Gated Memory Policy

Yihuai Gao Jinyun Liu Shuang Li Shuran Song

Stanford University

https://gated-memory-policy.github.io

***Abstract*—Robotic manipulation tasks exhibit varying memory requirements, ranging from Markovian tasks that require no memory to non-Markovian tasks that depend on historical information spanning single or multiple interaction trials. Surprisingly, simply extending observation histories of a visuomotor policy often leads to a significant performance drop due to distribution shift and overfitting. To address these issues, we propose Gated Memory Policy (GMP), a visuomotor policy that learns both when to recall memory and what to recall. To learn when to recall memory, GMP employs a learned memory gate mechanism that selectively activates history context only when necessary, improving robustness and reactivity. To learn what to recall efficiently, GMP introduces a lightweight cross-attention module that constructs effective latent memory representations. To further enhance robustness, GMP injects diffusion noise into historical actions, mitigating sensitivity to noisy or inaccurate histories during both training and inference. On our proposed non-Markovian benchmark MemMimic, GMP achieves a 30.1% average success rate improvement over long-history baselines, while maintaining competitive performance on Markovian tasks in RoboMimic. All code, data and in-the-wild deployment instructions are available on our project website.**

I. Introduction

Robotic manipulation tasks exhibit a wide range of memory requirements. Simple pick-and-place tasks are largely Markovian [34, 30] and require minimal history. More complex tasks demand **in-trial memory**, where the policy needs to recall past actions and visual observations within a single execution episode [5, 10]. The most challenging tasks require **cross-trial memory**[1] that aggregates information across multiple interactions to infer unobservable physical properties of objects and environments [6]. For example, in Fig. 1 (c), the robot needs to infer the object surface friction from repeated casting iterations and adapt its subsequent actions in context.

Designing a policy formulation that handles all memory requirements is challenging. Naively extending observation histories enables non-Markovian behavior but often degrades performance on Markovian tasks. A longer history substantially increases the input dimensionality, raising the risk of overfitting [47]. This also exacerbates distribution shift between training and deployment. In particular, the presence of noisy or unfamiliar observations in a few history frames can shift the entire input sequence out of distribution. Additionally, attending to long histories (especially cross-trial memory) incurs significant computational overhead. Together, these issues make history-based policy designs both unreliable and inefficient.

[1]In-trial and cross-trial memory are often referred to as working and reference memory in psychology [24, 16].

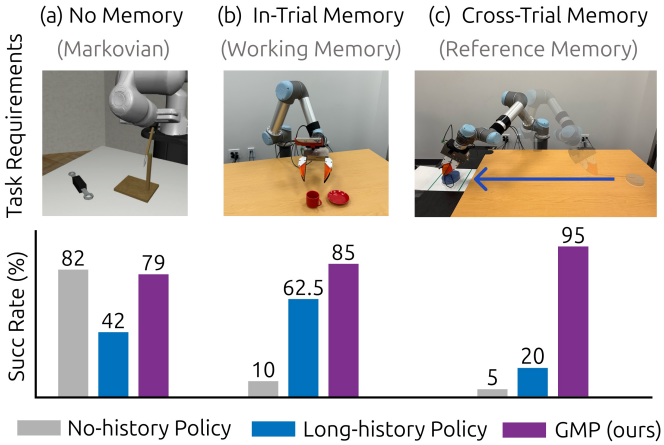


Fig. 1: **Memory Requirements** in robotics range from (a) *Markovian* tasks requiring no memory; (b) *in-trial* memory for context within a single execution; and (c) *cross-trial* memory that summarizes information across multiple attempts and adapts in context[1]. Naively increasing policy history (i.e., using a long-history policy) often degrades policy performance in Markovian tasks and is computationally expensive for cross-trial memory tasks. Our Gated Memory Policy (GMP) selectively recalls task-relevant history, achieving strong performance across tasks.

To address these challenges, an effective memory-enabled policy must answer two questions:

When should memory be recalled? Memory is unnecessary during most of the task execution, and conditioning on it may distract the policy from the current observation, slowing down inference and reducing reactivity. A robust policy should therefore ignore history in these cases.

What information should be recalled? For complex manipulation tasks, it is difficult to specify explicit rules for which information to store and retrieve. For instance, the casting task described above cannot be reduced to either visual or action history alone; instead, it requires inferring object properties by combining visual and action history from multiple interactions.

In this work, we introduce **Gated Memory Policy (GMP)**, a practical and effective framework toward memory-focused imitation learning:

To learn *When* to recall memory, GMP employs a learned *memory gate* that adaptively regulates information flow between historical context and current observations. We calibrate the memory gate by comparing action prediction errors with and without memory on a validation set. The gated policy is able to selectively recall memory, which prevents over-reliance on past observations, improves generalization, and reduces unnecessary computation.

To learn *What* to recall efficiently, we leverage a cross-attention module that learns to focus on the most relevant history without additional supervision. Additionally, compared to feeding history to a bi-directional attention network, GMP eliminates redundant computation within the history itself, reducing both memory usage and runtime cost.

To further improve the policy's *Robustness* to history noise, GMP injects scheduled noise into history actions during training. This discourages dependence on clean or perfectly accurate histories and reduces sensitivity to accumulated errors at inference time. When combined with the memory gate, this strategy enables the policy to remain stable even in the presence of noisy or unnecessary history.

To rigorously evaluate long-term memory in manipulation, we introduce **MemMimic**, a suite of simulation and real-world benchmarks that explicitly require both in-trial and cross-trial memory. We additionally evaluate GMP on RoboMimic, which consists of Markovian tasks with minimal memory requirements. Overall, GMP consistently achieves strong performance compared to alternatives, demonstrating a **30.1%** improvement on non-Markovian tasks while maintaining competitive performance on Markovian benchmarks.

In summary, our contributions are:

- Gated Memory Policy (GMP), an efficient and practical memory-focused policy framework that selectively attends to history on demand, maintaining both robustness and reactivity. GMP enables both in-trial and cross-trial memory tasks without sacrificing performance on Markovian tasks.
- MemMimic, a benchmark designed to evaluate history usage in visuomotor policies with in-the-wild data and a focus on cross-trial memory tasks that are largely overlooked by existing benchmarks.
- A comprehensive empirical study in simulation and the real world, examining memory-related design choices and their impact on performance, robustness, and computational efficiency.

II. Related Work

A. Structured Memory for Robot Policy

Prior memory-based policies have typically utilized **action parameterization or trajectory tracking**, such as storing keyframe heatmaps [14], referencing object trajectories [6, 12], or visual trace overlays [53]. Alternatively, **semantic and latent representations** can be used to store history, such as environment dynamics estimation [25], LLM-generated textual plans [33], or vector database retrieval of observation and prompt embeddings [1]. Despite their success, structured memory often requires manual design, such as predefined keypoint tracking, specific keyframe selection, or language planning syntax. This makes it inherently task-dependent and time-consuming to deploy. In contrast, our proposed method, GMP, learns memory recall in a task-agnostic space – raw image observations and robot action trajectories – ensuring seamless generalizability to new tasks.

B. Generalizable Long Context Visuomotor Policies

The role of temporal context in visuomotor policies remains a subject of active research. While RNN-based policies have been explored [34, 15, 5], recent studies indicate that shorter or even no history generally achieves better performance on Markovian tasks [7, 3, 32, 22, 23]. However, for non-Markovian tasks, including in-context learning [17] and adaptation [31, 51], long-context policies are essential.

Recent attempts to scale context length leverage efficient memory retrieval and auxiliary training objectives. Some approaches utilize external memory banks to retrieve action and image histories [41, 29], while others employ keyframe-selection [44] and attention aggregation [20] to integrate critical information into the policy input. To enhance policy robustness, noise injection is applied to history tokens during training [4, 43]. Furthermore, prediction of history actions [47] and future videos [26, 27] is also used as auxiliary tasks to enhance temporal understanding.

While these methods improve performance on non-Markovian tasks, they face significant trade-offs. Including memory only in high-level VLM planners [41, 44] often lacks fine-grained action-observation alignment, and is therefore unable to infer complex object dynamics and iteratively refine actions. Furthermore, including additional generation objectives slows down policy inference [47, 43, 26], impairing policy reactivity. Our work addresses these gaps by attending to both action and image tokens within a simple cross-attention module, achieving linear complexity in history length.

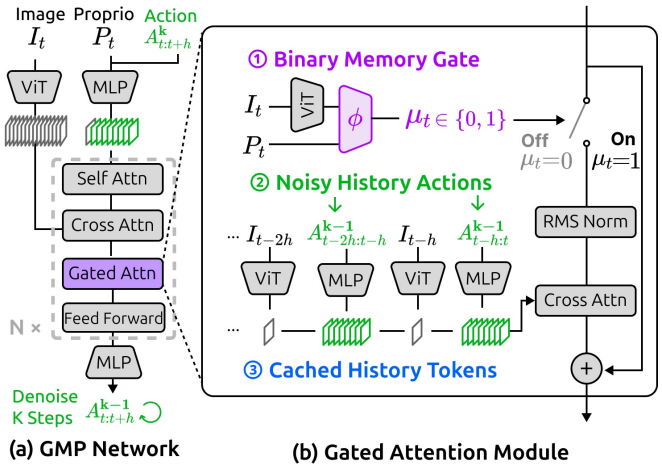
C. Gated Networks for Temporal Sequence Modeling

Gated architectures have long been a foundational approach for modeling temporal dependencies in sequential data. Classical recurrent models such as LSTM [19] and GRU [9] introduce gating mechanisms to regulate information flow across time, enabling the retention or forgetting of past states. Extensions to the Transformer paradigm incorporate similar principles: GTrXL [37] augments self-attention with gating and recurrence to stabilize long-horizon reinforcement learning, while recent state-space and gated sequence models such as Mamba [18, 11] achieve linear-time sequence modeling by selectively propagating information through learned gates. More recently, gated attention mechanisms [39] have been proposed to modulate attention weights dynamically, improving efficiency and robustness on long-context tasks. Our work incorporates such a gating mechanism to trigger memory retrieval only on demand, improving robustness and reactivity.

III. Method

A. Background: Transformer-based Diffusion Policy

We adopt the Transformer-based Diffusion Policy [7, 32] as our network backbone. Given the current image observation I_t and robot proprioception P_t, the policy predicts a trajectory of future actions $A_{t:t+h} = \{A_t, A_{t+1}, \cdots, A_{t+h-1}\}$, where h is the action prediction horizon. During training, Gaussian noise $\epsilon \sim \mathcal{N}(0, 1)$ is progressively added to the ground-truth



(a) GMP Network (b) Gated Attention Module

Fig. 2: **Gated Memory Policy Network (a)** Based on Diffusion Transformer (DiT) [38, 32], we add a gated attention module to selectively recall memory. **(b)** The gated attention module features three key designs: (1) Binary memory gate μ_t that determines whether history cross-attention is skipped or applied. (2) Noised history action condition to improve robustness and reduce overfitting. (3) Cached history tokens during inference time to reduce computational cost.

action $A^0_{t:t+h}$. We train a denoising network φ_θ to predict the ground-truth action and calculate the action prediction loss as

$$\mathcal{L}_{\text{action}} = \mathbb{E}_{A^0_{t:t+h},\epsilon,k}\left[\|A^0_{t:t+h} - \varphi_\theta(A^k_{t:t+h}, I_t, P_t, k)\|_2^2\right], \tag{1}$$

where k is the diffusion step. At test time, actions are denoised for K steps using the DDIM [42] noise scheduler.

While Diffusion Policy performs well on Markovian tasks, its performance degrades when history-dependent capabilities are required, such as history-aware planning or in-context adaptation. This is because standard diffusion policies condition only on the most recent one or two observations, making them **incapable of leveraging extended history**.

A straightforward way to incorporate longer history is to extend the observation-action window, that is, conditioning on $I_{t-H:t}$ and $P_{t-H:t}$, where H is the history length. However, this approach introduces two significant drawbacks: **(1) Overfitting.** As H increases, the input space expands while the training data remains fixed, making it harder for the model to generalize. Consequently, the model overfits more easily and performs poorly on long-history sequences at inference time. **(2) High computational cost.** In Transformer-based architectures, self-attention over H historical tokens scales quadratically as $\mathcal{O}(H^2)$, making both training and inference increasingly expensive as H grows.

B. Gated Memory Policy

We aim to develop a simple and effective method that enables long-horizon conditioning without overfitting or incurring high computational cost. We use an MLP to encode a set of n historical action trajectories $\{A_{t-nh:t-(n-1)h}, \cdots, A_{t-h:t}\}$, where each chunk consists of a sequence of h actions. Since encoding images requires significantly more computation and high-frequency frames are often redundant, we sample only one image observation per

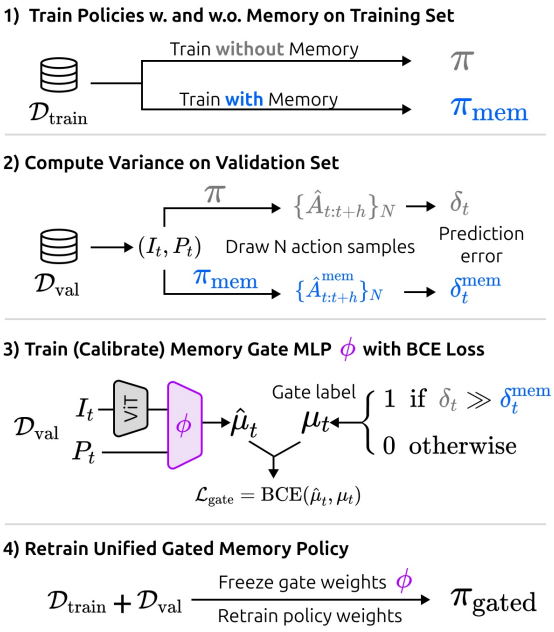


Fig. 3: **Training Procedure with Memory Gate Calibration.** We split the dataset into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$. **1)** Train two policies on $\mathcal{D}_{\text{train}}$ with memory gate always off π or always on π_{mem}. **2)** Evaluate the two policies on $\mathcal{D}_{\text{val}}$ for N rounds and calculate the error of the predicted actions at each timestep t. **3)** At timestep t, if the error of the no-memory policy δ_t is significantly larger than the error of the memory policy δ_t^{mem}, we label *memory-required* ($\mu_t = 1$) for this timestep; otherwise, $\mu_t = 0$. This stage is referred to as **Calibration** of the memory gate. **4)** We freeze the memory gate weights and retrain the policy on the full dataset to obtain the final Gated Memory Policy π_{gated}.

trajectory, yielding $I_{t-nh:h:t} = \{I_{t-nh}, I_{t-(n-1)h}, \cdots, I_{t-h}\}$. We use the pretrained ViT encoder [13, 48] to extract visual features from each image, followed by multi-head attention pooling (MAP) to aggregate all image patches into a single token, substantially reducing the number of visual tokens.

Cross-Attention with Cached Tokens for History Conditioning. Directly concatenating all historical action and visual tokens into a single long sequence and processing them with a Transformer is computationally expensive. We therefore introduce a cross-attention module specifically for history conditioning, as shown in Fig. 2 (b).

Inspired by the KV cache in causal Transformers [50, 36], we cache a sliding window of n history trajectories, where each trajectory consists of one aggregated image feature and h action tokens. GMP can therefore retrieve history tokens without additional computation.

Diffusion Noise Augmentation to Reduce Overfitting. Directly using clean history for training often leads to overfitting and poor generalization at test time [4], due to the expansion of the input space. To mitigate this, we propose to use augmented history actions, obtained by injecting diffusion-scheduled noise into past actions, as a history condition. This augmentation exposes the model to a significantly larger and more diverse input space than clean actions alone, improving robustness and generalization.

As shown in Fig. 2 (b), at diffusion step k, the model

predicts future actions $A^k_{t:t+h}$ at noise level k, conditioned on history actions $A^{k-1}_{t-nh:t}$ at noise level $k-1$, one step cleaner than the future actions. This creates a schedule where the noise level decreases as denoising progresses. In the early diffusion steps ($k = K, K-1, \ldots$), we add strong noise to history actions, providing substantial augmentation to prevent overfitting. In the final steps ($k = \ldots, 2, 1$), only minimal noise is added, allowing the policy to capture fine-grained information from the history. Compared to Diffusion Forcing [43], which uses random noise levels during training but no noise during inference, our method applies noise consistently during both training and testing, leading to more robust performance.

Memory Gating for On-Demand Memory Recall. In many manipulation tasks, a long history is informative only at occasional but critical moments. Always injecting history can introduce noise and encourage over-reliance on context that is often unnecessary. We therefore learn a *memory gate* that adaptively modulates the contribution of history at each deployment step, enabling the policy to perform well in both Markovian and non-Markovian settings.

Given the current image observation I_t and robot proprioception P_t, we use an MLP ϕ with a sigmoid activation σ to produce a binary gate value μ_t:

$$\mu_t = \mathbf{1}\{\sigma\left(\phi(I_t, P_t)\right) > 0.5\} \in \{0, 1\}. \tag{2}$$

As shown in Fig. 2(b), the gate μ_t is applied to the history cross-attention output $\mathbf{h}_{t:t+h} \in \mathbb{R}^{h\times d}$ before adding it to the residual connection $\mathbf{z}_{t:t+h} \in \mathbb{R}^{h\times d}$, where h is the prediction horizon and d is the DiT hidden dimension. Finally, $\bar{\mathbf{z}}_{t:t+h} = \mu_t \mathbf{h}_{t:t+h} + \mathbf{z}_{t:t+h}$ is passed to the feed-forward layer.

Memory Gate Calibration. The memory gate is challenging to train in an end-to-end manner: without regularization, the policy tends to use as much history as possible and overfit to the training data, leading to poor performance on Markovian tasks. Although we can apply a regularization term on μ to discourage unnecessary history use, it is difficult to find a weight that works well across all tasks. A large weight suppresses history usage and hurts performance on non-Markovian tasks, while a small weight fails to provide sufficient regularization, as shown in Fig. 13.

We therefore adopt a self-supervised calibration process to generate labels for the memory gate, as illustrated in Fig. 3. Specifically, we split the dataset into **two halves**, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$. We train two policies on the **training set** $\mathcal{D}_{\text{train}}$, one with the memory gate always off (π) and one with it always on (π_{mem}). We then draw N samples from the two policies on the **validation set** $\mathcal{D}_{\text{val}}$ and compute the action prediction error at each timestep t. We introduce a ratio threshold θ to determine memory dependency at timestep t: if the error of the no-memory policy δ_t is at least θ times larger than the error of the memory policy δ^{mem}_t, we label this timestep as "memory-required" ($\mu_t = 1$), otherwise, $\mu_t = 0$. We use Binary Cross Entropy (BCE) loss to calibrate the memory gate separately from the policy training.

Finally, we freeze the memory gate weights and retrain the policy on the full dataset ($\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$) to obtain the final Gated Memory Policy π_{gated}. This ensures that the policy uses history information only when necessary during both training and testing.

IV. Experiments

A. Benchmark Tasks

We propose MemMimic, a series of simulation and real-world non-Markovian tasks to evaluate a policy's memory capabilities, spanning both in-trial **(T1-3)** and cross-trial **(T4-6)** memory. We evaluate GMP on the Markovian task benchmark RoboMimic **(T7-9)** to ensure policy robustness. We also evaluate GMP on the existing memory-intensive benchmarks to compare with Vision-Language-Action (VLA) baselines [5, 41, 22, 35, 40] **(T10-14)**. For all of the simulation tasks, we train the model using 3 seeds and report the mean and standard deviation of the best checkpoints in each run.

MemMimic In-trial Memory Tasks: Evaluate the ability to retrieve relevant history within a single execution attempt.

- T1: `Match Color (Sim)`. Shown in Fig. 4, the robot picks up a cube from one of 4 bins with randomly assigned colors. Once the cube is picked up, the bin colors are shuffled, and the robot needs to put the cube back in the bin with the original color. This task tests the policy's **visual memory**.
- T1': `Match Color with Random Delay (Sim)`. Based on T1, we add a 5–600 second random delay after the initial colors disappear before the final colors appear. The robot should hold the cube in the air while waiting for the final colors to appear. This task tests the policy's **memory length**. Please refer to **Finding 7** for evaluation results.
- T2: `Discrete Place Back (Sim)`. Shown in Fig. 5, a cube is randomly placed in one of four bins, and the robot picks it up, holds it in the air for 2 seconds before returning it to the original bin. This task tests the policy's **spatial memory**.
- T3: `Continuous Place Back (Real)`. Shown in Fig. 6, a cup and a saucer are randomly placed on the table. The robot must first place the cup on the saucer and then return it to within 5 cm of its original position. This task tests the policy's **spatial memory** and **robustness to real-world noise**.
- T3': In-the-wild `Flip and Place Back (Real)`. Shown in Fig. 7, after the robot picks up the cup and places it on the saucer, a human flips it over by 90 degrees. The robot must flip it back and then return it to the original position. This task tests the policy's **spatial memory** and **generalizability across unseen environments and diverse objects**.

MemMimic Cross-trial Memory Tasks: Evaluate whether the policy can perform in-context adaptation – learn from past interactions and refine its actions through a trial-and-error process. Each episode fixes the object's property and

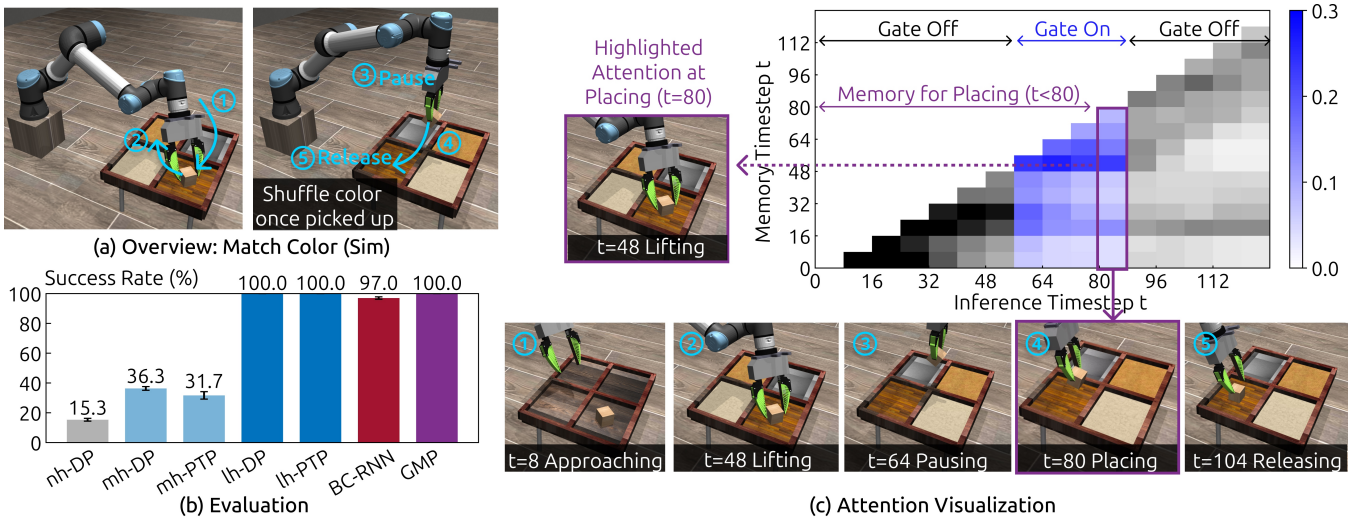


Fig. 4: **Task 1: Match Color (Sim)**. **(a)** The robot picks up a cube while observing four randomly colored bins. After lifting the cube, the bin colors are randomized; the robot must place the cube into the bin that matches the cube's original color. **(b) Baselines:** Diffusion Policy (DP) [7] and Past-Token-Prediction (PTP) [47] are evaluated across no, medium, and long-history [nh, mh, lh] settings. Our method, GMP, uses the [lh] configuration. **(c) Visualization:** We visualize the gated cross-attention weights for each 8-step action chunk. At $t = 80$, when the robot is placing the cube back, the attention focuses primarily on $t = 48$, when the robot first observed the bin colors. Blue indicates the memory gate is on ($\mu_t = 1$); gray indicates the gate is off ($\mu_t = 0$).

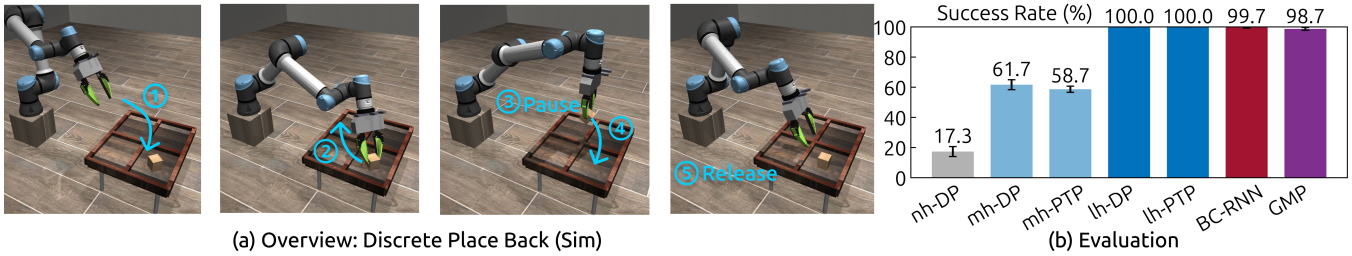


Fig. 5: **Task 2: Discrete Place Back (Sim)**. **(a)** The cube is randomly placed in one of the 4 bins. The robot must pick up the cube, hold it in the air for 2 seconds (creating a memory challenge), and return it to the original bin.

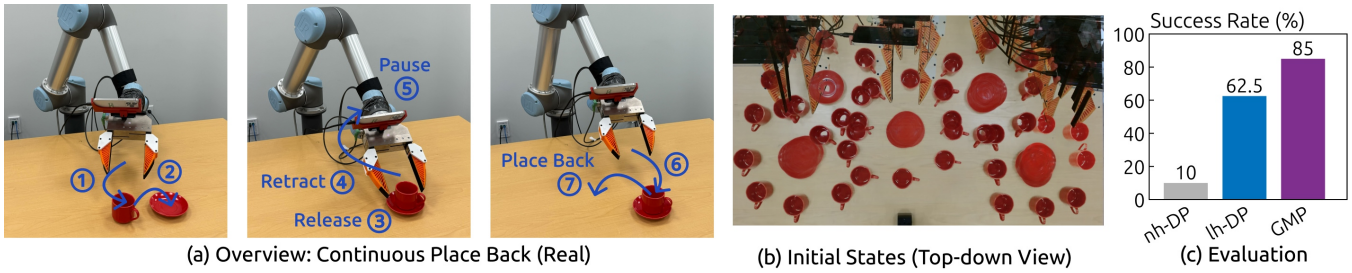
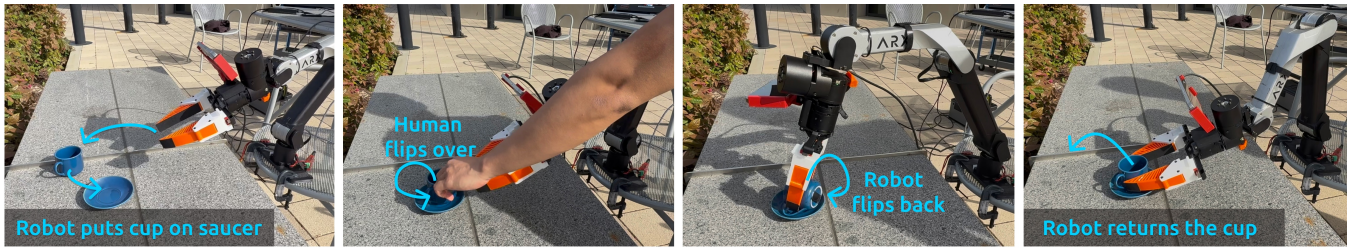


Fig. 6: **Task 3: Continuous Place Back (Real)**. **(a)** The robot picks up the cup, places it on the saucer, picks it up again, and returns it to the original position. **(b)** Initial positions of the cup and saucer tested across the continuous workspace.

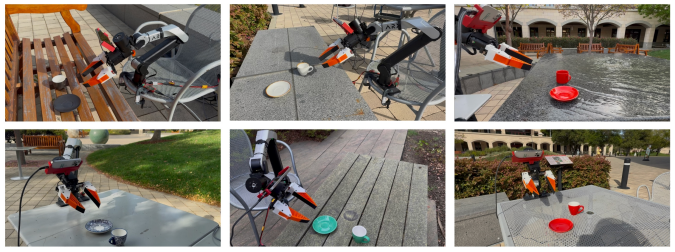
allows the policy to attempt N trials. An episode is considered successful if the last M trials ($M < N$) are correct.

- T4: `Iterative Pushing (Sim)`. Shown in Fig. 8, in each episode, the friction coefficient of the cube is randomly sampled from 0.005 to 0.015 and is unknown during testing. The policy is given 6 trials to push the object to the target region (red box). An episode is considered successful if the cube stops within the target region in all of the last 3 trials.
- T5: `Iterative Flinging (Sim)`. Shown in Fig. 9, the robot must fling the cloth so that its far edge lands in the target (black) area. In each episode, the cloth's mass is randomly sampled between 0.1 kg and 2.0 kg, and the robot is allowed five flings. Flinging too slowly prevents the cloth from fully extending, while flinging too fast causes it to fold back on itself—both resulting in failure. An episode is considered successful if the last 3 trials are all successful.
- T6: `Iterative Casting (Real)`. Shown in Fig. 10, the robot needs to cast the object [28] with an unknown friction coefficient so that the object stops sliding between two green lines. The robot stops at the same position in each trial and adaptively adjusts its casting speed. In each episode, the robot is allowed 3 casting attempts. The episode is considered successful if the last 2 trials are successful.

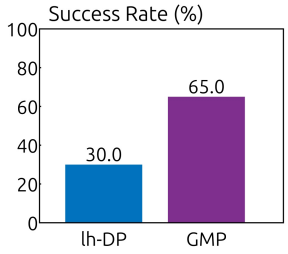


(a) Overview: In-the-wild Flip and Place Back (Real)

(b) In-the-wild Evaluation Setup

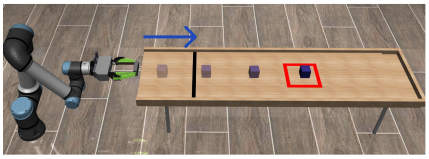


(c) Diverse Environments and Test Objects

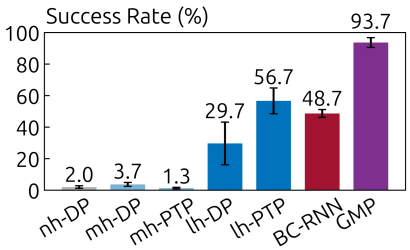


(d) Evaluation

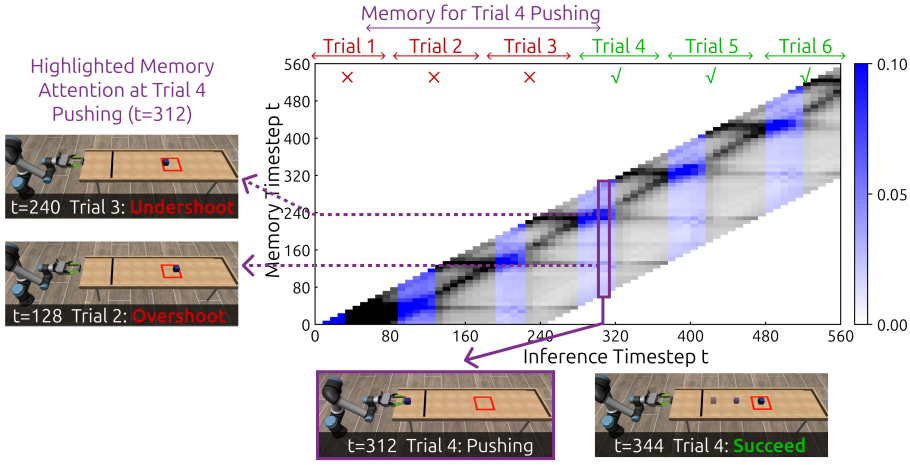
Fig. 7: **Task 3': In-the-wild Flip and Place Back (Real)**. **(a)** After the robot picks up the cup and places it on the saucer, a human flips it over by 90 degrees. The robot must flip it back then return it to the original position. **(b)** We use an ARX X5 robot arm with an iPhone for visual observation. **(c)** We evaluate our policy on 6 challenging unseen environments and over 10 cups, showing robustness and generalizability across objects and scenes.



(a) Single Trial Description



(b) Evaluation



(c) Attention Visualization for Multi-Trial Process

Fig. 8: **Task 4: Iterative Pushing (Sim)**. **(a)** The robot pushes a cube (with unknown friction) into the red box over 6 trials per episode. The environment resets after each trial. By observing how far the cube moves, the policy learns the physical dynamics and adjusts the pushing velocity accordingly for subsequent trials. **(b) Baselines:** Diffusion Policy (DP) [7] and Past-Token-Prediction (PTP) [47] are evaluated across no, medium, and long-history [nh, mh, lh] settings. Our method, GMP, uses the [lh] configuration. **(c)** Visualization of gated cross-attention weights for each 8-step action trajectory throughout one episode. Blue indicates the memory gate is on ($\mu_t = 1$) and gray indicates the memory gate is off ($\mu_t = 0$). During the pushing stage in Trial 4, attention focuses primarily on the outcomes from Trial 3 (undershoot) and Trial 2 (overshoot). The policy uses these past outcomes to adjust its action and achieve a successful push.

Markovian Tasks: We evaluate the proposed method on tasks that do not require history information, verifying that memory mechanisms do not harm performance on standard manipulation tasks.

- T7-9: **RoboMimic (Sim).** We evaluate three tasks from RoboMimic [34]: `Square`, `Transport`, and `Tool Hang`. Policies are trained for 400 epochs, with checkpoints saved every 20 epochs. Each checkpoint is evaluated on 100 episodes, and the best-performing checkpoint is used to report the final success rate.

Other In-trial Memory Benchmarks: We evaluate our method on other existing memory-intensive benchmarks to compare with VLA baselines [5, 41], shown in Fig. 12.

- T10-14: **MIKASA-Robo [5] (Sim).** We evaluate 5 tasks: `ShellGameTouch`, `InterceptMedium`, `RememberColor3`, `RememberColor5`, and `RememberColor9`. To match the training protocol in prior work, we use 250 training episodes per task generated by an oracle state-based PPO policy with image resolution 128×128. Policies are trained for 500 epochs, with checkpoints saved every 20 epochs. Each checkpoint is evaluated on 100 episodes, and the best-performing

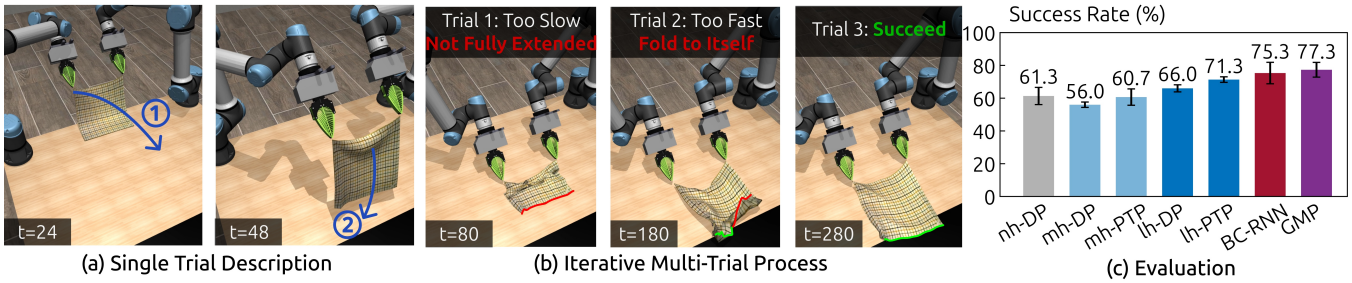


Fig. 9: **Task 5: Iterative Flinging (Sim)**. **(a)** The robot flings a cloth (with unknown mass) so that the far edge lands on the black target area. **(b)** Multi-attempt flinging process. Flinging too slowly leaves the cloth not fully extended; flinging too fast causes the cloth to fold back on itself. The policy learns to adjust the flinging velocity to match the unknown mass of the cloth.

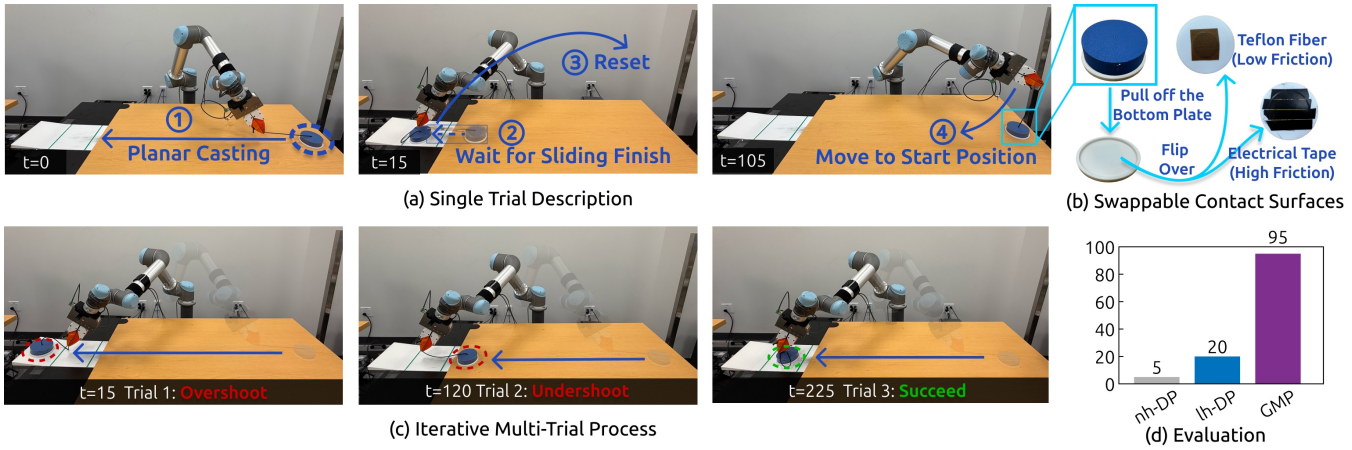


Fig. 10: **Task 6: Iterative Casting (Real)**. **(a)** The robot casts the object (with unknown friction) so that it stops between the two lines. After the object stops, the robot moves back for the next trial. **(b)** Two object contact surfaces with different friction coefficients are used across episodes, requiring the policy to infer object dynamics and apply different casting velocities.

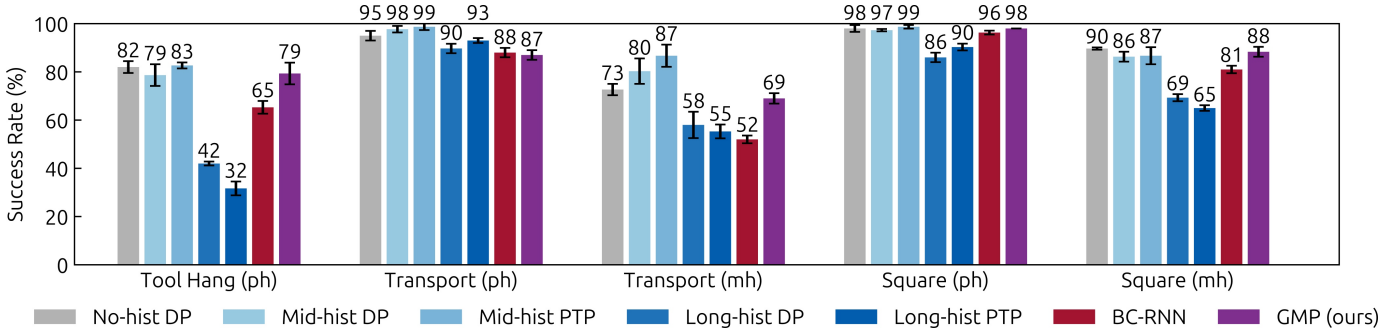


Fig. 11: **Evaluation results on RoboMimic.** We evaluate 3 tasks from the RoboMimic benchmarks [34]: *Tool Hang, Square, and Transport*. (ph, mh) indicate that the data were collected from proficient-human (ph) or multi-human (mh). While most long-history policies experience performance drops on these Markovian tasks, GMP maintains competitive performance by leveraging the gating mechanism.

checkpoint is used to report the final success rate. Please refer to Supplementary Material VIII-C for more details.

B. Algorithm Comparisons

We compare GMP with six baselines. For a fair comparison, all the models in our experiments use SigLIP2-B/16 [48] with pre-trained weights as the image encoder.

- **No-hist DP**: A standard diffusion policy with a Transformer backbone that includes only the most recent image I_t and robot proprioception P_t as input.
- **Mid-hist DP**: To evaluate the performance of DP with longer history, we extend the history input length from the original 1 step to 16 steps: using proprioception $P_{t-16:t}$ and images $I_{t-16:t}$ as input.
- **Mid-hist PTP [47]**: PTP is a recent method enabling long-history policy inference. Based on [Mid-hist DP], it predicts 16 past actions $A_{t-16:t}$ together with 16 future actions $A_{t:t+16}$. This configuration matches the longest context length in the original PTP implementation.
- **Long-hist DP**: We further extend the DP history length to

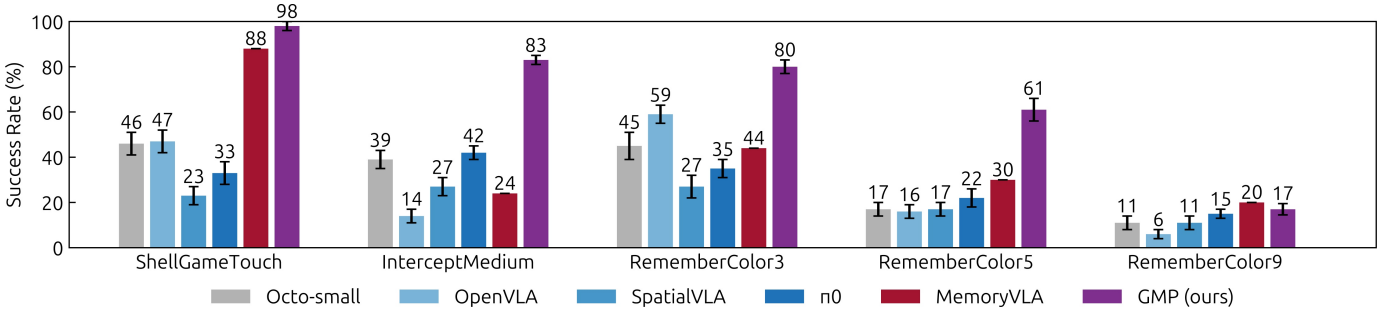


Fig. 12: **Evaluation results on MIKASA-Robo.** We evaluate 5 tasks on the MIKASA-Robo benchmarks [5], out-performing prior work MemoryVLA [41] by 26.6% on average. The baseline performance statistics are reported in [5] and [41].

120 steps. Due to the GPU memory limit during training, we sample every 8th image: using proprioception $P_{t-120:t}$ and images $I_{t-120:8:t}$ as input.

- **Long-hist PTP**: Based on [Long-hist DP], it predicts 120 past actions $A_{t-120:t}$ together with 16 future actions $A_{t:t+16}$. This history length is 7× longer than the longest history length reported in [47].
- **BC-RNN [34]**: An LSTM-based RNN policy, predicting an 8-step action chunk $A_{t:t+8}$ at every inference step. We do not reset the RNN states during training and rollout, thus the history length is unlimited.
- **GMP (Ours)**: We match the history configuration in [Long-hist DP], using 120 history actions $A_{t-120:t}$ and the corresponding 15 images $I_{t-120:8:t}$ as the cross-attention input.

For iterative tasks **T4-6**, we further extend the history length of GMP and the long-context baselines. See the Supplementary Materials for details.

C. Results and Findings

Finding 1: Naive history extension degrades performance. We observe that simply increasing the history length of robot policies degrades performance on Markovian tasks. As shown in Fig. 11, [Long-hist DP] (120 history steps) performs significantly worse than [Mid-hist DP] (16 history steps) and [No-hist DP]. This degradation persists even for methods explicitly designed to leverage long histories, such as [Long-hist PTP], which still struggles to effectively filter irrelevant or redundant history information to overcome the overfitting issue.

For non-Markovian tasks, while naive long-history policies perform well on simple tasks such as `Match Color` and `Discrete Place Back` (Fig. 4, 5), they are insufficient for more complex real-world tasks (Fig. 6, 10) or cross-trial memory tasks (Fig. 8, 9, 10). In contrast, our method selectively attends to task-relevant history and introduces diffusion-noise–augmented history actions during training, making it more robust to complex environments.

Finding 2: History cross-attention selectively attends to the right modality at the right time. To analyze how the policy uses history information, we visualize cross-attention weights from the gated cross-attention module at critical decision points (Fig. 4, 8).

For the in-trial memory task `Match Color` (Fig. 4), at timestep $t = 80$, when the robot places the cube back, the highest attention is assigned to timestep $t = 48$, when the colors were first observed. This shows that the model automatically identifies and attends to key frames without explicit supervision.

For cross-trial memory tasks such as `Iterative Pushing` (Fig. 8), attention in Trial 4 focuses on the outcomes of Trial 2 and Trial 3, corresponding to overshoot and undershoot behaviors, respectively. By attending to these past outcomes, the policy adapts its actions to achieve successful pushes in subsequent trials.

Finding 3: Memory gate helps skip memory recall when unnecessary for Markovian and non-Markovian tasks. When calibrating the memory gate (Fig. 3) for Markovian tasks, the action prediction error δ_t^{mem} of the policy with memory π_{mem} is similar to δ_t of the policy without memory π; thus, the memory gate is almost always off ($\mu = 0$) and the policy ignores history information, leading to competitive performance against the no-history policy. Results on the RoboMimic [34] benchmark are shown in Fig. 11.

Moreover, even for tasks that require history information, the memory gate is off ($\mu = 0$) most of the time (for example, 73% in `Match Color` and 58% in `Iterative Pushing`) and opens only when necessary, further speeding up policy inference. See Finding 5 for speed comparisons.

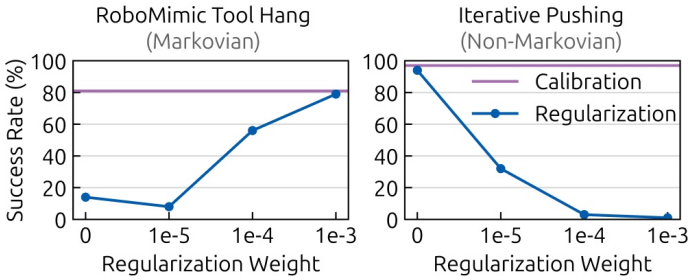


Fig. 13: **Calibration of Binary Memory Gate**. When training a binary memory gate jointly with the policy, no regularization (encourages the gate to stay on) leads to poor performance on Markovian tasks, while high regularization (encourages the gate to be off) hurts performance on non-Markovian tasks. Our method calibrates the binary memory gate independently and freezes it during policy training, achieving strong performance on both Markovian and non-Markovian tasks.

Finding 4: Calibration on validation set is crucial for mem-

ory gate training. An alternative approach to training a binary gate together with the policy model is to use straight-through estimators (STE) [2, 49]. Specifically, this involves using the binarized value in the forward pass while treating it as a continuous value in backpropagation. During training, since overfitting to more history generally reduces training error, the memory gate value tends to grow toward 1. This undermines policy generalizability and robustness on Markovian tasks. To mitigate this, one approach is to add a regularization loss to the memory gate value to suppress overuse. However, the gate value tends to drop to 0 if the regularization weight is large, impairing the performance of non-Markovian tasks, as shown in Fig. 13.

In contrast, calibrating the memory gate independently on a validation set avoids the dilemma mentioned above. By comparing the action prediction error with and without memory on validation samples, we can identify the critical moments that truly necessitate historical information, instead of overusing or underusing it.

Finding 5: GMP maintains low computational cost by leveraging the cross-attention and gating mechanisms. We run policy inference for 100 rounds on an NVIDIA RTX 3080 GPU and report the average inference time with standard deviation, as shown in Fig. 14. The computational cost of the self-attention DiT baseline scales quadratically with the context length, thus taking significantly longer as the history length increases. To reduce the inference time, GMP employs two techniques: 1) moving all the history tokens into a separate cross-attention module, making the computational cost scale linearly with the history length (see [GMP (Gate On)]); 2) skipping the history attention when the memory gate is off, reducing inference time to a short constant when memory is not needed (see [GMP (Gate Off)]).

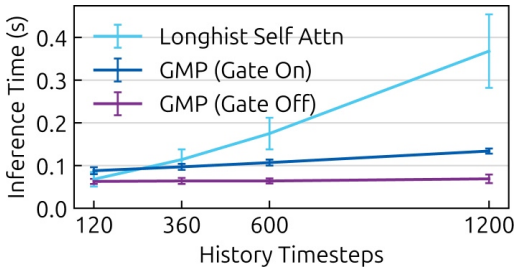


Fig. 14: **Inference Time Comparison**. The self-attention baseline's inference time increases significantly with the number of history timesteps. In contrast, GMP uses cross-attention, so the computational cost grows linearly when the memory gate is on. When the memory gate is off, GMP skips all history attention, keeping inference time constant and minimal.

Finding 6: The added diffusion noise in history actions improves policy robustness. Our proposed noise injection strategy [Diffusion Noising] uses the one-step-cleaner history actions $A^{k-1}_{t-nh:t}$ at diffusion step k both during training and testing. We compare our method with the following baselines: Using [No Noise] in both training and testing makes the policy overly reliant on clean history actions, and thus not robust to distribution shifts during evaluation; adding [Random Level] noise in both training and testing prevents the policy from accessing clean history actions and might result in losing critical information; [Diffusion Forcing] [4] injects random noise levels during training but no noise during testing, leading to inconsistency between training and testing. The results on `Iterative Pushing` are shown in Fig. 15.

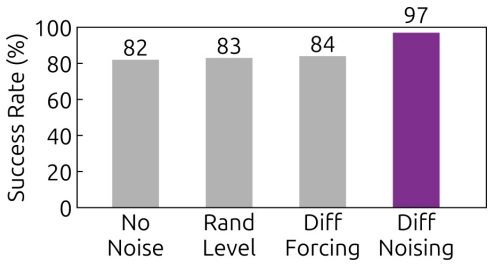


Fig. 15: **Noise Injection Ablation**. Our noise injection strategy [Diffusion Noising] uses the one-step-cleaner history actions $A^{k-1}_{t-nh:t}$ at diffusion step k during both training and testing. This added noise helps achieve better robustness than alternatives on the `Iterative Pushing` task.

Finding 7: Cross-attention is scalable and efficient towards long history length. Although the context window is not infinite, cross-attention is able to precisely attend to the most relevant observations and actions without additional supervision. In task **T1'** `Match Color with Random Delay`, after the initial color disappears, the environment randomly waits 5–600 seconds (10 fps) before the final color appears. To save training GPU memory, we follow [47] and freeze the vision encoder from the original `Match Color` task and cache all of the image features in the training dataset. After training on 500 episodes, we achieve a **99.0%±1.0%** success rate with a memory buffer of **6000 image frames and 6000 actions**, and the policy inference only takes **0.16s** for 8 denoising steps on a 5090 GPU. We believe this history length suffices for a visuomotor policy to complete short-term memory tasks. For long-term memory, a higher-level VLM can handle this more efficiently via text-based representations.

V. Limitations and Future Work

While we show that GMP can effectively leverage history information through the cross-attention mechanism, the current implementation is still limited to a finite attention window. Consequently, the model cannot leverage an "infinite" memory context and hence may lose access to critical information over extended sequences. Future work could investigate selective caching and token replacement strategies based on importance. By dynamically managing the history buffer, the model can attend to high-value tokens from the distant past even within a fixed attention budget.

Moreover, while we demonstrate that action prediction error serves as a useful proxy for memory requirements, this metric may be unreliable for tasks with intrinsic ambiguity or in environments characterized by high uncertainty. Future research could incorporate additional modalities into the memory retrieval criteria, for example, task semantics.

VI. Conclusion

We present Gated Memory Policy, a visuomotor policy that learns to selectively recall memory on demand. By introducing a cross-attention module for history tokens, calibrating a binary memory gate, and injecting diffusion noise into history actions, GMP can retrieve task-relevant information, perform in-context adaptation, remain robust to long history lengths, and maintain low computational cost. Our method outperforms other long-history baselines on the non-Markovian task suite MemMimic in both simulation and real-world settings, while avoiding performance degradation on Markovian tasks.

VII. Acknowledgments

We thank Austin Patel for the great iPhUMI app that enables in-the-wild data collection and deployment. The authors thank Chiling Han for the assistance with the baseline experiment. We thank Mengda Xu and Huy Ha for MuJoCo simulation instructions. We thank all REALab members for their valuable suggestions on manuscript writing and presentations. We also thank Yuejiang Liu, Moo Jin Kim, John Yao, Shang Yang, Genghan Zhang, and Dongchen Han for valuable discussions and feedback.

This work was supported in part by the NSF Award #2143601, #2037101, and #2132519. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors. We would like to thank TRI for providing the UR5 robot hardware and ARX for the X5 robot hardware. We thank Apple for providing the iPhone 15 Pro. We also thank Stanford Marlowe [21] for providing computational resources.

References

[1] Abrar Anwar, John Welsh, Joydeep Biswas, Soha Pouya, and Yan Chang. Remembr: Building and reasoning over long-horizon spatio-temporal memory for robot navigation, 2024. URL https://arxiv.org/abs/2409.13682.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

[4] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2025.

[5] Egor Cherepanov, Nikita Kachaev, Alexey K. Kovalev, and Aleksandr I. Panov. Memory, benchmark & robots: A benchmark for solving complex tasks with reinforcement learning, 2025. URL https://arxiv.org/abs/2502.10550.

[6] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy for goal-conditioned dynamic manipulation of deformable objects. In *Proceedings of Robotics: Science and Systems (RSS)*, 2022.

[7] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[8] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL https://arxiv.org/abs/1412.3555.

[10] Nhat Chung, Taisei Hanyu, Toan Nguyen, Huy Le, Frederick Bumgarner, Duy Minh Ho Nguyen, Khoa Vo, Kashu Yamazaki, Chase Rainwater, Tung Kieu, Anh Nguyen, and Ngan Le. Rethinking progression of memory state in robotic manipulation: An object-centric perspective, 2025. URL https://arxiv.org/abs/2511.11478.

[11] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

[12] Mark Van der Merwe and Devesh Jha. In-context iterative policy improvement for dynamic manipulation, 2025. URL https://arxiv.org/abs/2508.15021.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

[14] Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation, 2025. URL https://arxiv.org/abs/2501.18564.

[15] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[16] Karyn M Frick, Mark G Baxter, Alicja L Markowska, David S Olton, and Donald L Price. Age-related spatial reference and working memory deficits assessed in the

water maze. *Neurobiology of aging*, 16(2):149–160, 1995.

[17] Letian Fu, Huang Huang, Gaurav Datta, Lawrence Yunliang Chen, William Chung-Ho Panitch, Fangchen Liu, Hui Li, and Ken Goldberg. In-context imitation learning via next-token prediction. *arXiv preprint arXiv:2408.15980*, 2024.

[18] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[20] Qingda Hu, Ziheng Qiu, Zijun Xu, Kaizhao Zhang, Xizhou Bu, Zuolei Sun, Bo Zhang, Jieru Zhao, Zhongxue Gan, and Wenchao Ding. Resolving state ambiguity in robot manipulation via adaptive working memory recoding. *arXiv preprint arXiv:2512.24638*, 2025.

[21] Craig Kapfer, Kurt Stine, Balasubramanian Narasimhan, Christopher Mentzel, and Emmanuel Candes. Marlowe: Stanford's gpu-based computational instrument, January 2025. URL https://doi.org/10.5281/zenodo.14751899.

[22] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

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

[24] Ann-Katrin Kraeuter, Paul C. Guest, and Zoltán Sarnyai. *The Y-Maze for Assessment of Spatial Working and Reference Memory in Mice*, pages 105–111. Springer New York, New York, NY, 2019. ISBN 978-1-4939-8994-2. doi: 10.1007/978-1-4939-8994-2_10. URL https://doi.org/10.1007/978-1-4939-8994-2_10.

[25] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. 2021.

[26] Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control, 2026. URL https://arxiv.org/abs/2601.21998.

[27] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. In *Proceedings of Robotics: Science and Systems*, 2025.

[28] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *2022 International Conference on Robotics and Automation (ICRA)*, page 8282–8289. IEEE Press, 2022. doi: 10.1109/ICRA46639.2022.9811651. URL https://doi.org/10.1109/ICRA46639.2022.9811651.

[29] Min Lin, Xiwen Liang, Bingqian Lin, Liu Jingzhi, Zijian Jiao, Kehan Li, Yuhan Ma, Yuecheng Liu, Shen Zhao, Yuzheng Zhuang, and Xiaodan Liang. Echovla: Robotic vision-language-action model with synergistic declarative memory for mobile manipulation, 2025. URL https://arxiv.org/abs/2511.18112.

[30] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[31] Min Liu, Deepak Pathak, and Ananye Agarwal. Locoformer: Generalist locomotion via long-context adaptation. In *9th Annual Conference on Robot Learning*, 2025.

[32] 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. *arXiv preprint arXiv:2410.07864*, 2024.

[33] Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023.

[34] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[35] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

[36] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[37] Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning, 2019. URL https://arxiv.org/abs/1910.06764.

[38] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

[39] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free, 2025. URL https://arxiv.

org/abs/2505.06708.

[40] 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.

[41] 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, 2025. URL https://arxiv.org/abs/2508.19236.

[42] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. URL https://arxiv.org/abs/2010.02502.

[43] Kiwhan Song, Boyuan Chen, Max Simchowitz, Yilun Du, Russ Tedrake, and Vincent Sitzmann. History-guided video diffusion, 2025. URL https://arxiv.org/abs/2502.06764.

[44] Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Memer: Scaling up memory for robot control via experience retrieval, 2025. URL https://arxiv.org/abs/2510.20328.

[45] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012. doi: 10.1109/IROS.2012.6386025.

[46] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[47] Marcel Torne, Andy Tang, Yuejiang Liu, and Chelsea Finn. Learning long-context diffusion policies via past-token prediction. *arXiv preprint arXiv:2505.09561*, 2025.

[48] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features, 2025. URL https://arxiv.org/abs/2502.14786.

[49] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6309–6318, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[50] 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.

[51] Andrew Wagenmaker, Zhiyuan Zhou, and Sergey Levine. Behavioral exploration: Learning to explore via in-context adaptation, 2025. URL https://arxiv.org/abs/2507.09041.

[52] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

[53] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*, 2024.

VIII. Supplementary Materials

A. Overlapped Trajectory Training

Training a long-history policy jointly with a vision encoder introduces significant GPU memory overhead. Prior work [47] addresses this issue by first training the vision encoder with a short-history policy, then freezing the encoder while training the long-history policy. However, this prevents the vision encoder from adapting to long-term dependencies.

Inspired by the parallel training of autoregressive causal transformers [50], we address this challenge by introducing *Overlapped Trajectory Training*. Let an episode be a sequence of observations and actions $\mathcal{E} = \{(I_1, A_1), (I_2, A_2), \ldots, (I_T, A_T)\}$. Instead of sampling trajectories independently, we construct a trajectory batch as a collection of overlapped subsequences of *history length* H:

$$\tau_s = \{(I_s, A_s), \ldots, (I_{s+H-1}, A_{s+H-1})\}$$

where $s = 1, \ldots, T - H + 1$. Consecutive subsequences share most of their observations; e.g., τ_s and τ_{s+1} overlap in $H - 1$ steps. This allows each image I_t to be encoded by the vision encoder only once per forward pass through a mini-batch, with the resulting feature reused across all subsequences in the batch that include I_t. After each optimization step, the encoder is updated, and in the next batch all images are re-encoded in the same manner.

As a result, memory scales with the number of *unique* images per batch rather than the total number of image occurrences across H-length subsequences, yielding substantial efficiency gains while still training the vision encoder end-to-end on long-history dependencies.

B. MemMimic Details

We report the task details, including robot observation and action spaces, for each task in our proposed MemMimic. For all the simulation tasks, we use the MuJoCo simulator [46] to set up the environment. Each checkpoint is evaluated on 100 episodes and we report the mean and standard deviation of the best checkpoint in 3 different training seeds. For in-domain real-world tasks, we use a UR5 robot and a WSG50 gripper with fin-ray fingers [8]. For the in-the-wild real-world task (T3'), we use an ARX X5 with the same UMI setup. For detailed training configurations, please refer to the corresponding dictionary embedded in the checkpoint files for each task.

T1: Match Color (Sim)

Task Setup. To demonstrate that our model attends to the correct moment rather than exhibiting the Attention Sink [52] effect, we initialize the bin colors only after the robot approaches the cube. We randomly initialize the cube position and the colors of the 4 bins, yielding $4 \times 4! = 96$ different configurations. To reduce training time while still keeping the visual memory requirement, we fix the final color scheme after the robot lifts the cube, otherwise there will be $4 \times 4! \times 4! = 2304$ different configurations.

Data Collection. We use a heuristic policy to collect training data. The robot starting pose, cube pose, relative grasping pose, and dropping pose are all randomly sampled within certain ranges, while the pausing pose remains fixed to avoid information leakage when training the no-memory policy. The heuristic policy applies linear interpolation between the key poses and pauses for 2 seconds after lifting the cube, ensuring that more than 2 seconds of memory are required. We use different ranges of starting and ending poses, so that even when the memory gate is off, the policy can learn whether the task is completed. We collect 300 training episodes for this task.

Training and Deployment. The policy uses a third-person camera view as visual observation and the absolute end-effector pose for proprioceptive feedback and action commands. The policy runs at 10Hz with a 500Hz MuJoCo physics update frequency. Applying action command interpolation from 10Hz to 500Hz significantly improves simulation stability compared to directly sending pose targets at 10Hz. The action prediction horizon of all policies is 16 steps (corresponding to 1.6s), and we only execute the first 8 steps (action execution horizon) before the next policy inference.

Baseline Failure Modes. At the pausing stage, [No-hist DP] has no memory of how long it has already paused, and thus consistently predicts a pausing trajectory within the first 8 actions. Therefore, it becomes stuck while holding the cube in the air and reaches the time limit. [Mid-hist DP] can escape the pausing trap, but still lacks sufficient memory to recall which colored bin the cube was originally on. Consequently, [Mid-hist DP] places the cube back into a random bin. [Long-hist DP] has sufficient memory to complete the task, achieving a 100% success rate.

T1': Match Color with Random Delay (Sim)

Task Setup. The task is similar to T1, but we add a 5–600 second random delay after the initial colors disappear before the final colors appear.

Data Collection. The heuristic policy will hold the cube in the air and put it back until the delay is over and the final colors appear. We collect 500 training episodes for this task.

Training and Deployment. We load the pretrained vision encoder from T1 and run it on all of the new data to extract image features. When training the policy, we directly use the cached image features with random gaussian noise augmentation. To save GPU memory, we only sample a subset of trajectories in an episode to train the policy, where each trajectory in the subset still has full access to the history. We train two policies, one with sparse image features (1 image per 8 steps) and one with dense image features (1 image per step). History context length is 6000 steps in total. Both policies perform similarly during evaluation with close to 100% success rate.

T2: Discrete Place Back (Sim)

The task description is similar to that of T1, but the bins are not initialized with different colors, thus there are only 4 different configurations. The data collection procedure, training configuration, and failure modes are identical to those of T1.

T3: Continuous Place Back (Real)

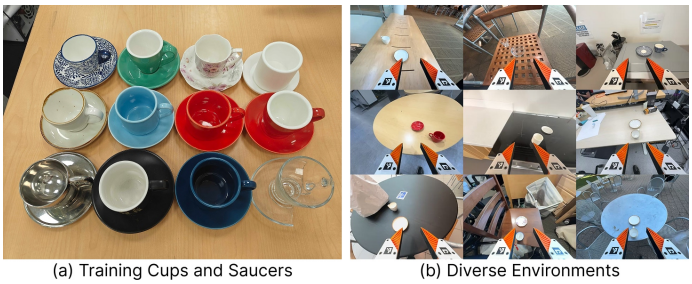
Data Collection. We use an iPhone-based UMI gripper [8] to manually collect 262 training episodes. We randomly place the saucer on the table, and place the cup next to the saucer in a variety of directions. For the same saucer position, we ensure the cup handle is pointing in the same direction, thus the policy cannot simply overfit to the cup handle direction and ignore the long-term dependencies. See Fig. 6 (b) for an example of the initial states.

Training and Deployment. The policy takes the iPhone ultra-wide camera as input and predicts relative end-effector poses as described in the UMI paper. The action prediction frequency is 20Hz during training, and we apply a random trajectory interval (instead of a fixed 8-step interval, we sample the next trajectory randomly between 4 and 8 steps after the previous trajectory), improving the policy's temporal robustness against imperfect policy execution. During GMP deployment, we reduce both the history and future action frequencies to 15Hz for better stability. We predict 16 future actions and execute the first 8 before the next policy inference. We enable asynchronous policy inference (the robot keeps moving the remaining steps during policy inference) for GMP to improve execution speed. We apply *dynamic latency matching* for a smooth transition between the executing trajectory and the new predicted trajectory. We evaluate the policy on 40 different initial states, shown in Fig. 6 (b) for each policy.

Baseline Failure Modes. For the baselines [No-hist DP] and [Long-hist DP], despite slowing down the execution frequency even further to 10Hz, the robot triggers the UR5 emergency stop frequently when pressing against the table, leading to failed episodes. Moreover, for [Long-hist DP], the history proprioceptions readily go out of distribution, leading to a 0% success rate, thus we only keep the history images as input. Additionally, [Long-hist DP] is unable to achieve asynchronous execution, as it jitters severely when the history is long. We must freeze the robot during policy inference. As a result, GMP achieves both higher performance and faster execution compared to the baselines.

T3': In-the-wild Flip and Place Back (Real)

Data Collection. The dataset consists of: 1274 trajectories for only picking up and placing back the cup; 588 trajectories for only flipping the cup; 304 trajectories for both picking up and flipping the cup. We used 12 cups during data collection



(a) Training Cups and Saucers (b) Diverse Environments

Fig. 16: **Training objects and scenes for In-the-wild Flip and Place Back**. We show the cups used for data collection in the real-world `Flip and Place Back` task and a subset of the environments used for data collection.

across more than 30 diverse scenes. Cups and environments during data collection are shown in Fig. 16.

Training and Deployment. We realize that the iPhone ultra-wide camera does not provide clear visual information of whether the cup is flipped or not. Therefore, we use both the ultra-wide and the main camera for policy inference, both resized and padded to 256x256 resolution. We notice that even without gate training, the policy performs well, thus we treat it as a special case of GMP when gate is always on. We deploy the policy at 10Hz action frequency on an ARX X5 robot arm with *dynamic latency matching*. We evaluate the policy in 6 challenging unseen environments and 10 cups, adding up to 20 episodes.

Failure Modes. We observe that most of our policy's failure cases are due to the challenging unseen environments: on the bench (the top left image in Fig. 7 (c)), the cup is very likely to fall into the gap on the bench; on the fountain (the top right image in Fig. 7 (c)), the policy tends to estimate the surface height incorrectly, dropping the cup too early when putting it back. In other environments, the policy is able to complete the task with a high success rate. The baseline [Long-hist DP] overfits much more easily and fails to put the cup back to the original position after the flipping perturbation.

T4: Iterative Pushing (Sim)

Data Collection. We use a heuristic policy to collect 500 training episodes. In each trial, the heuristic policy slowly approaches the cube, linearly accelerates to a pushing velocity v, maintains it for 0.2s, then starts to decelerate from a fixed position. After the cube stops sliding, the heuristic policy moves the robot back to the starting position, and the environment teleports the cube back to its starting position. One episode ends after the heuristic policy demonstrates a complete trial-and-error process for 6 trials.

The friction coefficient of the object is randomly sampled from 0.005 to 0.015. We empirically determine the optimal pushing velocity v_{opt}, ranging from 0.247m/s to 0.435m/s as the friction coefficient varies. The heuristic policy has access to the ground-truth object friction and approaches the optimal pushing velocity v_{opt} within 4 trials: beginning from the middle of the pushing velocity range and adjusting in a binary search manner. Once it reaches the optimal velocity, the policy repeats it in all remaining trials. This ensures that the heuristic policy successfully pushes the object into the target region for the last 3 trials. We record the robot end-effector poses and the top-down camera view for training. Both data collection and policy inference employ 10Hz action frequency and the action space is constrained to the y axis.

Training and Deployment. We extend the history length of GMP to 240 timesteps, including $A_{t-240:t}$ and $I_{t-240:8:t}$ as history conditioning. This ensures that the policy context includes 2 complete trials, achieving higher performance than with only 120 timesteps. However, the performance degrades when extending the history length for [Long-hist DP] and [Long-hist PTP], thus we report the performance with 120 timesteps for long-history baselines. Randomizing the trajectory interval does not turn out to be as helpful as in the real-world experiments, as the simulation includes significantly less noise. During testing, by referring to the last trial, GMP is able to move closer to the optimal pushing velocity through the binary search behavior distilled from the heuristic policy.

Baseline Failure Modes. [No-hist DP], [Mid-hist DP] and [Mid-hist PTP] lack sufficient memory to cover the last trial, and consequently push the object at a random velocity, unable to achieve continuous success in the last 3 trials. While [Long-hist DP] and [Long-hist PTP] are able to reach the optimal pushing velocity and succeed in a few trials, they struggle to maintain the same pushing velocity for multiple trials.

T5: Iterative Flinging (Sim)

Data Collection. We also use a heuristic policy to collect 500 training episodes. In MuJoCo, we model the cloth as a 13×13 grid and the mass is randomly sampled from the following 7 values: 0.1kg, 0.15kg, 0.2kg, 0.3kg, 0.5kg, 1.0kg, 2.0kg, with a small random perturbation $\mathcal{N}(0, 0.001)$kg to increase diversity. In the flinging process, the heuristic policy lifts the cloth, flings it along a cubic spline trajectory with a series of waypoints, and returns to the starting position after the cloth falls on the table. To demonstrate the trial-and-error process for a cloth with unknown mass, it iteratively adjusts the time interval between the waypoints and achieves continuous success for the last 3 trials. We record the end-effector poses for both robot arms and the top-down camera view for training. Both data collection and policy inference employ a 10Hz action frequency, and the action space is constrained to the x and z axes; the cloth is welded to the robots' grippers for stability.

Training and Deployment. To accelerate the simulation, we use the Projected Gauss-Seidel (PGS) solver in MuJoCo [45, 46] to solve cloth dynamics. Due to numerical instability, we observe that the cloth may still perform unrealistic motions, such as penetrating into the table or entangling with itself. Therefore, we relax the success criterion to require at least 10 (out of 13) edge grid points touching the black area. Despite the simulation being imperfect, GMP still learns to iteratively refine its actions based on past experiences, achieving higher performance than the other long-context baselines.

Baseline Failure Modes. As with T4, the flinging motion has higher velocity tolerance compared to pushing. [No-hist DP], [Mid-hist DP] and [Mid-hist PTP] learn to fling the cloth

at a medium velocity, and are thus able to achieve success on cloths with medium mass (0.2kg, 0.3kg, 0.5kg), achieving a 54% success rate, but fail on cloths with extremely light or extremely heavy masses.

T6: Iterative Casting (Real)

Data Collection. We use a heuristic policy to collect 80 training episodes. In each episode, we randomly choose the contact surface to be Teflon fiber (low friction, optimal casting velocity 1.4m/s) or electrical tape (high friction, optimal casting velocity 2.2m/s). In each trial, the heuristic policy casts the object, stops at the same position, lifts the object after the object stops sliding, and resets to the starting position. In each episode, the heuristic policy executes 3 trials. In the first trial, the heuristic policy casts at a medium velocity (1.8m/s), leading to either overshooting (Teflon fiber) or undershooting (electrical tape). In the second and third trials, it casts with the optimal velocity of the chosen contact surface to ensure success. We use a top-down camera to record videos at 30Hz and randomly perturb the camera pose every 20 episodes to reduce overfitting. We constrain the robot movement to the y-z plane during both data collection and policy inference. We visualize the position control for different casting velocities in Fig. 17.

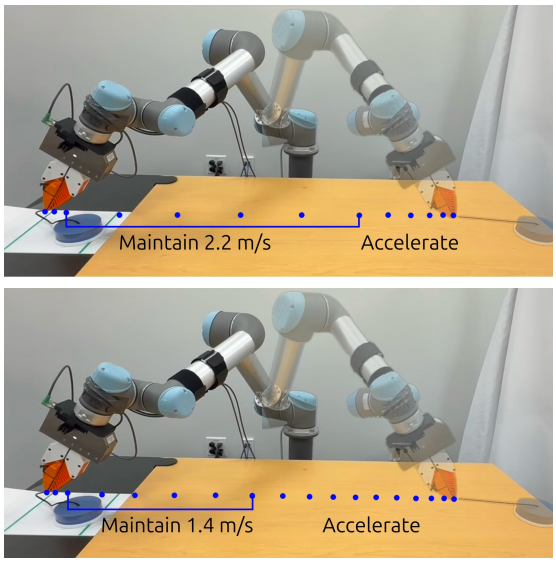


Fig. 17: **Position Control for Different Casting Velocities**. We present examples of waypoint-based position control to achieve different casting velocities. While starting and ending at the same position and running at the same frequency, the heuristic policy adjusts the waypoint distribution to modify the casting velocity. Before the robot decelerates, sparser waypoints lead to faster casting velocities, while denser waypoints lead to slower casting velocities.

Training and Deployment. We downsample the training data to 15Hz and execute GMP and all other baselines at 15Hz for consistent dynamics. The policy predicts 25 future actions and we execute the first 15 before the next policy inference. During training, we randomize the trajectory interval from 12 to 18 steps and apply a random image latency from -0.3s to 0.3s to improve temporal robustness against the noise and delay in the real-world system. We evaluate the policy on 2 different contact surfaces and 10 episodes per surface.

Baseline Failure Modes. We observe that [No-hist DP] consistently casts the object at the medium velocity (1.8m/s), leading to failure in 95% of the episodes. Although Diffusion Policy is able to learn a multi-modal action distribution, the training error generally remains the lowest if it always chooses the medium velocity, and thus it collapses to a single mode. [Long-hist DP] still fails to include past proprioceptions due to real-world noise, leading to severe jittering with a 0% success rate; therefore, we only keep history images as input. We notice that [Long-hist DP] learns to cast at the medium velocity for the first trial, and chooses a different velocity in the second and third trials. However, when switching from one trajectory (start of casting) to another (end of casting), [Long-hist DP] often gives inconsistent velocity predictions. Moreover, [Long-hist DP] fails to maintain the same casting velocity for the last two trials, resulting in a 20% success rate.

C. MIKASA-Robo Evaluation Details

We evaluate our method on the MIKASA-Robo benchmark [5] for 5 tasks: `ShellGameTouch`, `InterceptMedium`, `RememberColor3`, `RememberColor5`, and `RememberColor9`. We notice that the action generated by the PPO oracle policy is very noisy. Instead of using delta joint actions, we calculate the equivalent absolute joint actions and use them as training supervision. Due to the different task settings, we try different action prediction and execution horizons and report the best performing checkpoint. To match the training protocol in prior work, we only use 250 episodes per task, which is insufficient for the remember color tasks, especially for `RememberColor9`. More training episodes (e.g. 1000) will lead to significantly better performance. Another approach to achieve higher performance is to use rule-based heuristics or human teleoperation to generate training data, which has much lower action noise.

D. Visualization of the Memory Gate Calibration

In this section, we visualize the statistics of the memory gate calibration process for one example episode per task. This demonstrates that the calibration process can identify whether a state requires memory and generate corresponding labels, ensuring the functionality of the memory gate during deployment.

After training two policies with and without memory, π_{mem} and π, we sample 500 trajectory batches from each episode for π_{mem} and 20000 samples from each episode for π on the validation set to calculate the average action prediction error. We apply a sliding-window average of 20 timesteps to both δ_t^{mem} and δ_t to reduce noise from outliers. Fig. 18 is the result before applying the sliding-window average. After applying, the active labels are generally wider than the current active labels. Finally, we label μ_t as 1 if $\delta_t > \theta\delta_t^{\text{mem}}$ for a threshold $\theta = 10.0$ across all tasks, otherwise $\mu_t = 0$. We visualize the

action prediction errors and memory gate labels along with task progress in Fig. 18.

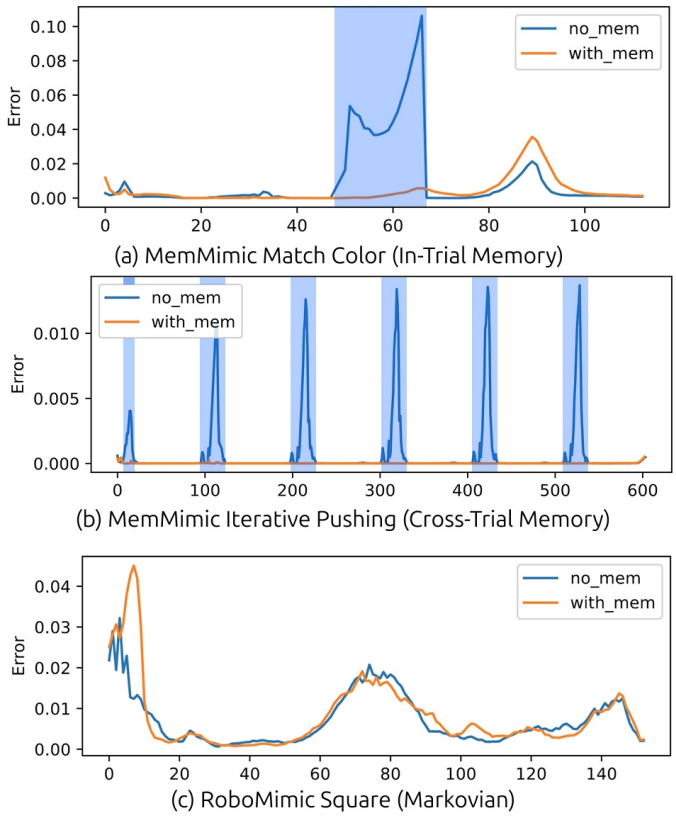


Fig. 18: **Example Statistics of Memory Gate Label Generation**. We present the action prediction errors for both the no-memory policy (δ_t) and memory policy (δ_t^{mem}) for one example episode per memory requirement category. The x-axis represents the timestep of each episode. We use a blue background to indicate when the memory gate label is set to 1, and no background when it is set to 0. We observe that in **(a)** `Match Color`, $\delta_t > \theta\delta_t^{\text{mem}}$ in the middle of the episode when the robot is placing the cube back; in **(b)** `Iterative Pushing`, $\delta_t > \theta\delta_t^{\text{mem}}$ in every trial before the robot pushes; while in **(c)** `Square`, the action prediction error is similar between the two policies, indicating that the memory gate should remain off throughout.

We observe that the memory gate is on (i.e., $\mu_t = 1$) earlier than the moments that require memory. This is because we are calculating the error for the entire predicted action trajectory $\delta_t = \|\hat{A}_{t:t+h} - A_{t:t+h}\|_2$, which aligns with the policy deployment requirement—the policy should recall memory when predicting actions right before the critical moment. For the RoboMimic tasks, we observe that the prediction error with or without memory is similar, indicating that the memory gate can identify the critical moments that require memory and omit memory recall when it is not needed.

E. Why Not Use Continuous Gating

While prior methods mostly use a continuous gate value to control the amount of history to be recalled, we find that simply training a continuous gate together with the policy model is insufficient for memory usage control. Similar to **Finding 4**, without applying a regularization loss term, the gate value tends to converge to 1, leading to poor performance on Markovian tasks, as shown in Fig. 19. The success rate difference in `Tool Hang` is not as significant as that of the binary gate, because we do not need to apply Straight-Through Estimator (STE) during training and the training gradients are more stable. However, when applying regularization, there is also a significant trade-off between performance on Markovian and non-Markovian tasks, making it difficult to find a regularization weight that works well across all tasks.

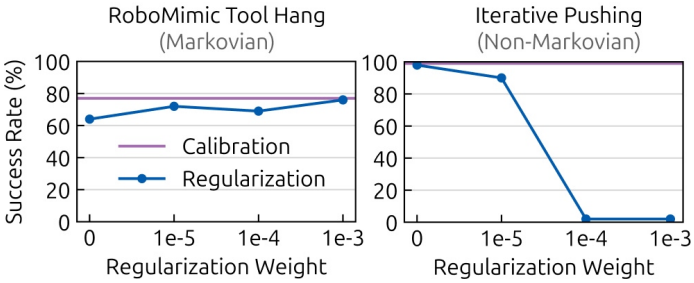


Fig. 19: **Continuous Memory Gate Ablation**. When training a continuous memory gate jointly with the policy, using no regularization, which encourages the gate to stay on, degrades performance on Markovian tasks, while using high regularization, which encourages the gate to be off, hurts performance on non-Markovian tasks.

While it is possible to apply the same calibration process to a continuous gate, replacing the BCE loss with an MSE loss and keeping all other stages the same, we find that the continuous-gated policy works similarly to the binary-gated policy. However, using continuous values limits the ability to skip memory recall when unnecessary, thereby preventing speedup of policy inference.

F. Training Cost and Calibration Overhead

Full model convergence is not required for label generation. Therefore, a checkpoint from the first few epochs is sufficient to observe prediction error differences. We also don't need to run the calibration process on the full dataset, as a subset of the validation set is sufficient. GPU-hours on H100 GPUs are shown for the most challenging `Iterative Pushing` to achieve the best performance checkpoint.

Train π	Train π_{mem}	Rollout	Train Gate	Train GMP
1.2h	6.7h	6.4h	1.3h	31.2h

However, we do notice that the gate is not necessary for all tasks. In tasks that do not require high-precision actions, for example `Mikasa-Robo` tasks and `Continuous Place Back`, the policy without the gate already achieves decent performance. A general rule of thumb is to first train the policy without the gate and run evaluation. If there are clear signals that the policy fails because of overfitting, for example in the `Robomimic Tool Hang` task, then we train the gate and consequently the gated policy.