example mask rather than learned as zero-motion targets. An absent example type contributes zero to its corresponding loss.

Training scopes. The final recipe contains 19,277,381 effective training records (19.28M). The final 4B and 8B configurations in Table 9 share the one-epoch schedule and global batch size.

C VLA Components, Model Size, and Data Scaling

Codec context-allocation ablation. Table 10 compares positional allocation, our reimplementation of the Qwen-RobotNav observation interface (Qwen Team, 2026b), with the *Codec* training

Table 10: **Codec context-allocation ablation.** Full Val-Unseen results.

Size	Configuration	R2R-CE Val-Unseen				RxR-CE Val-Unseen			
		NE↓	OSR↑	SR↑	SPL↑	OSR↑	nDTW↑	SR↑	SPL↑
4B	Positional allocation	**3.91**	75.2	68.1	62.9	78.8	71.4	73.3	64.9
	Codec	4.04	**76.9**	**69.2**	**63.9**	**80.6**	**72.1**	**74.7**	**66.5**
8B	Positional allocation	3.88	79.4	69.5	62.7	80.9	70.9	74.4	65.1
	Codec	**3.57**	**80.8**	**71.6**	**65.1**	**82.7**	**73.1**	**77.0**	**67.6**

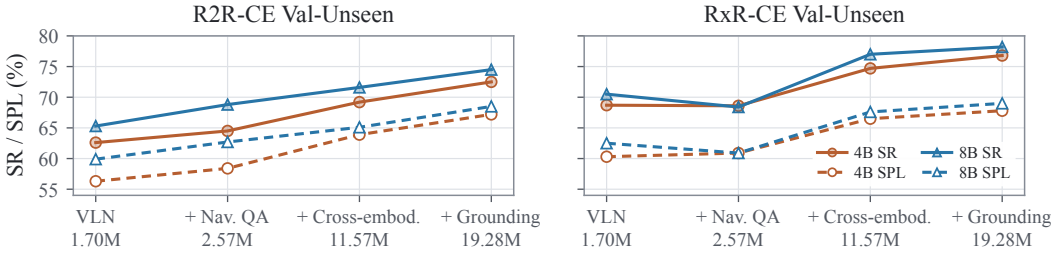


Figure 5: **Model and data scaling.**

configuration. Codec denotes change-aware allocation. Both 4B and 8B checkpoints are evaluated on the full R2R-CE and RxR-CE Val-Unseen splits. This table examines the trained execution policy beneath the harness; inference-budget controls are supplementary (Section B.7).

Codec improves over positional allocation at both model sizes. At 8B it leads on every metric: 2.1 SR and 2.4 SPL on R2R-CE, 2.6 SR and 2.5 SPL on RxR-CE, and 0.31 m lower NE. At 4B the gains are smaller, 1.1 SR and 1.0 SPL on R2R-CE and 1.4 SR and 1.6 SPL on RxR-CE, while positional allocation keeps a slightly lower NE (3.91 vs. 4.04 m). The gains are larger on RxR-CE at both sizes, consistent with its longer instructions and trajectories, whose retained histories more often exceed the visual budget and thus engage the allocator.

These are intermediate checkpoints trained before visual grounding was added. The final recipe used in the main comparison includes grounding and reaches 72.54/74.51 R2R-CE SR and 76.77/78.19 RxR-CE SR for 4B/8B, so the main results and this table reflect different training mixtures, not different evaluations of the same checkpoint.

Model and data scaling. Figure 5 plots 4B and 8B Codec policies across four cumulative training scopes on full Val-Unseen.

Cross-embodiment data matters more than model size: with it, the 4B model reaches 69.2 R2R-CE SR, surpassing the 8B model trained on VLN data only (65.3 SR). Because the added sources change data volume and exposure together, this comparison measures the cumulative recipe rather than the quality of any single source.

D Navigation Interface Implementation

The main method describes navigation capabilities independently of API naming. The primary `planner_vla_memory` profile instantiates these capabilities with eight registered operations (Table 12). The harness registry validates operation names and arguments before authorized dispatch, NavGPT Environment commits physical transitions, and harness state services maintain route and node records. The Planner cannot add capabilities through an instruction or directly mutate navigation state.

D.1 Implemented Responsibility Boundaries

The seven boundaries in Figure 6 describe responsibilities across the NavGPT-3 system. They separate state ownership and visibility, independently of the runtime's logical threads and the deployment of model services. The main task thread invokes Planner reasoning, VLA execution, and other tools as its current operation. Context construction is distributed across observation capture, route-state maintenance, result serialization, and the provider conversation. Scenario and event monitors are example tasks rather than required components.

Responsibility boundary	**Owned state or decision**
Thread coordination	Runtime: schedule threads, manage permissions, and control motion authority.
Semantic control	Planner: interpret evidence, select operations, accept the endpoint.
Context materialization	Harness: select and render model-visible images and state.
Persistent navigation state	Harness: retain route references, nodes, and annotations.
Capability registry	Harness: validate requests and dispatch authorized operations.
Executable navigation skill	VLA: predict waypoints using stateful visual history.
Physical substrate	Environment: commit motion, sensing, and collision outcomes.

Figure 6: Responsibility boundaries in the NavGPT-3 system.

D.2 Capabilities and Execution Contracts

Capability	**Navigation purpose**	**Interface contract**
Local observation	Ground landmarks and inspect the current surroundings	Relate forward or panoramic appearance to the current viewing direction and pose.
Instruction delegation	Execute an extended route with the learned policy	Pass the requested navigation instruction; preserve temporal state; return executed-route evidence and status.
Route inspection	Relate current progress to previously encountered places	Expose the observed map and ordered spatial references without representing unknown space as free.
Spatial revision	Revisit a recorded place or correct a local endpoint error	Resolve a route reference or relative motion request into validated movement and report its realized outcome.
Semantic annotation	Retain an interpretation of a visited place	Attach semantic metadata to existing spatial state while preserving its physical provenance.
Task termination	Accept the final navigation state	Make episode completion explicit and distinct from the end of a delegated rollout.

Table 11: Navigation capabilities and their interface contracts.

Execution and return semantics. Instruction delegation runs as the main task thread's active operation: it passes the Planner-selected navigation instruction to the stateful VLA and executes predictions until an advisory stop, the delegation cap, or an episode limit. The Planner may supply the task instruction or an instruction for a subtask. The observed map distinguishes unknown from observed free space, and ordered nodes give stable return references. Semantic annotations attach names, captions, or instruction clauses to existing nodes without changing measured topology. Because episode termination may include a return to a selected node, capability restrictions cover that embedded return argument as well as the dedicated return operation. The final profile excludes primitive control, arbitrary historical-image recall, and detector tools.

D.3 Harness Development Through Recursive Self-Improvement

We formulate recursive self-improvement (RSI) of the harness as a KDLoop-style tree search over executable harness versions (Zhou et al., 2026a). Each node specifies the Planner briefing, available

Registered operation	State effect	Model-visible return
`observe_forward()`	None	Current forward RGB for fine semantic inspection.
`observe_panorama()`	Updates observed local coverage, not pose	Four labelled views with bearings, clearance, pose displacement, and node information.
`observe_map()`	Updates observed coverage, not pose	Observed-only top-down map, map status, and route-ordered node table.
`navigate_by_instruction()`	Executes a stateful VLA leg	Up to eight sampled route keyframes, endpoint panorama, optional observed map, and execution status.
`navigate_relative()`	Executes a validated local turn and translation	Realized motion status, endpoint panorama, and nearby node candidates.
`navigate_to_node()`	Returns to a visited node or latest-rollout waypoint	Endpoint panorama, observed map, return status, and updated node table.
`annotate_node()`	Attaches semantic metadata to an existing node	Annotation status and updated node table.
`terminate_episode()`	Optionally returns to a selected node, then stops	Terminal or retrace status; this is the irreversible episode-ending operation.

Table 12: Implementation of the navigation capabilities.

tools, persistent spatial state, and model-visible observation and execution-return formats; each edge is a candidate revision. THINK proposes edits using the current harness and accumulated episode evidence; CRITIC checks them against interface constraints and earlier failures; EXPERIMENT implements and scores the surviving branches; and DISTILL records confirmed findings, failed edits, and unresolved hypotheses. Scores guide selection of the next incumbent, while REFLECT revisits its intervention families when progress stalls. Search operates at development time: the chosen interface is fixed during evaluation, without online Planner or VLA weight updates. Figure 7(a) summarizes the research loop; panel (b) illustrates three intervention families: tool interfaces, visual context, and execution returns, branching from a common harness, undergoing revisions, and merging the selected branch into the next incumbent. These are schematic search branches, not recovered historical parent links.

The search evaluates candidate harnesses with Claude Opus 5 on one fixed 100-episode subset sampled from the R2R-CE training split, whose scenes are disjoint from Val-Unseen, using success rate as the selection signal and episode traces to diagnose failures. Figure 7(c) displays the 14 completed 100-episode harness configurations with $\mathrm{SR} \geq 50\%$, sharing the same episode IDs; the accompanying catalogue groups completed pilots, standalone-policy controls, later interface studies, and full-split evaluations separately. The frontier is the cumulative maximum in this display order, while dashed orange segments mark decreases between adjacent measured candidates.

The archived configurations trace the harness's development: front-view observation and primitive stepping score 66% SR; VLA delegation scores 72% with the 4B policy and 76% with the 8B policy; and combining VLA execution with spatial tools reaches a development best of 81%. These scores are measured on training episodes and are not comparable to the Val-Unseen results in the main text. Early topology scores 54% and the consolidated configuration scores 77%. One interface review found that the briefing described local movement although its tool was absent from the registry; later revisions registered movement and replaced the stitched panorama with four separate 512-pixel views. Iteration labels enumerate the archived candidates in display order. The resulting tool and visual-prompt studies are reported in Table 1 and Figure 4, and the final eight-operation interface in Table 12.

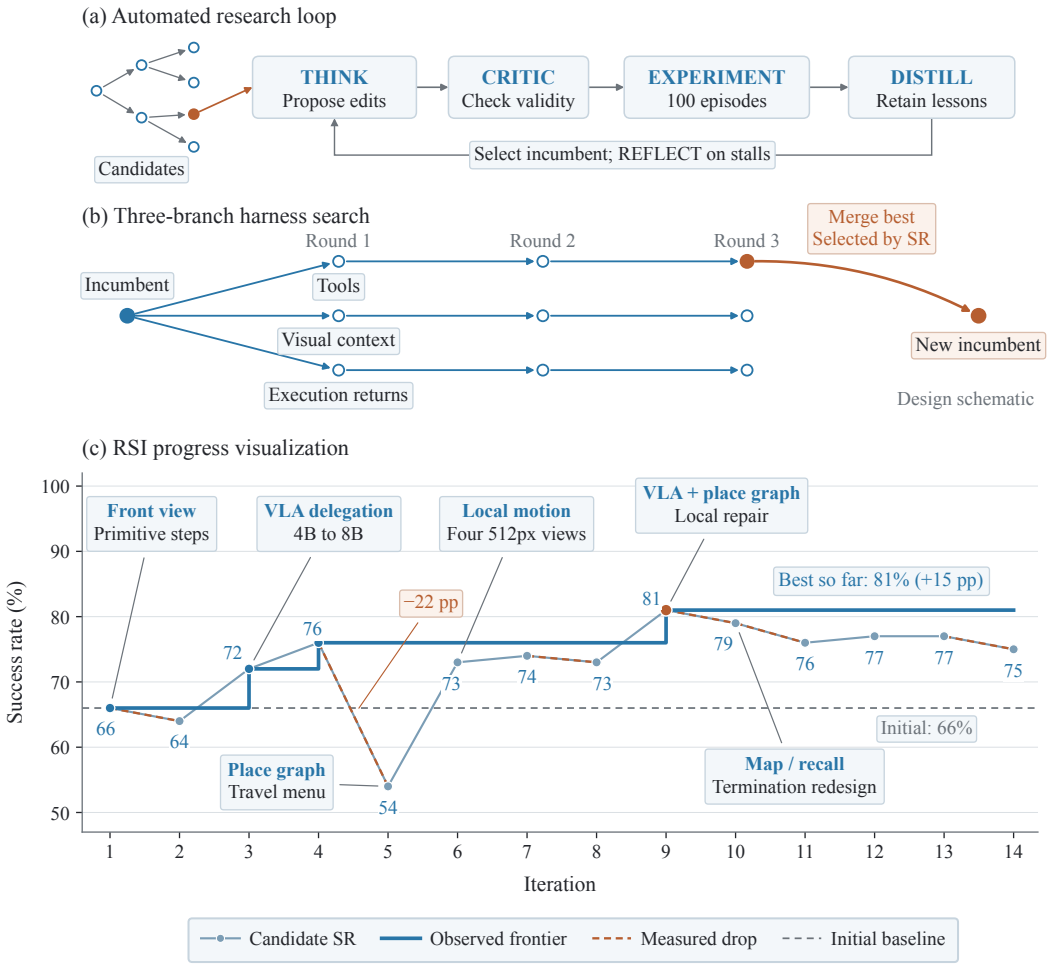


Figure 7: Automated research loop, candidate branching, observed frontier, and measured score drops. Annotations identify selected interface revisions.

E Evaluation Coverage and Validation

E.1 R2R Ablation Subset

The 100-episode ablation set follows the public OpenNav subset (Qiao et al., 2025). We match its ordered episode identifiers against the saved experiment manifest and its instructions and scenes against the saved execution records: all 100 agree. Joining these identifiers to the R2R-CE v1.3 Val-Unseen release yields identical instruction text, scene, start position, reference path, and stored geodesic distance for every selected episode. Table 13 compares these episode attributes with the full 1,839-episode split. Numeric entries give mean ± standard deviation. The absolute standardized mean difference (SMD) compares the subset with its disjoint 1,739-episode complement, using the square root of their equally weighted variances; D is the maximum difference between the subset and full empirical cumulative distributions. Directional mentions count `left`, `right`, and forms of `turn`; reference-path turns are horizontal direction changes of at least 30°, rather than executed primitive turns. These are interpretable task-complexity proxies, not a complete semantic-complexity measure.

Path distance and directional mentions have small mean differences; instructions in the subset are shorter and have a narrower length distribution. The subset covers 10 of 11 scenes, omitting a scene containing 18 full-split episodes, and its scene-frequency total-variation distance from the full split

Table 13: R2R ablation-subset coverage and distributions.

Statistic	Full	Subset	|SMD|	D
Episodes	1,839	100	–	–
Scenes	11	10	–	–
Unique reference trajectories	613	97	–	–
Geodesic distance (m)	8.90 ± 2.67	8.99 ± 3.06	0.033	0.072
Instruction length (words)	26.65 ± 11.48	24.68 ± 8.28	0.207	0.103
Directional mentions	2.14 ± 1.80	2.20 ± 1.75	0.034	0.040
Reference-path turns	1.56 ± 1.08	1.46 ± 1.21	0.095	0.100

is 0.097. In particular, one scene accounts for 3% of the subset versus 9.63% of the full split. A 10,000-draw, without-replacement reference calculation gives a scene-frequency Pearson statistic of 8.14 and a tail fraction of 0.567. This calculation calibrates the observed imbalance against random subsets of the same size; a non-small tail fraction does not establish representativeness, especially for a curated subset. The public OpenNav and v1.3 files also differ in the initial headings of all 100 selected episodes, and we fix them in our evaluation. Thus this comparison establishes coverage and similarity of the listed attributes, not full execution-protocol or performance equivalence. Full-split results remain the basis for benchmark comparisons.

E.2 Matched ObjectNav Evaluation

The validation in Table 14 runs Qwen-RobotNav 8B and NavGPT-3 on exactly the same 500 episodes for each dataset, with identical scene versions, sensing, motion budgets, collision handling, and stop/success rules. It is a system comparison; the complete system additionally uses its hosted Planner. Selection is fixed before observing either system's outcomes. SR and SPL are percentages. ΔSR vs. full is the 500-episode SR minus the full-split SR published for Qwen-RobotNav 8B, in percentage points (-1.0 on MP3D, $+1.2$ on HM3D v2); it indicates how closely the selected episodes track the full split. NavGPT-3 has no full-split run, so its entry is a dash.

Table 14: ObjectNav comparison on identical episodes.

Dataset	System	N	SR	SPL	ΔSR vs. full
MP3D	Qwen-RobotNav 8B	500	47.8	17.5	-1.0
MP3D	NavGPT-3	500	59.2	25.2	–
HM3D v2	Qwen-RobotNav 8B	500	72.4	33.1	$+1.2$
HM3D v2	NavGPT-3	500	75.3	40.8	–

E.3 EXPRESS Judge Validation

We compare the GPT-6 Astra judge with the GPT-4o-mini judge used by prior EXPRESS-Bench results on the same saved answers and final observations, scored against human judgments. Table 15 reports agreement on 50 human-labeled answers: Astra agrees with the human label on 46 and GPT-4o-mini on 36, a 20.0-point difference. Without item-level pairing, the exact McNemar p depends on how the two judges' errors overlap; for these totals it lies between 0.002 and 0.031. This issue applies to open-answer EXPRESS scoring; HM-EQA and MT-HM3D multiple-choice accuracy uses the reference option.

Table 15: EXPRESS judge agreement with human labels.

Judge	Labeled N	Agreement (%)
GPT-4o-mini	50	72.0
GPT-6 Astra	50	92.0
Astra − 4o-mini		+20.0

F Real-World Deployment

F.1 Deployment Setup

Robot and sensing. We use the Unitree Go2 platform and onboard VLA inference. NavGPT VLA receives the robot's front-facing RGB stream, with visual context allocated over the retained single-view history. This differs from the four-view simulator configuration in Section 3.2.1. The onboard LiDAR supplies geometric hazard detections to our runtime monitor; it is not an additional input to the VLA.

Remote and onboard inference. In the remote configuration, the robot compresses and transmits observations together with the Planner-selected navigation instruction to a server, which predicts waypoints and returns them for execution. The onboard configuration runs the 4B NavGPT VLA on an NVIDIA Jetson Thor with FP8 quantization and TensorRT acceleration, following Qwen Team (2026b). Both configurations expose the same waypoint interface to the harness. Onboard inference removes the network round trip for VLA predictions; the hosted Planner remains a separate service. Model inference, communication, and physical execution contribute separately to end-to-end response time and are measured separately below.

Harness and runtime integration. The harness manages observations, spatial state, tool calls, and execution returns, while the runtime above it coordinates independently paced reasoning, execution, and monitoring. Moving the VLA between the server and the robot does not change this division of responsibilities. The deployment contract requires task motion to pass through the command gateway under the active runtime grant. The device evaluation tests whether a LiDAR event can revoke that grant without waiting for Planner inference, and whether execution resumes only after fresh sensing and reauthorization (Section A.6).

F.2 Glass-Door Execution Visualization



Figure 8: Recorded glass-door execution. The outlined fourth panel highlights the near-door forward pause, followed by lateral correction and continued passage.

Approach and pause. Figure 8 follows a recorded instruction to enter a shop. Panels 1–3 show the approach from an indoor concourse toward the already-open glass doorway, beside a freestanding sign. Panel 4 (5.48 s) highlights the interruption of forward progress near the glass. Sensor poses place the pause and correction between 5.48 and 7.36 s; lateral movement continues, so the robot is not completely stationary.

Correction and continuation. Panel 5 (6.85 s) shows the lateral adjustment. Forward motion resumes in panel 6 (7.71 s), the robot passes the entrance in panel 7, and panel 8 shows it inside the shop without visible contact with the glass door. The connected points illustrate an interrupted approach followed by revised paths through the opening: the task remains to enter the shop while the immediate route changes.

Connection to runtime control. This sequence illustrates the pause–revise–continue pattern targeted by the runtime: suspend the active motion operation, inspect fresh observations, and resume with an updated path under renewed authorization.

F.3 Runtime Evaluation

Controlled inputs. Conditions share the forward RGB stream, LiDAR detector and timestamped events, speed limit, routes, and manufacturer safeguards. The VLA placement and precision are held

fixed within each runtime comparison, together with the single-view adapter, checkpoint, Planner briefing, detector threshold, and firmware. An independent operator stop remains active; operator intervention counts as a failed trial.

Table 16: Go2 scenarios and measurements.

Scenario	Controlled event	Primary measurements
A: Route revision	A detour or endpoint correction becomes necessary during an extended rollout; vary event timing relative to Planner inference.	Completion SR; event-to-authorized-repair time; route efficiency; Planner calls; execution idle time.
B: LiDAR protection	A soft obstacle enters a calibrated protected region during motion, including while the Planner is busy. Use matched harmless passages as negatives.	Missed and false stops; trigger-to-halt p50/p95; post-trigger distance; minimum clearance; operator interventions.
C: Priority and recovery	Trigger a hazard during repair or near VLA completion; delay a model reply, replay an obsolete command at the gateway, then clear the hazard.	Accepted stale commands; event ordering; halt latency; unauthorized resumption; correct resumption and task completion.

Comparisons. We compare VLA-only and Planner-with-tools baselines with a matched 2×2 runtime comparison: sequential versus overlapping Planner–VLA execution, crossed with queued versus priority monitor handling. All four runtime configurations share the same models, tool schemas, route evidence, and LiDAR detections. Queued handling processes a detected event at the next ordinary control boundary; priority handling immediately revokes the active grant through the command gateway. Keeping a priority monitor in the sequential condition tests whether protection can be explained by a simple independent stop gate, while the overlap contrast tests progress during reasoning. The final runtime configuration combines overlap with priority handling. These controls separate detector quality, scheduling, and authority transfer.

Table 17: Go2 runtime results.

	A: Revision		B: Protection		C: Recovery	
Condition	SR	Repair (s)	Missed/N	Halt p95 (ms)	Stale/N	Resume SR
VLA-only	60.0	–	–	–	–	–
Planner + tools	70.0	14.2	9/30	2150	11/30	53.3
Sequential + queued	73.3	12.8	7/30	1840	8/30	60.0
Sequential + priority	76.7	11.5	2/30	420	3/30	70.0
Overlapping + queued	76.7	9.6	5/30	1310	6/30	66.7
Overlapping + priority	83.3	7.9	0/30	180	1/30	80.0

Results. Overlap and priority contribute separately (Table 17). Overlapping Planner reasoning with VLA execution shortens repair from 12.8 to 9.6 s under queued handling and from 11.5 to 7.9 s under priority handling, because execution continues while the Planner reviews the route. Priority handling reduces missed stops from 7/30 to 2/30 in the sequential runtime and from 5/30 to 0/30 with overlap, and lowers p95 halt latency from 1840 to 420 ms and from 1310 to 180 ms, respectively. Combining both yields the highest revision SR (83.3 vs. 73.3 for sequential queued handling), the fewest accepted stale commands (1/30), and the highest resume SR (80.0).

Event timing and outcomes. We log observation compression, network transfer, and VLA inference durations separately, and record sensor acquisition, detector trigger, review release/completion, grant revocation, last accepted nonzero command, stop-command dispatch, physical halt, reauthorization, and resumed motion on a synchronized monotonic timeline. A successful SDK return alone is not a physical halt. Halt is defined from independently verified motion as speed below a preregistered threshold sustained for a fixed interval. A stale command is one carrying a revoked generation; Stale/N counts trials in which such a command was accepted. Resume success requires fresh sensing, a new grant, and completion of the remaining task without operator intervention.

Trial allocation and uncertainty. Five pilot trials per condition and scenario validated logging and are excluded from the comparison. The evaluation uses five routes, three event timings, and two repetitions per condition and scenario (30 trials), randomized in matched blocks. Detection false negatives and false positives are annotated independently of the detector's output.

G QUALITATIVE NAVIGATION RESULTS

Each case includes its complete instruction. Maps T1 onward show position after each tool return; numbered views sample the first VLA rollout. Solid lines trace foreground control and dashed lines mark waiting. Planner quotations are recorded; VLA descriptions are editorial. Token counts include cached input and output.

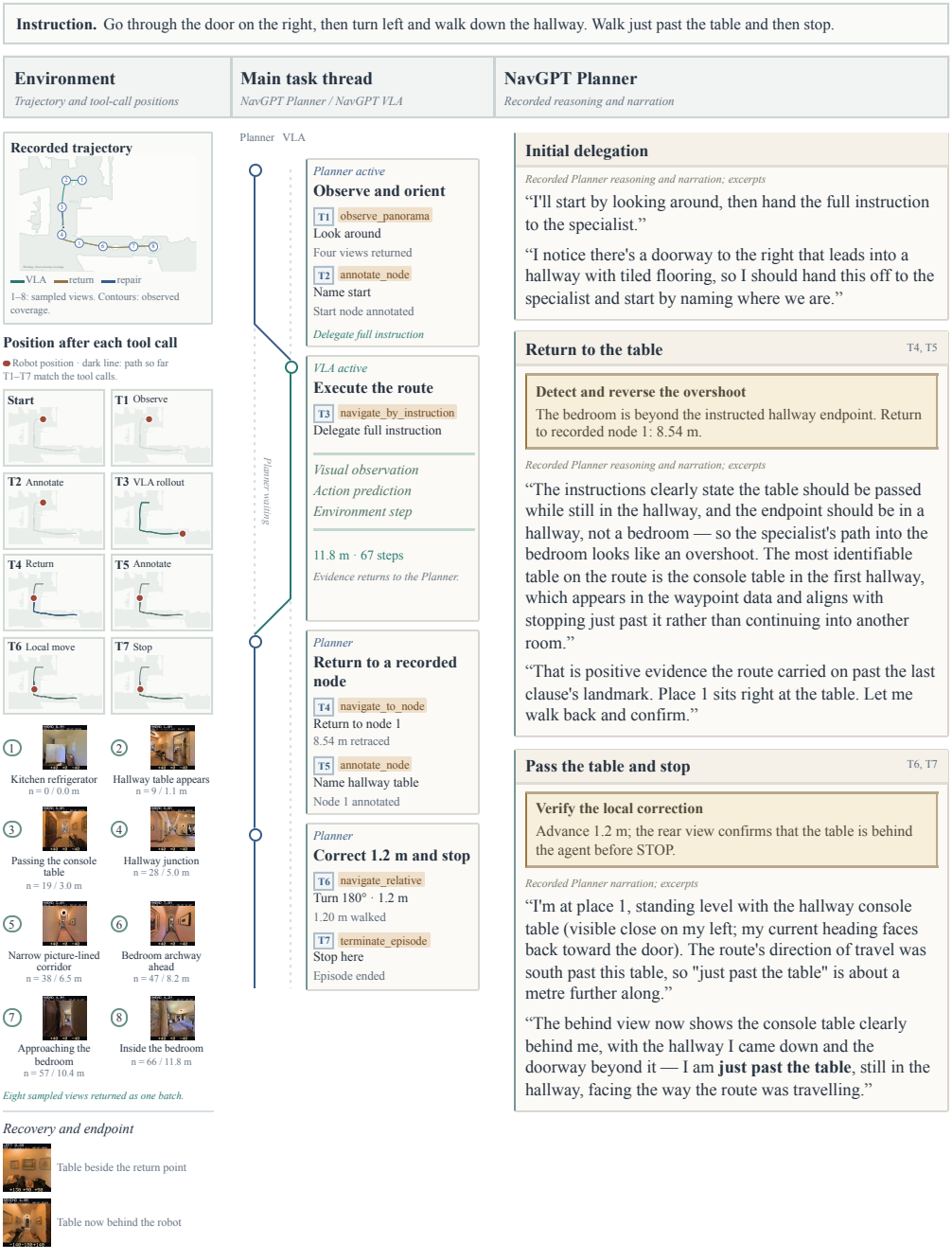


Figure G.1: **Backtracking to an overshot endpoint.** Claude Opus 5 on R2R-CE, episode 144. The Planner identifies the hallway table before the VLA's bedroom stop, retraces 8.54 m to node 1, and verifies a 1.2 m local correction before stopping.

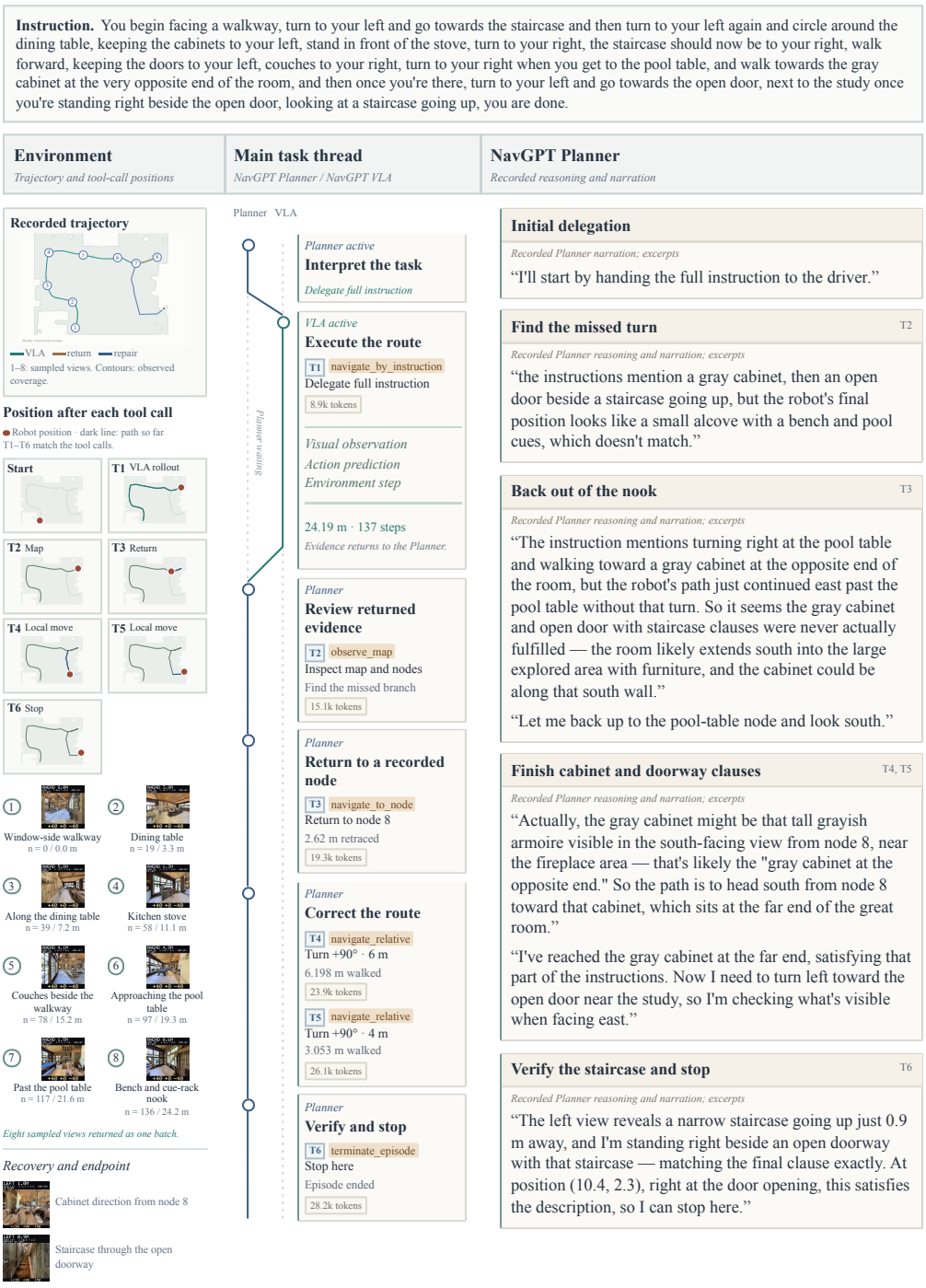


Figure G.2: **Completing omitted route clauses.** Claude Opus 5 on RxR-CE, episode 2523. The Planner returns from a dead-end nook to node 8, then uses relative motion to reach the gray cabinet and the open doorway with a rising staircase.

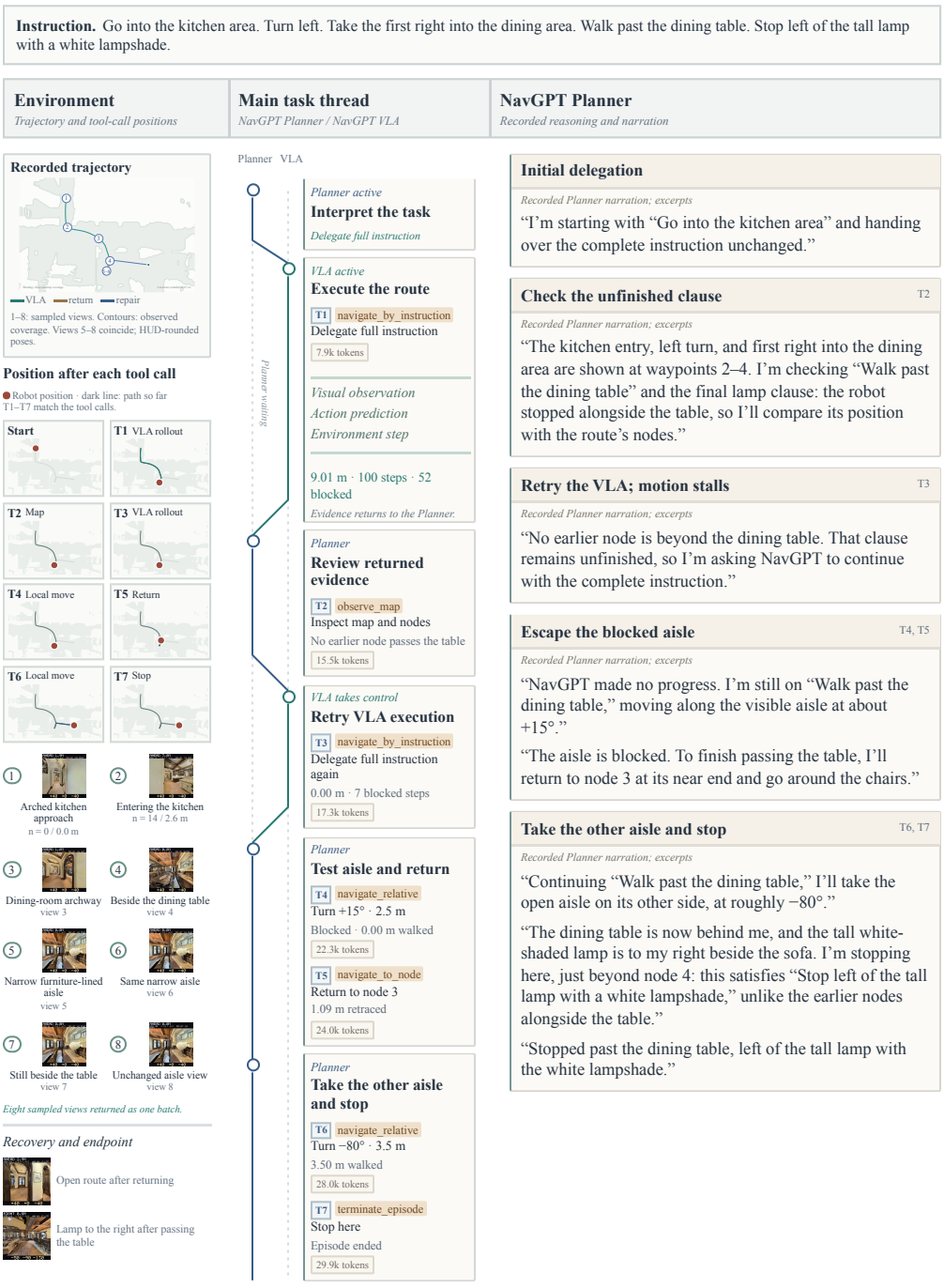


Figure G.3: **Recovering from a blocked aisle.** GPT-6 Astra on R2R-CE, episode 43. After repeated VLA execution and a direct motion attempt stall, the Planner returns to node 3, traverses the other aisle, and stops beyond the table beside the lamp.



Figure G.4: **Resuming delegation after unsuccessful local correction.** Claude Opus 5 on RxR-CE, episode 5782. With the stair clause still unfinished, the Planner delegates the full instruction again. The second rollout reaches the upstairs bathroom, which the Planner verifies before stopping.