HiMem-WAM: Hierarchical Memory-Gated World Action Models for Robotic Manipulation

Xiaoquan Sun HUST
Ruijian Zhang HUST
Chen Cao HKU
Yihan Sun HUST
Jiahui Chen HKU
Zetian Xu HKU
Bo Chen HKU
Haijier Chen Wuhan University
Zhen Yang HKU
Jiarun Zhu SUST
Yijun Hong SUST
JingZhe Xu HUST
Jingrui Pang THU
Mingqi Yuan HKU
Jiayu Chen HKU & Infiforce
jiayuc@hku.hk

Abstract: We present a hierarchical latent action framework for unified world models that bridges short-horizon motion learning and long-horizon robotic manipulation. Our method retains motion-centric low-level latent actions learned from multi-view optical flow and weak action supervision, and builds a second abstraction layer that discovers variable-length latent skills from latent-action sequences. An online planner-executor policy predicts high-level skills, low-level latent-action chunks, and embodiment-specific controls from RGB observations, language, and proprioception, while a boundary-triggered memory mechanism writes compact key-frame states only at skill transitions. This design preserves strong motion priors from action-free videos while introducing temporal hierarchy and sparse memory for long-horizon control. We target a unified evaluation setting spanning long-horizon manipulation, domain-randomized bimanual control, and memory-intensive robot benchmarks.

Keywords: robot learning, latent actions, unified world models

1 Introduction

Large-scale VLA models have made language-conditioned robot policies increasingly scalable by transferring semantic priors from vision-language pretraining to action prediction [1, 2, 3, 4, 5, 6, 7, 8, 9]. Meanwhile, world-model-based methods use future prediction as dense supervision and provide a natural way to learn from heterogeneous visual data beyond fully action-labeled robot demonstrations [10, 11, 12, 13, 14, 15]. These two trends point toward a unified embodied model that jointly leverages semantic understanding, visual dynamics, and executable control. Despite this progress, current models still lack an explicit mechanism for temporal abstraction and memory. Most VLA policies predict actions or action chunks at a single control scale, while many unified world-action models focus on short-horizon motion prediction. This is effective for local execution, but long-horizon manipulation also requires identifying the active skill, detecting when a subtask changes, and retaining task-relevant observations for later decisions. Without a structured hierarchy, these long-horizon dependencies must be absorbed implicitly by a flat policy. We build on motion-centric latent actions and introduce hierarchy where it is most useful: on top of the low-level latent-action sequence. Motus [15] shows that optical-flow-based latent actions can bridge action-free videos and robot control, but its latent space remains primarily short-horizon. HiLAM [16] shows that temporally extended skills can be discovered from latent-action sequences, but does not study this idea inside a unified world-model policy with multi-view robot observations and memory-intensive tasks. Our method connects these directions by retaining the Motus-style

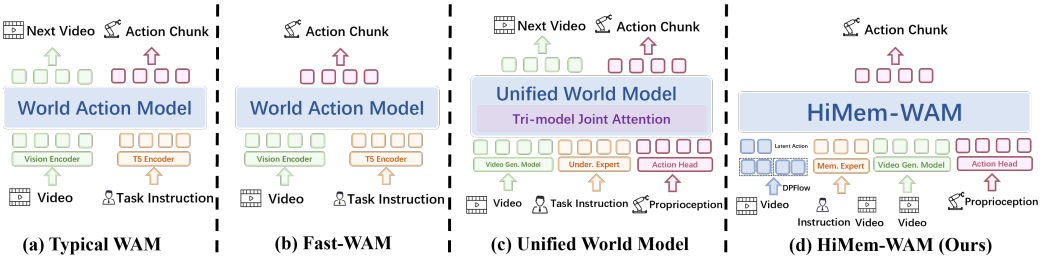

Figure 1: **From WAM to HiMem-WAM.** HiMem-WAM extends unified world action modeling with a memory expert, enabling action prediction conditioned on both current observations and task history.

low-level motion prior, discovering variable-length high-level skill latents, and using learned skill boundaries to trigger sparse key-frame memory updates. The resulting policy follows a planner-executor structure. A high-level planner predicts the current skill from RGB observations, language, proprioception, and memory; a low-level executor converts the skill into a latent-action chunk; and an embodiment-specific decoder maps the chunk to robot controls. This decomposition preserves executable motion priors while making long-horizon structure and memory updates explicit. We evaluate on LIBERO [17], RoboTwin 2.0 [18], and RMBench [19], which respectively emphasize long-horizon language following, domain-randomized bimanual control, and memory-dependent manipulation. Our contributions are summarized as follows:

- We propose a task-centric hierarchical latent action framework that extends a motion-centric unified world model with variable-length skill abstraction.
- We introduce a boundary-triggered memory mechanism that aligns memory writing with learned skill transitions.
- We establish a unified evaluation setting across LIBERO, RoboTwin 2.0, and RMBench for analyzing low-level motion priors, high-level skills, and memory in long-horizon manipulation.

2 Related Work

Vision-Language-Action Models. VLA models combine visual observations, language instructions, and action supervision in a unified policy architecture [1, 2, 3, 4, 5, 6, 7, 8, 9]. Their strength lies in transferring broad semantic priors to robot control, but most methods still predict actions at a single temporal scale. Our work is complementary: we preserve the direct policy interface of VLAs while adding an intermediate latent hierarchy that separates skill selection from motion execution.

World Models and Unified World Action Models. World-model-based approaches improve control by predicting future observations or using imagined futures as supervision [10, 11, 12, 13, 14, 15]. Unified models such as UWM [14] and Motus [15] further connect policy learning, inverse dynamics, and visual prediction within a shared framework. We follow this unified modeling direction, but focus on the missing temporal structure: how short-horizon latent actions can be organized into high-level skills and used for memory-aware control.

Latent Actions, Hierarchical Skills, and Memory. Latent action models infer compact motion representations from action-free videos, enabling policy pretraining and cross-embodiment transfer. Motus [15] demonstrates the effectiveness of optical-flow-based low-level latent actions, while HiLAM [16] shows that high-level skills can be discovered from latent-action sequences. RMBench [19] highlights that long-horizon manipulation also requires selective memory. Our method combines these ideas by using motion-centric low-level latents as the substrate for skill discovery and using learned skill boundaries as memory write signals.

3 Method

3.1 Overview and Problem Formulation

We study language-conditioned long-horizon manipulation with multi-view RGB observations, proprioception, and a sparse task memory. At timestep t, the robot observes $o_t = \{I_t^{(v)}\}_{v=1}^{V}$, proprioception p_t, instruction ℓ, and memory bank $\mathcal{M}_t$. The policy predicts an action chunk $\mathbf{a}_t = a_{t:t+K-1}$. Instead of directly mapping observations to actions at a single temporal scale, HiMem-WAM decomposes control into three levels:

$$\hat{z}_t^{\text{high}} = \pi_\theta^{\text{plan}}(o_t, p_t, \ell, \mathcal{M}_t), \qquad \hat{\mathbf{Z}}_t^{\text{low}} = \pi_\theta^{\text{exec}}(o_t, p_t, \hat{z}_t^{\text{high}}, \mathcal{M}_t), \qquad \hat{\mathbf{a}}_t = D_{\text{act}}(\hat{\mathbf{Z}}_t^{\text{low}}, o_t, p_t). \tag{1}$$

Here z_t^{high} represents the current skill, and $\mathbf{Z}_t^{\text{low}} = z_{t:t+K-1}^{\text{low}}$ denotes a chunk of short-horizon latent actions. Future observations and optical flow are used only to construct training supervision; inference is fully causal and requires only $(o_t, p_t, \ell, \mathcal{M}_t)$.

4 Method

4.1 Low-Level Latent Actions

We first learn a motion-centric latent-action tokenizer from short-horizon visual dynamics. For each transition, DPFlow [20] estimates multi-view optical flow $\mathbf{\Phi}_t = \{\Phi_t^{(v)}\}_{v=1}^{V}$ from $I_t^{(v)}$ to $I_{t+1}^{(v)}$. Given a short-horizon context c_t built from the available elements of $(o_t, o_{t+1}, p_t, \ell, \mathbf{\Phi}_t)$, the tokenizer encodes a latent action by

$$q_\phi(z_t^{\text{low}} \mid c_t) = \mathcal{N}(\mu_t, \text{diag}(\sigma_t^2)), \qquad z_t^{\text{low}} = \mu_t + \sigma_t \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{2}$$

The tokenizer is trained to reconstruct motion and, when action labels are available, weakly align the latent with executable controls:

$$\mathcal{L}_{\text{low}} = \|D_{\text{flow}}(z_t^{\text{low}}, o_t) - \mathbf{\Phi}_t\|_1 + \lambda_a \mathbb{I}_t^{\text{act}} \|D_{\text{align}}(z_t^{\text{low}}, o_t, p_t) - a_t\|_2^2 + \beta D_{\text{KL}}(q_\phi \| \mathcal{N}(0, I)). \tag{3}$$

Here $\mathbb{I}_t^{\text{act}}$ indicates whether the transition has an action annotation. After training, the tokenizer is frozen and applied offline to both robot trajectories and action-free videos to produce $Z^{\text{low}} = (z_1^{\text{low}}, \ldots, z_{T-1}^{\text{low}})$. Thus, action-free videos contribute motion supervision without requiring robot control labels.

4.2 High-Level Skill Discovery

Low-level latent actions capture local motion, but long-horizon tasks also require temporally extended skills. We therefore discover variable-length skill latents by dynamically chunking the low-level latent-action sequence. Let $Z^{(0)} = Z^{\text{low}}$. At hierarchy stage s, an encoder maps each token to $h_i^{(s)} = E_s(z_i^{(s)})$. We detect segment starts from adjacent-token dissimilarity:

$$r_i^{(s)} = \begin{cases} 1, & i = 1, \\ \frac{1}{2}\big(1 - (\hat{q}_{i-1}^{(s)})^\top \hat{k}_i^{(s)}\big), & i > 1, \end{cases} \qquad b_i^{(s)} = \mathbb{I}[r_i^{(s)} \geq \delta_s], \tag{4}$$

where $\hat{q}_i^{(s)}$ and $\hat{k}_i^{(s)}$ are normalized projections of $h_i^{(s)}$, and $b_i^{(s)} = 1$ indicates the start of a new segment. For each segment $\mathcal{I}_j^{(s)}$, we obtain the next-stage token by attention pooling,

$$z_j^{(s+1)} = \text{Pool}_{\text{attn}}\big(\{h_i^{(s)} \mid i \in \mathcal{I}_j^{(s)}\}\big). \tag{5}$$

Stacking H stages gives a shorter skill sequence $Z^{\text{high}} = Z^{(H)} = (z_1^{\text{high}}, \ldots, z_S^{\text{high}})$ with $S \ll T$. We unfold this sequence back to the original control timeline by assigning all timesteps in the same discovered segment the same skill target $\bar{z}_t^{\text{high}}$. The skill module is trained with

$$\mathcal{L}_{\text{skill}} = \mathcal{L}_{\text{next}} + \lambda_m \mathcal{L}_{\text{motion}} + \lambda_r \mathcal{L}_{\text{ratio}} + \lambda_c \mathcal{L}_{\text{cons}}, \tag{6}$$

where $\mathcal{L}_{\text{next}}$ predicts the next low-level latent, $\mathcal{L}_{\text{motion}}$ preserves visual-motion semantics, $\mathcal{L}_{\text{ratio}}$ prevents degenerate segment lengths, and $\mathcal{L}_{\text{cons}}$ encourages within-segment skill consistency.

4.3 Memory-Gated Planner-Executor

HiMem-WAM introduces memory as an external adapter to the planner-executor policy. We encode the current state as $x_t = E_\theta(o_t, p_t, \ell)$ and retrieve a memory context from $\mathcal{M}_t$:

$$c_t^{\text{mem}} = \text{Attn}(W_q x_t, W_k \mathcal{M}_t, W_v \mathcal{M}_t), \qquad \tilde{x}_t = x_t + \alpha_t^{\text{read}} W_m c_t^{\text{mem}}, \quad \alpha_t^{\text{read}} = \sigma(G_{\text{read}}(x_t, c_t^{\text{mem}})). \tag{7}$$

The high-level planner, instantiated with Qwen3-VL-4B-Instruct, predicts the current skill and a boundary score from the memory-adapted state:

$$\hat{z}_t^{\text{high}}, \hat{b}_t = \pi_\theta^{\text{QwenPlan}}(\tilde{x}_t, c_t^{\text{mem}}), \qquad \hat{\mathbf{Z}}_t^{\text{low}} = \pi_\theta^{\text{exec}}(\tilde{x}_t, \hat{z}_t^{\text{high}}), \qquad \hat{\mathbf{a}}_t = D_{\text{act}}(\hat{\mathbf{Z}}_t^{\text{low}}, \tilde{x}_t). \tag{8}$$

The boundary score also controls memory writing. A compact memory token γ_t is produced from the current state, predicted skill, and predicted latent-action chunk. The memory bank is updated only when the write gate is active:

$$\alpha_t^{\text{write}} = \sigma(G_{\text{write}}(\tilde{x}_t, \hat{z}_t^{\text{high}}, \hat{b}_t)), \qquad \mathcal{M}_{t+1} = \begin{cases} U_\psi(\mathcal{M}_t, \gamma_t), & \alpha_t^{\text{write}} > \eta, \\ \mathcal{M}_t, & \text{otherwise}, \end{cases} \tag{9}$$

where $\gamma_t = \Gamma_\psi(\tilde{x}_t, \hat{z}_t^{\text{high}}, \text{Pool}(\hat{\mathbf{Z}}_t^{\text{low}}))$. This sparse update stores task-relevant key states mainly at skill transitions, reducing memory noise during long-horizon execution.

4.4 Training Pipeline

We train HiMem-WAM in three stages. **Stage I** learns the low-level tokenizer with Eq. (3) and extracts offline latent-action sequences for all training videos and robot trajectories. **Stage II** discovers high-level skill latents and pretrains the planner and executor without external memory. This stage provides three pseudo-labels: the unfolded skill target $\bar{z}_t^{\text{high}}$, the low-level latent-action chunk $\mathbf{Z}_t^{\text{low}}$, and the boundary label $\bar{b}_t$. The pretraining loss is

$$\mathcal{L}_{\text{latent}} = \lambda_h \|\hat{z}_t^{\text{high}} - \bar{z}_t^{\text{high}}\|_2^2 + \lambda_l \|\hat{\mathbf{Z}}_t^{\text{low}} - \mathbf{Z}_t^{\text{low}}\|_2^2 + \lambda_b \text{BCE}(\hat{b}_t, \bar{b}_t). \tag{10}$$

Stage III fine-tunes the full policy on target robot demonstrations and activates gated memory. We optimize

$$\mathcal{L}_{\text{ft}} = \mathcal{L}_{\text{act}} + \alpha_h \mathcal{L}_{\text{plan}} + \alpha_l \mathcal{L}_{\text{exec}} + \alpha_b \mathcal{L}_{\text{bd}} + \alpha_m \Big(\text{BCE}(\alpha_t^{\text{write}}, \bar{b}_t) + \lambda_s \big(\|\alpha_t^{\text{read}}\|_1 + \|\alpha_t^{\text{write}}\|_1 \big) \Big), \tag{11}$$

where $\mathcal{L}_{\text{act}}$ is either negative log-likelihood for stochastic action heads or MSE for deterministic action heads, and $(\mathcal{L}_{\text{plan}}, \mathcal{L}_{\text{exec}}, \mathcal{L}_{\text{bd}})$ are the Stage-II auxiliary losses. At inference, the policy retrieves memory, predicts a skill, generates a low-level latent-action chunk, decodes executable actions, and writes memory only when $\alpha_t^{\text{write}} > \eta$. Neither future video generation nor optical-flow estimation is required during deployment.

Low-level latent actions. We first learn a low-level latent-action tokenizer from short-horizon visual motion. For each transition, we compute multi-view optical flow using DPFlow [20], $\mathbf{\Phi}_t = \{\Phi_t^{(v)}\}_{v=1}^V$, where $\Phi_t^{(v)}$ denotes the optical flow from $I_t^{(v)}$ to $I_{t+1}^{(v)}$. The tokenizer encodes adjacent observations, optical flow, and available robot context into a compact latent action:

$$q_\phi(z_t^{\text{low}} \mid c_t) = \mathcal{N}(\mu_t, \text{diag}(\sigma_t^2)), \qquad z_t^{\text{low}} = \mu_t + \sigma_t \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{12}$$

Here c_t denotes the fused short-horizon motion context constructed from the available elements of $(o_t, o_{t+1}, p_t, \ell, \mathbf{\Phi}_t)$. The latent is trained to reconstruct visual motion and, when action annotations are available, weakly align with real robot actions:

$$\mathcal{L}_{\text{low}} = \|\hat{\mathbf{\Phi}}_t - \mathbf{\Phi}_t\|_1 + \lambda_a \mathbb{I}_t^{\text{act}} \|\hat{a}_t - a_t\|_2^2 + \beta D_{\text{KL}}\big(q_\phi(z_t^{\text{low}} \mid c_t) \,\|\, \mathcal{N}(0, I)\big). \tag{13}$$

Here $\hat{\mathbf{\Phi}}_t = D_{\text{flow}}(z_t^{\text{low}}, o_t)$ is the reconstructed multi-view flow, $\hat{a}_t = D_{\text{align}}(z_t^{\text{low}}, o_t, p_t)$ is an auxiliary action prediction used only for action alignment, and $\mathbb{I}_t^{\text{act}} \in \{0, 1\}$ indicates whether an action annotation is available at timestep t. After training, the tokenizer is applied offline to all robot trajectories and videos without action annotations to produce low-level latent-action sequences

$Z^{\text{low}} = (z_1^{\text{low}}, \ldots, z_{T-1}^{\text{low}})$. DPFlow is only used in this offline tokenizer-learning stage; the deployed policy does not require flow estimation.

High-level skill latents. Low-level latent actions describe local motion, but long-horizon manipulation requires temporally extended abstractions. We therefore build a second abstraction layer on top of Z^{low}. Let $Z^{(0)} = Z^{\text{low}}$ and denote the length of $Z^{(s)}$ by L_s, where $L_0 = T-1$. At hierarchy stage s, each token is encoded as $h_i^{(s)} = E_s(z_i^{(s)})$ for $i = 1, \ldots, L_s$. We compute normalized query and key features $\hat{q}_i^{(s)} = W_q^{(s)} h_i^{(s)} / \|W_q^{(s)} h_i^{(s)}\|_2$ and $\hat{k}_i^{(s)} = W_k^{(s)} h_i^{(s)} / \|W_k^{(s)} h_i^{(s)}\|_2$, and detect skill boundaries by measuring the motion change between adjacent latent-motion tokens:

$$r_i^{(s)} = \begin{cases} 1, & i = 1, \\ \frac{1}{2}\left(1 - (\hat{q}_{i-1}^{(s)})^\top \hat{k}_i^{(s)}\right), & i > 1, \end{cases} \qquad b_i^{(s)} = \begin{cases} 1, & i = 1, \\ \mathbb{I}\left[r_i^{(s)} \geq \delta_s\right], & i > 1. \end{cases} \tag{14}$$

Here $r_i^{(s)}$ measures the motion dissimilarity between two adjacent tokens, and $b_i^{(s)}$ indicates whether a new skill segment starts at position i of the stage-s sequence. Given the ordered boundary indices $\mathcal{B}^{(s)} = \{i_j^{(s)}\}_{j=1}^{L_{s+1}} = \{i \mid b_i^{(s)} = 1\}$, we set $i_{L_{s+1}+1}^{(s)} = L_s + 1$ and define the j-th segment as $\mathcal{I}_j^{(s)} = \{i \mid i_j^{(s)} \leq i < i_{j+1}^{(s)}\}$. Each variable-length segment is summarized into a coarser token by attention pooling:

$$\alpha_{j,i}^{(s)} = \frac{\exp((w_s)^\top h_i^{(s)})}{\sum_{k \in \mathcal{I}_j^{(s)}} \exp((w_s)^\top h_k^{(s)})}, \qquad z_j^{(s+1)} = \sum_{i \in \mathcal{I}_j^{(s)}} \alpha_{j,i}^{(s)} h_i^{(s)}. \tag{15}$$

Here w_s is a learnable attention vector at hierarchy stage s. After stacking H hierarchy stages, we obtain a temporally coarser skill sequence $Z^{\text{high}} = Z^{(H)} = (z_1^{\text{high}}, \ldots, z_S^{\text{high}})$, where $S = L_H \ll T$. For per-timestep policy supervision, we unfold the final skill sequence back to the original control timeline. Let $\bar{b}_t$ be the final boundary indicator at the original temporal resolution, with $\bar{b}_1 = 1$. We define $\kappa_t = \sum_{\tau=1}^{t} \bar{b}_\tau$ and assign $\bar{z}_t^{\text{high}} = z_{\kappa_t}^{\text{high}}$ for $t = 1, \ldots, T-1$. Thus, all original timesteps within the same discovered segment share the same high-level skill target. The hierarchy is trained with

$$\mathcal{L}_{\text{skill}} = \mathcal{L}_{\text{next}} + \lambda_m \mathcal{L}_{\text{motion}} + \lambda_r \mathcal{L}_{\text{ratio}} + \lambda_c \mathcal{L}_{\text{cons}}. \tag{16}$$

Here $\mathcal{L}_{\text{next}}$ predicts the next low-level latent action, $\mathcal{L}_{\text{motion}}$ preserves the visual-motion semantics of predicted latents, $\mathcal{L}_{\text{ratio}}$ prevents degenerate segment lengths, and $\mathcal{L}_{\text{cons}}$ encourages timesteps within the same segment to share a coherent skill representation.

Gated External Memory. We introduce memory as an external adapter rather than a separate policy branch. Given observation, proprioception, and instruction, we first compute a state representation $x_t = E_\theta(o_t, p_t, \ell)$. The memory bank $\mathcal{M}_t = \{m_i\}_{i=1}^{N_t}$ stores compact skill-level tokens. A read gate retrieves a memory context $c_t^{\text{mem}} = \text{Attn}(W_q x_t, W_k \mathcal{M}_t, W_v \mathcal{M}_t)$ and forms a memory-adapted state $\tilde{x}_t = x_t + \alpha_t^{\text{read}} W_m c_t^{\text{mem}}$, where $\alpha_t^{\text{read}} = \sigma(G_{\text{read}}(x_t, c_t^{\text{mem}}))$.

We instantiate the high-level planner with Qwen3-VL-4B-Instruct. The planner takes the current visual-language state and the retrieved memory context as input, and predicts the current high-level skill latent together with a boundary score:

$$\hat{z}_t^{\text{high}}, \hat{b}_t = \pi_\theta^{\text{QwenPlan}}(\tilde{x}_t, c_t^{\text{mem}}).$$

Here $\hat{z}_t^{\text{high}}$ specifies the skill to execute, while $\hat{b}_t$ estimates whether the current timestep corresponds to a skill transition. Conditioned on this skill, the executor predicts a low-level latent-action chunk and the action decoder maps it to executable robot actions:

$$\hat{\mathbf{Z}}_{t:t+K-1}^{\text{low}} = \pi_\theta^{\text{exec}}(\tilde{x}_t, \hat{z}_t^{\text{high}}), \qquad \hat{\mathbf{a}}_{t:t+K-1} = D_{\text{act}}(\hat{\mathbf{Z}}_{t:t+K-1}^{\text{low}}, \tilde{x}_t).$$

Memory writing is controlled by a write gate:

$$\alpha_t^{\text{write}} = \sigma(G_{\text{write}}(\tilde{x}_t, \hat{z}_t^{\text{high}}, \hat{b}_t)), \qquad \gamma_t = \Gamma_\psi(\tilde{x}_t, \hat{z}_t^{\text{high}}, \text{Pool}(\hat{\mathbf{Z}}_{t:t+K-1}^{\text{low}})).$$

The memory bank is updated as

$$\mathcal{M}_{t+1} = \begin{cases} U_\psi(\mathcal{M}_t, \gamma_t), & \alpha_t^{\text{write}} > \eta, \\ \mathcal{M}_t, & \text{otherwise.} \end{cases}$$

This sparse write rule stores skill-level events only when the planner detects meaningful task transitions.

Training pipeline. We train the model in three stages.

Stage I: Latent-action tokenizer learning. We train the tokenizer with Eq. (13) to reconstruct multi-view motion and weakly align latent actions with real robot actions when action annotations are available. After training, the tokenizer is applied offline to robot trajectories and videos without action annotations to produce low-level latent-action sequences $\mathbf{Z}^{\text{low}} = (z_1^{\text{low}}, \ldots, z_{T-1}^{\text{low}})$. Optical flow is only used in this offline stage; the deployed policy does not require flow estimation.

Stage II: hierarchical latent-action pretraining. We learn high-level skill latents by dynamically chunking $\mathbf{Z}^{\text{low}}$ with Eq. (14) and Eq. (15). This stage produces three pseudo-labels for each trajectory: the per-timestep skill target $\bar{z}_t^{\text{high}}$, the low-level latent-action target $\mathbf{Z}_{t:t+K-1}^{\text{low}}$, and the skill-boundary label $\bar{b}_t$. We then pretrain the planner and executor without memory:

$$\mathcal{L}_{\text{latent}} = \lambda_h \mathcal{L}_{\text{plan}} + \lambda_l \mathcal{L}_{\text{exec}} + \lambda_b \mathcal{L}_{\text{bd}}. \tag{17}$$

where

$$\mathcal{L}_{\text{plan}} = \left\| \hat{z}_t^{\text{high}} - \bar{z}_t^{\text{high}} \right\|_2^2, \quad \mathcal{L}_{\text{exec}} = \left\| \hat{\mathbf{Z}}_{t:t+K-1}^{\text{low}} - \mathbf{Z}_{t:t+K-1}^{\text{low}} \right\|_2^2, \quad \mathcal{L}_{\text{bd}} = \text{BCE}(\hat{b}_t, \bar{b}_t).$$

This stage trains the policy to predict which skill to execute, how to expand it into low-level latent motions, and when a skill transition occurs.

Stage III: action grounding with gated memory. Finally, we finetune the policy on robot demonstrations with action annotations and activate the gated external memory module. The executor and action decoder are trained to ground latent motions into executable actions, while the memory write gate is supervised by the skill-boundary label $\bar{b}_t$:

$$\mathcal{L}_{\text{ft}} = \mathcal{L}_{\text{act}} + \alpha_h \mathcal{L}_{\text{plan}} + \alpha_l \mathcal{L}_{\text{exec}} + \alpha_b \mathcal{L}_{\text{bd}} + \alpha_m \mathcal{L}_{\text{gate}}, \tag{18}$$

where $\mathcal{L}_{\text{act}} = -\log \pi_\theta(\mathbf{a}_{t:t+K-1} \mid o_t, p_t, \ell, \mathcal{M}_t)$ for stochastic policies, or $\mathcal{L}_{\text{act}} = \|\hat{\mathbf{a}}_{t:t+K-1} - \mathbf{a}_{t:t+K-1}\|_2^2$ for deterministic policies. The gate loss is

$$\mathcal{L}_{\text{gate}} = \text{BCE}(\alpha_t^{\text{write}}, \bar{b}_t) + \lambda_{\text{read}} \|\alpha_t^{\text{read}}\|_1 + \lambda_{\text{write}} \|\alpha_t^{\text{write}}\|_1. \tag{19}$$

The first term trains the write gate to store memory primarily at skill transitions, while the sparsity terms discourage dense memory reading and writing. At inference time, the policy is fully causal: it retrieves memory through the read gate, predicts a skill latent and a low-level latent-action chunk, decodes executable robot actions, and writes memory only when $\alpha_t^{\text{write}} > \eta$. No future video generation and optical flow estimation is required during deployment.

5 Experiments

We design our experiments to evaluate whether HiMem-WAM improves robotic manipulation performance. In particular, we focus on the following questions:

- **Q1:** *How does HiMem-WAM compare with existing VLA and WAM on standard and Memory manipulation benchmarks?*
- **Q2:** *Do latent actions provide effective motion priors for robotic manipulation?*
- **Q3:** *Does the gated memory module enhance long-horizon memory in robotic manipulation?*
- **Q4:** *Can HiMem-WAM be effectively deployed on real-world robotic tasks?*

5.1 Simulation Experimental Setup

Benchmark Selection. We evaluate HiMem-WAM on three benchmarks: LIBERO [17], LIBERO-PLUS [21], and RMBench [19]. LIBERO [17] evaluates language-conditioned manipulation across

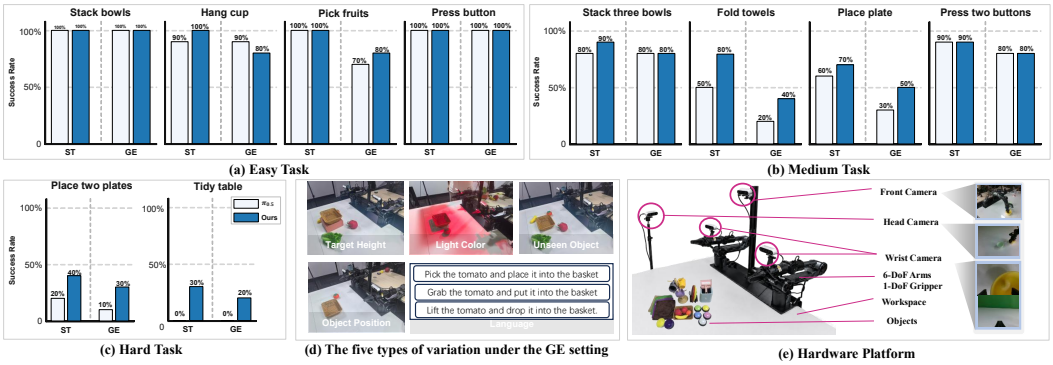

Figure 2: **Real-world evaluation on 10 tasks.** We evaluate HiMem-WAM on 10 real-world tasks under both the **ST** and **GE** settings. (a)–(c) report **SR** across three task categories with different manipulation requirements, (d) illustrates the evaluation variations in the **GE** setting, (e) illustrates the hardware platform.

four task suites, including spatial, object, goal, and long-horizon tasks. LIBERO-PLUS [21] evaluates policy robustness under deployment-time perturbations across vision, language, initialization, and scene layout. RMBench [19] evaluates memory-dependent manipulation tasks that require the agent to retain and reuse task-relevant information across long horizons.

Evaluation Metrics. We use **S**uccess **R**ate (**SR**) as the evaluation metric across all benchmarks. On LIBERO [17] and LIBERO-PLUS [21], we evaluate 50 rollouts per task. On RMBench [19], we evaluate 100 rollouts per task.

Algorithmic Baselines. We evaluate HiMem-WAM against a collection of representative and competitive baselines, including DP [22], ACT [23], $\pi_{0.5}$ [7], OpenVLA [24], X-VLA [8], MEM-0 [19], AtomVLA [25], WorldVLA [26], LingBot-VA [27], Fast-WAM [28] and other baselines.

5.2 Results Analysis

Key Finding 1: Latent actions improve robustness by capturing transferable motion patterns. As shown in Tables 2 and 3, *Stage II* latent action pretraining achieves gains of **+1.1%** on the standard LIBERO benchmark and **+3.8%** on the LIBERO-PLUS benchmark (Zero-Shot), where models are trained only on the standard LIBERO dataset. The larger gain under deployment perturbations, including camera and viewpoint variations and observation noise, suggests that latent actions capture task completion motion rather than overfitting to clean visual trajectories. This provides a stable motion prior for action prediction and improves robustness under diverse deployment perturbations.

Key Finding 2: Gated memory improves task state tracking, while stronger memory planning is still needed. As shown in Table 1, HiMem-WAM achieves a total average SR of **26.3%** on RMBench and reaches **31.5%** on $M(1)$ tasks. This shows that the memory module is effective: the read gate retrieves task history to form a memory adapted state, while the write gate stores compact key states only when meaningful task transitions are detected. As a result, the Qwen3-VL-4B planner can condition action prediction on both the current observation and previously observed task states, which improves target tracking, phase recognition, and memory dependent manipulation. However, performance drops to **19.8%** on $M(n)$ tasks and remains below Mem-0 [19], indicating that repeated trials and multi step state updates still require stronger memory reasoning. This gap is reasonable because Mem-0 uses a larger 8B planner and a more specialized memory planning design, whereas HiMem-WAM integrates gated memory into a general world action policy together with latent action priors.

Table 1: Comparisons with other baselines on RMBench [19] benchmark.

Tasks	TMC	DP	ACT	$\pi_{0.5}$	X-VLA	Mem-0	**HiMem-WAM (Ours)**
Observe and Pick Up	$M(1)$	1%	1%	9%	9%	4%	**28%**
Rearrange Blocks	$M(1)$	0%	29%	13%	13%	**89%**	33%
Put Back Block	$M(1)$	0%	0%	11%	18%	**90%**	32%
Swap Blocks	$M(1)$	11%	2%	24%	16%	**67%**	38%
Swap T	$M(1)$	2%	2%	15%	3%	14%	**27%**
Average	$M(1)$	6.4%	6.8%	14.4%	11.8%	**52.8%**	31.5%
Battery Try	$M(n)$	10%	19%	16%	26%	28%	**28%**
Blocks Ranking Try	$M(n)$	10%	0%	6%	1%	18%	**24%**
Cover Blocks	$M(n)$	0%	0%	2%	0%	**68%**	19%
Press Button	$M(n)$	0%	0%	1%	2%	0%	**8%**
Average	$M(n)$	5.0%	4.8%	5.5%	7.3%	**28.5%**	19.8%
Total Average	/	5.8%	5.9%	10.4%	9.8%	**42.0%**	26.3%

Note: We report **SR** (%). RMBench includes nine manipulation tasks across the $M(1)$ and $M(n)$ levels of Task Memory Complexity (TMC). **Bold** denotes the best performance among all methods.

5.3 Real-World Experimental Setup

Hardware Platform. We conduct real-world experiments on a static tabletop dual-arm platform built with two AgileX Piper 6-DoF robotic arms. The system is equipped with four Intel RealSense D435i RGB cameras, including two wrist cameras, one head camera, and one front-view camera, as shown in Figure 2(e).

Tasks Settings. We evaluate real-world deployment on a suite of manipulation tasks organized into three difficulty categories: **Easy**, **Medium**, and **Hard**. Easy tasks focus on short-horizon single-arm manipulation, Medium tasks require basic bimanual coordination, and Hard tasks involve long-horizon bimanual manipulation with more complex object interactions.

Evaluation Settings. Each task is evaluated under two settings: Standard **(ST)**, where the scene configuration is clean and consistent with the training data, and Generalization **(GE)**, where we introduce a range of deployment time perturbations to evaluate robustness in more challenging environments. We report **S**uccess **R**ate (**SR**) separately for each difficulty category and evaluation setting. Detailed task definitions and the full specification of the **GE** setting are provided in Appendix B.

Training Details. Each task consists of 100 demonstrations. The model undergoes supervised fine-tuning (SFT) on the real-world dataset for 2 epochs, followed by post-training, where we sample 100K trajectories with 10 candidates per state. During evaluation, we conduct 20 trials for each task. Specifically, under the GE setting, the environmental conditions of these 20 trials are randomly distributed across the four perturbation types to rigorously assess robustness. Detailed hyperparameters are provided in the supplementary material.

5.4 Results Analysis

Key Finding 3: HiMem-WAM improves robustness and generalization in real-world robotic tasks. As shown in Figure 2 and Table 4, HiMem-WAM performs comparably to $\pi_{0.5}$ [7] on Easy tasks, where both policies already achieve high success rates, confirming that $\pi_{0.5}$ remains a strong baseline for basic manipulation. The advantage of HiMem-WAM becomes clearer on Medium tasks, with gains of **+12.5%** under ST and **+10.0%** under GE, suggesting that latent action pretraining provides predictive motion priors that help stabilize real-world action prediction under object, light and instruction variations. The advantage becomes most pronounced on Hard tasks, with gains of **+25.0%** under ST and **+20.0%** under GE. These tasks require long-horizon execution and persistent task-state maintenance, while SFT adaptation often struggles with history forgetting and fails to track the correct task progress. In contrast, the gated memory module helps HiMem-WAM retain

Table 2: Comparisons with other baselines on LIBERO benchmark.

Method	Spatial	Object	Goal	Long	Avg.
Vision-Language-Action Models					
OpenVLA	84.7	88.4	79.2	53.7	76.5
SpatialVLA	88.2	89.9	78.6	55.5	78.1
π_0	96.8	98.8	95.8	85.2	94.2
NORA-1.5	97.3	96.4	94.5	89.6	94.5
AtomVLA	96.4	99.6	97.6	94.4	97.0
World Action Models					
WorldVLA	87.6	96.2	83.4	60.0	81.8
Fast-WAM	**98.2**	**100.0**	97.0	**95.2**	97.6
Ours					
w/o Stage II	96.0	99.6	97.1	93.8	96.6
HiMem-WAM	**98.2**	99.8	**98.4**	94.5	**97.7**

Table 3: Comparisons with other baselines on LIBERO-PLUS benchmark. (Zero-Shot, train on the standard LIBERO dataset only)

Method	Cam.	Init	Lang.	Light	BG.	Noise	Layout	Avg.
Vision-Language-Action Models								
OpenVLA	0.8	3.5	23.0	8.1	34.8	15.2	28.5	16.3
OpenVLA-OFT	56.4	31.9	**79.5**	88.7	**93.3**	75.8	74.2	71.4
π_0	13.8	6.0	58.8	85.0	81.4	79.0	68.9	56.1
UniVLA	1.8	46.2	69.6	69.0	81.0	21.2	31.9	45.8
RIPT-VLA	55.2	31.2	77.6	88.4	91.6	73.5	74.2	70.2
World Action Models								
WorldVLA	0.1	27.9	41.6	43.7	17.1	10.9	38.0	25.6
HoloBrain-0	65.5	**58.2**	78.7	88.1	90.3	66.9	**79.5**	75.3
Ours								
w/o Stage II	77.2	37.9	71.7	91.1	83.6	73.2	70.7	72.2
HiMem-WAM	**78.2**	38.1	76.6	**92.2**	91.0	**80.7**	74.9	**76.0**

Note: We report **SR** (%). **Bold** denotes the best performance among all methods. **Cam.** denotes camera perturbation; **Init.** denotes initial-state perturbation; **Lang.** denotes language perturbation; **Light** denotes lighting perturbation; **BG.** denotes background perturbation; **Noise** denotes observation noise; **Layout** denotes layout perturbation; **Avg.** denotes average performance.

Table 4: Category-level real-world success rates of HiMem-WAM.

Task Category	Tasks	ST SR (%)	GE SR (%)
Easy	4	100.0	90.0
Medium	4	82.5	62.5
Hard	2	35.0	25.0

historical information and task states, enabling the policy to continue long-horizon task execution.
Key Finding 4:

6 Conclusion

We presented HiMem-WAM, a hierarchical memory gated world action model for long horizon robotic manipulation. HiMem-WAM combines latent action pretraining, skill abstraction, and gated memory to connect motion execution with task state retention. It learns motion priors from multi-view visual dynamics and uses discovered skill boundaries to control sparse memory writing, enabling causal action prediction from current observations and compact task history. Experiments in simulation and on real robots show that latent actions improve robustness under deployment perturbations, gated memory supports memory-dependent manipulation, and the full system brings larger gains as task difficulty increases. These results validate the benefit of coupling latent motion priors with task memory for long horizon robot control.

References

[1] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[2] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024.

[3] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu, J. Luo, et al. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[4] H. Wu, Y. Jing, C. Cheang, G. Chen, J. Xu, X. Li, M. Liu, H. Li, and T. Kong. Unleashing large-scale video generative pre-training for visual robot manipulation. *ICLR*, 2024.

[5] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *The Thirteenth International Conference on Learning Representations*, 2024.

[6] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[7] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. R. Equi, C. Finn, N. Fusai, M. Y. Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[8] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[9] W. Song, J. Chen, X. Sun, H. Lei, Y. Qin, W. Zhao, P. Ding, H. Zhao, T. Wang, P. Hou, et al. Rethinking the practicality of vision-language-action model: A comprehensive benchmark and an improved baseline. *arXiv preprint arXiv:2602.22663*, 2026.

[10] Y. Du, S. Yang, B. Dai, H. Dai, O. Nachum, J. Tenenbaum, D. Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems*, volume 36, pages 9156–9172, 2023.

[11] S. Zhou, Y. Du, J. Chen, Y. Li, D.-Y. Yeung, and C. Gan. Robodreamer: Learning compositional world models for robot imagination. In *International Conference on Machine Learning*, pages 61885–61896, 2024.

[12] Y. Feng, H. Tan, X. Mao, C. Xiang, G. Liu, S. Huang, H. Su, and J. Zhu. Vidar: Embodied video diffusion model for generalist manipulation. *arXiv preprint arXiv:2507.12898*, 2025.

[13] Q. Lv, W. Kong, H. Li, J. Zeng, Z. Qiu, D. Qu, H. Song, Q. Chen, X. Deng, and J. Pang. F1: A vision-language-action model bridging understanding and generation to actions. *arXiv preprint arXiv:2509.06951*, 2025.

[14] C. Zhu, R. Yu, S. Feng, B. Burchfiel, P. Shah, and A. Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. *arXiv preprint arXiv:2504.02792*, 2025.

[15] H. Bi, H. Tan, S. Xie, Z. Wang, S. Huang, H. Liu, R. Zhao, Y. Feng, C. Xiang, Y. Rong, H. Zhao, H. Liu, Z. Su, L. Ma, H. Su, and J. Zhu. Motus: A unified latent action world model. *arXiv preprint arXiv:2512.13030*, 2025.

[16] H. Kim, L. Pinto, and S. J. Kim. Hierarchical latent action model. *arXiv preprint arXiv:2603.05815*, 2026.

[17] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[18] T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Q. Liang, Z. Li, X. Lin, Y. Ge, Z. Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[19] T. Chen, Y. Wang, M. Li, Y. Qin, H. Shi, Z. Li, Y. Hu, Y. Zhang, K. Wang, Y. Chen, et al. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *arXiv preprint arXiv:2603.01229*, 2026.

[20] H. Morimitsu, X. Zhu, R. M. Cesar-Jr., X. Ji, and X.-C. Yin. DPFlow: Adaptive optical flow estimation with a dual-pyramid framework. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[21] S. Fei, S. Wang, J. Shi, Z. Dai, J. Cai, P. Qian, L. Ji, X. He, S. Zhang, Z. Fei, J. Fu, J. Gong, and X. Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

[22] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[23] T. Zhao et al. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[24] M. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[25] X. Sun, Z. Xu, C. Cao, Z. Liu, Y. Sun, J. Pang, R. Zhang, Z. Yang, K. Pang, D. He, et al. Atomvla: Scalable post-training for robotic manipulation via predictive latent world models. *arXiv preprint arXiv:2603.08519*, 2026.

[26] J. Cen, C. Yu, H. Yuan, Y. Jiang, S. Huang, J. Guo, X. Li, Y. Song, H. Luo, F. Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

[27] L. Li, Q. Zhang, Y. Luo, S. Yang, R. Wang, F. Han, M. Yu, Z. Gao, N. Xue, X. Zhu, Y. Shen, and Y. Xu. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

[28] T. Yuan, Z. Dong, Y. Liu, and H. Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

A Supplementary Method Details

This section complements the method description in the main paper. We focus on implementation details that are omitted from the main text for clarity: how training-time motion signals are converted into low-level latent-action pseudo-labels, how hierarchical skill labels are unfolded back to the original control timeline, how the Qwen3-VL-Instruct planner uses the external memory bank, and how losses are applied across the three training stages.

A.1 Separation Between Offline Supervision and Online Inputs

A key design choice of HiMem-WAM is to use rich motion supervision during training while keeping inference causal and lightweight. Table 5 summarizes which signals are used in each stage.

Table 5: Signals used during training and inference. Optical flow and future observations are used only to construct supervision for latent-action learning; they are not required by the deployed policy.

Signal	Stage I	Stage II	Stage III	Inference
RGB observations o_t	✓	✓	✓	✓
Proprioception p_t	✓	✓	✓	✓
Instruction ℓ	✓	✓	✓	✓
Optical flow $\mathbf{\Phi}_t$	✓	supervision only	–	–
Action annotations a_t	optional	–	✓	–
Low-level latents Z^{low}	output	supervision	supervision	predicted
High-level skills $\bar{z}_t^{\text{high}}$	–	supervision	auxiliary supervision	predicted
Boundary labels $\bar{b}_t$	–	supervision	gate supervision	predicted
External memory $\mathcal{M}_t$	–	disabled	✓	✓

Stage I uses optical flow to learn a compact latent-action space. Stage II uses the extracted latent actions to discover high-level skills and pretrain the planner and executor without memory. Stage III activates the gated external memory module and grounds the latent policy into executable actions. At inference time, the policy receives only RGB observations, proprioception, instruction, and the current memory bank.

A.2 Low-Level Latent-Action Tokenizer Details

The main paper defines the low-level latent posterior and training objective. Here we specify the preprocessing and masking details used to construct the tokenizer supervision.

Optical-flow preprocessing. For each view v, DPFlow produces a dense flow field

$$\Phi_t^{(v)} = \text{DPFlow}(I_t^{(v)}, I_{t+1}^{(v)}).$$

Before being passed to the flow encoder, the horizontal and vertical flow components are normalized by image width and height:

$$\widetilde{\Phi}_t^{(v)}(x,y) = \left[\frac{\Phi_{t,x}^{(v)}(x,y)}{W_v}, \frac{\Phi_{t,y}^{(v)}(x,y)}{H_v} \right], \tag{20}$$

where W_v and H_v are the image width and height of view v. This normalization keeps the flow scale comparable across camera views and resolutions.

Multi-view fusion. Each view is encoded independently before fusion. We use a view embedding e_v to preserve camera identity:

$$m_t^{(v)} = E_{\text{flow}}(\widetilde{\Phi}_t^{(v)}) + e_v, \qquad s_t^{(v)} = E_{\text{vis}}(I_t^{(v)}) + e_v. \tag{21}$$

The motion and semantic features are fused across views:

$$m_t = \text{Fuse}_{\text{mot}}(\{m_t^{(v)}\}_{v=1}^{V}), \qquad s_t = \text{Fuse}_{\text{sem}}(\{s_t^{(v)}\}_{v=1}^{V}). \tag{22}$$

Together with proprioception and instruction features, they form the short-horizon context c_t used by the latent-action posterior in the main text.

Action-annotation masking. The low-level tokenizer can be trained on both trajectories with action annotations and videos without action annotations. Let $\mathbb{I}_t^{\text{act}} \in \{0, 1\}$ indicate whether a_t is available. The action-alignment term is applied only when $\mathbb{I}_t^{\text{act}} = 1$:

$$\mathcal{L}_{\text{align}} = \frac{1}{\sum_{t=1}^{T-1} \mathbb{I}_t^{\text{act}}} \sum_{t=1}^{T-1} \mathbb{I}_t^{\text{act}} \left\| D_{\text{align}}(z_t^{\text{low}}, o_t, p_t) - a_t \right\|_2^2. \tag{23}$$

Thus, videos without action annotations still contribute to flow reconstruction and KL regularization, while robot trajectories with action annotations additionally align the latent space with executable controls.

Offline latent extraction. After Stage I, the tokenizer is frozen and applied offline to all training trajectories and videos. For each sequence, we store the low-level latent-action sequence

$$Z^{\text{low}} = (z_1^{\text{low}}, \ldots, z_{T-1}^{\text{low}}).$$

These stored latents are used as pseudo-labels in Stage II and Stage III. DPFlow is not called during policy training after this offline extraction step, and it is not used during inference.

A.3 Skill Boundary Unfolding and Pseudo-Label Construction

The main paper describes dynamic chunking at each hierarchy stage. This subsection clarifies how the final high-level skill sequence is aligned back to the original control timeline.

Let $Z^{(0)} = Z^{\text{low}}$ and let L_s be the length of $Z^{(s)}$. At each hierarchy stage s, the boundary predictor outputs $b_i^{(s)}$ for positions $i = 1, \ldots, L_s$. Since higher stages operate on progressively shorter sequences, we maintain an index map ψ_s from a stage-s token index to its starting timestep in the original low-level sequence. At the bottom level,

$$\psi_0(i) = i, \qquad i = 1, \ldots, L_0. \tag{24}$$

If the boundary indices at stage s are

$$\mathcal{B}^{(s)} = \{i_j^{(s)}\}_{j=1}^{L_{s+1}},$$

then the index map for the next stage is updated by

$$\psi_{s+1}(j) = \psi_s(i_j^{(s)}), \qquad j = 1, \ldots, L_{s+1}. \tag{25}$$

After H hierarchy stages, the high-level skill sequence is

$$Z^{\text{high}} = Z^{(H)} = (z_1^{\text{high}}, \ldots, z_S^{\text{high}}), \qquad S = L_H.$$

The final boundary indicator at the original temporal resolution is

$$\bar{b}_t = \mathbb{I}\left[t \in \{\psi_H(j)\}_{j=1}^{S}\right], \qquad t = 1, \ldots, T-1, \tag{26}$$

with $\bar{b}_1 = 1$. The segment index and per-timestep skill target are then

$$\kappa_t = \sum_{\tau=1}^{t} \bar{b}_\tau, \qquad \bar{z}_t^{\text{high}} = z_{\kappa_t}^{\text{high}}, \qquad t = 1, \ldots, T-1. \tag{27}$$

This construction ensures that all original timesteps within the same discovered segment share the same high-level skill target.

Low-level chunk pseudo-labels. For action-chunk policy learning, the low-level target at timestep t is

$$\mathbf{Z}_{t:t+K-1}^{\text{low}} = (z_t^{\text{low}}, \ldots, z_{t+K-1}^{\text{low}}). \tag{28}$$

When $t + K - 1 > T - 1$, we use the valid suffix and mask invalid positions in the executor loss. This avoids introducing artificial padding latents into the target sequence.

A.4 Skill-Discovery Loss Terms

The main paper summarizes the skill-discovery objective as

$$\mathcal{L}_{\text{skill}} = \mathcal{L}_{\text{next}} + \lambda_m \mathcal{L}_{\text{motion}} + \lambda_r \mathcal{L}_{\text{ratio}} + \lambda_c \mathcal{L}_{\text{cons}}.$$

Here we provide the concrete reduction used for each term.

Next-latent prediction. A prediction head estimates the next low-level latent action from the current low-level latent and its unfolded skill target:

$$\hat{z}_{t+1}^{\text{low}} = P_\omega(z_t^{\text{low}}, \bar{z}_t^{\text{high}}). \tag{29}$$

The next-latent loss is

$$\mathcal{L}_{\text{next}} = \frac{1}{T-2} \sum_{t=1}^{T-2} \left\| \hat{z}_{t+1}^{\text{low}} - z_{t+1}^{\text{low}} \right\|_1. \tag{30}$$

Motion preservation. To ensure that predicted low-level latents remain motion-consistent, we reuse the frozen flow decoder from the tokenizer:

$$\hat{\mathbf{\Phi}}_{t+1}^{\text{pred}} = D_{\text{flow}}(\hat{z}_{t+1}^{\text{low}}, o_{t+1}). \tag{31}$$

The motion loss is

$$\mathcal{L}_{\text{motion}} = \frac{1}{V(T-2)} \sum_{t=1}^{T-2} \sum_{v=1}^{V} \left\| \hat{\Phi}_{t+1}^{(v),\text{pred}} - \Phi_{t+1}^{(v)} \right\|_1. \tag{32}$$

Boundary-ratio regularization. The ratio loss prevents degenerate segmentation patterns, such as assigning every token as a boundary or assigning almost no boundaries:

$$\mathcal{L}_{\text{ratio}} = \sum_{s=0}^{H-1} \left(\frac{1}{L_s} \sum_{i=1}^{L_s} b_i^{(s)} - \rho_s \right)^2, \tag{33}$$

where ρ_s is the target boundary ratio at hierarchy stage s.

Within-segment consistency. For each discovered segment $\mathcal{I}_j^{(s)}$, the segment-level token $z_j^{(s+1)}$ should summarize the token features inside that segment. We use a projection R_s to match dimensions and define

$$\mathcal{L}_{\text{cons}} = \sum_{s=0}^{H-1} \frac{1}{L_s} \sum_{j=1}^{L_{s+1}} \sum_{i \in \mathcal{I}_j^{(s)}} \left\| R_s h_i^{(s)} - z_j^{(s+1)} \right\|_2^2. \tag{34}$$

This term encourages tokens inside the same discovered skill segment to share a coherent representation.

A.5 Qwen3-VL-Instruct Planner and Memory Interface

The main text writes the planner compactly as

$$\hat{z}_t^{\text{high}}, \hat{b}_t = \pi_\theta^{\text{plan}}(\tilde{x}_t).$$

In our implementation, this planner is instantiated with Qwen3-VL-Instruct. The planner is responsible for high-level skill reasoning and memory-conditioned boundary prediction; it is not the memory bank itself.

Planner input formatting. At timestep t, the planner receives four sources of information:

1. current multi-view RGB observation o_t;
2. task instruction ℓ;
3. proprioceptive summary p_t projected into planner token space;
4. retrieved memory context c_t^{mem} projected into planner token space.

We denote the resulting planner hidden state as

$$h_t^{\mathrm{plan}} = \mathrm{QwenPlan}_\theta(o_t, \ell, P_p(p_t), P_m(c_t^{\mathrm{mem}})), \tag{35}$$

where P_p and P_m are learned projection layers for proprioception and memory context.

Continuous skill and boundary heads. The output hidden state is mapped to a continuous skill latent and a boundary score by lightweight heads:

$$\hat{z}_t^{\mathrm{high}} = H_z(h_t^{\mathrm{plan}}), \qquad \hat{b}_t = \sigma(H_b(h_t^{\mathrm{plan}})). \tag{36}$$

The predicted skill latent is used by the executor to generate the low-level latent-action chunk. The boundary score is used both as a skill-transition prediction and as an input to the memory write gate.

Memory retrieval. The external memory bank is a set of continuous skill-level tokens:

$$\mathcal{M}_t = \{m_i\}_{i=1}^{N_t}. \tag{37}$$

Given the current state representation $x_t = E_\theta(o_t, p_t, \ell)$, the retrieved memory context is

$$c_t^{\mathrm{mem}} = \mathrm{Attn}(W_q x_t, W_k \mathcal{M}_t, W_v \mathcal{M}_t), \tag{38}$$

with $c_t^{\mathrm{mem}} = 0$ if $\mathcal{M}_t$ is empty. The read gate forms the memory-adapted state:

$$\alpha_t^{\mathrm{read}} = \sigma(G_{\mathrm{read}}(x_t, c_t^{\mathrm{mem}})), \qquad \tilde{x}_t = x_t + \alpha_t^{\mathrm{read}} W_m c_t^{\mathrm{mem}}. \tag{39}$$

Memory writing. After the planner predicts $\hat{z}_t^{\mathrm{high}}$ and $\hat{b}_t$, the executor produces

$$\hat{\mathbf{Z}}_{t:t+K-1}^{\mathrm{low}} = \pi_\theta^{\mathrm{exec}}(\tilde{x}_t, \hat{z}_t^{\mathrm{high}}).$$

The write gate computes

$$\alpha_t^{\mathrm{write}} = \sigma(G_{\mathrm{write}}(\tilde{x}_t, \hat{z}_t^{\mathrm{high}}, \hat{b}_t)). \tag{40}$$

The candidate memory token is

$$\gamma_t = \Gamma_\psi\left(\tilde{x}_t, \hat{z}_t^{\mathrm{high}}, \mathrm{Pool}(\hat{\mathbf{Z}}_{t:t+K-1}^{\mathrm{low}})\right). \tag{41}$$

The memory bank is updated by

$$\mathcal{M}_{t+1} = \begin{cases} U_\psi(\mathcal{M}_t, \gamma_t), & \alpha_t^{\mathrm{write}} > \eta, \\ \mathcal{M}_t, & \text{otherwise.} \end{cases} \tag{42}$$

The update operator U_ψ appends γ_t to the bank and compresses the bank if it exceeds a fixed budget $N_{\max}$.

A.6 Training Details for the Three Stages

Stage I. Stage I trains only the low-level tokenizer. The output of this stage is a frozen tokenizer and a set of offline low-level latent-action pseudo-labels. The downstream planner, executor, action decoder, and memory gates are not trained in this stage.

Stage II. Stage II learns high-level skill pseudo-labels and pretrains the planner and executor without external memory. We set $\mathcal{M}_t = \emptyset$, $c_t^{\text{mem}} = 0$, and $\tilde{x}_t = x_t$. The loss is

$$\mathcal{L}_{\text{latent}} = \lambda_h \mathcal{L}_{\text{plan}} + \lambda_l \mathcal{L}_{\text{exec}} + \lambda_b \mathcal{L}_{\text{bd}}. \tag{43}$$

The terms are computed as

$$\mathcal{L}_{\text{plan}} = \frac{1}{T-1} \sum_{t=1}^{T-1} \left\| \hat{z}_t^{\text{high}} - \bar{z}_t^{\text{high}} \right\|_2^2, \tag{44}$$

$$\mathcal{L}_{\text{exec}} = \frac{1}{K(T-K)} \sum_{t=1}^{T-K} \left\| \hat{\mathbf{Z}}_{t:t+K-1}^{\text{low}} - \mathbf{Z}_{t:t+K-1}^{\text{low}} \right\|_2^2, \tag{45}$$

and

$$\mathcal{L}_{\text{bd}} = \frac{1}{T-1} \sum_{t=1}^{T-1} \text{BCE}(\hat{b}_t, \bar{b}_t). \tag{46}$$

Stage III. Stage III activates the gated external memory module and fine-tunes the policy on robot demonstrations with action annotations. The full objective is

$$\mathcal{L}_{\text{ft}} = \mathcal{L}_{\text{act}} + \alpha_h \mathcal{L}_{\text{plan}} + \alpha_l \mathcal{L}_{\text{exec}} + \alpha_b \mathcal{L}_{\text{bd}} + \alpha_m \mathcal{L}_{\text{gate}}. \tag{47}$$

For deterministic action prediction, the action loss is

$$\mathcal{L}_{\text{act}} = \left\| \hat{\mathbf{a}}_{t:t+K-1} - \mathbf{a}_{t:t+K-1} \right\|_2^2. \tag{48}$$

For stochastic policies, it is replaced by the negative log-likelihood

$$\mathcal{L}_{\text{act}} = -\log \pi_\theta(\mathbf{a}_{t:t+K-1} \mid o_t, p_t, \ell, \mathcal{M}_t). \tag{49}$$

The gate loss is

$$\mathcal{L}_{\text{gate}} = \frac{1}{T-1} \sum_{t=1}^{T-1} \left[\text{BCE}(\alpha_t^{\text{write}}, \bar{b}_t) + \lambda_{\text{read}} \|\alpha_t^{\text{read}}\|_1 + \lambda_{\text{write}} \|\alpha_t^{\text{write}}\|_1 \right]. \tag{50}$$

The write gate is supervised by the discovered skill-boundary label $\bar{b}_t$, while the read gate is learned through the downstream action and latent prediction losses with sparsity regularization.

Teacher-forced memory warmup. To stabilize the beginning of Stage III, we optionally initialize memory updates using the discovered boundary label rather than the predicted write gate. During this warmup, memory is updated by

$$\mathcal{M}_{t+1}^{\text{teach}} = \begin{cases} U_\psi(\mathcal{M}_t^{\text{teach}}, \gamma_t^{\text{teach}}), & \bar{b}_t = 1, \\ \mathcal{M}_t^{\text{teach}}, & \text{otherwise}, \end{cases} \tag{51}$$

where

$$\gamma_t^{\text{teach}} = \Gamma_\psi \left(x_t, \bar{z}_t^{\text{high}}, \text{Pool}(\mathbf{Z}_{t:t+K-1}^{\text{low}}) \right). \tag{52}$$

After warmup, memory writing is controlled by α_t^{write} as in Eq. (42).

A.7 Causal Inference Procedure

At inference time, HiMem-WAM does not generate future videos and does not estimate optical flow. The policy executes the following causal procedure at each decision step:

1. Receive current RGB observations o_t, proprioception p_t, instruction ℓ, and memory bank $\mathcal{M}_t$.
2. Compute $x_t = E_\theta(o_t, p_t, \ell)$ and retrieve c_t^{mem} from $\mathcal{M}_t$.
3. Form $\tilde{x}_t = x_t + \alpha_t^{\text{read}} W_m c_t^{\text{mem}}$.
4. Use Qwen3-VL-Instruct planner to predict $\hat{z}_t^{\text{high}}$ and $\hat{b}_t$.

5. Use the executor to generate $\hat{\mathbf{Z}}^{\mathrm{low}}_{t:t+K-1}$.
6. Decode $\hat{\mathbf{a}}_{t:t+K-1} = D_{\mathrm{act}}(\hat{\mathbf{Z}}^{\mathrm{low}}_{t:t+K-1}, \tilde{x}_t)$.
7. If $\alpha_t^{\mathrm{write}} > \eta$, write γ_t into the memory bank.

This procedure uses only current observations and stored memory, preserving a causal test-time interface.

B Real-World Setting Details

B.1 Generalization Setting

We provide HiMem-WAM definition of the **GE** setting used in our real-world evaluation. Depending on the task, one or more of the following perturbations may be introduced:

- **Object position variation:** The initial positions of target objects and receptacles are changed within the workspace.
- **Unseen distractor objects:** Novel objects that are not present in the training demonstrations are placed near the target objects, resulting in a more cluttered scene.
- **Target layout and height variation:** The spatial layout of task-relevant objects is changed, and target objects or receptacles may be placed at different heights using support structures.
- **Instruction variation:** The original task instructions are replaced with semantically equivalent paraphrases to evaluate robustness to language variation.

B.2 Task Descriptions

We evaluate HiMem-WAM on 10 real-world manipulation tasks grouped into three difficulty categories: **Easy**, **Medium**, and **Hard**. Below, we describe the tasks used in our evaluation.

Easy tasks. Easy tasks are short-horizon single-arm manipulation tasks involving basic actions such as reaching, grasping, picking, and placing.

- **Stack bowls**: Pick up the yellow bowl and place it on top of the green bowl.
- **Hang cup**: Pick up the cup and hang it on the mug rack.
- **Put fruit into basket**: Pick up the fruit and place it into the basket.
- **Press button**: Press the target button until it is activated.

Medium tasks. Medium tasks require basic bimanual coordination, while the overall task horizon remains moderate and the manipulation procedure is still relatively structured.

- **Stack three bowls**: Provide a one-sentence definition of the task.
- **Fold towels**: Provide a one-sentence definition of the task.
- **Place plate**: Provide a one-sentence definition of the task.
- **Press two button**: Provide a one-sentence definition of the task.

Hard tasks. Hard tasks involve longer-horizon bimanual manipulation and more complex object interactions, and typically require more precise coordination across multiple steps.

- **Place two plate**: Provide a one-sentence definition of the task.
- **Tidy the table**: Provide a one-sentence definition of the task.

B.3 Full Real-World Results

Table 6: Full real-world results on 15 manipulation tasks using the Galaxea R1 Lite. Tasks are grouped into **Easy**, **Medium**, and **Hard**. We report **SR** (%) under the **ST** and **GE** settings. Bold denotes the better result between $\pi_{0.5}$ and HiMem-WAM under the same setting.

Task	π_0		**HiMem-WAM**	
	ST	**GE**	**ST**	**GE**
Easy Tasks				
Stack bowls	00.0	00.0	**00.0**	**00.0**
Open drawer	00.0	00.0	**00.0**	**00.0**
Hang up cup	00.0	00.0	**00.0**	**00.0**
Put fruit into basket	00.0	00.0	**00.0**	**00.0**
Press button	00.0	00.0	**00.0**	**00.0**
Easy Avg.	00.0	00.0	**00.0**	**00.0**
Medium Tasks				
Medium Task 1	00.0	00.0	**00.0**	**00.0**
Medium Task 2	00.0	00.0	**00.0**	**00.0**
Medium Task 3	00.0	00.0	**00.0**	**00.0**
Medium Task 4	00.0	00.0	**00.0**	**00.0**
Medium Task 5	00.0	00.0	**00.0**	**00.0**
Medium Avg.	00.0	00.0	**00.0**	**00.0**
Hard Tasks				
Hard Task 1	00.0	00.0	**00.0**	**00.0**
Hard Task 2	00.0	00.0	**00.0**	**00.0**
Hard Task 3	00.0	00.0	**00.0**	**00.0**
Hard Task 4	00.0	00.0	**00.0**	**00.0**
Hard Task 5	00.0	00.0	**00.0**	**00.0**
Hard Avg.	00.0	00.0	**00.0**	**00.0**
Overall Avg.	00.0	00.0	**00.0**	**00.0**

C Baselines

DP: Diffusion Policy is a diffusion-based visuomotor imitation learning method that represents robot actions as a conditional denoising process. It predicts action sequences conditioned on visual observations and executes them in a receding-horizon manner, enabling expressive multi-modal action generation for contact-rich manipulation.

ACT: Action Chunking Transformer is an imitation learning policy originally developed for fine-grained bimanual manipulation. Instead of predicting a single action at each step, ACT generates temporally coherent action chunks with a Transformer-based policy and applies temporal ensembling to reduce compounding errors and improve execution smoothness.

X-VLA: X-VLA is a soft-prompted flow-matching VLA framework designed for cross-embodiment robot learning. It introduces learnable prompt embeddings to encode robot- and dataset-specific variations, allowing a shared Transformer policy to adapt across heterogeneous sensors, action spaces, and robotic platforms with limited additional parameters.

MEM-0: MEM-0 is a modular memory-aware manipulation policy introduced for memory-dependent robotic tasks. It explicitly incorporates memory components to retain task-relevant historical observations, making it suitable for evaluating manipulation scenarios where the correct action depends on past states rather than only the current visual input.

Motus: Motus is a unified latent action world model that combines visual prediction, action modeling, and robot control within a shared latent formulation. By leveraging motion-centric representations and pretrained generative priors, it aims to provide a unified modeling interface for world modeling, inverse dynamics, and VLA-style action prediction.

OpenVLA: OpenVLA is a 7B-parameter open-source vision-language-action model trained on 970K real-world robot demonstrations from the Open X-Embodiment dataset. It builds on a pretrained vision-language backbone with DINOv2 and SigLIP visual encoders and a Llama-2 language model, and predicts discretized robot actions from image observations and language instructions.

OpenVLA-OFT: OpenVLA-OFT is an optimized fine-tuning variant of OpenVLA. It replaces the original token-by-token action generation with a faster and more control-oriented recipe that combines parallel decoding, action chunking, continuous action regression, and an L1 objective, substantially improving inference speed and downstream manipulation success.

SpatialVLA: SpatialVLA is a spatially enhanced VLA model trained on large-scale real-robot episodes. It augments visual-language-action modeling with Ego3D Position Encoding for explicit 3D spatial grounding and Adaptive Action Grids for discretizing spatial robot motions, improving cross-robot transfer and manipulation in geometry-sensitive tasks.

π_0: π_0 is a generalist vision-language-action flow model built on top of a pretrained VLM. It uses flow matching to generate continuous action trajectories and is trained on diverse data from multiple robotic platforms, including single-arm, dual-arm, and mobile manipulation settings, enabling language-conditioned dexterous control and fine-tuning to new skills.

$\pi_{0.5}$: $\pi_{0.5}$ extends π_0 toward open-world generalization by co-training on heterogeneous robot and non-robot data. In addition to low-level action prediction, it incorporates multimodal supervision such as semantic predictions, object information, and high-level task cues, enabling long-horizon manipulation in unfamiliar real-world environments.

NORA-1.5: NORA-1.5 builds upon the NORA VLA backbone and augments it with a flow-matching action expert. It further applies reward-guided post-training using world-model-based and action-based preference rewards, improving action reliability and task success across simulation and real-robot settings.

NORA-LONG: NORA-LONG is a long-horizon variant of NORA, an efficient 3B-parameter VLA model based on Qwen2.5-VL-3B and trained on Open X-Embodiment demonstrations. Compared

with the standard NORA configuration, NORA-LONG is pretrained with a longer action horizon, making it more suitable for evaluating temporally extended manipulation policies.

AtomVLA: AtomVLA is a subtask-aware VLA post-training framework for long-horizon robotic manipulation. It decomposes high-level instructions into atomic subtasks and uses a predictive latent world model to evaluate candidate action chunks, enabling scalable offline policy optimization and reducing compounding errors during multi-step execution.

UniVLA: UniVLA is a unified vision-language-action model that represents visual observations, language instructions, and robot actions as token sequences within a single autoregressive framework. By jointly modeling perception, world dynamics, and action generation, it aims to improve long-horizon policy learning and multimodal grounding for robotic manipulation.

WorldVLA: WorldVLA is an autoregressive action world model that integrates VLA policy learning and world modeling into one framework. It jointly reasons about robot actions and future visual states, allowing the model to use predicted environmental dynamics as an intermediate structure for more temporally consistent action generation.

LingBot-VA: LingBot-VA is a causal video-action world model for robot control. It uses an autoregressive diffusion framework with a shared latent space for visual and action tokens, closed-loop rollout with real observations, and asynchronous inference, enabling future-aware policy execution and stronger long-horizon temporal memory.

FAST-WAM: FAST-WAM is a fast world-action model that studies whether explicit future imagination is necessary at test time. It retains video-modeling objectives during training to learn stronger world representations, but skips future-frame generation during inference, achieving competitive manipulation performance with substantially lower control latency.

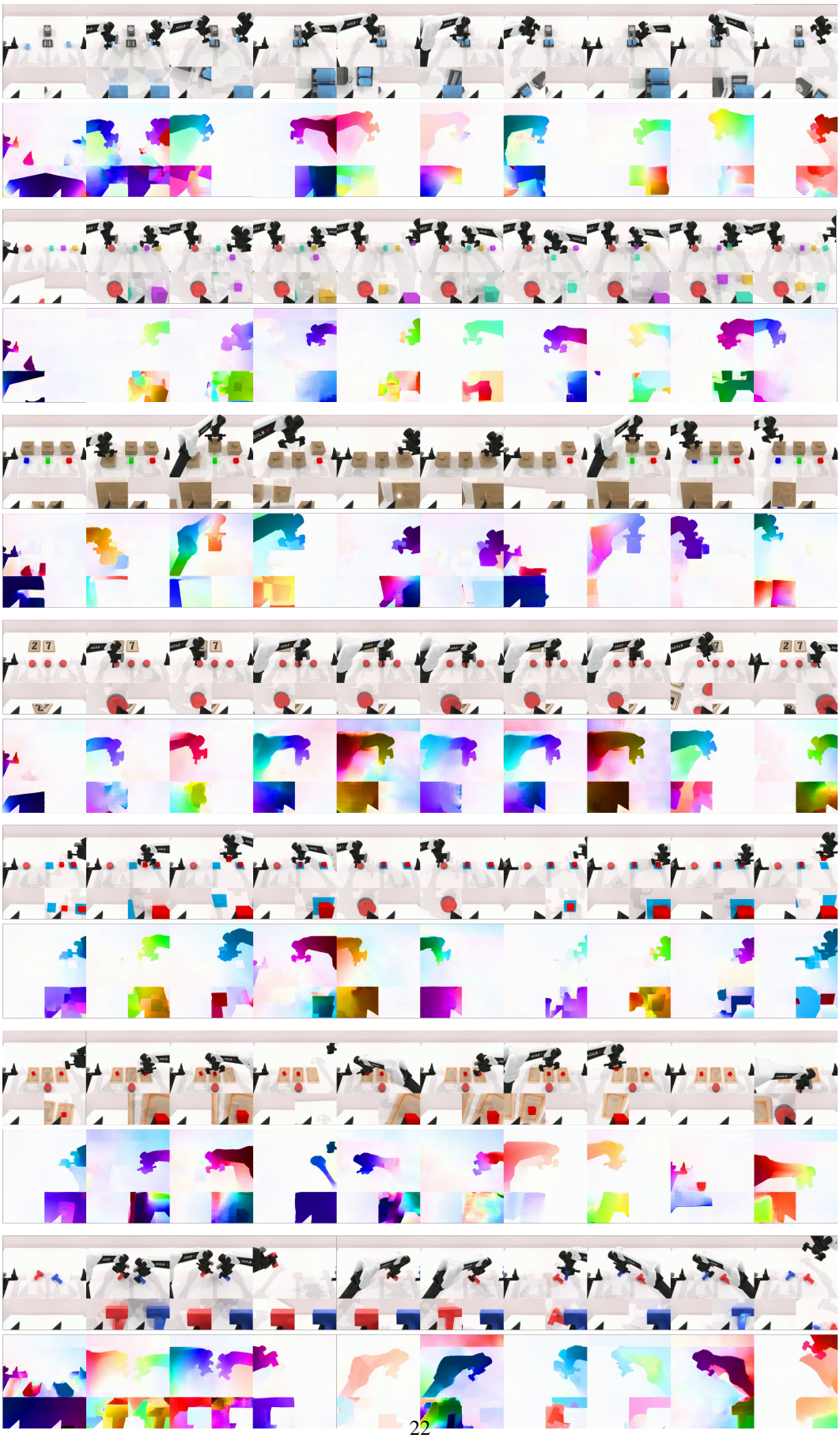

Figure 3: Appendix RMBench tasks

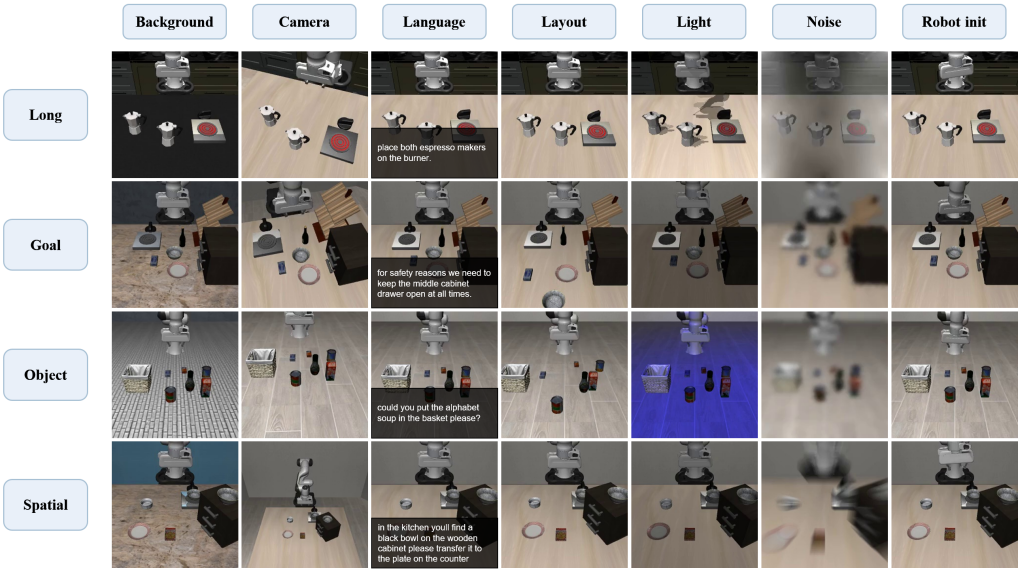

Figure 4: LIBERO-Plus perturbation gallery