FLASH: Efficient Visuomotor Policy via Sparse Sampling

Jiaqi Bai* **Jindou Jia*** **Yuxuan Hu** **Gen Li**
Xiangyu Chen **Tuo An** **Kuangji Zuo** **Jianfei Yang†**
MARS Lab, Nanyang Technological University, Singapore
baij0018@ntu.edu.sg, jianfei.yang@ntu.edu.sg

Abstract

Generative models such as diffusion and flow matching have become dominant paradigms for visuomotor policy learning, yet their reliance on iterative denoising incurs high inference latency incompatible with real-time robotic control. We present **F**ast **L**egendre-polynomial **A**ction policy via **S**parse **H**istory-anchored flow (**FLASH** Policy), which replaces discrete action-chunk generation with continuous *Legendre* polynomial trajectory representation. Specifically, by fitting expert demonstrations under sparse temporal sampling, FLASH enables a single inference to cover a significantly extended action horizon. To further accelerate generation, FLASH initiates the flow matching process from history polynomial coefficients rather than uninformative *Gaussian* noise, shortening the transport distance and enabling accurate single-step inference. Moreover, analytic polynomial differentiation directly provides desired velocity feed-forward signals to the torque controller without numerical approximation. Evaluation spans 12 methods, five simulated manipulation tasks, four real-robot tasks, and two controlled 2D stress tests. FLASH achieves 96.8% mean success on the five simulated tasks, outperforming all baseline method. On *Stack Cube*, its inference cost is $0.32\,ms$ per environment step (up to $113\times$ faster than diffusion policies and $21\times$ faster than prior flow matching policies), and reduces controller tracking error through analytic velocity feed-forward.

Project: https://b1ue-jay.github.io/FLASH
Code: https://github.com/NTUMARS/FLASH-Policy

1 Introduction

Imitation learning has enabled robots to acquire complex manipulation skills through expert demonstrations. Alongside recent work on world models [7], generative modeling such as diffusion policy [5] and flow matching [15] has emerged as a dominant paradigm for policy representation in imitation learning, exhibiting exceptional multi-modal distribution modeling capabilities. However, their reliance on iterative denoising or ordinary differential equation (ODE) solving leads to high inference latency, making it difficult to satisfy real-time control requirements in robotic systems.

To mitigate inference latency, prevailing methods primarily aim to reduce the number of inference steps [29, 31, 18, 25, 26, 10, 14]. Despite these advancements, they still rely on discrete action-point generation that requires high-frequency inference over short action chunks, where each chunk typically spans only a short duration (e.g., ~ 0.8 seconds in π_0 [2]). A natural approach is to increase the prediction horizon to reduce inference frequency and improve real-time efficiency. However,

*Equal contribution.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

doing so for a single prediction inevitably leads to a substantial expansion in output dimensionality, resulting in the "curse of dimensionality" and significantly increased computational cost. This fundamental trade-off raises a key question: **Can low-frequency inference meet high-frequency control demands without sacrificing control fidelity or increasing model complexity?**

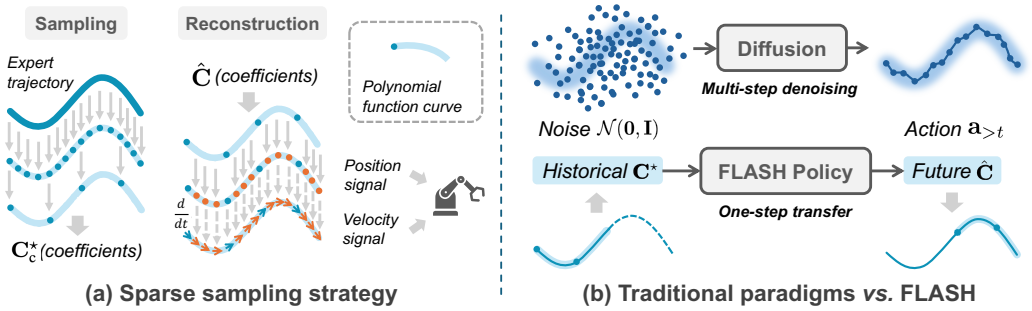


Figure 1: **Overview of FLASH Policy. (a) Sparse sampling strategy**: expert trajectories are fitted to polynomial coefficients $\mathbf{C}_c^\star$ at a significant long temporal stride. During deployment, we perform dense temporal upsampling on the predicted coefficients $\hat{\mathbf{C}}$. Both position and velocity signals are fed to the controller. **(b) Traditional paradigms vs. FLASH**: conventional generative policies (top) generate discrete, non-smooth action points through multi-step denoising from uninformative noise. FLASH (bottom) operates in the coefficient space, transferring the historical coefficients to future coefficients with only one step, yielding a smooth continuous trajectory.

Our key observation is that robot motions are inherently smooth and low-frequency, and a simple *Legendre* polynomial can represent the discrete action points with only a few coefficients. While there exist recent works adopting polynomial-based representations for trajectory planning [4, 27], they still use the original number of action points to fit the polynomial. To achieve highly efficient inference, we propose **F**ast **L**egendre-polynomial **A**ction policy via **S**parse **H**istory-anchored flow (**FLASH**) that incorporates sparsity and generation mode for polynomial policy learning. Firstly, a *sparse temporal sampling* strategy is proposed to fit polynomial coefficients at a significantly long temporal stride, as Fig. 1 (a), so that each inference can cover a substantially extended action horizon without increasing model scale, directly answering the question above. Secondly, we introduce FLASH flow matching mechanism that initiates generation from the polynomial fitted to the measured action history, as Fig. 1 (b). This keeps the entire generative process in a compact, structured coefficient space, dramatically shortening the transport distance and enabling accurate single-step inference. Moreover, *cross-horizon kinematic continuity* constraints enforce C^1 continuity (position and velocity) [1] on fitted training targets, encouraging smooth transitions between predicted polynomial chunks.

The polynomial representation offers additional benefits for low-level controllers. First, its differentiable nature enables analytic computation of desired velocity via simple differentiation, which we leverage as a feedforward signal to the low-level torque controller to substantially reduce joint-position tracking errors. Second, this representation fundamentally guarantees the spatial smoothness of the trajectory and aligns with the varying control frequencies across different robotic platforms, which is hardware-friendly to low-level controllers. Third, during inference, task execution can be accelerated or decelerated by simply adjusting the evaluation temporal stride, satisfying the specific timing constraints of different tasks. We thoroughly evaluate FLASH in both simulated environments and real-world robotic systems, against 10 baselines and an internal control variant. The empirical results show that FLASH exhibits superior performance regarding the success rate, efficiency, controller tracking error, and smoothness, consistently outperforming state-of-the-art (SOTA) baselines.

To recap, the primary contributions of this work are as follows:

- We propose **F**ast **L**egendre-polynomial **A**ction policy via **S**parse **H**istory-anchored flow (**FLASH** Policy), which compresses a significantly long trajectory as only a few *Legendre* polynomial coefficients fitted under sparse temporal sampling. Meantime, FLASH initiates the generative flow from a history prior in the coefficient space, enabling accurate single-step inference.

- We introduce *cross-horizon* C^1 *constraints* on fitted training targets to encourage smooth predicted chunk transitions, and use polynomial differentiation to supply analytic velocity feedforward to the torque controller without numerical approximation or additional velocity supervision.
- Extensive experiments across 11 tasks and 10 baselines demonstrate that FLASH achieves state-of-the-art success rates, an amortized inference cost of $0.32\,ms$ per environment step (up to $113\times$ faster than diffusion policy [5] and $21\times$ faster than prior flow matching [15]), up to $4\times$ faster training convergence than ACT [35], and $5\times$ to $7\times$ controller tracking error reductions.

2 Related work

2.1 Generative visuomotor policies

Diffusion policy [5] first formulates visuomotor control as conditional denoising diffusion process [12], generating action chunks via iterative DDPM sampling. Flow matching [15] offers a simulation-free alternative with more flexible ODE solvers. Subsequent efforts focus on accelerating diffusion and flow inference through faster samplers [29, 30, 21], consistency models [31], diffusion distillation [25, 18, 26], MeanFlow [10], more expressive architectures (DiT [22]), and visual tokenization with flow-based generation (VITA [8]). While A2A [14] enables faster generation by replacing the uninformative *Gaussian* prior with the previous *action* chunk, it still operates in the discrete action space. Instead, FLASH operates in the *functional* level. The generative flow is initiated from the polynomial coefficients fitted to the measured action history, a trajectory-level prior that preserves richer kinematic structure than a flat action-chunk prior. More broadly, all the above methods predict fixed-length discrete action chunks that inherently suffer from inter-chunk jitter and require high-frequency re-inference. In contrast, FLASH lifts the representation to continuous polynomial coefficients, decoupling inference frequency from control frequency.

2.2 Polynomial trajectory representations

Recent learning-based methods have begun to adopt polynomial and spline parameterizations in robot trajectory planning. MPD [4] trains diffusion models over B-spline control points, employing analytically derived velocity and acceleration solely as smoothness cost functions during the denoising process. FlowMP [20] extends this idea to flow matching, jointly learning position, velocity, and acceleration fields from B-spline training data, which triples the supervision dimensionality and increases training overhead. BEAST [36] employs B-splines purely as an efficient tokenization scheme for transformer-based imitation learning, without exploiting derivative information. Crowd-FM [27] parameterizes trajectories as *Bernstein* polynomials for mobile robot crowd navigation, yet does not feed the analytic velocity to any low-level controller. A common limitation across these works is that *none* actually leverages the analytic differentiability of the polynomial representation to supply velocity feedforward signals to a torque-level controller. FLASH fills this gap. It uses *Legendre*-coefficient trajectories with C^1-constrained training targets, from which desired velocity is derived by analytic differentiation without additional velocity supervision.

3 Fast Legendre-polynomial action policy via sparse history-anchored flow

We consider a visuomotor imitation-learning setting in which a policy maps historical observations to a future action trajectory. Let $\mathbf{a}_{\leq t} = \{\mathbf{a}_{t-T_o+1}, \ldots, \mathbf{a}_t\}$ denote the history actions [3] observed up to step t with observation horizon T_o, $\mathbf{I}_{\leq t}$ denote the corresponding image stream, and $\mathbf{a}_{>t} = \{\mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+T_a}\}$ denote the generated actions with action horizon T_a. As illustrated in Fig. 1, FLASH departs from discrete action-chunk generation [35]: rather than learning T_a discrete action points, FLASH generates a small set of polynomial coefficients from which the entire continuous trajectory segment is analytically reconstructed. The key idea to improve inference efficiency is twofold. On the one hand, discrete action chunks are parameterized by compact polynomials (Sec. 3.1), decoupling the network's output dimensionality from the prediction horizon. On the other hand, the generative flow is initiated from the polynomial coefficients fitted to the action

[3] We assume history actions are observable, a common setting in manipulation tasks where a direct semantic correspondence exists between actions and proprioceptive states (e.g. joint angle, end-effector pose, and their differential forms).

history rather than from noise (Sec. 3.2), shortening the transport distance and enabling accurate single-step generation. Sec. 3.3 details the learning objective.

3.1 Sparse polynomial parameterization

***Legendre* trajectory representation.** Instead of generating a sequence of discrete samples, we parameterize the action trajectory as a continuous function of a normalized time variable $s \in [0, 1]$. Each action dimension is expanded on a set of orthogonal basis functions $\{\Phi_j\}_{j=0}^{K}$:

$$\mathbf{a}(s) \;=\; \sum_{j=0}^{K} \mathbf{c}_j \, \Phi_j(s), \qquad s = \frac{\tau - \tau_{\text{start}}}{\tau_{\text{end}} - \tau_{\text{start}}} \in [0, 1], \tag{1}$$

where $\Phi_j(s) \in \mathbb{R}$ is a scalar basis function, $\mathbf{c}_j \in \mathbb{R}^{d_a}$ is the corresponding coefficient vector, K is the polynomial degree, d_a is the action dimensionality, and τ_{start}, τ_{end} are the first and last raw-data timesteps of the fitting window, including padding when used. The predicted segment contains H_{poly} sparse nodes; in its simplest form $H_{\text{poly}} = T_a$. We adopt the shifted *Legendre* polynomials [1, 33] as $\{\Phi_j\}$, which are defined through the *Bonnet* recurrence [32] ($n = 1, \ldots, K-1$):

$$(n+1)P_{n+1}(x) = (2n+1)\, x\, P_n(x) - n\, P_{n-1}(x), \qquad P_0(x) = 1,\; P_1(x) = x. \tag{2}$$

Evaluated at $x = 2s - 1$, the *Legendre* family is orthogonal on $[-1, 1]$. Continuous orthogonality provides a structured basis, but the conditioning of a sampled *Gram* matrix also depends on the degree and sampling window. Coefficients are scale-normalized for learning, and the history fit is regularized below. With this continuous representation in place, we next describe how to fit the coefficients to expert demonstrations efficiently.

Sparse temporal sampling fitting. To extend the physical prediction horizon without increasing the network's output dimensionality, we apply a sparse temporal sampling strategy to the expert demonstrations. We extract nodes from the expert trajectory using a temporal stride k, so that H_{poly} nodes span $k(H_{\text{poly}} - 1)$ raw-data intervals. Let H_{fit} count all fitting nodes, including any padding. Then $\mathbf{Q} \in \mathbb{R}^{H_{\text{fit}} \times d_a}$ stacks their actions, and $\mathbf{S} \in \mathbb{R}^{H_{\text{fit}} \times (K+1)}$ is the *Legendre* basis matrix evaluated at those nodes. The optimal polynomial coefficients $\mathbf{C}^\star$ can be obtained in closed-form via ordinary least squares [3, 34] (OLS):

$$\mathbf{C}^\star = (\mathbf{S}^\top \mathbf{S})^{-1} \mathbf{S}^\top \mathbf{Q}. \tag{3}$$

The inverse in Eq. (3) requires $\mathbf{S}$ to have full column rank, with at least $K + 1$ distinct fitting nodes. This strategy leverages the analytic continuity of polynomials to compactly represent a trajectory segment spanning a longer physical horizon.

Cross-horizon continuity and two-tier extension. Solving $\mathbf{C}^\star$ independently for adjacent sub-segments introduces position or velocity discontinuities at the junctions. To enforce C^1 continuity, we extend the fitting window with overlapping regions on both sides of the execution interval and place exact kinematic anchors at the segment boundaries. A closed-form *KKT* correction [17] then projects the unconstrained $\mathbf{C}^\star$ onto the constraint manifold, yielding the corrected coefficients $\mathbf{C}_{\text{c}}^\star$. We further append fit-padding steps to pull these anchors into the stable interior of the *Legendre* fitting window, preventing boundary sensitivity. The corrected coefficients are then row-wise scale-normalized to stabilize the flow matching objective. Detailed derivations are provided in Appx. A. The above pipeline produces the training targets. We now describe how the predicted coefficients are decoded into trajectories at deployment.

Temporal upsampling and frequency decoupling. Sparse sampling extends the time between the predicted segment's first and last nodes by a factor of k. During deployment, we perform dense temporal upsampling on the predicted coefficients $\hat{\mathbf{C}}$, evaluating the executable trajectory $\hat{\mathbf{q}}(s_i)$ and its exact higher-order kinematic derivatives either at the original frequency used to collect the expert trajectories, or at any arbitrary high-frequency grid required by different low-level controllers:

$$\hat{\mathbf{a}}(s_i) = \sum_{j=0}^{K} \hat{\mathbf{c}}_j \Phi_j(s_i), \qquad \hat{\mathbf{v}}(s_i) = \frac{1}{T} \sum_{j=0}^{K} \hat{\mathbf{c}}_j \Phi_j'(s_i), \tag{4}$$

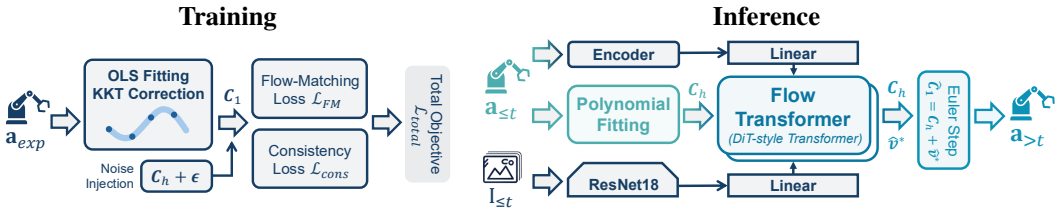


Figure 2: **Overview of FLASH's pipeline. Left: Training.** Expert actions are fitted to target coefficients using OLS and KKT correction, with flow-matching and consistency objectives. **Right: Inference.** A DiT-style transformer combines history coefficients with visual features; a single Euler step predicts future coefficients for trajectory decoding.

where $T = (\tau_{\text{end}} - \tau_{\text{start}})/f_{\text{expert}} = k(H_{\text{fit}} - 1)/f_{\text{expert}}$ is the physical duration between the first and last fitting nodes, and f_{expert} is the original demonstration sampling rate. With δ padding nodes on each side, $H_{\text{fit}} = H_{\text{poly}} + 2\delta$ (Appx. A). For a fixed number of generated coefficients, the extended horizon can reduce the frequency of policy calls. Moreover, the continuous polynomial representation naturally supports post-hoc temporal scaling: a single inference produces coefficients $\hat{\mathbf{C}}$ that fix the trajectory shape on the normalized time $s \in [0, 1]$, while the physical duration T over which this shape is played satisfies $T \propto k$ (Eq. (4)); the velocity command $\hat{\mathbf{v}} \propto 1/T$ is automatically rescaled by the same analytic derivative. Hence, simply setting the evaluation stride k_{eval} to a value different from the training stride k_{train} allows the same generated polynomial to be executed at any desired speed, with no retraining and no extra computation. The evaluated implementation chooses this speed before each rollout, enabling acceleration of simple, safe tasks for shorter execution time and deceleration of high-precision, high-risk maneuvers. The achievable speed range is empirically validated in Sec. 4.5.

3.2 History-anchored flow

With the polynomial representation, we employ a flow matching formulation to predict the transition of coefficients from adjacent history to the future timesteps, conditioned on the observations.

Flow matching. Flow matching [15] learns a time-dependent velocity field f_θ that transports a source distribution p_0 to a target distribution p_1 along the probability path induced by the linear interpolant $\mathbf{a}_\tau = (1-\tau)\mathbf{a}_0 + \tau\mathbf{a}_1$, $\tau \in [0, 1]$. The corresponding conditional velocity is $\mathbf{v}^\star = \mathbf{a}_1 - \mathbf{a}_0$, and training minimizes $\mathbb{E}_{\tau,\mathbf{a}_0,\mathbf{a}_1} \| f_\theta(\mathbf{a}_\tau, \tau, \mathbf{e}) - \mathbf{v}^\star \|^2$, where τ is the flow time uniformly sampled from $[0, 1]$ and $\mathbf{e}$ denotes the conditioning observation embedding. Standard flow matching policies fix $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$, which obliges the network to learn long transport paths and motivates the many-step ODE integration used at inference.

History-anchored flow. In visuomotor control, consecutive action segments share strong kinematic structure [14]. We leverage this informative prior by defining the generative flow directly between history and future polynomials, rather than originating from uninformative *Gaussian* noise. Specifically, we fit the observed proprioceptive history $\mathbf{a}_{\leq t}$ to the *Legendre* basis to obtain the history coefficients $\mathbf{C}_h$. We use the same scheme as Eq. (3), but introduce *Tikhonov* regularization $\lambda_h > 0$. This formulation not only allows for fitting when the history window is configured to be short, but crucially, it mitigates the numerical ill-conditioning caused by dense discrete sampling over a narrow time horizon. By preventing coefficient explosion, it ensures a numerically stable and bounded starting distribution for the subsequent flow matching:

$$\mathbf{C}_h \;=\; \left(\mathbf{S}_h^\top \mathbf{S}_h + \lambda_h \mathbf{I}\right)^{-1} \mathbf{S}_h^\top \, \mathbf{a}_{\leq t}. \tag{5}$$

FLASH then constructs a straight-line flow path from $\mathbf{C}_h$ to the target future coefficients $\mathbf{C}_1$:

$$\mathbf{C}_\tau \;=\; (1-\tau)\,\mathbf{C}_h + \tau\,\mathbf{C}_1, \qquad \mathbf{v}^\star(\mathbf{C}_\tau, \tau) \;=\; \mathbf{C}_1 - \mathbf{C}_h. \tag{6}$$

A DiT-style transformer [22] $f_\theta(\mathbf{C}_\tau, \tau, \mathbf{e})$ regresses this constant velocity $\mathbf{v}^\star$, where the global condition $\mathbf{e}$ is obtained by encoding the multi-view image stream $\mathbf{I}_{\leq t}$ with a ResNet-18 encoder [11], concatenating each frame's visual features with the corresponding proprioceptive history $\mathbf{a}_{\leq t}$, and

flattening across the T_o observation steps, as shown in Fig. 2. Because $\mathbf{C}_h$ is typically close to $\mathbf{C}_1$, the transport distance is short and a single *Euler* step $\hat{\mathbf{C}}_1 = \mathbf{C}_h + f_\theta(\mathbf{C}_h, 0, \mathbf{e})$ is sufficient at inference. The resulting $\hat{\mathbf{C}}_1$ is then decoded into the executable action trajectory $\hat{\mathbf{a}}_{>t}$ together with its analytical velocity profile via Eq. (4). During training, we inject minimal *Gaussian* noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_h^2 \mathbf{I})$ into $\mathbf{C}_h$ to improve robustness against off-policy histories at deployment. Standard evaluation uses the fitted history without additional inference-time noise; explicit noise perturbations are evaluated separately in Appx. J.9. With the parameterization (Sec. 3.1) and flow architecture defined, we now formalize the learning objectives.

3.3 Learning objectives

Flow matching loss. Following Eq. (6), the network is trained to regress the constant transport velocity along every intermediate point of the polynomial flow,

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{\tau \sim \mathcal{U}[0,1],\, \mathbf{C}_h,\, \mathbf{C}_1} \Big\| f_\theta\big(\mathbf{C}_\tau, \tau, \mathbf{e}\big) - (\mathbf{C}_1 - \mathbf{C}_h) \Big\|_2^2. \tag{7}$$

Polynomial consistency loss. Eq. (7) supervises the vector field at arbitrary τ, but does not explicitly constrain the *one-step Euler* update used at inference. We therefore add a consistency term that enforces transport of $\mathbf{C}_h$ to $\mathbf{C}_1$ in a single evaluation of f_θ at $\tau = 0$:

$$\mathcal{L}_{\text{cons}} = \mathbb{E}_{\mathbf{C}_h,\, \mathbf{C}_1} \Big\| \mathbf{C}_h + f_\theta\big(\mathbf{C}_h,\, 0,\, \mathbf{e}\big) - \mathbf{C}_1 \Big\|_2^2. \tag{8}$$

This term bridges the gap between the continuous-time flow matching objective and the discrete one-step ODE solver used at deployment, and improves accuracy in the $N_{\text{NFE}} = 1$ regime (Table 20 in Appx. J).

Total objective. With a scalar weight $\lambda_{\text{cons}} > 0$, the training loss reads

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{FM}} + \lambda_{\text{cons}}\, \mathcal{L}_{\text{cons}}. \tag{9}$$

All parameters of the vision encoder, condition projector, and flow transformer f_θ are optimized jointly. The scale-normalization matrix, least-squares solver (Eq. (3)), *Tikhonov*-regularized history solver (Eq. (5)), and *KKT* correction (Eq. (11)) are all precomputed and remain fixed during training.

4 Experiments

We evaluate FLASH on five simulated manipulation tasks in *Roboverse* [9], four real-world *Franka* robot tasks (Fig. 3), and two 2D stress tasks in Sec. 4.6.

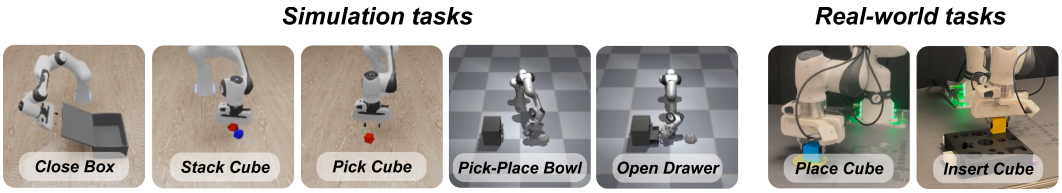


Figure 3: **Left:** Five simulated tasks from *Roboverse* [9] platform. **Right:** Representative real-world tasks: *Place Cube* and *Insert Cube*.

We compare 10 external baselines: DDPM-UNet [5, 12], DDPM-DiT [12, 22], DDIM-UNet [5, 29], FM-UNet and FM-DiT [15], Score-UNet [30], ACT [35], VITA [8], MIP [21], and A2A-Noise [14]. FLASH-G is an internal control sharing FLASH's sparse polynomial representation and architecture but generating from Gaussian noise with $NFE = 10$. MIP is evaluated at NFE=1 and NFE=2. Task protocols, architecture and parameter counts, and shared training settings are in Appx. B.

Table 1 shows that FLASH achieves $\geq 92\%$ success on every task and 96.8% on average, 4.8 percentage points above A2A-Noise. Each entry uses 50 rollouts (Appx. C); the internal FLASH-G comparison is included in the ablation study (Sec. 5).

Table 1: **Five-task success rates (%) at the 10,000-step checkpoint.** Each entry uses 50 rollouts and a 15,000-step training/LR schedule. NFE is the number of generator evaluations per policy call. *P-P Bowl* denotes *Pick-Place Bowl*. Bold denotes the best value in each task.

Method	NFE	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
Score-UNet	100	44	58	24	2	8	27.2
DDPM-UNet	100	84	68	44	76	90	72.4
DDIM-UNet	40	80	74	42	76	92	72.8
DDPM-DiT	100	68	78	36	50	76	61.6
FM-DiT	10	70	92	46	36	68	62.4
FM-UNet	10	78	82	30	70	76	67.2
ACT	1	70	**98**	20	80	0	53.6
VITA	6	94	86	86	68	86	84.0
A2A-Noise	1	98	90	92	88	92	92.0
MIP	2	86	88	28	44	18	52.8
MIP	1	68	92	48	46	2	51.2
FLASH	1	**100**	**98**	**96**	**92**	**98**	**96.8**

4.1 Training efficiency

Training efficiency is measured at checkpoints from 2,500 to 10,000 steps. Figure 4 shows representative learning curves; the task curves and MIP results at both inference budgets are reported together in Appx. D.

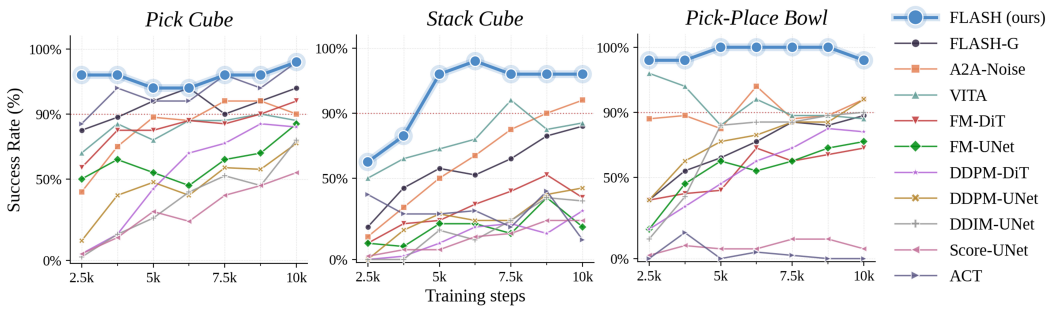


Figure 4: **Training efficiency.** Success rate versus training steps on three simulated tasks. The plotted methods comprise nine baselines, FLASH-G, and FLASH; MIP curves are tabulated in Appx. D.

On *Pick Cube*, FLASH reaches 96% success after 2,500 steps, while ACT requires 10,000 steps for a comparable result: $4\times$ fewer optimizer steps. On *Stack Cube*, FLASH reaches 98% at 6,250 steps versus 72% for VITA. On *Pick-Place Bowl*, FLASH reaches 100% after 5,000 steps. These comparisons demonstrate faster convergence under the matched training schedule.

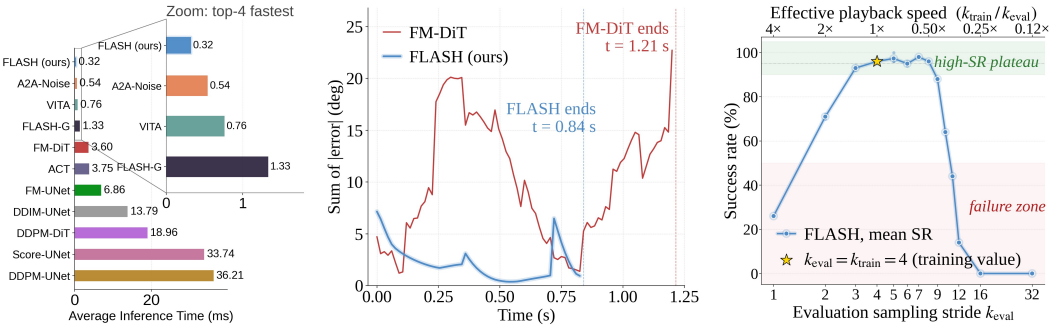


Figure 5: **Simulation experiments. Left:** amortized inference cost on *Stack Cube* (ms/environment step, RTX 5090). **Middle:** sum of seven-joint absolute tracking errors on *Pick Cube*; dashed lines mark task completion. **Right:** playback-speed modulation on *Stack Cube*.

4.2 Inference speed

All timing comparisons use the same NVIDIA RTX 5090 GPU. Our primary metric is *amortized inference cost*: accumulated policy-inference time divided by executed environment steps. On *Stack Cube*, FLASH requires 0.32 ms/environment step, versus 0.54 for A2A-Noise, 1.33 for FLASH-G, and 36.21 for DDPM-UNet (Fig. 5, left). This metric accounts for chunk reuse.

Cumulative episode inference cost and physical task-completion time are complementary deployment measures, reported in Appx. E. They depend on execution schedules and trajectory duration as well as the cost of each policy call.

4.3 Controller tracking error

The polynomial representation supplies high-frequency positions and analytic velocity feed-forward (Eq. (4)). In the representative simulated *Pick Cube* rollout (Fig. 5, middle), FLASH reduces mean and peak seven-joint tracking errors by factors of 4.70 and 3.18 relative to FM-DiT. In real *Place Cube*, its tracking MAE is $0.274 \pm 0.004°$, versus $0.460 \pm 0.028°$ for FM-DiT over five rollouts. These comparisons evaluate the policies together with their controller interfaces (Appx. F).

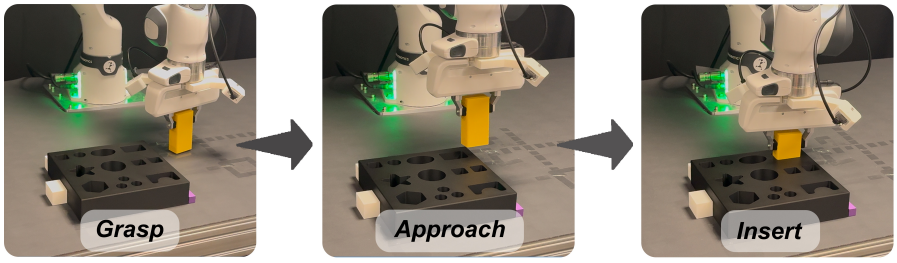


Figure 6: **Real-world *Insert Cube* task.** Execution snapshots (grasp → approach → insert) under millimeter-level insertion tolerance. Success rates are reported in Table 9.

The real-world *Insert Cube* task requires millimeter-level insertion precision (Fig. 6). FLASH succeeds in all ten trials, exceeding the other seven policies by 47.1 percentage points on average. Complete success rates appear in Appx. F (Table 9).

4.4 Contact-rich manipulation

Keyboard Pressing tests repeated free-space/contact transitions by pressing and releasing Up, Space, and Esc in order without unintended contacts (Fig. 7). All three policies use vision and proprioception, 30 demonstrations, 7,500 training steps, and ten trials. Full-sequence success is 90% for FLASH, 10% for FM-DiT, and 0% for DDPM-DiT (Table 2); full protocols are in Appx. H.



Figure 7: ***Keyboard Pressing*.** Press-and-release sequence: Up → Space → Esc.

Table 2: ***Keyboard Pressing* success (%).** Full success requires all three keys in order.

Policy	≥ 1 key	≥ 2 keys	All 3 keys
FLASH	**100**	**100**	**90**
FM-DiT	**100**	60	10
DDPM-DiT	90	20	0

Table 3: **Five-task component ablations.** Mean success (%) at the 10,000-step checkpoint of a 15,000-step schedule. NFE=1 except FLASH-G (NFE=10).

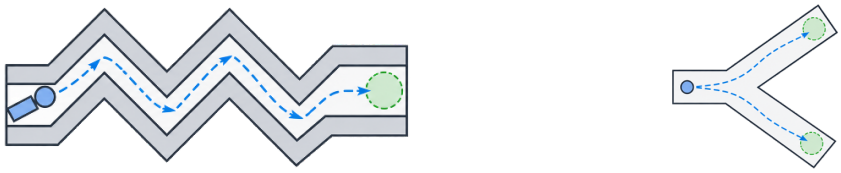
Configuration	Mean SR	Change (pp)
FLASH (default)	**96.8**	—
Dense sampling, $k = 1$	54.0	-42.8
Gaussian source (FLASH-G, NFE=10)	79.2	-17.6
No history-noise augmentation, $\sigma_h = 0$	52.4	-44.4
No history-fit regularization, $\lambda_h = 0$	87.6	-9.2
No consistency loss, $\lambda_{\text{cons}} = 0$	89.6	-7.2
No fit padding, no KKT	78.8	-18.0
Fit padding, no KKT	92.0	-4.8
No analytic velocity feed-forward	93.2	-3.6

4.5 Post-hoc execution speed modulation

Changing k_{eval} rescales playback without retraining. Tested *Stack Cube* speeds of $0.5\times$–$1.33\times$ training pace achieve $\geq 93\%$ success (Fig. 5, right). Extreme-speed results and feedback schedules are in Appx. G and K.

4.6 2D stress tests

We probe sharp within-chunk corrections in *Corridor Push* and alternative-goal coverage in *Fork Reach* (Fig. 8). Both use 100 demonstrations and 3,000-step checkpoints; protocols and comparisons are in Appx. I.



(a) *Corridor Push*: sharp within-chunk reversals (b) *Fork Reach*: alternative goals

Figure 8: **2D stress tests.** *Corridor Push* requires weaving through a narrow corridor; *Fork Reach* admits two distinct successful branches.

Sharp corrections. *Corridor Push* requires several reversals per eight-step open-loop chunk. Different degree FLASH models achieve 12% ($K = 6$) and 100% ($K = 9$) success over 50 episodes, exposing a degree-dependent limit. The latter matches A2A and VITA with lower RMS second-order action difference (21.7 versus 23.6 and 23.2; Table 10).

Alternative goals. *Fork Reach* pairs opposite-goal demonstrations from identical initial states; either goal succeeds. FLASH achieves 100% success over 48 episodes with 21/27 branch choices (balance 0.78), matching FM-DiT's balance; ACT selects one branch (Fig. 16).

5 Ablation study

Tables 3 and 4 summarize five-task ablations (50 episodes/task); Fig. 9 shows complementary sampling and control studies. Full results and protocols are in Appx. J.

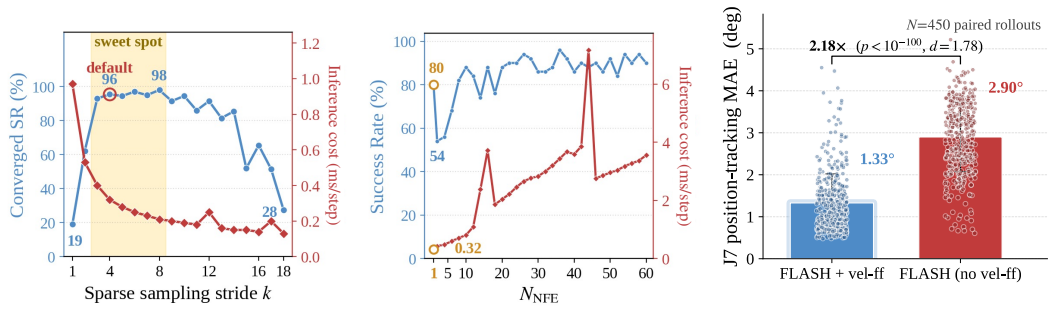


Figure 9: **Sampling, inference-budget, and control studies. Left:** stride on *Stack Cube* (400 demonstrations). **Middle:** NFE on *Close Box* (2,500-step checkpoint); red axes show amortized inference cost (ms/environment step). **Right:** velocity feed-forward over 450 paired rollouts. Protocols are in Appx. J.

Table 4: **Inference budget at 2,500 steps.** Five-task means; cost is ms/environment step. Complete sweeps are in Tables 21 and 22.

NFE	1	10	50	100
Mean success (%)	78.8	88.8	89.2	85.2
Mean cost (ms/step)	0.35	0.89	3.40	6.95

Representation and history. Sparse sampling and history-noise augmentation give the largest gains; FLASH-G changes both source and native NFE. Appx. J reports degree, stride, and history-sensitivity sweeps.

Fitting, supervision, and control. Padding and KKT correction improve mean success; regularization and consistency loss are beneficial but not essential. Velocity feed-forward improves success and tracking (Appx. J).

Inference budget. One-step Euler minimizes measured cost. Larger budgets can improve success at this checkpoint but incur higher cost and non-monotonic accuracy changes. Thus the default favors efficiency (Table 4).

6 Conclusion

FLASH combines sparse polynomial trajectory representation, history-anchored flow, and analytic velocity feed-forward to connect efficient single-step policy inference with precise high-frequency control. Evaluation across 11 tasks supports its efficiency and contact-rich manipulation performance. The results also motivate selecting the representation and feedback schedule for the motion demands of the task.

7 Limitations

The fixed polynomial degree remains task-dependent: $K = 6$ performs well on the nine manipulation tasks but reaches only 12% success on *Corridor Push*, where a $K = 9$ model reaches 100%. Severe history corruption also degrades performance, and the coefficient-noise probe does not establish recovery from unseen task failures. The current vision-and-proprioception setting for visuomotor policies does not evaluate force-regulated control or arbitrary high-speed impacts.

Acknowledgments and Disclosure of Funding

This work is jointly supported by the Singapore Ministry of Education (MOE) Tier 1 grants RG83/25 and RS36/24, and a start-up grant from Nanyang Technological University. The authors have no other competing interests to declare.

References

[1] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*.

[2] 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.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[4] João Carvalho, An T. Le, Piotr Kicki, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and adapting robot motion planning with diffusion models. *IEEE Transactions on Robotics*, 41:4881–4901, 2025.

[5] 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*, 44(10–11):1684–1704, 2025.

[6] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[7] Jiahua Dong, Qi Lyu, Baichen Liu, Xudong Wang, Wenqi Liang, Duzhen Zhang, Jiahang Tu, Hongliu Li, Hanbin Zhao, Henghui Ding, et al. Learning to model the world: A survey of world models in artificial intelligence. *TechRxiv preprint*, 2026.

[8] Dechen Gao, Boqi Zhao, Andrew Lee, Ian Chuang, Hanchu Zhou, Hang Wang, Zhe Zhao, Junshan Zhang, and Iman Soltani. VITA: Vision-to-action flow matching policy. In *International Conference on Learning Representations*, 2026.

[9] Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, et al. RoboVerse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

[10] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[13] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5 (2):3019–3026, 2020.

[14] Jindou Jia, Gen Li, Xiangyu Chen, Tuo An, Yuxuan Hu, Jingliang Li, Xinying Guo, and Jianfei Yang. Action-to-action flow matching. *arXiv preprint arXiv:2602.07322*, 2026.

[15] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

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

[17] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

[18] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023.

[19] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021.

[20] Khang Nguyen, An T. Le, Tien Pham, Manfred Huber, Jan Peters, and Minh Nhat Vu. FlowMP: Learning motion fields for robot planning with conditional flow matching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 11291–11297. IEEE, 2025.

[21] Chaoyi Pan, Giridharan Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. In *International Conference on Learning Representations*, 2026.

[22] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[23] Alfredo Reichlin, Giovanni Luca Marchetti, Hang Yin, Ali Ghadirzadeh, and Danica Kragic. Back to the manifold: Recovering from out-of-distribution states. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 8660–8666. IEEE, 2022.

[24] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[25] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[26] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *Proceedings of the European Conference on Computer Vision*, pages 87–103. Springer, 2024.

[27] Antareep Singha, Laksh Nanwani, Samkit Jain, Phani Teja Singamaneni, Arun Kumar Singh, K Madhava Krishna, et al. Crowd-FM: Learned optimal selection of conditional flow matching-generated trajectories for crowd navigation. *arXiv preprint arXiv:2602.06698*, 2026.

[28] Rohan Sinha, Apoorva Sharma, Somrita Banerjee, Thomas Lew, Rachel Luo, Spencer M. Richards, Yixiao Sun, Edward Schmerling, and Marco Pavone. A system-level view on out-of-distribution data in robotics. *arXiv preprint arXiv:2212.14020*, 2022.

[29] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[30] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[31] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proceedings of the International Conference on Machine Learning*, pages 32211–32252, 2023.

[32] Murray R. Spiegel. *Mathematical handbook of formulas and tables*. Schaum's Outline Series. McGraw-Hill, New York, 1968.

[33] Gábor Szegő. *Orthogonal polynomials*, volume 23. American Mathematical Society, 1939.

[34] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[35] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[36] Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, et al. BEAST: Efficient tokenization of B-splines encoded action sequences for imitation learning. *arXiv preprint arXiv:2506.06072*, 2025.

A Cross-horizon continuity and two-tier extension

This section provides the full derivation of the cross-horizon continuity mechanism summarized in Section 3.1.

First extension: overlap and exact anchors. Let T_{exec} be the number of sparse action nodes executed before replanning. The predicted polynomial extends this execution slice by μ_{pre} nodes backward and μ_{post} nodes forward. This forms an extended evaluation window with H_{poly} sparse nodes:

$$H_{\text{poly}} \;=\; \mu_{\text{pre}} + T_{\text{exec}} + \mu_{\text{post}}. \tag{10}$$

These $\mu_{\text{pre}} + \mu_{\text{post}}$ overlap steps are supervised during training, but discarded before deployment. On the padded fitting grid defined below, the front anchor has zero-based index $\delta + \mu_{\text{pre}}$ and the rear anchor has index $\delta + \mu_{\text{pre}} + T_{\text{exec}}$, the first node after the execution slice. At these anchors, we enforce the fitted polynomial's kinematic states to match the expert trajectory.

For two distinct anchors, matching derivatives through order r gives $2(r+1)$ constraints per action dimension. Let $\mathbf{A}$ stack their basis evaluations and $\mathbf{b}$ their expert targets. Provided that the fitting matrix $\mathbf{S}$ has full column rank and $\mathbf{A}$ has full row rank, the closed-form *KKT* solution [17] corrects the coefficients:

$$\mathbf{C}_{\text{c}}^{\star} \;=\; \mathbf{C}^{\star} \;-\; (\mathbf{S}^{\top}\mathbf{S})^{-1}\mathbf{A}^{\top}\big[\mathbf{A}(\mathbf{S}^{\top}\mathbf{S})^{-1}\mathbf{A}^{\top}\big]^{-1}\big(\mathbf{A}\mathbf{C}^{\star} - \mathbf{b}\big). \tag{11}$$

These rank conditions require at least $K+1$ distinct fitting nodes and $K+1 \geq 2(r+1)$. Thus C^1 fitting requires at least degree three, while C^3 fitting generally requires at least degree seven. Numerical conditioning remains relevant even when these dimension requirements hold. The exact constraints apply to fitted training targets; the neural predictions are not projected onto this constraint set at inference.

Second extension: fit padding. For *Legendre* bases, the functions reach extreme values at the boundaries, making the least-squares solution highly sensitive to target perturbations. To prevent degradation of the fit quality near the anchors, we append δ padding steps to both sides, expanding the full regression window:

$$H_{\text{fit}} \;=\; H_{\text{poly}} + 2\delta \;=\; \delta + \mu_{\text{pre}} + T_{\text{exec}} + \mu_{\text{post}} + \delta. \tag{12}$$

The basis matrix $\mathbf{S}$ and the normalized time grid are constructed on all H_{fit} nodes. Their first-to-last duration therefore contains $H_{\text{fit}} - 1$ sampling intervals, which sets T in Eq. (4). These 2δ padding nodes serve purely as a data-preprocessing method. They act as additional regression observation points to pull the constraint anchors into the stable interior of the fitting window, without receiving direct training supervision or being output by the network.

B Evaluation setup and details

Tasks and demonstrations. We evaluate FLASH on 11 tasks: five simulated manipulation tasks, four real-world Franka tasks (*Place Cube*, *Insert Cube*, *Keyboard Pressing*, and *Push-T*), and two 2D stress tests (*Corridor Push* and *Fork Reach*). The real-world protocols are detailed in Appx. H, with tracking and insertion analyses in Appx. F; the 2D protocols are in Appx. I. The five simulated manipulation tasks are *Close Box* from RLBench [13], *Stack Cube* and *Pick Cube* from ManiSkill [19] (using the Isaac simulator), as well as *Pick-Place Bowl* and *Open Drawer* from LIBERO [16] (using the MuJoCo simulator). We use 99 valid expert demonstrations for *Close Box*, 100 each for *Pick Cube* and *Stack Cube*, and 40 each for *Pick-Place Bowl* and *Open Drawer*. For the five-task comparison, all policies use a training schedule configured for $15{,}000$ optimizer steps, with checkpoints saved every $1{,}250$ steps. Table 1 reports the $10{,}000$-step checkpoint, evaluated over 50 rollouts with randomized initial states. The learning-rate schedule retains its $15{,}000$-step horizon when evaluating intermediate checkpoints.

Comparison methods and hyperparameters. The comparison includes ten external baselines: DDPM-UNet, DDPM-DiT, DDIM-UNet, FM-UNet, FM-DiT, Score-UNet, ACT, VITA, A2A-Noise, and MIP. FLASH-G is an internal Gaussian-source control. All ten external baselines are evaluated on the five simulated manipulation tasks; the real-world and 2D experiments use the

method subsets specified in their respective protocols. For the five-task manipulation comparison, the methods share a ResNet-18 [11] visual encoder, observation horizons, batch size 32, and optimizer schedule. The encoder is trained from scratch with group normalization, except ACT, which uses ImageNet initialization and frozen batch normalization; the only varying factors are the algorithmic core and method-specific settings. Through extensive hyperparameter search, we find that A2A-Noise achieves its best performance with a noise scale of $\epsilon = 0.02$ and a single-step (NFE $= 1$) inference, which outperforms A2A with 6-step inference. We therefore report results with this best-performing configuration. For FLASH and FLASH-G, we use temporal sampling stride $k = 4$, Legendre degree $K = 6$, and FLASH's unique setting is: a single-step Euler solver, the noise scale injected to the history $\sigma_h = 0.5$, *Tikhonov* regularization term $\lambda_h = 0.1$, and the weight for consistency loss $\lambda_{\text{cons}} = 1.0$.

The training recipe uses AdamW with learning rate 10^{-4}, $\beta = (0.95, 0.999)$, $\epsilon = 10^{-8}$, and weight decay 10^{-6}. A cosine learning-rate schedule follows 500 warmup steps. The FLASH manipulation configuration uses a horizon of 16, including eight observation steps and six executed sparse action steps per prediction, with one prefix point, one suffix point, and one fit-padding point on each side. FLASH training for the five manipulation tasks used a single NVIDIA RTX 5090. The public repository provides the training configuration and reproduction commands at `https://github.com/NTUMARS/FLASH-Policy`.

Gripper representation in simulation. The two finger-joint position targets are continuous action dimensions, represented with the same Legendre basis, coefficient normalization, and training loss as the arm joints. The decoded targets are tracked directly by the joint-position controller, with no separate discrete head or execution-time thresholding. Expert signals are step-like in *Pick Cube*, *Stack Cube*, and *Close Box*; *Pick-Place Bowl* has a short multi-step closing transition, while *Open Drawer* changes little. The polynomial provides a continuous approximation to these demonstrated trajectories.

B.1 Baseline architectures and parameter counts

Counting protocol and visual encoders. We instantiate each policy with one head camera, a 9-dimensional proprioceptive state, 9-dimensional actions, and an action horizon of 16. Table 5 counts trainable parameters at initialization, excluding buffers and data-dependent `LinearNormalizer` statistics. The shared visual encoder is a randomly initialized ResNet-18 with GroupNorm, containing 11,176,512 parameters. Its 512-dimensional image feature is concatenated with the 9-dimensional state to give a 521-dimensional observation feature. ACT is an explicit exception: it uses an ImageNet-initialized ResNet-18 with frozen BatchNorm in its DETR pipeline. Its normalization affine terms are buffers, yielding 11,166,912 encoder parameters, 9,600 (approximately 0.09%) fewer than the GroupNorm encoder.

Table 5: **Exact parameter counts and native inference budgets.** Encoder, generator-head, and total counts use the protocol above. All encoders use scratch initialization and GroupNorm except ACT (ImageNet initialization and frozen BatchNorm). FLASH-G is an internal ablation. MIP is evaluated at its native NFE=2 and with an NFE=1 variant.

Method	Generator	Encoder	Head	Total	NFE
FLASH	DiT	11,176,512	21,839,369	33,015,881	1
FM-DiT	DiT	11,176,512	21,839,369	33,015,881	10
DDPM-DiT	DiT	11,176,512	21,839,369	33,015,881	100
MIP	DiT	11,176,512	21,839,369	33,015,881	2
FLASH-G	DiT	11,176,512	21,839,369	33,015,881	10
VITA	MLP + VAE	11,176,512	20,608,144	31,784,656	6
A2A-Noise	MLP + VAE	11,176,512	23,480,392	34,656,904	1
ACT	DETR CVAE	11,166,912	7,542,602	18,709,514	1
FM-UNet	UNet-1D	11,176,512	124,974,857	136,151,369	10
DDIM-UNet	UNet-1D	11,176,512	122,745,993	133,922,505	40
DDPM-UNet	UNet-1D	11,176,512	124,974,857	136,151,369	100
Score-UNet	UNet-1D	11,176,512	124,974,857	136,151,369	100

Generator architectures. FLASH, FLASH-G, FM-DiT, DDPM-DiT, and MIP share a four-layer AdaLN DiT with hidden width 512 and eight attention heads. Its 21,839,369 parameters are independent of token-sequence length: the input and output projections are applied per token ($9 \to 512$ and $512 \to 9$), and the rotary-position cache is a buffer. Thus FLASH's seven coefficient tokens and the discrete-action models' 16 action tokens have the same parameter count. VITA and A2A-Noise use an MLP flow network with an action VAE in latent action space. ACT uses a DETR CVAE transformer with width 256, two encoder layers, four decoder layers, and four attention heads. The conditional UNet-1D baselines use channel widths $[256, 512, 1024]$.

Which baseline is closest? FM-DiT, DDPM-DiT, and MIP tie exactly with FLASH in encoder and total parameter count. Among these external baselines, FM-DiT also shares the flow-matching learning paradigm, making it the closest comparator on that additional axis. Table 6 separates these shared architectural properties from algorithmic differences. Equal counts establish capacity parity, not identical learned weights or training objectives. In particular, the comparison also differs in native NFE, consistency supervision, and the use of continuity-constrained polynomial training targets. FLASH-G shares the polynomial pipeline but uses a Gaussian source; its source comparison is presented in Appx. J.1.

Table 6: **FLASH versus FM-DiT.** Shared architecture and capacity do not imply identical generation or training procedures.

Property	FLASH	FM-DiT
Encoder	ResNet-18, scratch, GN	ResNet-18, scratch, GN
Generator	DiT, 512×4, 8 heads	DiT, 512×4, 8 heads
Total parameters	33,015,881	33,015,881
Learning paradigm	Flow matching	Flow matching
Source	History coefficients	Gaussian noise
Representation	Legendre coefficients	Discrete actions
Native NFE	1	10
Consistency loss	Included	Not included
Training-target continuity	KKT-constrained fit	No polynomial constraint

MIP implementation. Our MIP implementation ports the official loss and sampler [21], with numerical-oracle checks, into the same backbone, encoder, and training recipe used for FM-DiT. Its training and learning-rate schedules are configured for 15,000 optimizer steps, including when an intermediate checkpoint is evaluated. Table 1 reports both NFE=2 and NFE=1 at the 10,000-step checkpoint with 50 rollouts per task. Its checkpoint curves appear in Appx. D.

Controller interface and comparison scope. In the reported real-robot torque-control comparison, every method supplies commands to the same 1,000 Hz controller. Discrete-action baselines use 1,000 Hz linear interpolation and numerical velocity estimates, whereas polynomial policies are evaluated directly at the controller rate with analytic derivatives. This matches the controller rate and interface. Table 1 reports each method's inference budget, including both the one- and two-step MIP configurations.

C Success rate analysis

Table 1 presents the success rates of the ten external baselines and FLASH across the five simulated tasks at the 10k-step checkpoints of runs configured for 15k optimizer steps:

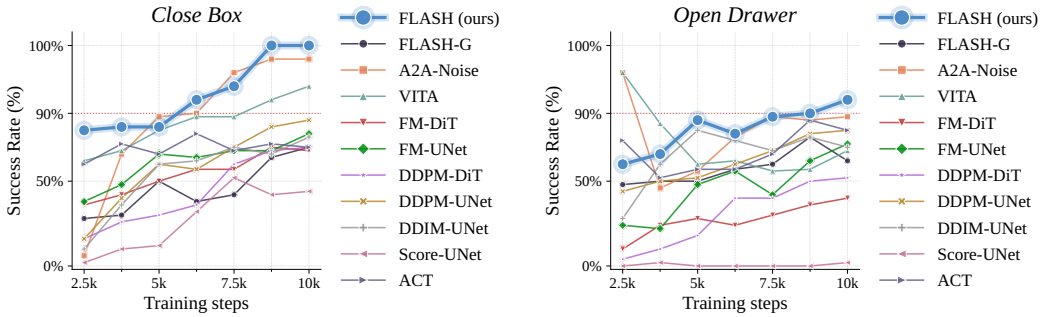
- **State-of-the-art performance:** FLASH ranks first on 4 out of 5 tasks, ties for first with ACT on *Pick Cube*, and is the only method to achieve $\geq 92\%$ success rate across all five tasks, requiring only a single sampling step (NFE=1) during inference.
- **Advantage over the strongest baseline:** FLASH's success rates are 2 to 8 percentage points higher than A2A-Noise's (4.8 points on average). Both use NFE=1, while their representations and training procedures differ.

FLASH combines smooth within-chunk polynomial trajectories with a temporally downsampled H_{poly} window that extends the physical horizon by a factor of $k = 4$. The KKT-constrained fit

(Eq. (11)) additionally imposes position and velocity continuity on the fitted training targets. These design choices may contribute to the observed success rates, but this comparison does not isolate their individual effects or guarantee continuity between predicted chunks. The component comparisons in Appx. J.1 examine these effects and their remaining confounds.

D Training efficiency: full curves

Fig. 10 shows the training curves for the two tasks not included in the main text. On *Close Box*, FLASH reaches 100% at 10 k steps while most baselines plateau below 90%. On *Open Drawer*, FLASH converges to 92% at 10 k steps, while the strongest competitor A2A-Noise reaches 88%, a gap of 4 percentage points.



(a) *Close Box* (b) *Open Drawer*

Figure 10: **Training efficiency on remaining tasks.** Same protocol as Fig. 4.

MIP learning curves. Tables 7 and 8 report MIP's checkpoint results at NFE=2 and NFE=1, respectively, using 50 rollouts per task and the shared 15,000-step training and learning-rate schedule. The 10,000-step results in Table 1 give mean success rates of 52.8% and 51.2%, respectively, compared with FLASH's 96.8%. Thus a small NFE alone does not reproduce FLASH's performance in this protocol. Fifty trials per task limit the resolution of small success-rate differences.

Table 7: **MIP learning curves at NFE=2.** Success rates (%) at the indicated optimizer step.

Task	2,500	3,750	5,000	6,250	7,500	8,750	10,000
Close Box	78	74	56	42	74	86	86
Pick Cube	46	48	64	82	86	88	88
Stack Cube	0	2	4	6	12	22	28
Open Drawer	0	6	22	34	32	36	44
Pick-Place Bowl	66	46	46	34	36	24	18

Table 8: **MIP learning curves at NFE=1.** Success rates (%) at four training checkpoints.

Task	2,500	5,000	7,500	10,000
Close Box	42	34	46	68
Pick Cube	62	86	92	92
Stack Cube	4	32	38	48
Open Drawer	58	36	44	46
Pick-Place Bowl	54	6	4	2

At 10,000 steps, a second MIP function evaluation improves *Close Box* and *Pick-Place Bowl* by 18 and 16 percentage points, respectively, but lowers *Stack Cube* by 20 points. The five-task means are therefore close (52.8% at NFE=2 and 51.2% at NFE=1), despite task-dependent differences. The checkpoint curves also show that a later checkpoint need not improve every task. These results describe the shared training protocol used here and do not establish MIP's performance under different training budgets or model-selection procedures.

E Inference cost and task completion time

We distinguish amortized inference cost from cumulative episode compute and physical completion time. The logged `Average Inference Time` is accumulated inference time divided by environment steps, as defined in Sec. 4.2.

Amortized inference cost. In Fig. 5, FLASH requires 0.32 ms/environment step, compared with 36.21 for DDPM-UNet, 0.54 for A2A-Noise, and 1.33 for the Gaussian-source ablation FLASH-G. These correspond to cost ratios of approximately $113\times$, $1.7\times$, and $4.2\times$, respectively.

Cumulative episode inference cost. Figure 11 accumulates the time of all inference calls within successful *Stack Cube* rollouts. FLASH requires 31.4 ms per successful episode, compared with 5,476 ms for Score-UNet, 69 ms for A2A-Noise, and 159 ms for FLASH-G. The corresponding cost ratios are $174.4\times$, $2.2\times$, and $5.1\times$. This deployment-level quantity depends on the episode and execution schedule.

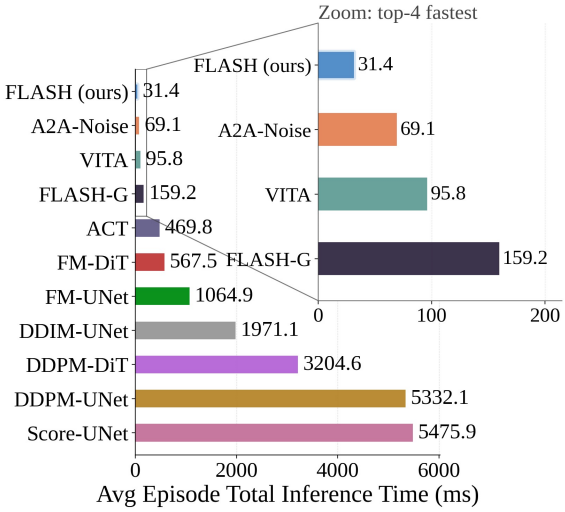


Figure 11: **Cumulative episode inference cost on *Stack Cube*.** Total inference time within successful rollouts for the measured policy subset on an NVIDIA RTX 5090; the inset enlarges the four fastest policies.

Real-world task wall time. Fig. 12 presents execution snapshots of the real-world *Place Cube* task and compares the wall-clock time across 6 representative policies over 5 rollouts. Task-completion time includes computation, interpolation/communication, and mechanical response. FLASH completes the task in 8.00 ± 0.24 s on average, the fastest among the 6 policies:

- $2.16\times$ faster than DDPM-UNet (17.29 s) and $2.13\times$ faster than DDPM-DiT (17.00 s);
- $1.29\times$ faster than the homologous FLASH-G (10.34 s);
- Saves 45% of wall-clock time compared to the baseline average of 14.53 s.

FLASH's completion-time standard deviation across these 5 rollouts is 0.24 s, approximately $13\times$ smaller than DDPM-DiT's 3.21 s. This describes the observed variability in this set of trials.

Sources of the inference-cost reduction. Sparse sampling and history anchoring affect different parts of the computation. With $k = 4$, a polynomial spans a longer physical interval, allowing its coefficients to be reused for more environment steps. Separately, FLASH uses one generator evaluation per call, whereas FLASH-G uses ten. FLASH-G's 1.33 ms/environment step and 159 ms/episode versus FLASH's 0.32 and 31.4 illustrate the combined effect of the source and native sampling budget.

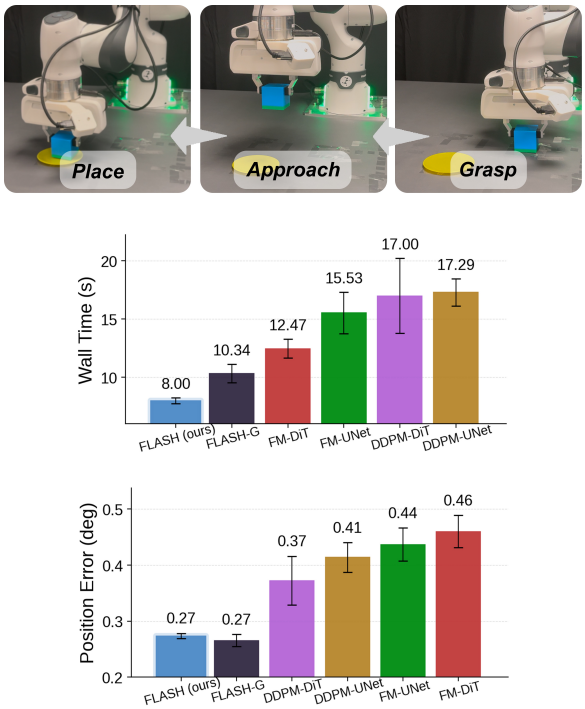


Figure 12: **Real-world *Place Cube*.** Top: Execution sequence. Middle: Task completion wall time. Bottom: Tracking error.

F Trajectory smoothness and tracking error: detailed analysis

Table 9: **Real-world *Insert Cube* success rates (%).** Each policy is evaluated over ten trials. FLASH-G is the internal Gaussian-source control.

Policy	Success rate (%)
FLASH	**100**
DDIM-UNet	80
A2A-Noise	60
FLASH-G	60
FM-UNet	60
FM-DiT	50
DDPM-DiT	50
DDPM-UNet	10

This section provides the full analyses summarized in Section 4.3.

Time-series tracking error (full analysis). In Fig. 5 (middle), we compute the sum of absolute tracking errors across the 7 arm joints (J3 to J9), given by $\sum_{j=3}^{9} |e_j(t)|$. Both the mean and peak errors of FLASH are lower than those of FM-DiT (a mean gap of $4.70\times$ and a peak gap of $3.18\times$). In this representative rollout, FM-DiT's error curve exhibits spikes at chunk boundaries, consistent with discontinuities in the supplied command sequence. FLASH completes the task in 0.84 s, compared with FM-DiT's 1.22 s ($1.45\times$ longer), and has both a shorter duration and lower tracking errors in this comparison.

Cumulative tracking error IAE. We also employ the Integral of Absolute Error (IAE), a standard metric in control theory defined as $\text{IAE} = \int \sum_{j=3}^{9} |e_j(t)|\, dt$ (7 joints / unit: deg · s), to aggregate

the compound cost of "duration $\times$ magnitude". For the same *Pick Cube* rollout, FLASH achieves 1.80 deg $\cdot$ s, while FM-DiT scores 12.23 ($6.80\times$ FLASH).

Real-world control stack (Fig. 12(bottom)). We operate the Franka robot in **joint torque control** mode, where the host directly computes and sends torque signals to the Franka. In this real-robot comparison, FLASH and FLASH-G are evaluated directly at 1000 Hz, while the other 4 baselines use 1000 Hz linear interpolation. FLASH and FLASH-G provide the desired velocity $\dot{q}_d$ through analytic polynomial differentiation (Eq. 4); the discrete-action baseline implementations use numerical velocity estimates. The resulting tracking errors reflect the policies together with these controller interfaces and the robot dynamics. The rates in this paragraph describe the real-robot experiment; the five-task simulation study resamples the decoded polynomial at 66.7 Hz (Table 13).

We measure the control-level 7-joint average tracking MAE over 6 policies $\times$ 5 rollouts. FLASH (0.274 $\pm$ 0.004 deg) and FLASH-G (0.266 $\pm$ 0.011 deg) differ in mean by 0.008 deg. DDPM-DiT, DDPM-UNet, FM-UNet, and FM-DiT yield 0.373, 0.414, 0.437, and 0.460 deg, respectively (representing a $1.36\times$ to $1.68\times$ error increase relative to FLASH). FLASH's inter-rollout standard deviation is 0.004 deg, the smallest among the evaluated policies and approximately an order of magnitude lower than DDPM-DiT's 0.043 deg in these five-rollout comparisons.

High-precision real-world *Insert Cube* task (full analysis). We validate FLASH under strict high-precision constraints on the real-world *Insert Cube* task (a precision insertion with millimeter-level tolerance). All models are trained with 10k optimizer steps and evaluated over ten trials per policy. The three subfigures of Fig. 6 chronologically illustrate the execution process (grasp $\rightarrow$ approach $\rightarrow$ insert). Table 9 reports success rates for all eight policies: FLASH reaches 100%, exceeding the other seven policies by 47.1 percentage points on average. DDPM-UNet reaches 10% on this task.

The simulated tracking comparison (Fig. 5, middle), real-world tracking measurements (Fig. 12, bottom), and *Insert Cube* success rates (Table 9) favor FLASH over the evaluated non-polynomial baselines. These results support the combined polynomial pipeline under the tested controller interfaces.

G Post-hoc execution speed modulation

We empirically validate the post-hoc speed modulation property established in Section 3.1 on *Stack Cube* using the 10 k step FLASH checkpoint. We fix the training stride $k_{\text{train}} = 4$ and sweep the evaluation stride $k_{\text{eval}} \in \{1, \ldots, 12, 16, 32\}$, which spans playback speeds from $4\times$ acceleration to $8\times$ deceleration relative to the training pace. Figure 5 (right) summarizes the resulting success rates.

FLASH maintains $\geq 93\%$ success across $k_{\text{eval}} \in \{3, 5, 6, 7, 8\}$, spanning $0.5\times$ to $1.33\times$ playback speed. The observed success rates at $k_{\text{eval}} \in \{5, 7\}$ are 98%, compared with 96% at the training value $k_{\text{eval}} = 4$. At the extremes, $4\times$ acceleration yields 26% success, whereas the tested slowdowns of $2.5\times$ or more yield no successes. This sweep establishes sensitivity to playback speed.

H Real-world task protocols

The four real-world tasks test placement (*Place Cube*), precision insertion (*Insert Cube*), ordered key contacts (*Keyboard Pressing*), and repeated object pushing (*Push-T*). *Place Cube* is evaluated across six representative policies with five rollouts each, with wall time and tracking error reported in Appx. E and F. *Insert Cube* uses ten trials per policy after 10,000 optimizer steps (Table 9). *Keyboard Pressing* and *Push-T* each use 30 demonstrations, 7,500 optimizer steps, and 10 evaluation trials per policy. For *Keyboard Pressing* and *Push-T*, FLASH, FM-DiT, and DDPM-DiT use the same task-specific protocol. These experiments use visual observations and proprioception, without force or tactile feedback.

Keyboard Pressing. The robot must press and release Up, Space, and Esc in that order. A valid press must avoid unintended key contacts, and the subsequent retraction must avoid neighboring keys. The task alternates free-space motion and contact, including sharp press–release reversals. Table 2 reports nested completion criteria: at least one of the three target keys correctly pressed and released, at least two such keys, and all three in the prescribed order. The first two levels need not be the first one or two keys in the sequence; each counted press and release must avoid neighboring keys. Complete success additionally requires no unintended key contact anywhere in the trajectory.



Figure 13: ***Keyboard Pressing.*** Execution snapshots of the ordered three-key task. Each press is followed by a retraction before the next contact.

FLASH completes the full sequence in 9 of 10 trials, compared with 1 of 10 for FM-DiT and 0 of 10 for DDPM-DiT. The intermediate criteria distinguish successful initial contact from maintaining precision across repeated contacts. These results establish performance on this protocol.

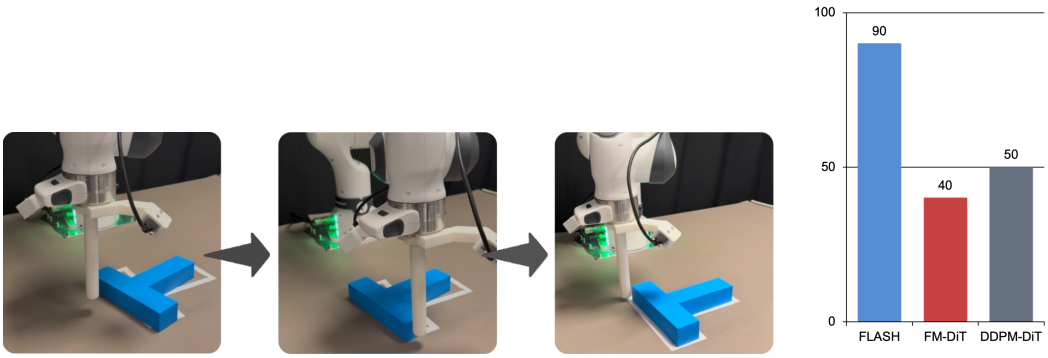
Push-T. The robot pushes a T-shaped block from different sides and angles through repeated contacts until it fully overlaps a target region. Figure 14a shows execution snapshots, and Fig. 14b reports success rates. FLASH succeeds in 9 of 10 trials (90%), FM-DiT in 4 of 10 (40%), and DDPM-DiT in 5 of 10 (50%). This complements *Keyboard Pressing* with repeated changes in pushing direction and object contact.

I 2D stress tests

We use two controlled tasks to probe different properties: sharp within-chunk changes in *Corridor Push*, and coverage of alternative successful strategies in *Fork Reach*. Each task uses 100 expert demonstrations. These tests complement the robot manipulation benchmarks.

I.1 *Corridor Push*: sharp reversals within a chunk

A pusher moves a puck through a sawtooth corridor with 8.5-pixel clearance per side and a sharp reversal every 50 pixels. Every policy executes an eight-step action chunk without replanning inside that chunk. A chunk spans approximately 134 pixels and must realize two to three reversals, and up to four in some segments. All reported methods use the same 100 demonstrations, each 20–23 steps long. We evaluate the 3,000-step checkpoint over 50 episodes with environment seed 10000. The smoothness statistic called *jerk* is the RMS second-order action difference; it is a discrete action-space measure, not a physical third derivative of position.



(a) Successive execution stages (b) Success rate (%)

Figure 14: **Real-world *Push-T*.** The block is pushed into its target through multiple contacts. Each policy is evaluated in ten trials after training on 30 demonstrations for 7,500 optimizer steps.



(a) *Corridor Push* (b) *Fork Reach*

Figure 15: **Two-dimensional stress-test environments.** *Corridor Push* requires repeated sharp reversals within an open-loop action chunk. *Fork Reach* admits two distinct successful goal choices.

Table 10: ***Corridor Push*.** Success rate and RMS second-order action difference; lower values of the latter indicate smoother action changes.

Policy	Success rate (%) ↑	Jerk ↓
FLASH, $K = 6$	12	29.6
FLASH, $K = 9$	**100**	**21.7**
A2A	**100**	23.6
VITA	**100**	23.2
DDIM-UNet	96	26.1
DDPM-UNet	94	27.0
FM-UNet	92	26.8
FM-DiT	66	37.0
Score-UNet	22	54.7
DDPM-DiT	12	45.2

The default $K = 6$ fails on most episodes (12%), whereas retraining with a fixed $K = 9$ reaches 100% with lower action-difference RMS than the evaluated baselines. This is a concrete limit of the default degree. $K = 9$ reduces average success on the five simulated manipulation tasks from 96.8% to 65.2% (Table 16).

I.2 *Fork Reach*: alternative successful goals

Reaching either the upper or lower goal region is successful. The training dataset contains 100 expert demonstrations organized as 50 opposite-goal pairs sharing identical initial states: 50 trajectories reach the top goal and 50 the bottom goal. Each trajectory has 18 steps and is collected without

retries. FLASH, FM-DiT, and ACT all use the same 10,000-step training and learning-rate schedule; we report their intermediate 3,000-step checkpoints, not models trained with a separate 3,000-step schedule. Evaluation uses 48 held-out episodes with evaluation seed 10000.

Direction choices and success. At the end of every episode, including failures, we classify the agent's final vertical coordinate y as *top* if $y < 210$, *bottom* if $y > 302$, and *middle* otherwise. The thresholds are 46 pixels above and below the initial center line $y = 256$, halfway toward the respective goals. Middle episodes are counted separately and excluded from the two branch counts. Mode balance is $\min(n_{\text{top}}, n_{\text{bottom}}) / \max(n_{\text{top}}, n_{\text{bottom}})$. The 24/24 expert split is a reference over the 48 evaluation episodes; the training dataset instead has a 50/50 split. Success rate separately measures goal arrival.

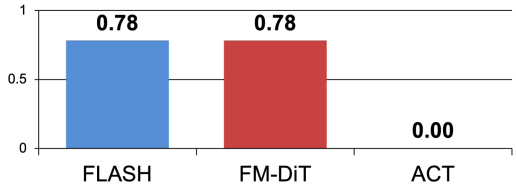


Figure 16: ***Fork Reach* mode balance.** Top/bottom direction counts are 21/27 for FLASH and 27/21 for FM-DiT (balance 0.78 for both); ACT chooses the top branch in all 48 trials (48/0; balance 0). These counts include failed episodes; all three methods have zero middle episodes. Success rates are 100%, 98%, and 100%, respectively.

FM-DiT succeeds in 47 of 48 episodes (98% after rounding); its single failed episode is included in the bottom count. FLASH and FM-DiT cover both branches in this test, whereas ACT selects a single branch despite achieving 100% task success. The reported action-difference RMS is 3.3 pixels for FLASH and 8.4 pixels for FM-DiT. Thus success rate alone does not characterize branch coverage over the evaluation set. FLASH adds no inference-time history noise in this test. These aggregate counts establish that both branches occur across the held-out episodes.

J Ablation studies

This section groups component comparisons, sensitivity sweeps, and control studies by design choice. The five-task sweeps vary one setting at a time, with all other settings fixed at their defaults. These runs use seed 42 and a 15,000-step training and learning-rate schedule. Unless otherwise stated, we evaluate the 10,000-step checkpoint over 50 episodes per task; the NFE sweep instead uses the 2,500-step checkpoint. Success rates are in percent, and means weight the five tasks equally. Bold rows identify the default configuration. *P-P Bowl* denotes *Pick-Place Bowl*. The *Stack Cube* learning-curve study uses 400 demonstrations; the *Close Box* continuity study averages four checkpoints; and the velocity control study uses 450 paired rollouts. Their protocols are specified with the corresponding results below.

J.1 Component overview

Table 11 summarizes the component comparisons across all five simulated manipulation tasks. The topic-specific subsections report the full task results together with learning curves, checkpoint-averaged continuity measurements, and tracking errors under their respective protocols.

Table 11: **Five-task component comparisons.** Mean success rates (%). FLASH-G is an internal control using Gaussian-source flow in the same sparse polynomial representation; it uses its native NFE of 10, whereas FLASH uses 1.

Configuration	Mean SR	Change from FLASH (pp)
FLASH (default)	**96.8**	—
Dense sampling, $k = 1$	54.0	-42.8
FLASH-G (Gaussian source, NFE=10)	79.2	-17.6
No consistency loss, $\lambda_{\text{cons}} = 0$	89.6	-7.2
No fit padding, no KKT	78.8	-18.0
Fit padding, no KKT	92.0	-4.8
No analytic velocity feed-forward	93.2	-3.6

Sparse sampling and continuity. Changing the stride from $k = 1$ to the default $k = 4$ improves mean success by 42.8 points. The five-task sweep has a broad high-performance region at $k \in [3, 6]$ (88.8–96.8%). Fit padding raises the no-KKT mean from 78.8% to 92.0%, and KKT correction of the training targets adds 4.8 points. The *Stack Cube* learning curves and *Close Box* checkpoint averages below assess these components under complementary protocols.

History source and polynomial representation. FLASH-G shares FLASH's architecture, sparse polynomial representation, and 33,015,881 parameters, but generates from Gaussian noise. Its five task scores are 70/94/82/62/88%, averaging 79.2%, versus FLASH's 96.8%. Both FLASH-G and FM-DiT use NFE=10; their 79.2% versus 62.4% means compare the sparse polynomial pipeline with discrete actions. These contrasts support the combined design.

Table 12: **Polynomial-pipeline and source comparisons.** Success rates (%) at 10,000 steps. Native NFE differs between FLASH and FLASH-G; FLASH-G and FM-DiT also differ in representation, horizon, and controller signals.

Method	NFE	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
FLASH	1	100	98	96	92	98	96.8
FLASH-G	10	70	94	82	62	88	79.2
FM-DiT	10	70	92	46	36	68	62.4

Degree, regularization, and history sensitivity. The fixed-degree comparison peaks at $K = 6$ (96.8%); $K = 4$ retains 89.6%, while $K = 8$ and 9 reach 73.2% and 65.2%. The same $\lambda_h = 0.1$ and $\lambda_{\text{cons}} = 1$ defaults are used across tasks. Removing training-time history noise lowers mean success to 52.4% (-44.4 points); removing consistency supervision lowers it by 7.2 points. Perturbing the history anchor only at inference lowers mean success from 96.8% to 89.2% at noise scale 0.5, and to 31.6% at scale 1.0, identifying a limit of robustness to inaccurate histories.

NFE and velocity feed-forward. At the early 2,500-step checkpoints, five-task mean success rises from 78.8% at NFE=1 to a maximum of 89.2% at NFE=50, while amortized inference cost rises from 0.35 to 3.40 ms/environment step (Appx. J.7). Removing analytic velocity feed-forward lowers five-task success by 3.6 percentage points. Its effect is more pronounced in tracking precision: the separate paired control study reports J7 MAE increasing from 1.33° to 2.90° across 450 rollout pairs (Appx. J.8). Success and tracking metrics therefore measure different benefits.

J.2 Sparse sampling stride k

Table 13: **Sparse-sampling sensitivity across all five tasks.** The ratio k (also denoted r in the experiment configuration) gives a training-data sampling rate of $66.7/k$ Hz; the decoded polynomial is resampled at the simulation control rate of 66.7 Hz regardless of k.

k	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
1	24	80	14	70	82	54.0
2	64	92	62	72	98	77.6
3	88	100	86	82	100	91.2
4	**100**	**98**	**96**	**92**	**98**	**96.8**
5	76	92	94	82	100	88.8
6	86	92	98	78	100	90.8
8	76	88	98	68	24	70.8
10	60	84	92	90	2	65.6
12	68	88	90	56	4	61.2
14	48	78	76	60	0	52.4
16	14	70	62	70	0	43.2

The default stride improves mean success by 42.8 percentage points over $k = 1$. Strides 3–6 retain 88.8–96.8%, while more aggressive subsampling reduces accuracy. The *Stack Cube* learning curves below use a larger demonstration set and report checkpoint-dependent behavior.

***Stack Cube* learning curves with 400 demonstrations.** We sweep the temporal stride k defined in Section 3.1 from 1 to 18 on the simulated *Stack Cube* task with 400 demonstrations, fixing the *Legendre* degree at $K = 6$, and evaluate 12 checkpoints every 1,250 training steps. Fig. 9 (left) reports the result.

- **Necessity of sparse sampling.** Without sparse sampling ($k = 1$) the success rate caps at 24% throughout training; raising k to 4 lifts it to 96%, a 72 pp gap that quantitatively justifies the strategy.
- **Bell-shaped optimum.** A sweet spot emerges at $k \in [3, 8]$ where every run converges to $\geq 93\%$, with $k = 8$ crossing 80% in only 5,000 steps (Fig. 9, left (a)). Beyond $k \geq 13$ performance collapses (52% at $k = 15$, 28% at $k = 18$). Two factors compound here: (i) the fixed-degree polynomial (Eq. 3) loses expressivity as the physical span $k(H_{\text{poly}} - 1)$ grows; (ii) a single inference covers too much of the task that only ~ 2.7 forward passes per successful episode remain at $k = 18$, leaving the policy little chance to revise its plan from fresh visual feedback.
- **Pareto trade-off and practical speed limit.** Amortized inference cost per environment step falls monotonically from 0.97 ms ($k = 1$) to 0.15 ms ($k = 14$), and per-episode total inference time tracks it from 116 ms to 14.5 ms. At $k = 8$, success reaches 98% while both timing metrics are about half those at $k = 4$, at 0.21 ms / 20.3 ms. We retain $k = 4$ as the shared default, combining success in this high-performance region with a $3.7\times$ reduction in amortized inference cost relative to $k = 1$ in this study; $k = 8$ offers a more aggressive cost–accuracy tradeoff.

J.3 Fit padding and cross-horizon continuity

Table 14: **Fit padding and KKT continuity across all five tasks.** Success rates at the 10,000-step checkpoint; Table 15 reports the complementary *Close Box* average over the last four checkpoints.

Configuration	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
$\delta = 0$, KKT off	74	90	68	68	94	78.8
$\delta = 1$, KKT off	86	94	98	86	96	92.0
$\delta = 1$**, KKT on**	**100**	**98**	**96**	**92**	**98**	**96.8**

Fit padding improves mean success by 13.2 percentage points, with a further 4.8-point gain from KKT continuity correction of the training targets. The latter gain is not uniform: *Stack Cube* decreases from 98% to 96%, while the other four tasks improve.

***Close Box* checkpoint-averaged continuity.** We isolate the contribution of the two-tier extension introduced in Section 3.1 on the *Close Box* task, sharing identical training pipelines across the three configurations (last-4-checkpoint average success rate reported in Table 15).

Table 15: **Fit padding and KKT continuity on *Close Box*.** Mean success rate over the last four training checkpoints.

Configuration	**Success Rate (%)**
$\delta = 0$, no KKT continuity	75.0
$\delta = 1$, no KKT continuity	86.0 (+11.0)
$\delta = 1$ + KKT continuity (default)	**97.5** (+11.5)

Fit padding pulls the constraint anchors into the stable interior of the *Legendre* fitting window and yields an 11.0 pp gain; stacking the KKT C^1 correction (Eq. 11) on top contributes another 11.5 pp, jointly closing a 22.5 pp gap to a near-saturated 97.5%. Both components improve this comparison.

J.4 Fixed polynomial degree

Table 16: **Fixed-degree sensitivity across all five tasks.** Each model is trained and evaluated at a fixed degree K.

K	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
4	74	96	88	92	98	89.6
6	**100**	**98**	**96**	**92**	**98**	**96.8**
8	44	78	90	76	78	73.2
9	26	78	68	72	82	65.2

The default $K = 6$ yields the highest mean among the reported degrees. It attains or ties the best success rate on each task; $K = 4$ ties it on *Open Drawer* and *Pick-Place Bowl*. The fixed $K = 9$ that succeeds on *Corridor Push* lowers the five-task mean to 65.2%. Increasing degree changes both representational capacity and the difficulty of fitting and predicting coefficients; greater capacity alone does not ensure better closed-loop performance.

J.5 Training-time history noise

Table 17: **Training-time history-noise sensitivity.** Only the augmentation scale σ_h varies. The standard five-task evaluation does not add inference-time history noise.

σ_h	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
0	58	64	54	44	42	52.4
0.1	80	92	98	76	100	89.2
0.5	**100**	**98**	**96**	**92**	**98**	**96.8**
1	86	90	100	72	98	89.2
2	8	60	80	16	84	49.6

Removing training noise reduces mean success from 96.8% to 52.4% (44.4 percentage points). The means at $\sigma_h = 0.1$ and 1.0 are both 89.2%, within 7.6 points of the default over a tenfold scale range. This average tolerance does not hold identically for every task: *Close Box* falls from 100% to 80% at $\sigma_h = 0.1$, and *Open Drawer* from 92% to 72% at $\sigma_h = 1.0$. At $\sigma_h = 2.0$, mean success falls to 49.6%.

Table 18: **Fine-grained training-noise sweep on *Close Box*.** Same checkpoint and evaluation protocol as Table 17.

σ_h	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	2
SR (%)	58	80	98	90	92	100	82	82	84	76	86	8

J.6 History-fit regularization and consistency loss

Table 19: **History-fit regularization.** Success rates (%) for the Tikhonov coefficient λ_h.

λ_h	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
0	74	96	94	74	100	87.6
0.01	96	88	94	78	98	90.8
0.1 (default)	100	98	96	92	98	**96.8**
1.0	84	92	92	82	100	90.0

The shared default $\lambda_h = 0.1$ gives the highest five-task mean without per-task selection. The unregularized setting remains operational in these experiments, but its mean success is 9.2 percentage points lower.

Table 20: **Consistency-loss weight across all five tasks.** Other settings remain fixed.

λ_{cons}	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
0	80	98	86	90	94	89.6
0.1	74	92	90	86	96	87.6
1	**100**	**98**	**96**	**92**	**98**	**96.8**
10	86	88	94	70	100	87.6

Removing the consistency loss lowers mean success by 7.2 percentage points, but still achieves 89.6%; the term is beneficial rather than indispensable. Its endpoint supervision emphasizes the one-step update used at deployment, while the flow-matching objective already samples times near that endpoint. The default weight of 1.0 gives the best mean, although individual tasks need not peak at that weight.

J.7 Number of function evaluations

We vary NFE in $\{1, 2, 4, 6, 10, 20, 30, 40, 50, 100\}$ using each task's 2,500-step FLASH checkpoint and 50 evaluation episodes. Tables 21 and 22 report success and amortized inference cost. The logged `Average Inference Time` divides accumulated inference time by environment steps; its units are ms/environment step, not ms/policy call.

Table 21: **FLASH success versus NFE at the 2,500-step checkpoint.** The NFE=1 row is the default inference budget; the bold row need not be the most accurate at this early checkpoint.

NFE	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
1	**80**	**96**	**60**	**60**	**98**	**78.8**
2	54	90	56	90	100	78.0
4	56	78	72	76	100	76.4
6	68	88	62	88	100	81.2
10	88	86	72	98	100	88.8
20	88	86	68	98	100	88.0
30	86	86	64	96	100	86.4
40	86	88	60	92	100	85.2
50	92	88	70	96	100	89.2
100	86	86	60	94	100	85.2

Table 22: **Amortized inference cost versus NFE.** Units are ms/environment step. Task values and equal-task means are rounded to two decimal places; these are not full-call latencies.

NFE	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
1	**0.32**	**0.33**	**0.30**	**0.39**	**0.42**	**0.35**
2	0.43	0.45	0.53	0.48	0.50	0.48
4	0.48	0.87	0.52	0.64	0.68	0.64
6	0.60	1.23	0.93	0.77	0.71	0.85
10	0.81	0.95	0.89	0.90	0.92	0.89
20	2.04	1.40	1.36	1.37	1.41	1.52
30	2.83	2.09	2.04	2.04	1.97	2.19
40	3.59	2.49	2.52	2.35	2.45	2.68
50	2.96	4.68	3.11	3.23	3.03	3.40
100	7.59	5.76	10.62	5.27	5.50	6.95

One step gives 78.8% mean success at 0.35 ms/environment step. The mean peaks at 89.2% for NFE=50, a 10.4-point gain at 3.40 ms/environment step; NFE=100 reaches 85.2% at 6.95 ms/environment step. Thus the measurements show an accuracy–cost tradeoff, not flat accuracy or an accuracy optimum at one step. Improvements are task-dependent: *Close Box* rises from 80% at one step to 92% at 50, *Open Drawer* from 60% to 98% at 10 or 20, and *Stack Cube* from 60% to 72% at 4 or 10. *Pick-Place Bowl* is 98% at one step and 100% at every tested larger budget. Individual timing entries fluctuate, but the mean cost rises strongly with NFE. The final 10,000-step model reaches 96.8% mean success at NFE=1 (Table 1); that later checkpoint is not interchangeable with this sensitivity sweep.

Fine-grained *Close Box* inference-budget curve. We sweep the inference $N_{\text{NFE}} \in \{1, 2, 4, \ldots, 60\}$ on a single *Close Box* checkpoint trained for $2{,}500$ steps; Fig. 9 (middle) reports success rate and amortized inference cost per environment step.

The single-step Euler ($N_{\text{NFE}} = 1$) reaches 80% success rate, *outperforming* the multi-step integrations at $N_{\text{NFE}} = 2$ (54%), 4 (56%), and 6 (68%). Larger inference budgets ($N_{\text{NFE}} \geq 10$) achieve higher success rates (mostly 86% to 94%, peaking at 96%). The advantage over small intermediate inference budgets is consistent with the role of the consistency loss $\mathcal{L}_{\text{cons}}$ (Eq. 8): by explicitly supervising the $\tau = 0$ one-shot Euler update at training time, the model is specialized for the $N_{\text{NFE}} = 1$ regime, whereas small intermediate N_{NFE} values lose this specialization without yet accumulating enough multi-step accuracy. Meanwhile, amortized inference cost rises from 0.32 to 3.56 ms/environment step between $N_{\text{NFE}} = 1$ and 60 (approximately $11\times$). This single-task curve and the five-task sweep above show that larger inference budgets can improve success at additional cost.

J.8 Velocity feed-forward

Removing analytic velocity feed-forward lowers mean success across the five simulated manipulation tasks from 96.8% to 93.2% (3.6 percentage points; Table 11). The paired control study below measures its effect on tracking precision.

FLASH derives the desired velocity signal analytically from the polynomial coefficients (Eq. 4), feeding it directly to the torque controller as a feed-forward term. To isolate the contribution of this component, we ablate the velocity feed-forward by sending only position commands (i.e., setting $\hat{\mathbf{v}} = \mathbf{0}$) while keeping all other components unchanged: the polynomial representation, sparse sampling, FLASH flow matching, and KKT continuity correction.

Experimental setup. We evaluate 9 checkpoints spanning 3 tasks $\times$ 3 training seeds, with 50 paired rollouts per checkpoint ($N = 450$ pairs total). For each rollout, we record the mean absolute error (MAE, in degrees) between the commanded and actual joint positions across all timesteps, then compare the *with-velocity* and *without-velocity* conditions in a pairwise fashion.

Results (Fig. 9, right). Removing the analytic velocity feed-forward increases the J7 (wrist) position-tracking MAE from 1.33° to 2.90°, a $\mathbf{2.18}\times$ increase. Across the 450 rollout pairs, the paired t-test gives $p < 10^{-100}$, with Cohen's $d = 1.78$ [6].

Six of seven arm joints exhibit the same pattern: the position-only controller produces larger tracking errors, with the degradation most pronounced in the distal joints (J6, J7). This supports the benefit of analytic velocity feed-forward for tracking precision in the evaluated torque-control setting.

J.9 Perturbations of the history anchor at inference

We inject Gaussian noise with standard deviation σ directly into the history coefficients $\mathbf{C}_h$ at inference. This perturbation probe is distinct from varying the training augmentation σ_h. Table 23 evaluates the 10,000-step checkpoint over 50 episodes per task.

Table 23: **History-anchor perturbations at inference.** Success rates (%). The unperturbed condition is $\sigma = 0$.

σ	*Close Box*	*Pick Cube*	*Stack Cube*	*Open Drawer*	*P-P Bowl*	Mean
0	100	98	96	92	98	96.8
0.25	90	84	98	88	98	91.6
0.5	84	88	96	78	100	89.2
0.75	40	86	88	54	84	70.4
1.0	8	74	42	14	20	31.6

Mean success drops by 7.6 percentage points at $\sigma = 0.5$, equal in scale to the training augmentation, and degrades more sharply at larger perturbations. The response is task-dependent: at $\sigma = 0.5$, *Stack Cube* and *Pick-Place Bowl* retain 96% and 100%, while *Close Box* falls to 84%.

K Failure regimes and scope of feedback

Representation and sampling. *Corridor Push* exposes a failure of fixed $K = 6$ when several sharp reversals must fit inside a single open-loop chunk. Fixed $K = 9$ succeeds there but performs substantially worse on the five simulated manipulation tasks. Separately, excessive sampling stride lowers success (Table 13), and extreme execution-speed scaling fails in Appx. G. These experiments show that degree, physical horizon, and playback speed must be chosen for the task regime.

History error and task-level recovery. The history anchor is a starting point for a learned update, not a constraint to repeat the previous motion: Eq. (6) does not impose a small residual bound. Nevertheless, severe history corruption degrades success (Table 23). Learning from interactive corrections [24], monitoring distribution shift [28], and learning a dedicated recovery policy [23] address related but different problems; our demonstration data do not establish such recovery capability.

Prediction horizon, feedback, and pauses. The polynomial horizon specifies how far a plan extends, while the executed prefix specifies when a new observation can trigger replanning. A policy can regenerate a plan after an executed prefix without waiting for the entire polynomial to finish. Dense interpolation at the controller rate supplies commands between policy calls; it does not itself provide fresh visual feedback. Likewise, rescaling k_{eval} changes the duration of a plan, but does not by itself implement event-triggered pauses or failure recovery. The reported speed-modulation experiment selects a constant speed within each rollout. Online speed selection and feedback-driven pausing remain to be evaluated.

Contact and sensing. The evaluated policies receive vision and proprioception, without force/tactile sensing; tasks requiring controlled normal force need a suitable feedback interface and separate evaluation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All quantitative claims in the abstract are supported by experimental results in Sec. 4 and the Appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The dedicated Sec. 7 discusses the limitations of this work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not present formal theorems or proofs. All equations (OLS fitting, KKT correction, flow matching objectives) are standard derivations with full details provided in Sec. 3 and Appx. A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Appx. B specifies all task descriptions, demonstration counts, baseline configurations, hyperparameters (stride k, degree K, σ_h, λ_h, λ_{cons}), shared encoder, optimizer schedule, and evaluation protocol (50 rollouts per method).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the full codebase, training scripts, and evaluation pipelines as supplemental material. Instructions for reproducing all main experiments are included.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sec. 4 summarizes the setup; Appx. B provides full details including tasks, demonstration counts, all hyperparameters for FLASH and baselines, shared encoder architecture, and optimizer configuration.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The five-task comparison uses 50 rollouts per task; *Corridor Push* uses 50, *Fork Reach* 48, and each new real-robot task ten trials per policy. These trial counts are stated beside the results. Real-world tracking MAE includes inter-rollout standard deviations (Appx. F). The velocity feed-forward ablation reports paired t-test, Cohen's d, and $N = 450$ (Appx. J.8).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All inference benchmarks are conducted on the same NVIDIA RTX 5090 GPU (Sec. 4.2). The original five-task study reports 10k-step checkpoints from 15k-step schedules; *Corridor Push* reports 3k steps. Per-environment-step inference cost is reported in Sec. 4.2; cumulative episode cost and robot completion time are reported separately in Appx. E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work focuses on robotic manipulation policy learning using standard simulation platforms and a single robot arm. It does not involve human subjects, private data, or dual-use concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: The paper presents a fundamental algorithm for robot visuomotor control. We do not foresee direct negative societal impacts beyond those generally associated with robotic automation.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The release consists of source code and reproduction instructions for robot manipulation policies, not pretrained checkpoints or scraped datasets. These released assets are not high-risk general-purpose generative models requiring the safeguards described in this question.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All simulation platforms (Roboverse, RLBench, ManiSkill/Isaac, LIBERO/MuJoCo) and baseline methods are properly cited. The visual encoder (ResNet-18) is cited with its original publication.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The new source-code release is accompanied by a README, environment setup instructions, policy configurations, a simulation reproduction guide, and an Apache-2.0 license at `https://github.com/NTUMARS/FLASH-Policy`.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as a component of the core methodology. Any use of LLMs was limited to writing assistance and does not impact scientific rigor or originality.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.