CONTROLLED PREDICTIVE MEMORY FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models need compact memories to distinguish execution histories that require different decisions despite similar current observations. Predicting future observations can guide this compression, but their dependence on subsequent behavior complicates comparisons between histories. We introduce Controlled Predictive Memory (CPM), which learns memory by predicting expected joint outcome features under a fixed set of short closed-loop interaction programs. The fixed programs provide a common behavioral reference for comparing histories, while joint features expose selected cross-step dependencies that separate endpoint means can miss. Response prediction and action imitation jointly train a fixed-size recurrent memory from additional program interactions and expert demonstrations; the programs and response decoder are used only during training. A prediction-risk decomposition characterizes the response information lost through history compression. CPM improves average success on each of five simulation benchmarks over two history-based baselines with the same VLA backbone and matched action-update budgets. On five single-arm and two bimanual real-world tasks, CPM achieves average success rates of 73.3% and 63.3%, respectively, exceeding the stronger history-based baseline on each platform by 6.7 and 8.3 percentage points.

1 INTRODUCTION

Vision-language-action (VLA) models bring pretrained visual and linguistic knowledge to robot control, enabling a single policy to perform diverse language-conditioned manipulation tasks (Black et al., 2024; Intelligence et al., 2025; Kim et al., 2025; Brohan et al., 2023; Kim et al., 2024; Jia et al., 2026). Successful manipulation, however, often depends on information that is not directly available in the current observation. Objects may become occluded after an interaction, and a scene may remain nearly unchanged after an operation is completed. In these cases, past observations and executed actions provide the evidence needed to interpret the current scene and select the next action (Shi et al., 2026; Dai et al., 2026). This poses a representation problem: a compact memory should retain the historical evidence needed for subsequent decisions.

Recent memory-augmented policies address this problem by retrieving, integrating, or recurrently encoding interaction histories (Shi et al., 2026; Sridhar et al., 2026; Li et al., 2026c). Reconstructing observation-action histories, predicting future visual features and actions, and answering task-oriented questions provide additional supervision for these representations (Zhou et al., 2026; Hu et al., 2026a; Shah et al., 2026). These objectives, however, do not directly test whether memory still distinguishes histories with different joint response statistics under the same subsequent control rule. Without specifying subsequent behavior, differences in future observations may reflect both historical information and differences in how subsequent actions are selected.

Building on predictive representations for control (Venkatraman et al., 2017; Hefny et al., 2018; Zheng et al., 2025; Lawson et al., 2026; Lee et al., 2023), we introduce Controlled Predictive Memory (CPM), which turns this criterion for distinguishing histories into an explicit training signal. CPM uses a fixed library of short closed-loop interaction programs: each program comprises two decisions, with the second adapting to the observed feedback from the first execution. Histories can thus be compared through their responses under the same rule, providing a common behavioral reference for deciding which distinctions to retain. Two histories may have identical mean features

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.

at each subsequent observation while differing in the relationship between the two observations. To include such selected cross-step distinctions in memory supervision, CPM jointly encodes the two observations with fixed nonlinear features and predicts their expectation.

We collect predictive supervision through additional interactions, randomly assigning a program before execution, and use expert demonstrations for action supervision. CPM recurrently encodes the commands actually issued to the robot and the observations before and after their execution. A program-conditioned decoder predicts the expected joint features from the memory state, while the action policy reads the same state to select task actions. The two objectives jointly train memory: response prediction encourages it to retain predictive historical evidence, and action imitation connects the representation to task decisions. A squared-error risk decomposition further explains how this supervision constrains compression: mapping histories with different program response statistics to the same memory introduces a compression error, distinct from outcome variability and decoding error. Memory capacity does not grow with the execution history, and the programs and response decoder are used only during training.

We evaluate CPM in simulation on LIBERO, SimplerEnv-Bridge, RoboMME, RoboTwin 2.0, and LIBERO-Plus (Liu et al., 2023; Li et al., 2024b; Dai et al., 2026; Chen et al., 2025; Fei et al., 2026). With the same VLA backbone and matched action-update budgets, CPM achieves higher mean success than both history-based baselines on all four manipulation benchmarks. Compared with the stronger history-based baseline on each benchmark, CPM improves success by 6.6 and 3.4 percentage points on Bridge and RoboMME, respectively. On LIBERO-Plus, it achieves an overall success rate of 88.4%. Ablations further support the contributions of program conditioning, joint outcome targets, and joint memory training to these gains. Real-world evaluations on five single-arm and two bimanual tasks further show gains of 6.7 and 8.3 percentage points, respectively, over the stronger history-based baseline on each platform.

Our main contributions are as follows:

- We formulate a history compression objective using joint responses under fixed closed-loop programs and analyze the distinctions between histories that this objective requires memory to retain.
- We introduce CPM, which jointly learns recurrent VLA memory through response prediction and action imitation, using a response decoder only during training and a compact state at deployment.
- We achieve higher mean success than two history-based baselines on five simulation benchmarks and on both single-arm and bimanual real-world platforms, and use component ablations to examine the roles of program conditioning, joint targets, and shared memory training.

Related work is discussed in Appendix A.

2 Controlled Predictive Memory

In tasks that require adapting behavior to execution feedback, identical current observations under the same instruction may correspond to different execution histories and require different subsequent actions. A policy conditioned only on the current input assigns the same action distribution to these situations, whereas issued commands and the observations before and after execution provide evidence that distinguishes the histories (Lian et al., 2026). We introduce Controlled Predictive Memory (CPM), which jointly learns a recurrent latent state through outcome prediction and action imitation, allowing action generation to use both the current scene and observed feedback from previous operations. As shown in Figure 1, fixed closed-loop interaction programs specify a common rule for subsequent behavior across histories, and the paired observations after the two action segments jointly supervise program-response prediction from memory. The action policy reads the same memory and learns from demonstrations how to use the distinctions between histories for action selection. At deployment, the memory is updated recurrently from actually issued commands and new observations to condition the next decision (Figure 2).

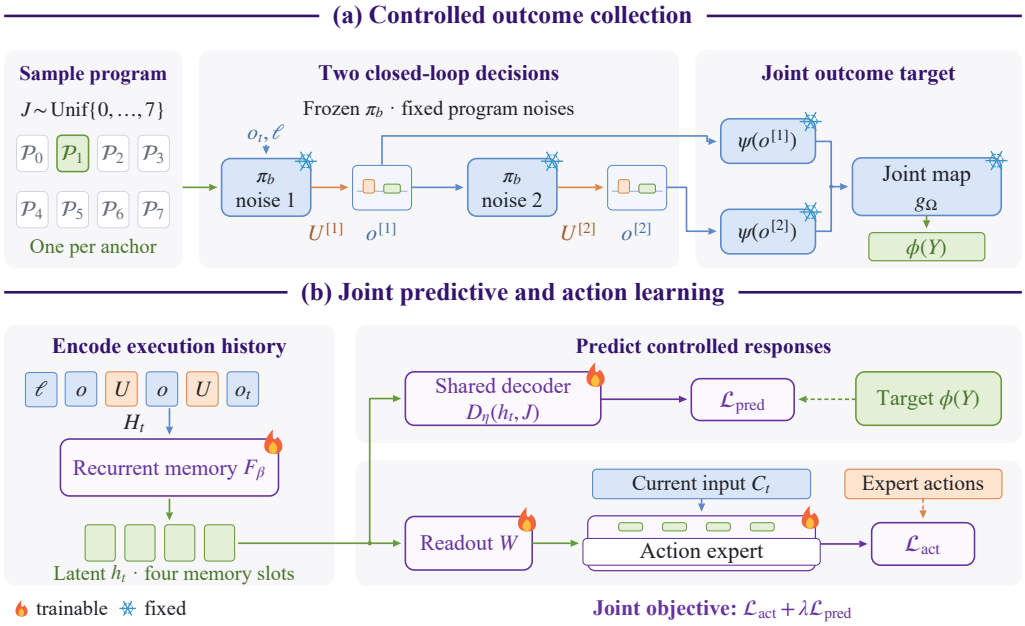


Figure 1: **Controlled outcome supervision and joint learning in CPM. (a)** A fixed closed-loop interaction program is randomly assigned at each collection anchor. The first action segment is executed to obtain $o^{[1]}$, which conditions the generation and execution of the second segment to obtain $o^{[2]}$. Features of the two observations are transformed by a fixed joint mapping to form the prediction target $\phi(Y)$. **(b)** The recurrent module F_β encodes execution history into a latent memory: the shared decoder D_η predicts the corresponding outcome conditioned on the program index, while the action expert reads the memory through the projection W to generate actions. Program records and expert demonstrations provide prediction and action supervision, respectively, and both losses train F_β.

2.1 Controlled Outcome Supervision

Task and execution history. We consider language-conditioned robotic manipulation, where ℓ denotes the instruction and the observation $o_t = (I_t, s_t)$ consists of RGB images and proprioceptive state. At decision step t, the policy predicts an action chunk A_t, the controller issues its prefix U_t, and the next observation o_{t+1} is then obtained. The current input is $C_t = (o_t, \ell)$, and the history $H_t = (\ell, o_0, U_0, \ldots, U_{t-1}, o_t)$ pairs issued commands with subsequent observations, recording the interaction evidence available before program execution.

Fixed control programs. Subsequent observations reflect the environment's response to an operation and also depend on the actions selected thereafter. Directly predicting future observations from demonstrations therefore incorporates how the demonstrator continues to act into the prediction target. To specify the behavioral conditions under which outcomes are obtained, we fix a set of short closed-loop interaction programs in advance and always predict outcomes obtained by following these rules.

The library $\{\mathcal{P}_j\}_{j=0}^{M-1}$ contains $M = 8$ programs constructed from a memoryless VLA policy π_b that is fine-tuned on demonstrations and then frozen. Each program fixes the flow sampling noise for each of two decisions in advance: it generates and issues the first action segment from the initial observation to obtain $o^{[1]}$, then generates and issues the second segment conditioned on $o^{[1]}$ to obtain $o^{[2]}$. Thus, each program specifies a fixed feedback rule, while its second action segment still varies with the observation after the first execution. We denote the two observations by $Y = (o^{[1]}, o^{[2]})$.

To make program assignment independent of anticipated outcomes, after each selected history H, we sample an index J uniformly at random before execution, execute only the assigned program, and record (H, J, Y). These additional interactions form the prediction dataset $\mathcal{D}_p$, while expert

demonstrations $\mathcal{D}_a$ separately provide action supervision. Failed and aborted executions are also retained so that prediction targets include the range of outcomes that a program may produce, rather than only successfully completed executions; endpoints for aborted executions are defined in Appendix B.

Joint outcome targets. Changes after the first execution, together with their persistence or alteration after the second, describe the program's short-term response. To include the relationship between the two endpoints in supervision, we jointly encode the observations before predicting the conditional mean of the resulting features. Specifically, a fixed endpoint mapping ψ extracts pretrained visual features and proprioceptive state, and a fixed nonlinear mapping g_Ω encodes the concatenated endpoint representations using random Fourier features (Rahimi & Recht, 2007), yielding $\phi(Y) = g_\Omega([\psi(o^{[1]}); \psi(o^{[2]})]/\sqrt{2})$. The prediction target for program j is defined as

$$r_j(H) = \mathbb{E}\big[\phi(Y^j) \mid H, \mathrm{do}(\mathcal{P}_j)\big], \tag{1}$$

where Y^j denotes the two observations obtained by continuing from history H with program $\mathcal{P}_j$, and $\mathrm{do}(\mathcal{P}_j)$ specifies the subsequent control rule. The feature $\phi(Y)$ obtained under randomized assignment provides sample-based supervision for the corresponding program response. Encoding the endpoints jointly before taking the expectation allows the target to capture cross-endpoint statistics, unlike separately predicting the feature mean at each endpoint; the full mapping is given in Appendix B.

2.2 Learning Predictive Latent Memory

To use program-response supervision for action selection, we encode execution history into a latent memory with a recurrent module and read out program responses with a separate decoder. The prediction objective thus constrains the execution information retained in memory, while the action objective directly shapes the same representation to support action selection in demonstrations (Venkatraman et al., 2017; Shah et al., 2026).

Recurrent encoding of execution feedback. The memory h_t consists of four latent vectors. Whenever execution feedback is received, the recurrent module F_β updates the memory using its previous state, the issued commands, and the observations before and after execution:

$$\begin{aligned} h_0 &= F_\beta^{\mathrm{init}}(e(o_0), \ell), \\ h_{t+1} &= F_\beta(h_t, e(o_t), U_t, e(o_{t+1}), \ell). \end{aligned} \tag{2}$$

Here, $e(o)$ combines frozen visual features with normalized proprioceptive state. Through self-attention among memory slots and cross-attention (Vaswani et al., 2017), F_β incorporates the newly observed transition into the existing memory; initialization uses only the initial observation and instruction.

Predicting program responses from memory. We train a response decoder D_η from scratch, with parameters shared across all programs. A learnable query associated with program index j reads the four memory slots to produce the response prediction $D_\eta(h_t, j)$. The decoder supplies the program conditioning, allowing the memory slots to jointly organize historical information without a one-to-one correspondence with programs. The recurrent module and decoder jointly minimize

$$\mathcal{L}_{\mathrm{pred}}(\beta, \eta) = \mathbb{E}_{(H,J,Y)\sim\mathcal{D}_p}\left[\|D_\eta(h_\beta(H), J) - \phi(Y)\|_2^2\right], \tag{3}$$

where $h_\beta(H)$ is obtained by replaying the history up to program assignment using Equation (2); observations after program execution are used only to construct supervision targets.

How predictive supervision constrains memory. When two histories yield different responses to the same program, compressing them into the same memory prevents the decoder from distinguishing those responses. For fixed programs, outcome mapping, and history distribution, the population risk can be written as

$$\mathcal{R}_{\mathrm{pred}} = V_H + C_{\mathrm{pred}}(\beta) + E_{\mathrm{dec}}(\beta, \eta). \tag{4}$$

V_H is the outcome variability that remains given the full history. C_{pred} measures the distinctions between responses lost through history compression, and E_{dec} measures the prediction error incurred

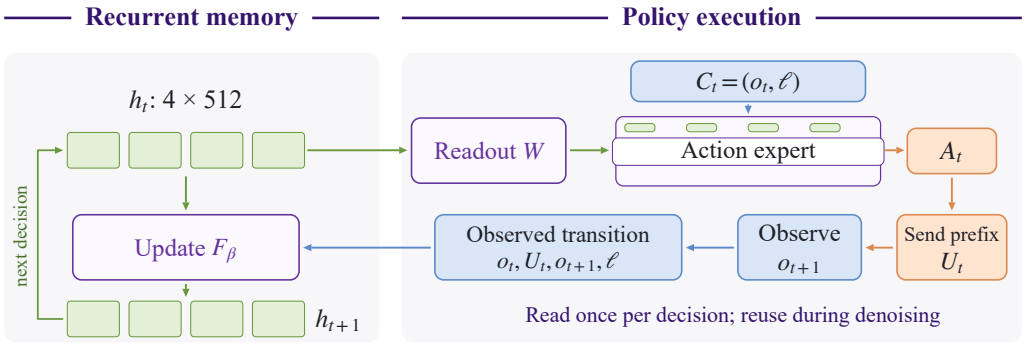


Figure 2: **Memory-conditioned closed-loop deployment.** The action expert uses the current observation and instruction C_t together with the memory readout $W(h_t)$ to generate an action chunk A_t. The controller issues the prefix U_t and then receives a new observation o_{t+1}. The recurrent module F_β combines the previous memory, the issued commands, and the observations before and after execution to obtain h_{t+1} for the next decision. The memory readout remains fixed within a decision, and neither the prediction decoder nor the control programs are used at deployment.

when the decoder reads out this information. Predictive learning therefore optimizes the sum of the latter two terms, with V_H fixed, constraining memory to retain program-response information. The conditions, definitions, and proof of this decomposition are provided in Appendix D.

2.3 Joint Policy Learning and Deployment

Memory-conditioned action generation. We build the action policy on $\pi_{0.5}$ (Intelligence et al., 2025) and use a shared affine projection W to map the four memory slots to input tokens for the action expert. The action expert generates an action chunk conditioned on the current input C_t and the memory readout $W(h_t)$. $W(h_t)$ is the policy's only historical input, and the attention mask prevents noisy actions from influencing the conditioning representations; architectural details are given in Appendix C.1.

Joint training. Let θ denote the action expert parameters and $\mathcal{L}_{\text{act}}(\theta, W, \beta)$ the memory-conditioned flow-matching loss on demonstrations (Lipman et al., 2022). We retain the base policy's action objective and jointly optimize

$$\mathcal{L}(\theta, W, \beta, \eta) = \mathcal{L}_{\text{act}}(\theta, W, \beta) + \lambda \mathcal{L}_{\text{pred}}(\beta, \eta), \tag{5}$$

where λ controls the weight of predictive supervision. The two losses use independent batches of expert demonstrations and program records, respectively, and both update the recurrent module F_β. The prediction loss also trains D_η, while the action loss also trains W and the full action expert θ; the vision-language backbone remains frozen. During training, memory is recomputed from the start of each episode using the current parameters, and gradients are backpropagated through the recorded prefix; loss reductions are specified in Appendix C.2.

We perform two rounds of collection and training so that outcome supervision covers histories visited by the memory-conditioned policy itself. In the first stage, we collect program records from histories generated by π_b and train CPM jointly on these records and expert demonstrations. We then freeze the first-stage policy as the history collection policy π_{col} and execute the original program library on histories it visits to collect a second round of records. The second stage continues training from the first-stage parameters using the same demonstrations and both rounds of program data, with prediction sampling balanced across tasks and collection rounds. The program library and outcome mapping remain unchanged between rounds, so the additional histories correspond to the same prediction tasks. Algorithm 1 in Appendix K summarizes the training procedure, where $\mathcal{D}_p^{(r)}$ denotes the program records collected in round $r+1$, $\Theta = (\beta, \eta, W, \theta)$ contains the trainable parameters, and T_r is the number of updates in the corresponding stage.

Closed-loop deployment. As shown in Figure 2, a deterministic memory readout $W(h_t)$ is computed once per decision and reused throughout action denoising for that decision. The controller issues the prefix U_t of the action chunk; after o_{t+1} is received, the memory is updated according to Equation (2). Memory is initialized at the start of an episode and reset when the episode ends or the instruction changes. The prediction decoder and control programs are used only during training.

3 Experiments

3.1 Experimental Setup

Benchmarks. We evaluate CPM on the Spatial, Object, Goal, and Long suites of LIBERO, which assess spatial relations, object recognition, goal discrimination, and long-horizon manipulation, respectively (Liu et al., 2023). Each suite contains ten tasks, with 50 evaluation episodes per task; we report per-suite success rates and the equally weighted average across the four suites. We also use the four SimplerEnv-Bridge tasks to evaluate manipulation on a different robot and in a different visual domain (Li et al., 2024b), and seven RoboMME tasks without preceding demonstration videos to evaluate counting, object permanence, and reference memory (Dai et al., 2026). For bimanual evaluation, we use the same sixteen RoboTwin 2.0 tasks as OptimusVLA, with 100 episodes per task under the Hard setting (Chen et al., 2025; Li et al., 2026c). For robustness evaluation, we directly apply the LIBERO policies to all seven perturbation categories in LIBERO-Plus without further adaptation (Fei et al., 2026). For methods evaluated by us in simulation, we use the same evaluation initializations and report mean success rates over three training seeds; we additionally report the standard deviation across seeds for overall robustness results.

Baselines. We implement two history-based baselines using the same demonstration-adapted $\pi_{0.5}$ backbone as CPM (Intelligence et al., 2025): **Triplet History**, inspired by Reflective VLA, conditions on previously executed observation–command–outcome transitions (Lian et al., 2026); **FrameSamp+Modul** adapts the visual history sampling and memory modulation mechanisms of MME-VLA to the shared backbone (Dai et al., 2026). External methods include OpenVLA-OFT, OptimusVLA, MemoryVLA, the continuous-action version of RynnVLA-002, CogACT, and RDT (Kim et al., 2025; Li et al., 2026c; Shi et al., 2026; Cen et al., 2025; Li et al., 2024a; Liu et al., 2025). We also include publicly reported results for $\pi_{0.5}$ fine-tuned on demonstrations; sources and protocol differences for external results are provided in Appendix E.2.

Implementation. We implement CPM in openpi, freezing the vision–language backbone and jointly training the full action expert and the added modules. Expert demonstrations provide action supervision, while separately collected control-program execution records provide prediction supervision. We train one policy per LIBERO suite, multitask policies for Bridge and RoboMME, and single-task policies for RoboTwin. Post-training consists of two stages, each with 30,000 updates. The same-backbone baselines share the initial weights, demonstration splits, current inputs, and action interface, with matched action-update budgets. We evaluate the final checkpoint of every retrained policy; data collection and optimization settings are provided in Appendix E.3.

3.2 Evaluation on Simulation Benchmarks

Table 1 summarizes results on the four simulation benchmarks; per-task results for Bridge, RoboMME, and RoboTwin are provided in Appendix F.

LIBERO. CPM achieves a mean success rate of 98.6% across the four suites, exceeding both same-backbone history-based baselines and matching the publicly reported average for OptimusVLA. On Long, CPM achieves 96.1%, improving over the two history-based baselines by 4.8 and 3.2 percentage points, respectively. The recurrent state carries prior execution feedback into subsequent decisions, providing historical context for action selection across stages; the gains on Long are consistent with this design.

SimplerEnv-Bridge. CPM achieves a mean success rate of 74.3% across the four tasks, exceeding Triplet History and FrameSamp+Modul by 7.3 and 6.6 percentage points, respectively, with higher success rates on all four tasks.

Table 1: Success rates (%) on simulation benchmarks. LIBERO Avg. is the equally weighted average across four suites; Bridge and RoboMME average over four and seven tasks, respectively, and RoboTwin over sixteen tasks under the Hard setting. † denotes results from public sources, and – denotes results not included in this table; all other results are means over three training seeds. RynnVLA-002-C is the continuous-action version.

	LIBERO					Bridge	RoboMME	RoboTwin
Method	Spatial	Object	Goal	Long	Avg.	Avg.	Avg.	Avg.
OpenVLA-OFT†	97.6	98.4	97.9	94.5	97.1	–	4.3	–
OptimusVLA†	99.6	99.8	98.4	96.4	98.6	–	–	30.1
MemoryVLA†	98.4	98.4	96.4	93.4	96.7	71.9	7.0	–
RynnVLA-002-C†	99.0	99.8	96.4	94.4	97.4	–	–	–
CogACT-Large†	–	–	–	–	–	57.3	–	–
RDT†	–	–	–	–	–	–	–	16.0
$\pi^{\dagger}_{0.5}$	98.8	98.2	98.0	92.4	96.9	57.2	22.2	23.5
Triplet History	99.6	98.9	97.7	91.3	96.9	67.0	45.8	26.3
FrameSamp+Modul	99.5	98.2	98.1	92.9	97.2	67.7	50.2	26.5
CPM	99.9	99.3	98.9	96.1	98.6	74.3	53.6	31.4

Table 2: Success rates (%) on LIBERO-Plus. Overall success is weighted by the number of test instances. † denotes results from public sources; all other results are means over three training seeds, with the standard deviation across seeds also reported for overall success.

Method	Camera	Initial state	Language	Lighting	Background	Noise	Layout	Overall
OpenVLA-OFT†	56.4	31.9	79.5	88.7	93.3	75.8	74.2	69.6
$\pi^{\dagger}_{0.5}$	75.8	79.4	83.3	95.5	95.0	89.6	87.0	85.7
Triplet History	74.9	79.9	82.4	95.2	95.2	90.4	88.2	85.8 ± 2.2
FrameSamp+Modul	74.4	80.1	83.6	96.1	94.5	90.5	87.2	85.8 ± 2.2
CPM	76.4	85.4	85.3	95.6	95.1	93.2	92.0	88.4 ± 2.3

RoboMME. CPM achieves a mean success rate of 53.6% across the seven tasks, exceeding the stronger history-based baseline, FrameSamp+Modul, by 3.4 percentage points. CPM outperforms both history-based baselines on five of the seven tasks.

RoboTwin 2.0. CPM achieves 31.4% across the sixteen tasks under the Hard setting, exceeding the two history-based baselines at 26.3% and 26.5%; on Click Bell, it improves over the stronger history-based baseline by 14.7 percentage points.

Representative rollouts are shown in Appendix F.1.

3.3 Robustness Evaluation

Table 2 reports results for direct transfer of the LIBERO policies to LIBERO-Plus (Fei et al., 2026).

CPM achieves an overall success rate of 88.4%, improving over the two history-based baselines by 2.6 and 2.5 percentage points, respectively. The gains are most pronounced under changes to the robot's initial state and the object layout: CPM improves over the stronger history-based baseline in each category by 5.3 and 3.8 percentage points, respectively. Initial-state and layout changes alter the starting conditions of manipulation; the gains under these two perturbations are consistent with CPM's continual memory updates using actually issued commands and subsequent observations. Evaluation and statistical details are provided in Appendix G.

Examples under different background textures are shown in Figure 5.

3.4 Ablation Studies

We examine the roles of execution history, prediction supervision, and joint optimization on SimplerEnv-Bridge and RoboMME. Each variant in Table 3 changes one factor relative to full CPM while keeping the action-training budget fixed; variants with prediction supervision share the same program records. Variant definitions and per-task results are provided in Appendix H.

Table 3: Component ablations of CPM. Success rates (%) are means over three training seeds. Bridge and RoboMME contain four and seven tasks, respectively.

ID	Variant	Bridge	RoboMME
A0	Full CPM	74.3	53.6
A1	Current input only	63.5	27.0
A2	Without command values	71.9	51.4
A3	Action supervision only	67.0	47.5
A4	Prediction without program conditioning	69.1	50.1
A5	Additive endpoint target	72.9	51.7
A6	Memory updated by prediction only	66.7	44.6

Execution history. Compared with the current-input-only variant, CPM improves mean success rates by 10.8 and 26.7 percentage points on Bridge and RoboMME, respectively. This variant retains prediction training; the gains from recurrent history support using prior execution feedback to distinguish situations with similar current inputs but different execution histories. Retaining command values yields further gains of 2.4 and 2.2 percentage points, respectively: commands complement observation transitions with control inputs, allowing memory to jointly use which commands were issued and what changes were subsequently observed.

Prediction supervision and targets. Compared with an architecturally identical recurrent memory trained only with action supervision, full CPM improves mean success rates by 7.3 and 6.1 percentage points on Bridge and RoboMME, respectively. The prediction loss constrains memory to distinguish histories with different expected responses under the same program, preserving information that affects subsequent execution outcomes. Retaining program conditioning yields gains of 5.2 and 3.5 percentage points, respectively; the program index specifies the subsequent closed-loop rule, allowing the decoder to distinguish responses under different rules. The joint endpoint target improves over the additive target by 1.4 and 1.9 percentage points, respectively, with its nonlinear joint mapping incorporating relations between the two execution outcomes into supervision.

Joint optimization. Compared with the variant in which memory is updated only by prediction, full CPM improves mean success rates by 7.6 and 9.0 percentage points on Bridge and RoboMME, respectively. This variant still trains the projection and action expert; the difference is whether action errors can propagate through the memory readout to the recurrent module. Joint training allows action and prediction supervision to shape the history representation together, connecting memory learning to downstream action selection. Comparisons of subsequent-command-conditioned prediction and prediction loss weights are provided in Appendix H.

3.5 Computational Efficiency

Table 14 in Appendix I compares inference costs with the same $\pi_{0.5}$ backbone, where $\pi_{0.5}$-SFT is the memory-free baseline. All methods use the same input and action-sampling settings on a single H100 80GB GPU, with mean latency measured over 300 calls after warm-up. CPM has a mean latency of 0.082 seconds, close to 0.080 seconds for Triplet History and below 0.101 seconds for FrameSamp+Modul; its peak allocated GPU memory of 7.96 GB is also lower than that of both history-based baselines. At deployment, history is maintained recurrently in four memory slots, and the readout is reused throughout action denoising within each decision; outcome prediction is confined to training, and neither the prediction decoder nor the control programs are used at inference. The measurement protocol is provided in Appendix I.

Table 4: Real-world full-task success rates (%). S1–S5 are single-arm tasks and B1–B2 are bimanual tasks, defined in Table 15. Task columns report means across three seeds; Avg. reports the task-averaged mean and sample standard deviation across seeds. Bold denotes the highest mean in each column; a dash denotes an unevaluated setting.

	Single-arm tasks						Bimanual tasks		
Method	S1	S2	S3	S4	S5	Avg.	B1	B2	Avg.
$\pi_{0.5}$-SFT	50.0	46.7	**85.0**	70.0	66.7	63.7 ± 2.9	43.3	56.7	50.0 ± 7.5
Triplet History	56.7	50.0	80.0	**76.7**	**70.0**	66.7 ± 2.5	48.3	56.7	52.5 ± 5.0
FrameSamp+Modul	58.3	48.3	83.3	73.3	66.7	66.0 ± 1.0	50.0	60.0	55.0 ± 5.0
MemoryVLA	58.3	56.7	83.3	75.0	63.3	67.3 ± 1.2	–	–	–
OpenVLA-OFT	50.0	41.7	78.3	71.7	65.0	61.3 ± 1.2	–	–	–
OpenVLA-OFT+	–	–	–	–	–	–	48.3	53.3	50.8 ± 11.3
CPM	**70.0**	**68.3**	81.7	**76.7**	**70.0**	$\mathbf{73.3 \pm 2.1}$	**55.0**	**71.7**	$\mathbf{63.3 \pm 5.8}$

3.6 Real-World Experiments

Setup. We evaluate CPM on five single-arm and two bimanual tasks against history baselines and representative VLA models (Table 4). All methods use the same demonstrations, and methods with the same backbone have matched action-update budgets. We run 20 online episodes per task for each of three training seeds. Success requires autonomous completion of all steps within the time limit. Appendix J provides platform and training details.

Single-arm results. CPM achieves 73.3% average success, exceeding the stronger history baseline, Triplet History, by 6.7 percentage points and MemoryVLA by 6.0 points (Table 4). Gains over the history baselines are concentrated in table clearing with confirmation and drawer cleaning and restoration. The former alternates object placement with button confirmation, while the latter requires returning the object and closing the drawer after cleaning. These transitions depend on completed actions and their outcomes, consistent with CPM's use of controlled response prediction to preserve history relevant to subsequent decisions.

Bimanual results. CPM achieves 63.3% average success, exceeding the stronger history baseline, FrameSamp+Modul, by 8.3 percentage points. On conveyor-based transfer of four parts, it reaches 71.7%, an 11.7-point gain over this baseline. The gain is consistent with the benefit of retaining execution context while tracking progress across four consecutive transfers. Figure 6 shows representative executions.

4 Conclusion

We introduced Controlled Predictive Memory (CPM) for vision-language-action control that depends on historical information. Using fixed closed-loop programs as a common behavioral reference, CPM predicts joint response statistics to incorporate selected cross-step dependencies into memory supervision. Response prediction and action imitation jointly train a fixed-capacity recurrent memory. A risk decomposition further characterizes the prediction error introduced by conflating histories with different response statistics. Across five simulation benchmarks and both single-arm and bimanual real-world platforms, CPM achieves higher mean success than two history-based baselines using the same VLA backbone. Ablations further support the roles of program conditioning, joint outcome targets, and joint memory training in these gains. These results support controlled joint response prediction as supervision for learning compact memories that serve subsequent decisions.

AI USE STATEMENT

We used generative AI tools to assist literature organization, manuscript drafting and translation, experimental-protocol clarification, result interpretation, and typesetting. The authors take responsibility for the final content.

REFERENCES

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Jun Cen, Siteng Huang, Yuqian Yuan, Kehan Li, Hangjie Yuan, Chaohui Yu, Bohan Hou, Yuming Jiang, Jiayan Guo, Xin Li, et al. Rynnvla-002: A unified vision-language-action and world model. *arXiv preprint arXiv:2511.17502*, 2025.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu 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.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. Robomme: Benchmarking and understanding memory for robotic generalist policies. *arXiv preprint arXiv:2603.04639*, 2026.

Shengliang Deng, Mi Yan, Yixin Zheng, Jiayi Su, Wenhao Zhang, Yitao Zeng, Xiaoguang Zhao, Heming Cui, Zhizheng Zhang, and He Wang. Stereovla: Enhancing vision-language-action models with stereo vision. *arXiv preprint arXiv:2512.21970*, 2025.

Cunxin Fan, Xiaosong Jia, Yihang Sun, Yixiao Wang, Jianglan Wei, Ziyang Gong, Xiangyu Zhao, Masayoshi Tomizuka, Xue Yang, Junchi Yan, et al. Interleave-vla: Enhancing robot manipulation with image-text interleaved instructions. In *International Conference on Learning Representations*, volume 2026, pp. 40872–40902, 2026.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: A progressive robustness benchmark for visual-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 38574–38583, 2026.

Ahmed Hefny, Zita Marinho, Wen Sun, Siddhartha Srinivasa, and Geoffrey Gordon. Recurrent predictive state policy networks. In *International Conference on Machine Learning*, pp. 1949–1958. PMLR, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Bing Hu, Zaijing Li, Rui Shao, Junda Chen, April Hua Liu, Wei-Shi Zheng, and Liqiang Nie. From abstraction to instantiation: Learning behavioral representation for vision-language-action model. *arXiv preprint arXiv:2605.22671*, 2026a.

Yucheng Hu, Jianke Zhang, Yuanfei Luo, Yanjiang Guo, Xiaoyu Chen, Xinshu Sun, Kun Feng, Qingzhou Lu, Sheng Chen, Yangang Zhang, et al. Bagelvla: Enhancing long-horizon manipulation via interleaved vision-language-action generation. *arXiv preprint arXiv:2602.09849*, 2026b.

Yutong Hu, Jan-Nico Zaech, Nikolay Nikolov, Yuanqi Yao, Sombit Dey, Giuliano Albanese, Renaud Detry, Luc Van Gool, and Danda Paudel. Ar-vla: Autoregressive action expert for vision-language-action models. In *Proceedings of Robotics: Science and Systems*, 2026c.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Xiaosong Jia, Bowen Yang, Zuhao Ge, Xian Nie, Yuchen Zhou, Cunxin Fan, Yufeng Li, Yilin Chai, Chao Jing, Zijian Liang, et al. Guidedvla: Specifying task-relevant factors via plug-and-play action attention specialization. *arXiv preprint arXiv:2605.12369*, 1, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

Daniel Lawson, Adriana Hugessen, Charlotte Cloutier, Glen Berseth, and Khimya Khetarpal. Self-predictive representations for combinatorial generalization in behavioral cloning. In *International Conference on Learning Representations*, volume 2026, pp. 101559–101586, 2026.

Jung Min Lee, Dohyeok Lee, Seokhun Ju, Taehyun Cho, Jin Woo Koo, Li Zhao, Sangwoo Hong, and Jungwoo Lee. Mvp-lam: Learning action-centric latent action via cross-viewpoint reconstruction. *arXiv preprint arXiv:2602.03668*, 2026.

Kuang-Huei Lee, Ted Xiao, Adrian Li, Paul Wohlhart, Ian Fischer, and Yao Lu. Pi-qt-opt: Predictive information improves multi-task robotic reinforcement learning at scale. In *Conference on Robot Learning*, pp. 1696–1707. PMLR, 2023.

Boyu Li, Chaoyi Xu, Haoqi Yuan, Xinrun Xu, Börje F Karlsson, Haoran Li, Zongqing Lu, and Dongbin Zhao. X-diffvla: X-embodied diffusion action heads for vision-language-action models. *arXiv preprint arXiv:2605.25044*, 2026a.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026b.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024b.

Zaijing Li, Bing Hu, Rui Shao, Gongwei Chen, Dongmei Jiang, Pengwei Xie, Jianye Hao, and Liqiang Nie. Global prior meets local consistency: Dual-memory augmented vision-language-action model for efficient robotic manipulation. *arXiv preprint arXiv:2602.20200*, 2026c.

Qing Lian, Kent Yu, and Lei Zhang. Reflective vla: In-context action consequences make vlas generalize. *arXiv preprint arXiv:2606.25215*, 2026.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Mengya Liu, Baoxiong Jia, Jiangyong Huang, Jingze Zhang, and Siyuan Huang. Lara: Latent action representation alignment for vision-language-action models. *arXiv preprint arXiv:2606.07100*, 2026.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Rutav Shah, Yisu Li, Femi Bello, Yuke Zhu, and Roberto Martín-Martín. Memory retrieval in visuomotor policies for long-horizon robot control. *arXiv preprint arXiv:2606.25136*, 2026.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 18567–18602, 2026.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up memory for robotic control via experience retrieval. In *International Conference on Learning Representations*, volume 2026, pp. 97142–97166, 2026.

Wen Sun, Arun Venkatraman, Byron Boots, and J Andrew Bagnell. Learning to filter with predictive state inference machines. In *International conference on machine learning*, pp. 1197–1205. PMLR, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Arun Venkatraman, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Martial Hebert, Byron Boots, Kris Kitani, and J Bagnell. Predictive-state decoders: Encoding the future into recurrent networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on robot learning*, pp. 1723–1736. PMLR, 2023.

Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, et al. Rise: Self-improving robot policy with compositional world model. *arXiv preprint arXiv:2602.11075*, 2026.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11941–11952. IEEE, 2023.

Xuanran Zhai, Zekai Huang, Longyan Wu, Qianyou Zhao, Qiaojun Yu, Jieji Ren, Ce Hao, and Harold Soh. Skillvla: Tackling combinatorial diversity in dual-arm manipulation via skill reuse. *arXiv preprint arXiv:2603.03836*, 2026.

Jianke Zhang, Xiaoyu Chen, Qiuyue Wang, Mingsheng Li, Yanjiang Guo, Yucheng Hu, Jiajun Zhang, Shuai Bai, Junyang Lin, and Jianyu Chen. Vlm4vla: Revisiting vision-language-models in vision-language-action models. *arXiv preprint arXiv:2601.03309*, 2026.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, et al. Flare: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

Linqing Zhong, Yi Liu, Yifei Wei, Ziyu Xiong, Si Liu, and Guanghui Ren. Acot-vla: Action chain-of-thought for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8152–8162, 2026.

Yijing Zhou, Qiwei Liang, Sitong Zhuang, Jiaxi Li, Xianpeng Wang, Boyang Cai, Yunyang Mo, and Renjing Xu. Action-effect memory pretraining for robot manipulation. *arXiv preprint arXiv:2606.12499*, 2026.

Fangqi Zhu, Zhengyang Yan, Zicong Hong, Quanxin Shou, Xiao Ma, and Song Guo. Wmpo: World model-based policy optimization for vision-language-action models. In *International Conference on Learning Representations*, volume 2026, pp. 62486–62502, 2026.

A Related Work

Vision-language-action models. Vision-language-action (VLA) models use pretrained vision-language knowledge to learn robot actions (Intelligence et al., 2025; Kim et al., 2025; Zhang et al., 2026; Pertsch et al., 2025; Qu et al., 2025; Li et al., 2026a; Zhai et al., 2026; Deng et al., 2025). To further exploit the relationship between actions and environmental changes, work on world models predicts subsequent outcomes to support action generation, planning, and policy improvement (Cen et al., 2025; Kim et al., 2026; Zhu et al., 2026; Yang et al., 2026; Hu et al., 2026b; Li et al., 2026b; Assran et al., 2025). For tasks with temporal dependencies, action selection also requires information from past interactions: similar current observations may correspond to different execution histories and require different subsequent actions (Dai et al., 2026; Shi et al., 2026; Hu et al., 2026c).

Temporal modeling in robotics. Memory-augmented policies address this ambiguity by retrieving, selectively retaining, or recurrently encoding historical context (Shi et al., 2026; Sridhar et al., 2026; Li et al., 2026c). Jointly representing executed actions and the observations before and after them connects scene changes to past interactions, allowing policies to use evidence from execution beyond a single image (Lian et al., 2026; Zhou et al., 2026). Learning objectives guide which information memory retains: history reconstruction, future visual feature and action prediction, and task-oriented question answering have provided explicit supervision for memory representations (Zhou et al., 2026; Hu et al., 2026a; Shah et al., 2026). These objectives, however, do not directly test whether memory conflates histories with different dependencies between outcomes under the same subsequent control rule. Even when mean outcomes at each time step are similar, the cross-step relationships associated with two histories may differ; ignoring this distinction during compression may discard interaction information useful for subsequent decisions.

Predictive representations. The predictive-state perspective provides a basis for this information retention problem: histories are characterized by statistics of future observations, including predictions conditioned on future actions (Venkatraman et al., 2017; Hefny et al., 2018; Sun et al., 2016). Recent auxiliary prediction and alignment objectives likewise connect future information to policy learning, showing that prediction can shape representations used for control (Zheng et al., 2025; Lawson et al., 2026; Liu et al., 2026; Lee et al., 2026). CPM defines its predictive objective for VLA history compression through joint responses under fixed closed-loop programs, using both behavioral conditions and cross-step outcome relationships to distinguish histories. Together with action imitation, this objective trains memory to retain historical information relevant to subsequent responses, supporting action selection.

B Controlled Outcomes and Supervision Data

B.1 Program Records and Observation Endpoints

A program $\mathcal{P}_j$ is specified by the frozen memoryless policy π_b, the fixed flow-sampling noise for each of its two decisions, and the platform's preprocessing, command-issuing, and stopping rules. Before execution, each collection anchor is independently assigned a program uniformly at random; we record the history H, program index J, assignment probability $1/M$, program version, and the two sequences of actually issued commands, together with the observations before and after execution and their temporal alignment. The program outcome Y consists of the observation endpoints after the two command sequences are issued; unissued portions of predicted action chunks are excluded from the execution history.

Each assignment has two endpoint slots. Ordinary failures retain the actual observations. If execution is aborted and a complete new observation is available, this final observation fills the current and remaining endpoint slots; otherwise, these slots are marked with the missing symbol $\perp$. Previously obtained endpoints remain unchanged. This rule defines a target for every assignment, including aborted executions; filled endpoints are used to encode outcomes, while memory updates use only actually observed transitions. Execution, preprocessing, and stopping rules within the program library remain unchanged across the two collection rounds.

B.2 Fixed Endpoint Features and Joint Mapping

We reuse the frozen SigLIP encoder in the $\pi_{0.5}$/openpi backbone (Zhai et al., 2023; Intelligence et al., 2025). Each observation is represented by sixteen spatially pooled image-region features $\bar{v}_r$, each normalized to unit norm, together with the normalized proprioceptive state $\bar{s}$. For an m-row frequency matrix B, paired random Fourier features (Rahimi & Recht, 2007) are defined as

$$g_B(x) = m^{-1/2}[\cos(Bx); \sin(Bx)]. \tag{6}$$

We use seven frequencies for each image region and sixteen for the state, giving the endpoint representation

$$\begin{aligned} \psi(o) &= 17^{-1/2}[g_{B_0}(\bar{v}_0); \ldots; g_{B_{15}}(\bar{v}_{15}); g_{B_s}(\bar{s})], \\ \psi(\perp) &= 0. \end{aligned} \tag{7}$$

Thus, $\psi(o)$ has $16 \times 14 + 32 = 256$ coordinates, and valid observations satisfy $\|\psi(o)\|_2 = 1$. The two endpoints are concatenated, scaled, and passed through a single nonlinear mapping:

$$z(Y) = 2^{-1/2}[\psi(o^{[1]}); \psi(o^{[2]})], \qquad \phi(Y) = g_\Omega(z(Y)). \tag{8}$$

Ω contains 256 frequencies and produces a 512-dimensional joint feature. Since $\cos^2 u + \sin^2 u = 1$, we have $\|\phi(Y)\|_2 = 1$ for all inputs, including those with missing endpoints. The mapping is applied to each recorded endpoint pair, and the conditional mean of the resulting features is then learned under the squared loss.

Entries of the frequency matrices are sampled independently from zero-mean Gaussian distributions with standard deviations equal to the reciprocals of the corresponding bandwidths. Region and state bandwidths are determined from demonstration training observations, and the joint bandwidth from the first round of program training records; each uses the median positive distance between sampled pairs. The endpoint mapping is fixed before the first round of program collection, and the joint mapping is fixed after the first-round records are collected but before joint training begins. The resulting frequencies and bandwidths remain unchanged during subsequent training and the second collection round.

C Memory Architecture and Training Implementation

C.1 Recurrent State and Its Two Readouts

The recurrent module F_β receives image-region features and states from observations before and after execution, the issued command sequence, and a pooled instruction embedding from the frozen

backbone. Modality, spatial-region, temporal-role, and command-position embeddings distinguish these inputs. The previous memory slots are projected to form queries; each block applies slot self-attention, cross-attention to valid input tokens, and a feedforward network (Vaswani et al., 2017), using pre-normalization and residual connections. An output projection followed by normalization produces four updated slots. At initialization, the previous-state slots are set to zero, and the absent previous observation and commands are masked; an episode boundary, an instruction change, or unrecoverable telemetry loss terminates the current recurrent sequence, and the next valid sequence is reinitialized. Padded transitions within a batch leave the memory unchanged.

The shared decoder D_η uses a learned program query to attend to all four projected slots, followed by a feedforward readout that produces a 512-dimensional outcome prediction. The program index and its query table are used only by the prediction decoder. The action expert reads the four memory slots through a shared affine projection W; the current proprioceptive state is provided together with the instruction using the backbone's discrete state tokens, and unused image slots are masked. Table 5 specifies the added components; the pretrained encoder and action expert follow $\pi_{0.5}$/openpi (Intelligence et al., 2025).

Table 5: Configuration of the components added by CPM.

Component	Configuration
Latent memory	4 slots, width 512
Recurrent update	2 attention blocks, width 256, 4 heads
Feedforward block	Hidden width 1024, GELU (Hendrycks & Gimpel, 2016)
Prediction readout	1 query-attention block; 512-dimensional output
Action conditioning	Shared affine projection to the action expert width

Denoting the current prefix, memory, and action tokens by P, M, A, respectively, the attention groups are

query\key	P	M	A
P	1	0	0
M	1	1	0
A	1	1	1

. (9)

Padded positions are masked, position indices follow the order of valid tokens, and the final action positions output the action chunk. The projected readout $W(h_t)$ is held fixed within each decision, without adding noise. The current prefix can be cached during that decision's denoising process; the memory and action suffixes, which are conditioned on the denoising time, are recomputed at every step. Only the recurrent state h_t is retained across decisions; historical tokens and the KV cache from the previous decision are not reused.

C.2 Loss Reduction and Gradient Paths

We follow the backbone's flow-matching formulation (Lipman et al., 2022; Intelligence et al., 2025), conditioning action generation on $(C, W(h))$. Let B_a, B_p denote the demonstration and prediction batch sizes, respectively, and let i, k, d index records, action rows, and action coordinates. Here, $v_{\theta,W,\beta,ikd}$ is the memory-conditioned predicted velocity, v^*_{ikd} is the target velocity, and m_{ikd} is the validity mask. For demonstration records with at least one valid action coordinate, the empirical losses are

$$\widehat{\mathcal{L}}_{\text{act}} = \frac{1}{B_a} \sum_i \frac{\sum_{k,d} m_{ikd} (v_{\theta,W,\beta,ikd} - v^*_{ikd})^2}{\sum_{k,d} m_{ikd}},$$
$$\widehat{\mathcal{L}}_{\text{pred}} = \frac{1}{B_p} \sum_i \sum_{q=1}^{512} [D_\eta(h_i, J_i)_q - \phi_q(Y_i)]^2. \tag{10}$$

The action error is averaged over valid action rows and coordinates within each record; the prediction error is summed over all feature coordinates and then averaged over records. With this scaling, we use $\lambda = 1$.

The two types of batches are sampled independently; their losses are computed separately, and the gradients are combined to update the parameters. Prediction gradients update F_β, D_η, while action gradients update F_β, W and the entire action expert θ; the vision–language backbone and outcome mapping remain frozen. For each training record, memory is recomputed from the start of the episode using the current parameters, and gradients are backpropagated through the full recorded prefix. Frozen observation features can be reused; supervision endpoints enter the prediction loss only after the memory at the anchor has been computed.

C.3 Two Rounds of History Collection

In the first round, fixed programs are assigned at random and executed at histories visited by the frozen policy π_b, yielding $\mathcal{D}_p^{(0)}$. After the first stage of joint training, the policy parameters are frozen as π_{col} to generate history anchors for the second round. At each anchor, we still execute a program defined by the original π_b, yielding $\mathcal{D}_p^{(1)}$. The second stage continues joint training from the first-stage parameters, using the same demonstrations and program records from both rounds.

For prediction batches, we first sample tasks and available collection rounds uniformly, then uniformly sample anchors with assigned programs within the selected group. Records are not filtered by program success or prediction error. This mixture over tasks and rounds defines the history population ν in Appendix D; the original program assignment probability at each anchor remains $1/M$.

D Prediction Risk and Joint Statistics

D.1 Response Supervision under Random Assignment

Fix the policy parameters, sampling noise, execution and endpoint rules of the program library, and the established outcome mapping ϕ. Let $H \sim \nu$ be a newly sampled history from the population defined by the prediction sampling design. For program j, let Y^j denote the potential endpoints obtained by executing the program after this history, and let $Z^j = \phi(Y^j)$. Uniform random assignment before execution and endpoint recording satisfy

$$J \perp (Z^0, \ldots, Z^{M-1}) \mid H, \qquad \Pr(J = j \mid H) = 1/M, \qquad Y = Y^J. \tag{11}$$

Independence follows from random assignment before program execution; positive assignment probabilities cover all programs, and the last equality expresses consistency between the recorded outcome and the executed fixed rule. This definition does not require executing every program from the same physical state.

With model parameters fixed, write $h = h_\beta(H)$ and $d_j(H) = D_\eta(h, j)$. Averaging over random assignments gives

$$\mathcal{R}_{\mathrm{pred}} = \mathbb{E}\|d_J(H) - Z^J\|_2^2 = \frac{1}{M}\sum_{j=0}^{M-1} \mathbb{E}\|d_j(H) - Z^j\|_2^2. \tag{12}$$

Thus, a single randomly assigned execution at each anchor provides sampled supervision for the equally weighted program risk, without additional importance weights. This equality holds for the population risk under the sampling design; program counts may differ in a finite set of records.

D.2 Prediction-Risk Decomposition

The conditional response given the full history and the optimal response prediction supported by memory are, respectively,

$$r_j(H) = \mathbb{E}[Z^j \mid H], \qquad \bar{r}_{\beta,j}(h) = \mathbb{E}[r_j(H) \mid h_\beta(H) = h]. \tag{13}$$

By the law of iterated expectations, $\bar{r}_{\beta,j}(h) = \mathbb{E}[Z^j \mid h_\beta(H) = h]$. When histories with different $r_j(H)$ are mapped to the same h, the optimal prediction based on that memory is their conditional average.

Proposition 1 (Prediction-risk decomposition)**.** *Under the conditions of Eq.* (11)*, fix* β, η *and assume* $\mathbb{E}\|d_j(H)\|_2^2 < \infty$ *for every program. Then*

$$\mathcal{R}_{\text{pred}} = V_H + C_{\text{pred}}(\beta) + E_{\text{dec}}(\beta, \eta), \tag{14}$$

where

$$\begin{aligned} V_H &= \frac{1}{M}\sum_j \mathbb{E}\|Z^j - r_j(H)\|_2^2, \\ C_{\text{pred}}(\beta) &= \frac{1}{M}\sum_j \mathbb{E}\|r_j(H) - \bar{r}_{\beta,j}(h)\|_2^2, \\ E_{\text{dec}}(\beta, \eta) &= \frac{1}{M}\sum_j \mathbb{E}\|d_j(H) - \bar{r}_{\beta,j}(h)\|_2^2. \end{aligned} \tag{15}$$

Proof. For each program, write the prediction residual as

$$\begin{aligned} Z^j - d_j(H) = [Z^j - r_j(H)] + [r_j(H) - \bar{r}_{\beta,j}(h)] \\ + [\bar{r}_{\beta,j}(h) - d_j(H)]. \end{aligned}$$

The first term has zero conditional mean given H, while the other two terms are functions of H; hence, it is orthogonal to both. The second term has zero conditional mean given h, and the third is a function of h, so these two terms are also orthogonal. Squaring and taking expectations eliminates the cross terms; averaging over programs using Eq. (12) yields the result. Since $\|\phi(Y)\|_2 = 1$, both the targets and their conditional means have finite second moments. □

Given the memory, $\bar{r}_{\beta,j}$ is the optimal squared-loss decoder without restrictions on the function class. Here, V_H is independent of the memory and decoder parameters, C_{pred} is the increase in optimal prediction risk caused by replacing the full history with memory, and E_{dec} is the error of the actual decoder relative to this optimal prediction. The prediction objective optimizes the sum of the latter two terms; it does not require either term to decrease monotonically during training. Since h encodes the current observation and prior execution, the compression term concerns the response information contained in both.

This decomposition explains how outcome supervision constrains the history representation, while the action objective trains its relationship to demonstration actions through the shared F_β. The analysis concerns population prediction risk under the selected finite program set and outcome features; it does not establish control sufficiency or guarantee task success.

D.3 Role of Joint Endpoint Statistics

Predicting endpoint means separately constrains only marginal statistics; a nonlinear encoding of each endpoint pair can additionally form statistics that depend on their relationship. Consider $B_1, B_2 \in \{-1, +1\}$ with uniform marginals. The cases $B_2 = B_1$ and $B_2 = -B_1$ have identical marginal distributions, but their joint cosine features satisfy

$$\mathbb{E}\cos(\omega_1 B_1 + \omega_2 B_2) = \begin{cases} \cos(\omega_1 + \omega_2), & B_2 = B_1, \\ \cos(\omega_1 - \omega_2), & B_2 = -B_1. \end{cases} \tag{16}$$

For frequencies that make the two values on the right-hand side differ, this feature distinguishes endpoint dependencies with identical marginals. CPM likewise encodes each endpoint pair before taking expectations, learning a finite set of joint statistics rather than the full outcome distribution.

E Experimental Details

E.1 Experimental Protocols

Tasks and evaluation. Table 6 summarizes the tasks and the number of evaluation episodes per checkpoint. We use each benchmark's success criterion and count timeouts as failures; success rates

are first computed within each task and then averaged with equal weight across tasks. In simulation, models with the shared backbone use the same evaluation initializations and control interface, and the full evaluation is performed for each of the three training seeds. For LIBERO-Plus, we use the policy trained on the corresponding original task suite and compute per-category and overall success rates on the official evaluation instances.

Table 6: Tasks and evaluation protocols for the simulation benchmarks. Episode counts specify the evaluation budget per checkpoint.

Benchmark	Tasks	Evaluation budget
LIBERO	Spatial, Object, Goal, and Long, with ten tasks each	50 episodes per task
SimplerEnv-Bridge	Spoon on Towel, Carrot on Plate, Stack Cube, Eggplant in Basket	24 episodes per task
RoboMME	BinFill, PickXtimes, SwingXtimes, StopCube, ButtonUnmask, ButtonUnmaskSwap, PickHighlight	50 episodes per task
RoboTwin 2.0	The fixed set of sixteen tasks in Table 10; Hard setting	100 episodes per task
LIBERO-Plus	Seven perturbation categories, with 10,030 instances in total	1 episode per instance

Bimanual tasks. On RoboTwin, we use the Aloha–AgileX platform and train a separate single-task policy on 50 clean expert demonstrations per task. The task set matches the sixteen tasks used by OptimusVLA and is fixed before training (Li et al., 2026c). Demonstrations and both rounds of program data collection use the clean scene distribution; evaluation uses the official Hard configuration, which varies distractors, backgrounds, lighting, and tabletop height, with the unseen instruction split used throughout. Current inputs comprise RGB images from the head and both wrist cameras, together with proprioceptive states of both arms; actions follow the official joint-space interface, including joint targets and gripper commands for both arms. CPM updates a shared memory using the full bimanual command and its subsequent observation, and the fixed programs are reconstructed for this action interface.

E.2 Baseline Implementations

Shared backbone. Triplet History, FrameSamp+Modul, and CPM use the same demonstration-adapted weights and action-training budget. Triplet History encodes the most recent 32 executed transitions, including frozen visual features, proprioceptive states, actual commands, and temporal and modality embeddings. FrameSamp+Modul retains a history of 32 main-camera frames, 512 visual history tokens, and the memory modulation mechanism (Dai et al., 2026). Both baselines train the action expert and added conditioning modules while keeping the vision–language backbone frozen. These baselines provide historical information but do not use CPM's loss for predicting program outcomes.

Results from prior work. The $\pi_{0.5}$ results on LIBERO use PI's official `pi05_libero` checkpoint at 30,000 steps. We use the continuous-action version of RynnVLA-002; for LIBERO Long, MemoryVLA jointly trains on Long and LIBERO-90, and selects the best validation checkpoints on LIBERO and Bridge (Cen et al., 2025; Shi et al., 2026). For $\pi_{0.5}$ on Bridge, we use the in-domain Text/Text result from Interleave-VLA at its best checkpoint, for which only the four-task average is available (Fan et al., 2026); the CogACT-Large and MemoryVLA results come from MemoryVLA, whose authors evaluated the publicly available CogACT-Large weights (Shi et al., 2026). For RoboMME, the OpenVLA-OFT and MemoryVLA results use the benchmark authors' adaptations to sixteen tasks, while the $\pi_{0.5}$ results use their joint training on sixteen tasks; all three are re-averaged over our seven-task subset and retain the evaluation repetition protocols of the original report. The training data, action representations, and training recipes of the first two differ from those of our same-backbone baselines (Dai et al., 2026). For RoboTwin, the RDT, OptimusVLA, and $\pi_{0.5}$ results are taken from the sixteen-task Hard evaluation reported by OptimusVLA and averaged with equal weight across tasks; that source does not specify the language split, whereas we use unseen instructions throughout (Li et al., 2026c). For LIBERO-Plus, the OpenVLA-OFT results come

from the benchmark (Fei et al., 2026), and the $\pi_{0.5}$ results come from ACoT-VLA's direct-transfer evaluation of the official checkpoint (Zhong et al., 2026).

E.3 Data Collection and Optimization

Initialization and data. We use expert demonstrations from LIBERO, RoboMME, and RoboTwin, together with BridgeData V2, for action supervision (Liu et al., 2023; Dai et al., 2026; Chen et al., 2025; Walke et al., 2023). Except for RoboTwin, where all 50 clean demonstrations are used for training, demonstrations are split into training and validation sets at a ratio of 90:10, by episode within each task. Starting from `pi05_base`, we fine-tune the policy on the training demonstrations for 30,000 updates to obtain the initial policy π_b shared across comparisons. CPM carries model parameters from the first stage into the second, while resetting the optimizer and learning-rate schedule between stages. Statistics for normalization and endpoint mapping are determined only from training data; the joint outcome mapping is determined from the first-round training records and then held fixed across both stages.

Program records. In each round, we collect 200 training anchors and 50 validation anchors per task, assigning collection episodes to their respective splits before execution. At an eligible decision point, a program is initiated with probability 0.1, after which one of the eight fixed programs is assigned uniformly and executed for two decisions; no further program assignment occurs within a program. Histories are generated by π_b in the first round and by the frozen first-stage CPM policy in the second. Programs in both rounds use the original frozen π_b, the same sampling noise, and the same outcome mapping. Records retain actually issued commands, observations before and after execution, the program index, and the termination status; failures and interruptions follow the endpoint rules in the method section. Prediction batches are sampled uniformly across tasks and available collection rounds.

Inputs and action updates. LIBERO and RoboMME use front-view and wrist RGB images, Bridge uses front-view RGB images, and RoboTwin uses head-camera and both wrist-camera RGB images. The current proprioceptive state is provided through the backbone's discrete state tokens; the memory and outcome encoders use main-camera RGB images and proprioceptive states. All policies with the shared backbone predict ten commands, issue at most the first five at each decision, and use ten action-denoising steps; action coordinates and control intervals follow the corresponding benchmark. Memory receives only actually issued commands and subsequent observations. Action and prediction losses use separate batches, with gradients combined before clipping; the four-slot memory is recomputed from the start of each episode, and gradients are backpropagated through the observation prefix.

Table 7: Optimization settings for CPM. Same-backbone baselines match the action-update budget and the optimization settings of the action expert.

Setting	Value
Joint updates per stage	30,000
Action/prediction global batch size	256/32
Optimizer	AdamW (Loshchilov & Hutter, 2017), $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$
Memory and decoder learning rate	Peak 10^{-4}, final 10^{-5}
Action expert and projection learning rate	Peak 5×10^{-5}, final 5×10^{-6}
Weight decay	Memory/decoder matrices: 0.01; expert/projection: 10^{-10}
Schedule	1,000 warm-up steps, followed by cosine decay (Loshchilov & Hutter, 2016)
Gradient norm limit/prediction loss weight	1.0/1.0

F PER-TASK RESULTS ON SIMULATION BENCHMARKS

Tables 8, 9, and 10 report per-task results for Bridge, RoboMME, and RoboTwin, respectively. Results from our evaluations are means over three training seeds, and Avg. is the equally weighted average across tasks. Means and improvements for Bridge, RoboMME, and RoboTwin are computed from unrounded success rates and then rounded to one decimal place.

Table 8: Per-task success rates (%) on SimplerEnv-Bridge. Spoon, Carrot, Stack, and Eggplant denote the four standard tasks. † denotes results from public sources; – indicates that the source does not report the result.

Method	Spoon	Carrot	Stack	Eggplant	Avg.
CogACT-Large†	58.3	45.8	29.2	95.8	57.3
MemoryVLA†	75.0	75.0	37.5	100.0	71.9
$\pi^{\dagger}_{0.5}$	–	–	–	–	57.2
Triplet History	81.9	54.2	45.8	86.1	67.0
FrameSamp+Modul	81.9	52.8	50.0	86.1	67.7
CPM	88.9	66.7	54.2	87.5	74.3

Table 9: Success rates (%) on seven RoboMME tasks. OFT, Memory, Triplet, and Frame denote OpenVLA-OFT, MemoryVLA, Triplet History, and FrameSamp+Modul, respectively. † denotes results from public sources.

Task	OFT†	Memory†	$\pi^{\dagger}_{0.5}$	Triplet	Frame	CPM
BinFill	8.0	10.0	30.0	43.3	44.0	48.7
PickXtimes	8.0	17.3	42.9	81.3	90.0	88.7
SwingXtimes	2.0	1.3	35.6	84.0	92.7	95.3
StopCube	2.0	0.0	6.7	40.0	47.3	52.0
ButtonUnmask	0.0	8.0	22.2	29.3	28.7	36.0
ButtonUnmaskSwap	0.0	4.7	6.7	20.0	22.7	29.3
PickHighlight	10.0	7.3	11.3	22.7	26.0	25.3
Avg.	4.3	7.0	22.2	45.8	50.2	53.6

Table 10: Success rates (%) on sixteen RoboTwin 2.0 tasks under the Hard setting. Optimus, Triplet, and Frame denote OptimusVLA, Triplet History, and FrameSamp+Modul, respectively. † denotes results from public sources.

Task	RDT†	Optimus†	$\pi^{\dagger}_{0.5}$	Triplet	Frame	CPM
Click Alarmclock	12.0	31.0	18.0	27.7	22.3	33.0
Click Bell	9.0	46.0	28.0	32.0	35.7	50.3
Dump Bin Bigbin	32.0	35.0	29.0	29.3	32.0	35.7
Open Laptop	32.0	48.0	38.0	38.0	34.7	45.3
Place Bread Skillet	1.0	4.0	2.0	4.7	2.7	5.0
Place Container Plate	17.0	37.0	30.0	42.3	33.7	41.0
Press Stapler	24.0	45.0	36.0	40.7	37.0	45.3
Stack Bowls Two	30.0	58.0	49.0	50.7	54.3	57.0
Beat Block Hammer	37.0	26.0	21.0	23.3	26.7	28.0
Lift Pot	9.0	31.0	28.0	29.3	30.0	28.7
Move Playingcard Away	11.0	32.0	25.0	22.7	32.3	39.0
Open Microwave	20.0	41.0	39.0	37.0	41.7	40.3
Pick Diverse Bottles	0.0	7.0	6.0	6.0	4.7	7.3
Turn Switch	15.0	16.0	11.0	16.7	17.3	17.0
Place Object Stand	5.0	13.0	11.0	12.7	11.3	15.0
Stack Blocks Two	2.0	11.0	5.0	7.3	8.0	14.7
Avg.	16.0	30.1	23.5	26.3	26.5	31.4

F.1 Representative Rollouts

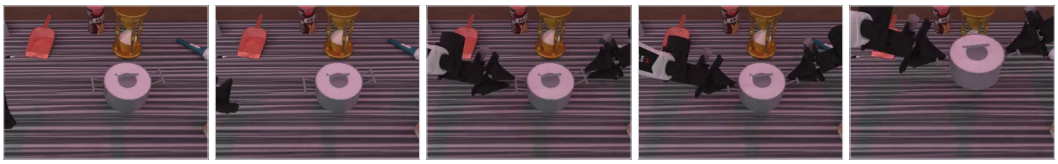
Figures 3 and 4 show representative CPM executions across simulation benchmarks.



(a) LIBERO-Long
Put the yellow and white mug in the microwave and close it.

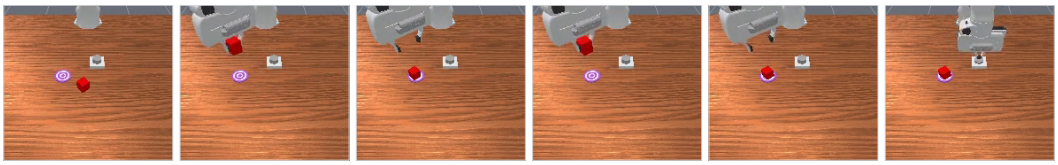


(b) SimplerEnv-Bridge
Put the carrot on the plate.

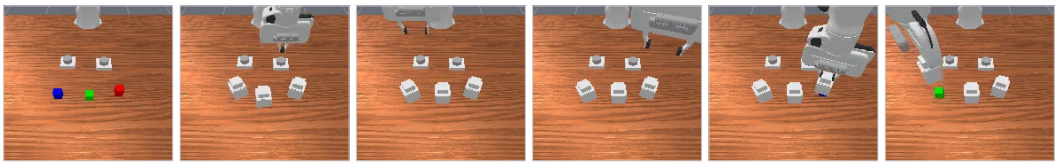


(c) RoboTwin 2.0
Lift the pot with both arms.

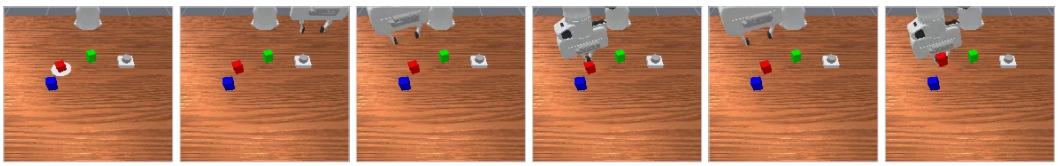
Figure 3: Representative CPM rollouts on LIBERO-Long (top), SimplerEnv-Bridge (middle), and RoboTwin 2.0 (bottom). Keyframes progress from left to right.



(a) PickXtimes - Counting
Pick up the red cube and place it on the target, repeating this action two times, then press the button to stop.



(b) ButtonUnmaskSwap - Object permanence
First press both buttons on the table, then pick up the container hiding the blue cube, finally pick up another container hiding the green cube.



(c) PickHighlight - Reference memory
First press the button, then pick up all cubes that have been highlighted with white areas on the table.

Figure 4: CPM rollouts on PickXtimes, ButtonUnmaskSwap, and PickHighlight, covering counting, object permanence, and reference memory, respectively. Each row shows keyframes from one episode in temporal order.

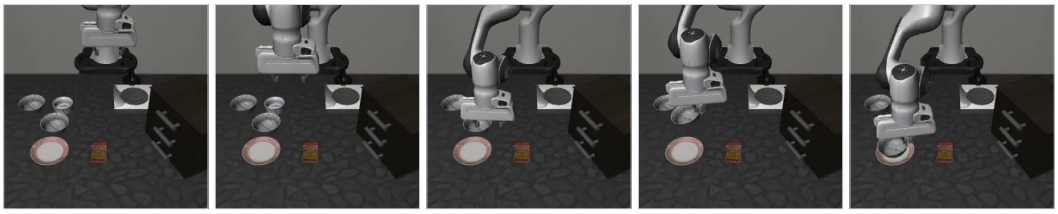
G LIBERO-PLUS EVALUATION DETAILS

Evaluation protocol. LIBERO-Plus contains 10,030 test instances: camera viewpoints, robot initial states, language instructions, lighting, background textures, sensor noise, and object layouts account for 1,599, 1,550, 1,537, 1,142, 1,076, 1,601, and 1,525 instances, respectively (Fei et al., 2026). Each instance uses the policy for its original task suite, with that policy's observation processing, command dispatch, and termination rules unchanged. Each instance is run once per training seed, for a total of 30,090 episodes across three seeds. All methods share the same instances and initializations, with memory reset at the start of each episode. Policies use the language instruction provided with each test instance; perturbation categories and difficulty labels are used only to aggregate results.

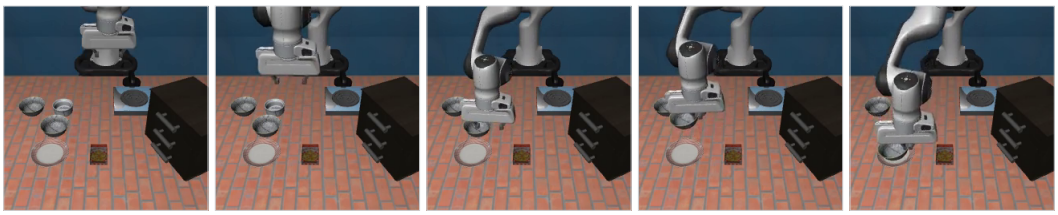
Statistics. Per-category and overall success rates are computed over the instances in each category and all 10,030 instances, respectively. We aggregate results independently for each training seed, then compute means, sample standard deviations, and improvements from the unrounded results.

Execution examples. Figure 5 shows executions of the same task under different background textures.

Pick up the black bowl between the plate and the ramekin and place it on the plate.



(a) Background variation 1



(b) Background variation 2

Figure 5: CPM executions of the same task under two background textures in LIBERO-Plus. Keyframes are ordered from left to right.

H Additional Ablation Studies

Setup. Ablations follow the training and evaluation protocols in Section 3.1, with each variant trained separately from the same initialization and with matched action-update budgets. For each training seed, variants with prediction supervision share the first-round program records collected from histories generated by π_b and the second-round records collected from histories generated by a frozen reference CPM policy after its first training stage; both rounds execute the original fixed program library. Each variant continues its second-stage training from its own first-stage parameters, allowing memory inputs, prediction targets, and gradient paths to be compared under a shared auxiliary-data distribution.

Execution history and joint optimization. A1 reinitializes the latent state from the current observation and instruction at each decision point, without reading historical states, observations, or commands, while retaining the prediction decoder and action readout branch. A2 zeros only the command values in the recurrent model's input, retaining observations before and after execution, the actually issued sequence length, and validity masks; action supervision labels and control outputs are unchanged. A3 retains the recurrent model, projection, and action expert, removing only the prediction loss. A6 uses $W(\mathrm{sg}(h_t))$ in the action branch: the prediction loss updates F_β, D_η, while the action loss updates only W and the action expert. The input modifications in A1 and A2 apply during both training and deployment.

Prediction conditioning and outcome features. A4 replaces the decoder's eight program queries with a single shared query, leaving program assignments, collected records, and outcome labels unchanged. A5 encodes the two endpoints separately and concatenates their encodings into additive outcome features. Let the original frozen frequency matrix be $\Omega = [\Omega_1\ \Omega_2] \in \mathbb{R}^{256\times512}$, and the endpoint encodings be $\psi_1, \psi_2 \in \mathbb{R}^{256}$. We define

$$\begin{aligned} v_1 &= \Omega_1^{1:128}\psi_1/\sqrt{2}, \qquad v_2 = \Omega_2^{129:256}\psi_2/\sqrt{2}, \\ g_i &= [\cos v_i; \sin v_i]/\sqrt{128}, \qquad \phi_{\mathrm{add}} = [g_1; g_2]/\sqrt{2}. \end{aligned} \tag{17}$$

Matrix superscripts indicate the selected rows, and missing endpoints retain their zero encodings. This replacement preserves the 512-dimensional output and unit norm of the outcome features while replacing the joint endpoint mapping with a concatenation of independent endpoint mappings.

Per-task results. Tables 11 and 12 provide per-task results for the two benchmarks, with Avg. denoting the equally weighted average across tasks. Variant identifiers match those in Table 3.

Table 11: Ablation results on the four Bridge tasks (success rate, %).

Task	A0	A1	A2	A3	A4	A5	A6
Spoon on Towel	88.9	80.6	84.7	87.5	83.3	90.3	80.6
Carrot on Plate	66.7	50.0	59.7	52.8	56.9	61.1	54.2
Stack Cube	54.2	44.4	51.4	44.4	48.6	50.0	45.8
Eggplant in Basket	87.5	79.2	91.7	83.3	87.5	90.3	86.1
Avg.	74.3	63.5	71.9	67.0	69.1	72.9	66.7

Table 12: Ablation results on the seven RoboMME tasks (success rate, %).

Task	A0	A1	A2	A3	A4	A5	A6
BinFill	48.7	36.7	42.7	40.0	44.7	47.3	39.3
PickXtimes	88.7	50.7	88.0	87.3	90.0	90.7	83.3
SwingXtimes	95.3	42.0	92.0	92.7	93.3	94.0	91.3
StopCube	52.0	8.0	47.3	40.7	43.3	46.7	35.3
ButtonUnmask	36.0	27.3	35.3	26.0	30.7	34.0	27.3
ButtonUnmaskSwap	29.3	8.7	28.7	20.0	22.7	24.7	18.7
PickHighlight	25.3	15.3	26.0	26.0	26.0	24.7	16.7
Avg.	53.6	27.0	51.4	47.5	50.1	51.7	44.6

Subsequent-command-conditioned prediction. We also replace the program query with the two command segments actually executed by the program. Commands are encoded in execution order with segment indices, positional encodings, and validity masks; a query attends to the command sequence, and its output is fed into the original decoder. Subsequent commands are used only to train the decoder; anchor memory is still computed before program assignment, and the action policy's inputs are unchanged. Because the second command segment depends on the first endpoint, this variant conditions on the realized action sequence and adds a command encoder used only during training. In Table 13, the program-query and subsequent-command conditions achieve success rates of 53.6% and 51.6%, respectively. The program index fixes the subsequent closed-loop feedback rule before execution, whereas subsequent-command conditioning describes the actual action sequence that depends on execution feedback.

Prediction loss weight. We set λ to 0.25 and 4, keeping loss reduction, gradient clipping, and all other training settings unchanged. Table 13 also includes results for the default weight 1 and zero weight, reusing full CPM and A3, respectively. Among the evaluated settings, $\lambda = 1$ achieves the highest mean success rate.

Table 13: Additional ablations on RoboMME. Success rates (%) are means over three training seeds. The shared-query, program-query, and zero-weight results reuse A4, A0, and A3, respectively.

Comparison	Setting	Success rate
Prediction conditioning	Shared query	50.1
	Program query	53.6
	Subsequent commands	51.6
Prediction loss weight	0	47.5
	0.25	50.1
	1	53.6
	4	49.8

I Efficiency Evaluation

Inference settings. We use a batch size of 1, dual-view 224×224 images, and the same language and state inputs; the action chunk length and number of denoising steps are both set to 10. All methods reuse input sequences from the ten LIBERO-Long tasks, with 30 consecutive decisions measured per task; timing includes visual encoding, history updates, and action generation. The history budget is 32 decisions, and these sequences contain 0–29 historical transitions. Generated-action throughput (actions/s) is the action chunk length divided by the unrounded mean latency and does not represent the actual control frequency. Peak allocated GPU memory is the peak memory in use as reported by the JAX allocator in a separate inference process, including loading and warm-up, and is measured in decimal GB. The added modules of the two adapted history-based baselines are randomly initialized for overhead measurement only.

Table 14: Inference efficiency on replayed LIBERO-Long inputs. Throughput counts generated actions, and GPU memory is the peak allocated memory of a separate inference process.

Method	Latency (s)↓	Action throughput (actions/s)↑	Peak allocated GPU memory (GB)↓
$\pi_{0.5}$-SFT	0.064	156.8	7.90
Triplet History	0.080	124.6	7.99
FrameSamp+Modul	0.101	99.1	8.38
CPM	0.082	121.3	7.96

J Real-World Experimental Details

J.1 Platforms and Tasks

The single-arm platform uses a 6-DoF RM65, an EG2-4C2 gripper, and two Intel RealSense D435i cameras with overhead and side views. The bimanual platform uses two 7-DoF Realman arms with EG2-4C2 grippers, front and rear D435i cameras, and a side-view RGB camera. All visual inputs are RGB. Within each platform, methods share the control interface, action coordinates, normalization rules, and control period.

An operator resets the scene and places the objects before each episode, after which the policy executes online with the instruction given only at the start. Stable placements must be maintained for two seconds after release. Drops, objects leaving the reachable workspace, safety stops, incorrect ordering, timeouts, and manual corrections count as failures. Brush paths and insertion endpoints are fixed before evaluation. The conveyor uses the demonstration-collection speed and preset feeding intervals, with timing starting at the first observation. The policy may recover from missed grasps, without the operator replenishing parts or pausing the timer.

We train one multitask policy per platform, sampling tasks uniformly before sampling a demonstration trajectory. Evaluation layouts are set separately from those used to collect demonstrations and program data.

J.2 Training and Evaluation

Shared-backbone methods. We adapt the official $\pi_{0.5}$ checkpoint (Intelligence et al., 2025) on each platform's demonstrations for 30,000 action updates. From this shared initialization, methods undergo two stages of post-training, each with 30,000 action updates. They use all RGB views and the same current inputs, with the model and optimization settings from the simulation experiments. All methods are evaluated at the final checkpoint after their fixed update budget.

Other baselines. MemoryVLA (Shi et al., 2026) uses overhead RGB and language, a memory capacity of 256, and 20,000 fine-tuning updates. OpenVLA-OFT (Kim et al., 2025) uses two RGB views, proprioception, and language on the single-arm platform. The bimanual platform uses OpenVLA-OFT+ with FiLM and three RGB views; both variants are fine-tuned for 100,000 updates.

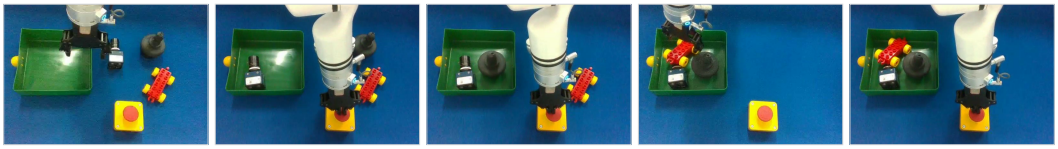
Table 15: Tasks, success criteria, and demonstration counts. All 1,100 demonstration trajectories are used for training. Time limits are in seconds.

Task	Success criteria	Demos	Time
S1 Clear table and confirm	Place all three objects in the tray; press the button exactly once after each placement and before the next grasp.	200	90
S2 Clean and restore drawer	Open the drawer, remove the green object, brush the prescribed interior path, replace the object, and close the drawer, in order.	200	120
S3 Replace food	Remove the strawberry, then place and release the banana on the plate; no strawberry remains on the plate.	100	60
S4 Nest cups in order	Place the blue cup inside the red cup, then both inside the green cup; maintain correct nesting after release.	100	60
S5 Peg insertion	Insert the yellow peg to the prescribed bottom position and maintain a stable placement after release.	100	45
B1 Bimanual handover	Transfer the red cylinder between grippers; the receiving arm places and releases it in the blue tray without dropping it.	200	90
B2 Conveyor part transfer	Transfer all four parts via the conveyor, one into each tray compartment, with no additional compartment-order constraint.	200	240

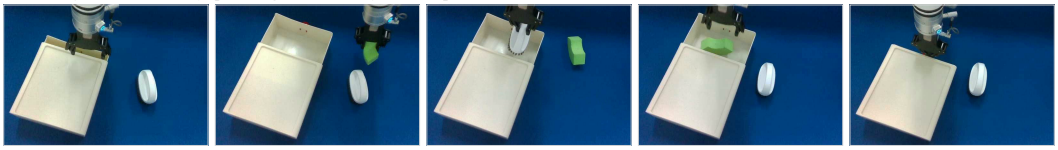
Baselines follow their public optimization recipes, with action outputs and normalization adapted to each platform.

Program interaction data. CPM uses eight fixed closed-loop programs on each platform with two decisions per program. The program backbone, sampling noise, and outcome mapping remain fixed across collection rounds. Per task and round, we collect 200 training anchors and 50 held-out anchors, uniformly assigning one program to each anchor and retaining failed or aborted executions. Records are split by collection episode, keeping each episode and its history prefixes in one split. Held-out records only monitor overfitting of auxiliary response prediction; they are not used to validate policy success or select checkpoints. First-round records come from the shared initial policy and are shared across seeds. Second-round records are collected separately with each seed's first-stage CPM policy. Stage two samples training records from both rounds equally. Program records provide response supervision only; their actions are not expert labels. Shared-backbone baselines do not use this auxiliary supervision.

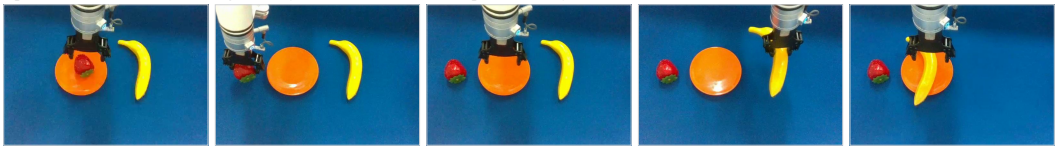
Evaluation and statistics. History buffers are reset before each episode. Task success rates are averaged across seeds. Platform results average tasks equally, with sample standard deviations computed over the three seed-level platform means. The three CPM seeds capture post-training randomness conditional on the shared initialization and first-round program data.



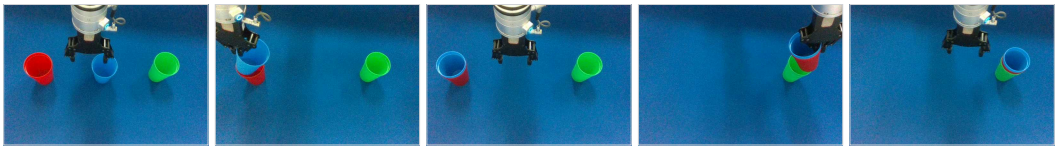
(a) Clear table and confirm
Put all objects into the tray, pressing the button after each placement.



(b) Clean and restore the drawer
Open the drawer, remove the green object, brush the inside, replace the object, and close the drawer.



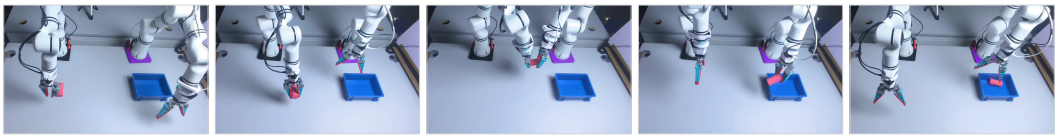
(c) Replace food
Remove the strawberry from the plate and put the banana on it.



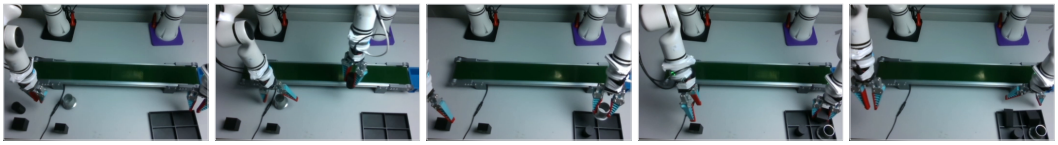
(d) Nest cups in order
Place the blue cup inside the red cup, then place both inside the green cup.



(e) Peg insertion
Insert the yellow peg into the hole in the light-blue block.



(f) Bimanual handover
Hand over the red cylinder between the two grippers, then place it in the blue tray.



(g) Conveyor-based part transfer
Transfer the four parts along the conveyor, placing one part in each tray compartment.

Figure 6: Representative real-world executions of CPM. Rows (a)–(e) correspond to single-arm tasks S1–S5, and (f)–(g) to bimanual tasks B1–B2. Each row shows key execution stages from left to right.

K TRAINING ALGORITHM

Algorithm 1 Two-stage joint training of CPM

Require: $\mathcal{D}_a, \mathcal{D}_p^{(0)}$; frozen $\pi_b, \mathcal{P}, \phi$

$\Theta \leftarrow \textsc{Initialize}(\pi_b)$
for $r = 0, 1$ **do**
  if $r = 1$ **then**
    $\mathcal{D}_p^{(1)} \leftarrow \textsc{Collect}(\mathrm{Freeze}(F_\beta, W, \pi_\theta), \mathcal{P})$
  end if
  for $n = 1, \ldots, T_r$ **do**
    $(\mathcal{B}_a, \mathcal{B}_p) \leftarrow \textsc{SampleBatches}(\mathcal{D}_a, \bigcup_{q=0}^{r} \mathcal{D}_p^{(q)})$
    $\{h_\beta(H)\} \leftarrow \textsc{Unroll}(F_\beta, \mathcal{B}_a \cup \mathcal{B}_p)$
    $\Theta \leftarrow \textsc{OptimizerStep}(\Theta, \nabla_\Theta(\widehat{\mathcal{L}}_{\mathrm{act}} + \lambda \widehat{\mathcal{L}}_{\mathrm{pred}}))$
  end for
end for
return (F_β, W, π_θ)