WORLDPIXELNAV: DEPLOYMENT-ORIENTED PIXEL-GOAL NAVIGATION WITH FUTURE-STATE-AWARE TRAJECTORY GENERATION AND HIERARCHICAL CORRECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Pixel-goal navigation bridges high-level reasoning and low-level control in vision-language navigation by selecting an image pixel as the next local target. Existing imitation-based planners, however, treat the pixel goal as a static target and do not model the spatial consequences of their motion, leading to trajectory drift and collision. We propose WorldPixelNav, a deployment-oriented pixel-goal navigation framework with future-state-aware trajectory generation and hierarchical correction. WorldPixelNav converts the pixel goal into a robot-centric planar coordinate, encodes depth observations, motion history, this coordinate, and a fixed instruction into shared conditional memory, predicts a future spatial latent from this memory, and decodes the latent into continuous waypoints. Training combines trajectory imitation, future-state prediction, and deployment-oriented supervision. During execution, safety verification triggers trajectory reconnection, in-place re-observation, or historical-viewpoint recovery without requesting a new high-level pixel goal. WorldPixelNav achieves 96.48% success rate and 0.07% collision-episode rate in unseen simulated scenes, and 83.3% and 6.7% in unseen real-world scenes.

1 INTRODUCTION

Pixel-goal navigation, which selects an image pixel as the next local target, bridges high-level reasoning and low-level control in vision-language navigation.(Bao et al., 2026; Gong et al., 2026) As VLN moves toward household service, public-space guidance, and human–robot collaboration, reliable operation in unseen environments becomes essential for real-world deployment.

Existing VLN systems mainly adopt end-to-end or decoupled architectures (Krantz et al., 2020). End-to-end methods directly map instructions and observations to actions or trajectories, coupling semantic reasoning with high-frequency control in the same inference loop, which incurs high latency and makes real-time closed-loop control difficult (Majumdar et al., 2020; Hong et al., 2021; Chen et al., 2021). Decoupled methods instead split the system into two loops: a vision–language model that performs high-level semantic decisions at low frequency, and a local planner that produces continuous motion at high frequency (Wei et al., 2026; Deng et al., 2026). This separation improves control efficiency, but introduces an interface between discrete decisions and continuous motion. Among possible interfaces, the pixel goal is widely used: the high-level model selects a pixel in the current observation, and the low-level planner converts it into executable motion.

This paper focuses on the low-level side of this interface: given a pixel goal from the high-level model, how to plan a safe and executable local trajectory toward it. Despite their differences, existing pixel-goal planners face two key issues (Cai et al., 2024; 2026; Wei et al., 2026; Huang et al., 2026). The first is a **training–deployment gap**: these planners are trained by imitating demonstrated trajectories, which only learns a mapping from the current observation and pixel goal to a trajectory. The model is not trained to predict the future spatial state change caused by its own actions, nor does it learn the key principles behind successful navigation. Consequently, in unseen scenes, the predicted trajectory often drifts or collides with obstacles (Cai et al., 2024; 2026; Wei et al., 2026;

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.

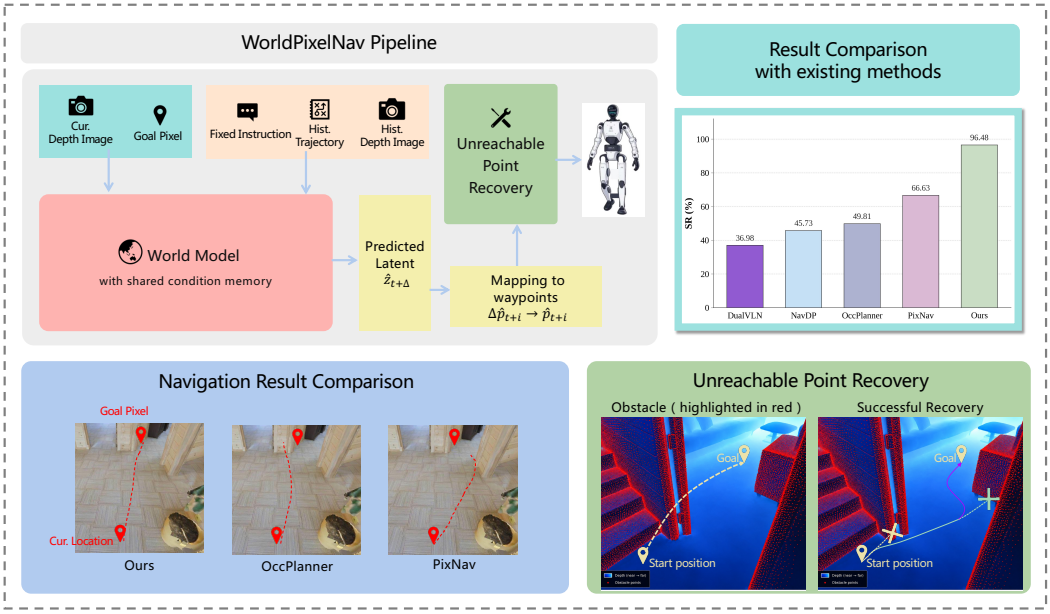

Figure 1: Overview of WorldPixelNav. The world-model backbone predicts a goal-conditioned future spatial latent from depth, a pixel goal, motion history, and a fixed instruction, and decodes it into continuous waypoints. Hierarchical correction repairs colliding trajectories. Together, they reduce drift and collision in unseen scenes and increase the proportion of episodes that complete the pixel-goal stage under a single high-level grounding. The right and bottom panels show quantitative comparisons, representative trajectories, and recovery from a collision-inducing prediction.

Huang et al., 2026). The second issue is **cumulative inference cost introduced by repeated re-planning**. To mitigate drift and collision, many systems execute only a short trajectory segment before re-observing and re-predicting, requiring multiple model calls to complete a pixel-goal stage (Bao et al., 2026; Wei et al., 2026; Deng et al., 2026).

To address these two issues, we propose WorldPixelNav, a deployment-oriented pixel-goal navigation framework with future-state-aware trajectory generation and hierarchical correction. Using the Cosmos Predict 2.5 (NVIDIA, 2025) backbone as a latent feature extractor, WorldPixelNav encodes the pixel goal, current and historical depth observations, historical motion increments, and a fixed instruction into a shared conditional space, predicts a future spatial state change from this memory, and decodes it into continuous waypoints. During training, deployment-oriented supervision and future spatial state change prediction jointly shape the shared representation toward execution outcomes. During execution, a hierarchical correction mechanism repairs trajectories that would cause a collision. Together, they reduce drift and collision in unseen scenes and raise the proportion of pixel-goal navigations completed under a single high-level grounding.

The main contributions of this paper are as follows.

- We propose WorldPixelNav for pixel-goal navigation, introducing world-model spatial modeling into pixel-goal trajectory planning. The framework learns to predict future spatial states, making generated trajectories more consistent with future spatial changes.
- We propose a deployment-oriented learning objective and a hierarchical correction mechanism to bridge the training–deployment gap. The learning objective augments trajectory imitation with deployment-oriented supervision, and the correction mechanism repairs colliding trajectories during execution. Together, they reduce drift and collision and raise the fraction of pixel-goal navigations completed under a single high-level grounding.
- We construct two evaluation protocols for unseen environments: a simulated protocol derived from InternVLA-CE-R2R, and a real-robot protocol collected on a physical platform. On both protocols, WorldPixelNav completes the vast majority of episodes under a single high-level grounding, achieving 96.48% success rate and 0.07% collision-episode rate in simulation, and 83.3% and 6.7% on a real robot in unseen real-world scenes.

2 Related Work

Vision-Language Navigation. Vision-language navigation requires an agent to perform goal-directed motion in an unknown environment using first-person observations according to a natural-language instruction. Existing methods mainly follow end-to-end or decoupled designs. End-to-end models use a unified model to map observations and instructions directly to actions or waypoints, as in VLN-BERT, HAMT, and FantasyVLN (Majumdar et al., 2020; Hong et al., 2021; Chen et al., 2021; Zuo et al., 2026). Recent work has extended VLN toward generalist, video-conditioned, adaptive, and foresight-aware navigation, including NaviLLM, RoomTour3D, Uni-NaVid, SAME, NavQ, JanusVLN, and OctoNav (Zheng et al., 2024; Han et al., 2025; Zhang et al., 2025; Zhou et al., 2025; Xu et al., 2025; Zeng et al., 2026; Gao et al., 2026). Decoupled systems separate high-level semantic decision-making from low-level motion planning. DualVLN combines low-frequency vision-language reasoning with high-frequency continuous trajectory planning, while SpaceVLN organizes navigation through high-level online spatial-cognitive memory and staged closed-loop reasoning (Wei et al., 2026; Deng et al., 2026). Recent evaluation further highlights the physical and visual gaps that arise when VLN agents are deployed with real embodiments (Wang et al., 2025). Our method follows the decoupled design and focuses on the collision avoidance and pixel-goal navigation accuracy of the low-level trajectory planner (Chen et al., 2022; An et al., 2023; Wang et al., 2023).

Pixel-Goal Navigation. Pixel-goal navigation refers to low-level trajectory planning conditioned on a pixel selected by a high-level model. The high-level model chooses a pixel in the current observation, and the low-level planner converts it into executable motion. Existing methods differ mainly in how this pixel interface is used. PixNav (Cai et al., 2024) learns a pixel-guided navigation skill that maps visual observations and a pixel goal to low-level actions; DualVLN and NavDP generate continuous local trajectories from pixel goals (Wei et al., 2026; Cai et al., 2026); Goal2Pixel back-projects navigable pixels into 3D waypoints (Bao et al., 2026); and OccPlanner constrains trajectory planning using local 3D occupancy information (Huang et al., 2026). These existing methods share a limitation: training relies on static imitation learning, with supervision concentrated on the mapping from observed inputs to demonstrated trajectories, and does not model the future spatial state. As a result, navigation accuracy and obstacle-avoidance capability can degrade in out-of-distribution scenes. Our method jointly addresses these two problems through world-model dynamic latent modeling, deployment-oriented supervision, and hierarchical correction during execution.

Combining World Models with VLN. World models provide future information for navigation by predicting changes in future space (Ha and Schmidhuber, 2018; Hafner et al., 2020; 2025; Hansen et al., 2024). In vision-language navigation, prior work has used world models to predict future observations or states to assist long-horizon planning. Navigation World Models uses a generative model to predict future visual observations as a basis for planning (Bar et al., 2025), while NavMorph maintains an evolving contextual world model for navigation in continuous environments (Yao et al., 2025). NavWAM jointly predicts future observations, goal progress, and action chunks in a shared latent sequence so that its predictions can be used directly for control (Azuma et al., 2026). WAM-Nav jointly models future visual prediction and long-horizon actions in a latent world-navigation model (Yang et al., 2026). FutureNav unifies action prediction, inverse dynamics, forward dynamics, and future spatial-state generation within a vision-language model. NavWM jointly models latent-world reasoning, action prediction, and visual generation (Mei et al., 2026). These studies show that world models can help navigation systems model future spatial-state change (Perincherry et al., 2025). However, these methods navigate mainly according to target objects or language instructions. In contrast, WorldPixelNav applies world-model spatial modeling to continuous pixel-goal trajectory planning and generates a safe trajectory toward the pixel goal.

3 Method

3.1 Task Definition and Overall Framework

We study low-level trajectory planning conditioned on a single pixel goal. At navigation time t, the low-level planner receives a pixel goal $g_t^{\text{pix}} = (u_t, v_t)$, a depth sequence $D_{t-H:t}$, historical planar-motion increments $A_{t-H:t-1}$, and a fixed instruction c_0. The low-level planner outputs a sequence of W waypoints. Each waypoint is represented by a planar displacement increment

$\Delta\hat{p}_{t,i} = (\Delta\hat{x}_{t,i}, \Delta\hat{y}_{t,i})$ and a yaw increment $\Delta\hat{\psi}_{t,i}$; the two are accumulated separately to obtain the waypoint position and heading:

$$\hat{p}_{t,i} = \hat{p}_{t,i-1} + \Delta\hat{p}_{t,i}, \qquad \hat{\psi}_{t,i} = \hat{\psi}_{t,i-1} + \Delta\hat{\psi}_{t,i}, \qquad i = 1, \ldots, W. \tag{1}$$

The goal is to generate a continuous local trajectory that reaches the target while avoiding collisions. The system does not rely on a prior global map.

The overall framework consists of two modules. The depth-conditioned world-navigation model receives depth, goal coordinate, motion increments, and fixed instruction in shared conditional memory, predicts a future spatial-state latent variable, and then generates a continuous local planar trajectory according to that future spatial state. At inference time, a closed-loop executor performs safety checks on the predicted trajectory and, when the trajectory would cause a collision, successively attempts trajectory reconnection, in-place re-observation, and historical-viewpoint recovery in increasing order of cost.

3.2 Depth-Conditioned World-Navigation Model

We use the Cosmos Predict 2.5 backbone as a latent feature extractor. The model inputs include a depth sequence and historical motion increments within the time window, a pixel goal, and a fixed instruction. It sequentially performs pixel goal conversion, conditional encoding, future spatial-state prediction, and trajectory generation.

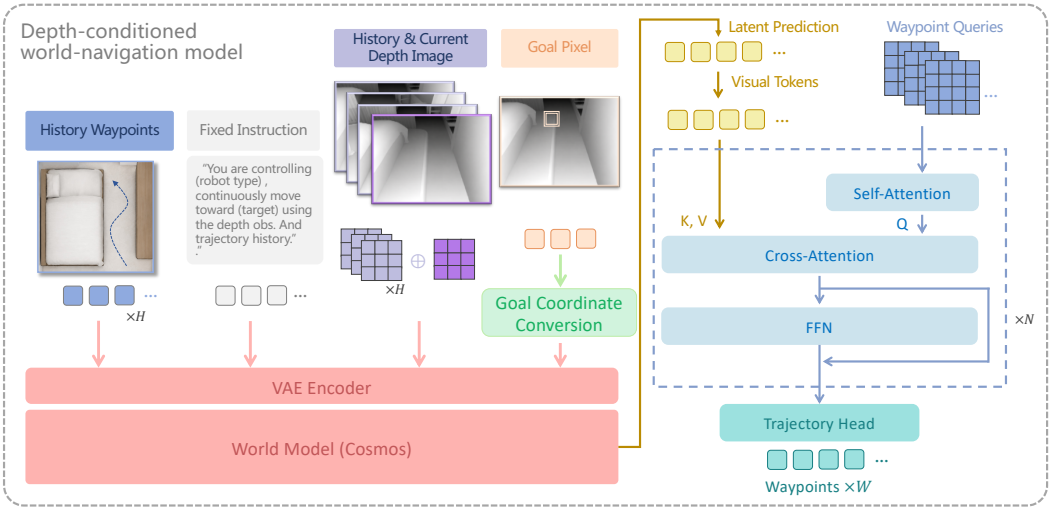

Figure 2: Architecture of the depth-conditioned world-navigation model. A fixed instruction, the current and historical depth observations, historical motion increments, and the goal coordinate are encoded into shared conditional memory. The world-model backbone first predicts a future spatial latent. Learnable waypoint queries then interact through self-attention and cross-attend to the predicted latent before the trajectory head outputs a sequence of continuous waypoints.

Input and Pixel-Goal Conversion. The inputs are a depth-observation sequence $D_{t-H:t}$, historical planar-motion increments $A_{t-H:t-1}$, a fixed instruction c_0, and a pixel goal. Let t_0 be the frame in which the pixel goal $g_{t_0}^{\text{pix}} = (u_{t_0}, v_{t_0})$ is selected. Its depth d_{t_0} is read from the corresponding depth frame. If the center depth is invalid, the median of valid depths in a local neighborhood is used. With camera intrinsics K, the pixel and its depth are back-projected into the camera frame:

$$p_{t_0}^{c} = d_{t_0} K^{-1} [u_{t_0}, v_{t_0}, 1]^{\top}. \tag{2}$$

The camera pose at the selection frame transforms this point into a persistent spatial target:

$$g^{w} = T_{t_0}^{w \leftarrow c} [p_{t_0}^{c}; 1]. \tag{3}$$

The same pixel-depth back-projection and pose transformation are used during training and inference, in simulation and on the physical robot. At each subsequent planning call, the persistent target

is expressed relative to the current robot pose:

$$g_t^{b,xy} = \Pi_{xy}\big((T_t^{w\leftarrow b})^{-1} g^w\big)\,. \tag{4}$$

Here $T_{t_0}^{w\leftarrow c}$ is the camera-to-world transform at the selection frame and $T_t^{w\leftarrow b}$ is the robot-to-world transform at time t. The planner receives the normalized form

$$\bar{g}_t^{b,xy} = \mathrm{clip}\left(\frac{g_t^{b,xy}}{s_g}, -1, 1\right), \tag{5}$$

where s_g is the position normalization scale. The normalized coordinate $\bar{g}_t^{b,xy}$ is used as the model condition, while the metric coordinate $g_t^{b,xy}$ is retained for loss computation and evaluation.

Conditional Encoding. The model encodes four types of conditions into shared conditional memory $C_t = [e^c; e_t^g; e_t^P; e_t^V]$.

- Depth token. The depth sequence $D_{t-H:t}$ is encoded by the depth encoder E^D into visual token e_t^V, which represents current and recent local geometry.
- Motion token. Historical planar-motion increments $a_j = (\Delta x_j, \Delta y_j, \Delta\psi_j)$ for $j \in [t-H, t-1]$ are encoded by the motion-history encoder E^P into e_t^P.
- Goal token. The normalized robot-centric planar coordinate $\bar{g}_t^{b,xy} \in \mathbb{R}^2$ is projected by the goal encoder E^g into e_t^g, representing the goal direction and relative planar distance.
- Task token. The fixed instruction c_0 is encoded by text encoder E_c into e^c, representing task context. This instruction is identical during training and inference.

The four types of tokens are concatenated into shared conditional memory, which serves both future spatial-state prediction and trajectory generation. Detailed encoder structures are provided in Appendix D.

Future Spatial-State Prediction. The model predicts a future spatial-state latent variable from the shared conditional memory:

$$\hat{z} = G_f(C_t). \tag{6}$$

Here G_f applies one denoising step of the Cosmos Predict 2.5 backbone to C_t and returns the resulting latent. During training, the supervision latent is obtained by applying the same depth encoder to the depth sequence near a future time:

$$z_{t+\Delta} = E^D(D_{t+\Delta-H:t+\Delta}), \qquad \Delta = 32. \tag{7}$$

The loss is the cosine distance between $\hat{z}$ and $z_{t+\Delta}$. During inference, the model still generates $\hat{z}$ from C_t and directly uses it as the key-value condition for trajectory decoding; future observations and their supervision are unavailable. This design supplies direct supervision through the future local depth representation, allowing the shared conditional memory to encode spatial changes relevant to subsequent navigation.

Trajectory Generation. The trajectory-generation branch uses learnable waypoint queries $Q^{(0)} \in \mathbb{R}^{W\times D}$, where $W = 32$ is the number of waypoints. The trajectory decoder consists of four Transformer blocks. Each block sequentially contains self-attention over waypoint queries, cross-attention using the future spatial-state latent $\hat{z}$ as keys and values, and a feed-forward network:

$$Q^{(l)} = \mathrm{Block}(Q^{(l-1)}, \hat{z}), \qquad l = 1, \ldots, 4. \tag{8}$$

The final $Q^{(4)}$ is mapped by the trajectory head to planar displacement increments and yaw increments:

$$\hat{X}_t = G_\tau(Q^{(4)}) = \{(\Delta\hat{p}_{t,i}, \Delta\hat{\psi}_{t,i})\}_{i=1}^{W}. \tag{9}$$

The planar displacement $\Delta\hat{p}_{t,i} = (\Delta\hat{x}_{t,i}, \Delta\hat{y}_{t,i})$ and yaw increment $\Delta\hat{\psi}_{t,i}$ are output separately. Accumulating planar displacements gives predicted waypoints $\hat{p}_{t,i} = \hat{p}_{t,i-1} + \Delta\hat{p}_{t,i}$, and accumulating yaw increments gives $\hat{\psi}_{t,i} = \hat{\psi}_{t,i-1} + \Delta\hat{\psi}_{t,i}$. Yaw increments are supervised by $\mathcal{L}_\psi$ and used for pose propagation and motion-direction control; planar positions are used for goal proximity, geometric constraints, and trajectory safety checks.

3.3 Deployment-Oriented Joint Learning Objectives

Merely fitting demonstrated trajectories does not guarantee deployment performance. A model may reproduce the local shape of a demonstration while ending away from the actual target, producing a jittery trajectory, or entering an obstacle region. To address these failure modes, we organize training around three objectives:

$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_f \mathcal{L}_{\text{future}} + \mathcal{L}_{\text{deploy}}, \tag{10}$$

where

$$\mathcal{L}_{\text{main}} = \mathcal{L}_{\text{xy}} + \lambda_\psi \mathcal{L}_\psi, \tag{11}$$

$$\mathcal{L}_{\text{deploy}} = \lambda_g \mathcal{L}_{\text{goal}} + \lambda_p \mathcal{L}_{\text{progress}} + \lambda_s \mathcal{L}_{\text{smooth}} + \lambda_{\text{collision}} \mathcal{L}_{\text{collision}}. \tag{12}$$

$\mathcal{L}_{\text{main}}$ supervises the demonstrated trajectory and yaw increments. $\mathcal{L}_{\text{future}}$ aligns the predicted latent with the future spatial state. $\mathcal{L}_{\text{deploy}}$ targets execution outcomes: $\mathcal{L}_{\text{goal}}$ penalizes endpoints far from the target, $\mathcal{L}_{\text{progress}}$ discourages backward or detouring motion, $\mathcal{L}_{\text{smooth}}$ reduces abrupt segment changes, and $\mathcal{L}_{\text{collision}}$ enforces obstacle clearance.

Because $\mathcal{L}_{\text{smooth}}$ and $\mathcal{L}_{\text{collision}}$ can conflict with $\mathcal{L}_{\text{main}}$ at turns and around obstacles, training projects their gradients onto the orthogonal complement of the primary trajectory gradient at every step, so that only the components orthogonal to $\mathcal{L}_{\text{main}}$ are retained. Complete definitions, masks, normalization, and weights are provided in Appendix E; the projection rule is detailed in Appendix F.

3.4 Hierarchical Closed-Loop Execution and Correction

The executor verifies each predicted waypoint and its connecting segment using the latest local Euclidean Signed Distance Field (ESDF), which provides the distance from any planar position to the nearest obstacle. The ESDF is constructed online from depth observations and robot poses, as detailed in Appendix G. A waypoint or segment is safe when its sampled obstacle distance is no smaller than the safety threshold ρ. Let o'_t and T'_t denote the position and pose at the last safe state before correction. If the next waypoint would cause a collision, the robot stops at the last safe state (o'_t, T'_t) and attempts three correction levels in increasing order of cost. Each level is attempted only if the previous one fails. Figure 3 illustrates the three correction levels.

Level 1: Trajectory reconnection. The executor searches the unexecuted trajectory for waypoints whose positions and connecting segments from o'_t are both safe:

$$\mathcal{C}_t = \{p_i \mid i \geq k,\ \text{SafePoint}(p_i) = 1,\ \text{SafeSegment}(o'_t, p_i) = 1\}, \tag{13}$$

$$p_t^{re} = \arg\min_{p \in \mathcal{C}_t} \|p - g_t^{b,xy}\|_2. \tag{14}$$

If $\mathcal{C}_t$ is nonempty, the robot reconnects to p_t^{re} and resumes the remaining trajectory without new model inference. If $\mathcal{C}_t$ is empty, the system proceeds to Level 2.

Level 2: In-place re-observation. The system evaluates whether an in-place rotation can bring the goal coordinate back into the camera field of view. If a feasible rotation exists, the robot rotates accordingly and checks whether the goal coordinate is re-observable in the updated view. Once the goal coordinate is re-observed and passes the depth-consistency check, the system updates the local observation and ESDF and replans the low-level trajectory. If no feasible rotation exists or the target remains invisible after rotation, the system proceeds to Level 3.

Level 3: Historical-viewpoint recovery. The system retrieves the most recent historical frame j^* in which the target was visible, together with its depth D_{j^*} and pose T_{j^*}. Using D_{j^*} and T_{j^*}, the system constructs a historical ESDF and selects a candidate region $\mathcal{R}_{j^*}$ whose points satisfy two conditions: the goal coordinate is observable from the candidate, and the candidate can be reached from the current safe stopping position o'_t through a safe connecting segment. From $\mathcal{R}_{j^*}$, the system selects the point v_{j^*} closest to the goal coordinate and transforms it into the current coordinate frame:

$$v_w = T_{j^*}^w v_{j^*}, \qquad v = (T_t'^w)^{-1} v_w. \tag{15}$$

The robot navigates to v and replans toward the same goal coordinate from this new starting position. If no safe candidate exists, or if a safe trajectory still cannot be generated after replanning, the episode terminates.

All correction levels retain the spatial target established from the initial pixel goal and update its robot-centric coordinates as the robot moves, without requesting a new pixel goal for the same local target. Complete safety definitions, coordinate transformations, and thresholds are provided in Appendix G.

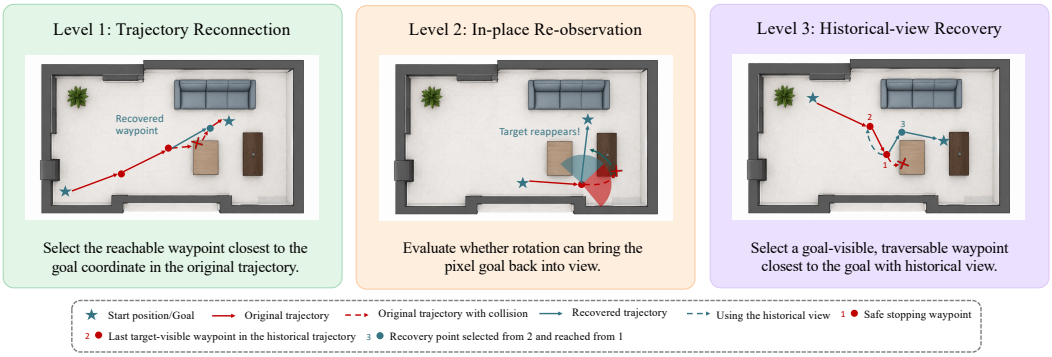

Figure 3: Hierarchical correction in increasing order of recovery cost. Level 1 reconnects the robot to the safely reachable unexecuted waypoint closest to the goal coordinate. Level 2 first evaluates whether an in-place rotation can bring the goal coordinate back into view and replans after re-observation. Level 3 uses the most recent valid historical observation to construct and filter a traversable recovery region, selects a safely reachable candidate closest to the goal coordinate, and replans after the goal becomes observable.

4 Experiments

4.1 Experimental Setup

Simulated Dataset and Protocol. We evaluate on the simulated IPGP protocol, built on InternVLA-CE-R2R. Training uses over 500,000 samples from 30 scenes; evaluation uses 12,000 samples from 31 disjoint unseen scenes. All methods share the same training data, observation history, initial pixel-goal annotation, and 32-waypoint output horizon, and are retrained to convergence from available checkpoints.Details on pixel goal definition, future-depth supervision, sample construction, and trajectory padding are provided in Appendix A.

Real-robot Dataset and Protocol. The real-robot protocol, TR-PGP, is collected on the Tiangong robot platform and contains a straight road, an after-turn corridor, and a path after a narrow passage. Each method is tested ten times per scenario. Before evaluation, WorldPixelNav and other baselines are adapted to the physical robot with LoRA on 50 self-collected trajectories from scenes disjoint from the evaluation scenarios. The simulation-trained backbone remains frozen. The success criterion matches simulation. Full real-robot setup details are provided in Appendix H.

Evaluation Metrics. We report success rate (SR), success weighted by path length (SPL), collision-episode rate (CR), and collision-free out-of-range rate (NOR). SR requires the final position to be within 0.3 m of the target with no collision. SPL weights successful episodes by the ratio of shortest to executed path length. CR is the fraction of episodes with at least one collision. NOR is the fraction of collision-free episodes ending more than 0.3 m from the target.

4.2 Main Simulation Results

Table 1 reports the main comparison. WorldPixelNav reaches 96.48% SR, 95.37% SPL, 0.07% CR, and 3.45% NOR, outperforming all baselines on all four metrics.

Table 1: Main simulation results (%).

Method	SR ↑	SPL ↑	CR ↓	NOR ↓
DualVLN	36.98	32.37	41.23	21.79
NavDP	45.73	40.13	26.54	27.73
OccPlanner	49.81	48.52	18.37	31.82
PixNav	66.63	59.27	12.93	20.44
WorldPixelNav (without correction)	92.72	92.67	3.78	3.50
WorldPixelNav (all three levels)	**96.48**	**95.37**	**0.07**	**3.45**

The large gap against DualVLN and NavDP is consistent with their native receding-horizon design. Both methods were built to execute a short trajectory segment and then re-ground the target from a new observation, which is not aligned with the fixed-goal protocol used here. Under a fixed goal, they cannot correct intermediate errors through later goal updates, and executing a longer predicted trajectory in one pass amplifies trajectory deviation. Errors therefore accumulate, and most episodes stop outside the target range. PixNav achieves the best baseline SR at 66.63%, but its CR and NOR remain high because its trajectory is regressed directly from pixel features without geometric constraints, making it prone to grazing obstacles or stopping outside the target. WorldPixelNav without correction already reaches 92.72% SR, showing that future-state prediction and deployment-oriented supervision alone provide most of the target-reaching accuracy. The three correction levels further reduce CR from 3.78% to 0.07% over 12,000 episodes. The residual eight collisions mainly occur near narrow obstacles and local free-space boundaries, where depth noise and local geometric estimation errors affect trajectory verification.

To compare success rates more fairly under similar low-level inference time, we align all methods to a matched cumulative low-level inference budget. Under this setting, a single WorldPixelNav call still achieves the best performance. This indicates that WorldPixelNav makes more effective use of a matched inference budget. Complete results are provided in Appendix B.

4.3 Real-Robot Experiments

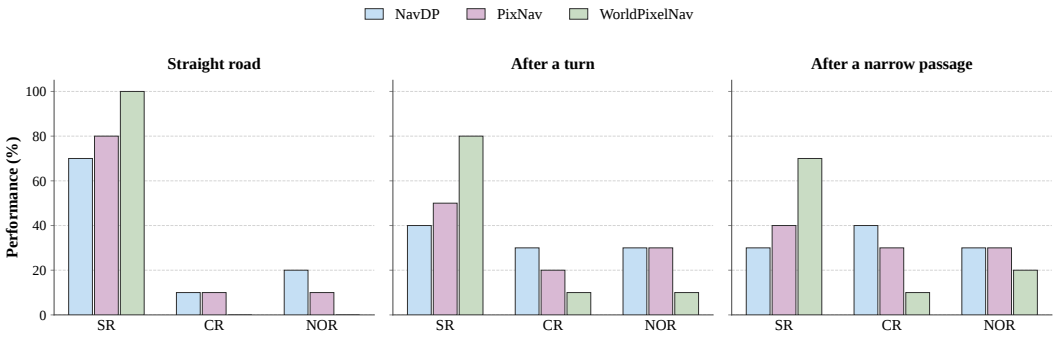

Figure 4: Real-robot results (%). Success rate (SR), collision-episode rate (CR), and collision-free out-of-range rate (NOR) across the three test scenarios.

Figure 4 reports real-robot results across three scenarios. In the straight-road scenario, the target remains visible and correction is rarely triggered; WorldPixelNav reaches 100.0% SR and 0.0% CR, with its advantage mainly coming from trajectory quality. In the after-turn scenario, the target temporarily leaves the field of view. PixNav and NavDP cannot recover it, whereas WorldPixelNav triggers the second correction level and reaches 80.0% SR, 30 and 40 points above the two baselines. In the after-narrow-passage scenario, traversable space is limited and trajectories are easily blocked. WorldPixelNav's first correction level triggers frequently, and the third level is used when the target is invisible at the passage exit. WorldPixelNav still reaches 70.0% SR, 30–40% points above the baselines. The pattern across scenarios is consistent with the correction mechanism contributing most when the target temporarily disappears or the path is locally blocked. Representative real-robot tasks are shown in Appendix H, Figure 5.

Table 2: Ablation of future spatial-state supervision (%).

Configuration	SR ↑	SPL ↑	CR ↓	NOR ↓
Without future-state supervision	73.14	69.22	11.79	15.07
WorldPixelNav (without correction)	92.72	92.67	3.78	3.50

4.4 Ablation Studies

Future Spatial-State Supervision. Removing this supervision decreases SR by 19.6 points and increases CR by 8.0 points. Without direct future-state supervision, the predicted latent is not explicitly aligned with the future local spatial state. Although the model can still fit local trajectory shapes, it becomes less effective at capturing spatial changes relevant to subsequent motion. The resulting degradation in both accuracy and safety confirms that future-state supervision is a core component of long-horizon trajectory prediction rather than a minor regularizer.

Table 3: Ablation of deployment-oriented supervision (%).

Configuration	SR ↑	SPL ↑	CR ↓	NOR ↓
Without $\mathcal{L}_{\text{goal}}$	30.36	25.23	9.88	59.76
Without $\mathcal{L}_{\text{progress}}$	46.12	42.14	8.75	45.13
Without $\mathcal{L}_{\text{smooth}}$	78.82	72.07	12.39	8.79
Without $\mathcal{L}_{\text{collision}}$	77.27	74.24	13.71	9.02
Without gradient orthogonalization	69.37	66.29	10.76	19.87
Only $\mathcal{L}_{\text{xy}}$ and $\mathcal{L}_{\psi}$	15.68	12.14	33.78	50.54
WorldPixelNav (without correction)	92.72	92.67	3.78	3.50

Deployment-Oriented Supervision. Table 3 reports the ablation of each deployment-oriented loss. Goal proximity dominates: removing it collapses SR to 30.36% and raises NOR to 59.76%, because the model can reproduce the demonstrated trajectory shape without any guarantee that the endpoint lands near the target. Trajectory progress mainly preserves path efficiency, and its removal pushes NOR to 45.13%. Smoothness and collision-avoidance losses each protect geometric safety, with CR rising above 12% when either is removed. Gradient orthogonalization prevents the smoothness and collision-avoidance terms from suppressing necessary turns, and its removal costs 23.4 SR points. Retaining only $\mathcal{L}_{\text{xy}}$ and $\mathcal{L}_{\psi}$ reduces SR to 15.68%, confirming that the auxiliary losses act jointly rather than independently. Complete definitions and additional analysis are provided in Appendix E.

Table 4: Cumulative ablation of hierarchical dynamic correction (%).

Configuration	SR ↑	SPL ↑	CR ↓	NOR ↓
WorldPixelNav (without correction)	92.72	92.67	3.78	3.50
Reconnection only	94.48	93.95	2.05	3.47
Reconnection + in-place re-observation	96.03	95.21	0.51	3.46
All three levels	96.48	95.37	0.07	3.45

Hierarchical Dynamic Correction. Table 4 reports the cumulative ablation of the three correction levels. Trajectory reconnection provides the largest gain, increasing SR from 92.72% to 94.48% and decreasing CR from 3.78% to 2.05%. It reuses safely reachable waypoints from the original trajectory without additional model inference. In-place re-observation adds 1.55 SR points by recovering the target after heading changes. Adding historical-viewpoint recovery further reduces CR from 0.51% to 0.07%, addressing cases in which the target remains invisible after in-place rotation. The three levels primarily reduce collision risk caused by occlusion and path blockage; their improvements therefore concentrate in CR, while NOR remains near 3.5%.

5 Conclusion

We proposed WorldPixelNav, a deployment-oriented dynamic latent correction framework for pixel-goal navigation. WorldPixelNav uses a world model as its backbone, generates continuous local trajectories by predicting a future spatial-state latent variable, and performs hierarchical correction in increasing order of cost during execution. Under the IPGP and TR-PGP evaluation protocols, WorldPixelNav achieves a success rate of 96.48% and a collision rate of 0.07% in unseen simulated environments. On the physical Tiangong robot platform, it achieves an average success rate of 83.3% and a collision rate of 6.7%.

AI USE STATEMENT

In preparing this manuscript, generative AI tools were used to assist with grammar checking and text polishing. All AI-assisted changes were reviewed by the authors for technical accuracy and consistency with the reported work. The authors take responsibility for the final content of the manuscript, including text and artifacts produced with AI assistance.

REFERENCES

Dong An, Yuankai Qi, Yangguang Li, Yan Huang, Liang Wang, Tieniu Tan, and Jing Shao. Bevbert: Multimodal map pre-training for language-guided navigation. In *ICCV*, 2023.

Daichi Azuma, Taiki Miyanishi, Koya Sakamoto, Shuhei Kurita, Yaonan Zhu, Petr Khrapchenkov, Motoaki Kawanabe, Yusuke Iwasawa, and Yutaka Matsuo. Navwam: A navigation world action model for goal-conditioned visual navigation. *arXiv preprint arXiv:2606.13494*, 2026.

Muyi Bao, Yuxin Cai, Hang Xu, Zongtai Li, Jinxi He, Jingfan Tang, Chen Lv, Ji Zhang, Yaqi Xie, and Wenshan Wang. Goal2pixel: Grounding goals to pixels for vision-language navigation. *arXiv preprint arXiv:2606.01621*, 2026.

Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models. In *CVPR*, pages 15791–15801, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023.

Wenzhe Cai, Siyuan Huang, Guangran Cheng, Yuxing Long, Peng Gao, Changyin Sun, and Hao Dong. Bridging zero-shot object navigation and foundation models through pixel-guided navigation skill. In *ICRA*, 2024.

Wenzhe Cai, Jiaqi Peng, Yuqiang Yang, Yujian Zhang, Meng Wei, Hanqing Wang, Yilun Chen, Tai Wang, and Jiangmiao Pang. Navdp: Learning sim-to-real navigation diffusion policy with privileged information guidance. In *ICRA*, 2026.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

Shizhe Chen, Pierre-Louis Guhur, Cordelia Schmid, and Ivan Laptev. History aware multimodal transformer for vision-and-language navigation. In *NeurIPS*, 2021.

Shizhe Chen, Pierre-Louis Guhur, Makarand Tapaswi, Cordelia Schmid, and Ivan Laptev. Think global, act local: Dual-scale graph transformer for vision-and-language navigation. In *CVPR*, 2022.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *RSS*, 2023.

Yucheng Deng, Pingrui Lai, Xinhai Li, Chenjia Bai, Xiaoheng Deng, Chengnuo Sun, Xuelong Li, and Hua Yang. Spacevln: A zero-shot vision-and-language navigation agent with online spatial cognitive memory and reasoning. *arXiv preprint arXiv:2606.08992*, 2026.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Chen Gao, Liankai Jin, Xingyu Peng, Jiazhao Zhang, Yue Deng, Annan Li, He Wang, and Si Liu. Octonav: Towards generalist embodied navigation. In *CVPR*, pages 40074–40084, 2026.

Ruiyan Gong, Yingnan Guo, Junjun Hu, Jintao Kong, Xiaoxu Leng, Tianlun Li, Weize Li, Fei Liu, Zhicheng Liu, Jia Lu, et al. Abot-n1: Toward a general visual language navigation foundation model. *arXiv preprint arXiv:2607.10383*, 2026.

David Ha and Jürgen Schmidhuber. World models. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2.

Mingfei Han, Liang Ma, Kamila Zhumakhanova, Ekaterina Radionova, Jingyi Zhang, Xiaojun Chang, Xiaodan Liang, and Ivan Laptev. Roomtour3d: Geometry-aware video-instruction tuning for embodied navigation. In *CVPR*, pages 27586–27596, 2025.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *ICLR*, 2024.

Yicong Hong, Qi Wu, Yuankai Qi, Cristian Rodriguez-Opazo, and Stephen Gould. Vln bert: A recurrent vision-and-language bert for navigation. In *CVPR*, 2021.

Binling Huang, Nianjin Ye, Xi Yang, Liang Hu, Zhou Huang, Shuang Wei, Longrui Yang, Yanchi Chen, and Lanpeng Jia. Occplanner: Goal-aware occupancy-conditioned diffusion planner for pixel-goal navigation. *arXiv preprint arXiv:2608.14160*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Grace Lam, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In *CoRL*, 2024.

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*, pages 104–120. Springer, 2020.

Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. Improving vision-and-language navigation with image-text pairs from the web. In *ECCV*, 2020.

Yanghong Mei, Longteng Guo, Ming-Ming Yu, Guiyu Zhao, Xingjian He, and Jing Liu. Navwm: A unified navigation world model for foresight-driven planning. *arXiv preprint arXiv:2606.24101*, 2026.

NVIDIA. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IROS*, 2017.

Akhil Perincherry, Jacob Krantz, and Stefan Lee. Do visual imaginations improve vision-and-language navigation agents? In *CVPR*, 2025.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Dhruv Shah, Benjamin Eysenbach, Nicholas Rhinehart, and Sergey Levine. Rapid exploration for open-world navigation with latent goal models. In *CoRL*, 2021.

Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. Vint: A foundation model for visual navigation. In *CoRL*, 2023.

Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. In *NeurIPS*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Liuyi Wang, Xinyuan Xia, Hui Zhao, Hanqing Wang, Tai Wang, Yilun Chen, Chengju Liu, Qijun Chen, and Jiangmiao Pang. Rethinking the embodied gap in vision-and-language navigation: A holistic study of physical and visual disparities. In *ICCV*, pages 9455–9465, 2025.

Zun Wang, Jialu Li, Yicong Hong, Yi Wang, Qi Wu, Mohit Bansal, and Stephen Gould. Scaling data generation in vision-and-language navigation. In *ICCV*, 2023.

Meng Wei, Chenyang Wan, Jiaqi Peng, Xiqian Yu, Yuqiang Yang, Delin Feng, Wenzhe Cai, Chenming Zhu, Tai Wang, Jiangmiao Pang, and Xihui Liu. Ground slow, move fast: A dual-system foundation model for generalizable vision-and-language navigation. In *ICLR*, 2026.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020.

Peiran Xu, Xicheng Gong, and Yadong Mu. Navq: Learning a q-model for foresighted vision-and-language navigation. In *ICCV*, pages 6327–6341, 2025.

Ning Yang, Yan Huang, Kaiwen Peng, Ziheng He, Kai Wang, Cui Miao, Kailin Lyu, Guo Li, Xiaofeng Wang, Zheng Zhu, Jing Liu, and Nianfeng Liu. Wam-nav: Asymmetric latent world-action modeling for unified visual navigation. *arXiv preprint arXiv:2606.04907*, 2026.

Xuan Yao, Junyu Gao, and Changsheng Xu. Navmorph: A self-evolving world model for vision-and-language navigation in continuous environments. In *ICCV*, pages 5536–5546, 2025.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *NeurIPS*, 2020.

Shuang Zeng, Dekang Qi, Xinyuan Chang, Feng Xiong, Shichao Xie, Xiaolong Wu, Shiyi Liang, Mu Xu, Xing Wei, and Ning Guo. Janusvln: Decoupling semantics and spatiality with dual implicit memory for vision-language navigation. In *ICLR*, 2026.

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. In *Robotics: Science and Systems*, 2025.

Duo Zheng, Shijia Huang, Lin Zhao, Yiwu Zhong, and Liwei Wang. Towards learning a generalist model for embodied navigation. In *CVPR*, pages 13624–13634, 2024.

Gengze Zhou, Yicong Hong, Zun Wang, Chongyang Zhao, Mohit Bansal, and Qi Wu. Same: Learning generic language-guided visual navigation with state-adaptive mixture of experts. In *ICCV*, pages 7794–7807, 2025.

Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *CVPR*, 2022.

Jing Zuo, Lingzhou Mu, Fan Jiang, Chengcheng Ma, Mu Xu, and Yonggang Qi. Fantasyvln: Unified multimodal chain-of-thought reasoning for vision-language navigation, 2026. URL `https://arxiv.org/abs/2601.13976`.

APPENDIX

A DATASET CONSTRUCTION

Pixel goal definition. We directly reuse the pixel goal annotations provided by InternVLA and do not regenerate them. These annotations are produced by the original InternVLA data-generation process. For each navigation trajectory, the depth map of one frame is taken as the current depth map. Future ground-truth trajectory points after that frame are projected onto the current image plane. Only future trajectory points that lie within the current camera field of view, remain unoccluded, and have valid observed depth at their projected pixels are retained. Among the visible future trajectory points, the one with the farthest 3D distance from the current robot position is selected as the pixel goal. We use these annotations as pixel goals for training and evaluation. The persistent spatial target is reconstructed from the selected pixel, its observed depth, and the camera pose, as described in Section 3.2. The same reconstruction procedure is used during training and inference, in simulation and on the physical robot.

Acquisition of future depth frames. $\mathcal{L}_{\text{future}}$ uses the spatial state of the 32nd future frame as supervision. If the trajectory ends before the 32nd frame, the final depth observation and pose at which the robot actually stops are used as the 32nd-frame state. This loss is skipped through a validity mask only when the final depth or pose is invalid.

Training and validation data. For every original training trajectory, all frames with a valid pixel goal are selected as current frames. Each sample contains:

- the historical poses and depth observations for the eight frames preceding the current frame;
- the pose and depth observation of the current frame;
- a fixed instruction;
- the pixel goal in the current frame; and
- the subsequent demonstrated trajectory from the current position to the target position.

Trajectory padding. The demonstrated trajectory of each sample has at most 32 steps. For samples with fewer than 32 steps, the first K steps are retained and zero displacement increments are padded from step $K+1$ through step 32. Padded positions are used only for batch alignment and are excluded from trajectory-position, progress, goal-proximity, smoothness, collision-avoidance, and yaw losses through waypoint-validity masks. The future-state loss uses its own latent-validity mask and follows the supervision rule described above.

Success criterion. A navigation episode is considered successful if the final position is no more than 0.3 m from the target and no collision occurs during execution. The simulated robot camera is 1.25 m high with a horizontal field of view of 45°, and the robot body is modeled as a cylinder with a radius of 0.1 m.

B INFERENCE-EFFICIENCY ANALYSIS

We measure batch-1 low-level planning latency for all methods on the same hardware and software environment. Each measurement includes the complete model computation required to generate the low-level output but excludes model loading, disk I/O, metric computation, and robot execution time. WorldPixelNav, DualVLN, NavDP, PixNav, and OccPlanner require 85, 42, 96, 82, and 94 ms per planning call, respectively.

To determine whether repeated short-horizon planning can close the performance gap without increasing the computational budget, we additionally evaluate a two-call DualVLN variant. It executes the first segment of its initial trajectory and then replans once from the updated observation while retaining the same spatial target. Its two low-level calls require 84 ms in total, closely matching the 85 ms required by one WorldPixelNav call.

Table 5: Batch-1 low-level planning latency and navigation performance. All latency measurements are obtained on the same hardware. "Calls" denotes the number of low-level planning calls.

Method	Calls	Time (ms)	SR ↑	SPL ↑	CR ↓	NOR ↓
DualVLN	1	42	36.98	32.37	41.23	21.79
NavDP	1	96	45.73	40.13	26.54	27.73
PixNav	1	82	66.63	59.27	12.93	20.44
OccPlanner	1	94	49.81	48.52	18.37	31.82
DualVLN, two-call replanning	2	84	51.78	46.25	20.37	27.85
WorldPixelNav, without correction	1	85	**92.72**	**92.67**	**3.78**	**3.50**

The two-call setting improves DualVLN from 36.98% to 51.78% SR and reduces CR from 41.23% to 20.37%. Nevertheless, under nearly identical cumulative low-level inference time, one-call WorldPixelNav achieves 92.72% SR, 92.67% SPL, 3.78% CR, and 3.50% NOR. The advantage therefore does not result from allocating more low-level inference time to WorldPixelNav. Instead, WorldPixelNav provides a substantially better accuracy–safety trade-off within a matched low-level inference budget. This comparison concerns low-level model computation and does not claim lower end-to-end system latency, which also depends on high-level grounding, communication, and physical execution.

C Detailed Evaluation Protocol

Metrics. Let N be the number of evaluation episodes. For episode n, let $S_n = 1$ if the robot terminates within 0.3 m of the target without collision, and $S_n = 0$ otherwise. Let $C_n = 1$ indicate that at least one collision occurs. We compute

$$\mathrm{SR} = \frac{1}{N}\sum_{n=1}^{N} S_n, \qquad \mathrm{CR} = \frac{1}{N}\sum_{n=1}^{N} C_n. \tag{16}$$

Let ℓ_n and p_n denote the shortest-path distance and executed-path length, respectively. SPL is defined as

$$\mathrm{SPL} = \frac{1}{N}\sum_{n=1}^{N} S_n \frac{\ell_n}{\max(\ell_n, p_n)}. \tag{17}$$

NOR measures collision-free episodes that terminate outside the target range:

$$\mathrm{NOR} = \frac{1}{N}\sum_{n=1}^{N} (1 - C_n)(1 - S_n). \tag{18}$$

Baseline adaptation. Because the baselines differ in their original output interfaces and execution strategies, their prediction heads are adapted to the common 32-waypoint output space and retrained to convergence from the available checkpoints. The comparison therefore isolates low-level trajectory-planning capability under the fixed pixel-goal protocol rather than reproducing each method's original system-level policy.

D Model and Training Details

Backbone model. We use the Cosmos Predict 2.5 backbone as a latent feature extractor. Only a single denoising step is executed, and the intermediate latent is used as the future spatial-state representation; the model does not perform iterative denoising or image generation. The depth sequence is encoded by the depth encoder E^D into a spatiotemporal latent space compatible with Cosmos. The depth encoder consists of three 3D convolutional layers with kernel size $3 \times 3 \times 3$ and 32, 64, and 128 channels, respectively, each followed by GroupNorm and SiLU activation. The temporal downsampling factor is 2 and the spatial downsampling factor is 4. The shared depth encoder E^D remains frozen during training. Stop-gradient is applied to future spatial-state supervision to prevent joint collapse of the prediction branch and encoder.

Trajectory-generation branch. The trajectory-generation branch uses learnable waypoint queries $Q \in \mathbb{R}^{W \times D}$, where $W = 32$ is the number of waypoints and $D = 256$ is the hidden dimension. The waypoint queries first use self-attention to model inter-waypoint dependencies and then query the future spatial state through cross-attention. The decoder consists of $N = 4$ Transformer blocks, each containing eight-head self-attention, eight-head cross-attention, and a feed-forward network with hidden dimension 1024. The total number of parameters is approximately 48M.

Training configuration. The model is trained with AdamW using a learning rate of 10^{-4}, weight decay of 0.01, and cosine learning-rate scheduling. The batch size is 64, and the model is trained for 100 epochs on eight NVIDIA A100 GPUs using bfloat16 mixed precision.

E Complete Loss Definitions

Let $M_i \in \{0, 1\}$ indicate whether waypoint i is valid, let $\epsilon = 10^{-6}$ be a small constant, and let $p_{t,i}$ and $\hat{p}_{t,i}$ denote demonstrated and predicted planar waypoints, respectively. Table 6 summarizes the roles and weights of the seven losses.

Table 6: Summary of loss terms.

Loss	Role	Weight
$\mathcal{L}_{\text{xy}}$	Keep predictions close to the demonstration distribution	1.0
$\mathcal{L}_{\text{goal}}$	Make the trajectory endpoint approach the spatial target	1.0
$\mathcal{L}_{\text{progress}}$	Encourage monotonic progress toward the target	0.5
$\mathcal{L}_{\text{smooth}}$	Reduce abrupt changes between adjacent segments	0.1
$\mathcal{L}_{\text{future}}$	Match the generated latent to the future spatial state	1.0
$\mathcal{L}_{\text{collision}}$	Penalize obstacle entry or insufficient clearance	1.0
$\mathcal{L}_{\psi}$	Supervise yaw increments	0.5

The losses are defined as follows.

Trajectory-position loss.

$$\mathcal{L}_{\text{xy}} = \frac{\sum_{i=1}^{W} M_i \|\hat{p}_{t,i} - p_{t,i}\|_2^2}{\sum_{i=1}^{W} M_i + \epsilon}. \tag{19}$$

Goal-proximity loss. Let i^* denote the last valid waypoint:

$$\mathcal{L}_{\text{goal}} = \|\hat{p}_{t,i^*} - g_t^{b,xy}\|_2. \tag{20}$$

Trajectory-progress loss. Let $d_{t,i} = \|\hat{p}_{t,i} - g_t^{b,xy}\|_2$ and $d_{t,0} = \|o_t - g_t^{b,xy}\|_2$:

$$\mathcal{L}_{\text{progress}} = \frac{\sum_{i=1}^{W} M_i \max(0, d_{t,i} - d_{t,i-1})}{\sum_{i=1}^{W} M_i + \epsilon}. \tag{21}$$

Trajectory-smoothness loss. Let $\Delta\hat{p}_{t,i} = \hat{p}_{t,i} - \hat{p}_{t,i-1}$:

$$\mathcal{L}_{\text{smooth}} = \frac{\sum_{i=1}^{W-1} M_i M_{i+1} \|\Delta\hat{p}_{t,i+1} - \Delta\hat{p}_{t,i}\|_2^2}{\sum_{i=1}^{W-1} M_i M_{i+1} + \epsilon}. \tag{22}$$

Future spatial-state loss. Let M_j^z be the validity mask for latent-space location j:

$$\mathcal{L}_{\text{future}} = \frac{\sum_j M_j^z [1 - \cos(\hat{z}^j, z_{t+32}^j)]}{\sum_j M_j^z + \epsilon}. \tag{23}$$

Collision-avoidance loss. Let $\delta_t(p)$ be the ESDF distance and ρ the safety threshold:

$$\mathcal{L}_{\text{collision}} = \frac{\sum_{i=1}^{W} M_i \max[0, \rho - \delta_t(\hat{p}_{t,i})]}{\sum_{i=1}^{W} M_i + \epsilon}. \tag{24}$$

Yaw-supervision loss. Let $\Delta\psi_{t,i}$ and $\Delta\hat{\psi}_{t,i}$ denote the demonstrated and predicted yaw increments:

$$\mathcal{L}_{\psi} = \frac{\sum_{i=1}^{W} M_i (\Delta\hat{\psi}_{t,i} - \Delta\psi_{t,i})^2}{\sum_{i=1}^{W} M_i + \epsilon}. \tag{25}$$

With $\mathcal{L}_{\text{main}} = \mathcal{L}_{\text{xy}} + \lambda_{\psi}\mathcal{L}_{\psi}$ and $\mathcal{L}_{\text{deploy}} = \lambda_g\mathcal{L}_{\text{goal}} + \lambda_p\mathcal{L}_{\text{progress}} + \lambda_s\mathcal{L}_{\text{smooth}} + \lambda_{\text{collision}}\mathcal{L}_{\text{collision}}$, the expanded objective is

$$\mathcal{L} = \mathcal{L}_{\text{xy}} + \lambda_g\mathcal{L}_{\text{goal}} + \lambda_p\mathcal{L}_{\text{progress}} + \lambda_s\mathcal{L}_{\text{smooth}} + \lambda_f\mathcal{L}_{\text{future}} + \lambda_{\text{collision}}\mathcal{L}_{\text{collision}} + \lambda_{\psi}\mathcal{L}_{\psi}. \tag{26}$$

The weights are $\lambda_g = 1.0$, $\lambda_p = 0.5$, $\lambda_s = 0.1$, $\lambda_f = 1.0$, $\lambda_{\text{collision}} = 1.0$, and $\lambda_{\psi} = 0.5$. They are determined through grid search on a fast validation set.

F Gradient Orthogonal Projection

The smoothness and collision-avoidance losses can conflict with the primary trajectory loss at turns and around obstacles. During training, the gradients of these two auxiliary losses are projected onto the orthogonal complement of the primary trajectory gradient at every step, so that only their orthogonal components are retained. For $k \in \{smooth, collision\}$,

$$\tilde{g}_k = g_k - \frac{g_k^{\top} g_{xy}}{\|g_{xy}\|_2^2 + \epsilon} g_{xy}. \tag{27}$$

The other three terms—goal proximity, trajectory progress, and future spatial state—are not projected and are combined directly in a weighted sum. The final parameter-update gradient is

$$g = g_{xy} + \lambda_g g_{goal} + \lambda_p g_{progress} + \lambda_s \tilde{g}_{smooth} + \lambda_f g_{future} + \lambda_{\text{collision}}\tilde{g}_{\text{collision}} + \lambda_{\psi} g_{\psi}. \tag{28}$$

This mechanism is executed at every training step. It changes only the gradient direction, not the loss value, and therefore does not affect inference-time computation.

G Closed-Loop Correction Details

Safety conditions. Let $\delta_t(p)$ denote the ESDF distance and ρ the safety threshold:

$$\text{SafePoint}(p) = 1 \iff \delta_t(p) \geq \rho, \tag{29}$$

$$\text{SafeSegment}(a, b) = 1 \iff \delta_t(q) \geq \rho, \quad \forall q \in \text{Interp}(a, b). \tag{30}$$

Starting from $\hat{p}_{t,0} = o_t$, the executor checks the trajectory, finds the first waypoint k that violates a safety condition, and executes only $\hat{p}_{t,1}, \ldots, \hat{p}_{t,k-1}$, reaching the safe stopping position o_t'.

ESDF construction. The local ESDF is built online from the current depth sequence and robot pose. In simulation, depth points are back-projected into the robot frame, accumulated into a local voxel grid, and converted into a distance field. On the physical robot, the same procedure is applied using the onboard depth stream.

Thresholds and update interval. We use $\rho = 0.25$ m in simulation and $\rho = 0.30$ m on the physical robot. The local ESDF is refreshed every $\Delta_u = 10$ frames. During candidate filtering, line-of-sight occlusion is checked using 3D ray casting with a ray-sampling interval of 0.05 m.

H Real-Robot Experimental Details

The real-robot experiments use the Tiangong robot platform. The robot obtains observations from a depth camera mounted at a height of 1.45 m with a horizontal field of view of 45°. The local geometric representation is constructed from online depth, historical depth, and robot poses. No pre-built global map is used throughout deployment. The test scenarios comprise a straight road, after a turn, and after a narrow passage. Each method is tested ten times in each scenario, giving 30 trials per method and 90 physical executions across all three methods. The success criterion is

identical to that in simulation. The control frequency is 10 Hz, and the fixed condition is the same as in the simulation setting.

All methods are adapted with the same LoRA rank $r = 8$, learning rate 10^{-4}, for $E = 200$ steps on the 50 collected trajectories. The depth encoder and world-model backbone remain frozen.

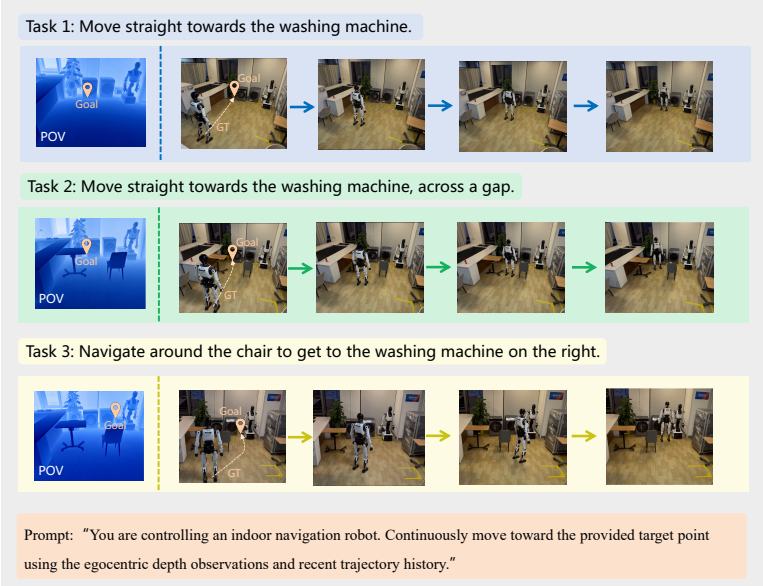

Figure 5: Real-robot pixel-goal navigation tasks. Each row shows the egocentric pixel goal and the corresponding execution sequence for a progressively more difficult setting: direct motion toward the goal, traversal across a constrained gap, and obstacle-aware navigation around a chair. Across all tasks, the initial goal coordinate remains fixed and the robot executes continuous motion using depth observations and recent trajectory history.