KINEMATIC MEANFLOW: ONE-STEP ACTION GENERATION POLICY FOR ROBOTIC FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we study how to achieve one-step action generation in Robotic Foundation Models (RFMs), aiming to overcome the high inference latency of multi-step flow matching. MeanFlow provides a promising framework for this goal, yet its direct application leads to performance collapse. We discover that this stems from two distinctive dynamics exhibited in the RFM velocity field: (1) the "local acceleration" exhibits stability early on, but surges sharply towards the end of the denoising process, and (2) the spread of its magnitudes across samples widens as denoising progresses. To address these issues, we introduce **Kinematic MeanFlow (K-MF)**, a novel one-step action policy tailored for RFMs. Specifically, grounded in a kinematic identity, K-MF decouples the time derivative term in the MeanFlow formulation into two sub-interval terms separated by an intermediate point. This decoupled formulation enables the two terms to capture early-stage and late-stage denoising dynamics, respectively, while mitigating the error amplification across the process. As a result, our K-MF empowers RFMs to achieve one-step action generation in both training from scratch and fine-tuning paradigms across diverse tasks, while outperforming multi-step flow matching in most settings. In terms of inference efficiency, K-MF reduces action-head latency of GR00T-N1.6 by 67.5%–74.4% across L40 and Jetson Orin in eager and compiled modes, yielding end-to-end latency reductions of 30.3%–54.9%.

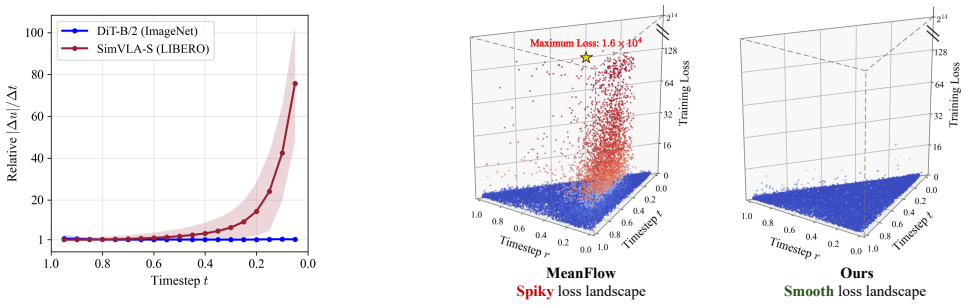
1 INTRODUCTION

Background. Robotic foundation models (RFMs) aim to establish a generalist robot manipulation policy capable of real-time execution across diverse robotic embodiments. Driven by this vision, the architecture of RFMs has been continuously evolved. The first evolution concerns the multi-modal backbone. Early RFMs (Jang et al., 2021; Brohan et al., 2023) employ FiLM-conditioned convolutional neural networks to fuse the language instructions and visual observations. Spurred by the remarkable generalization capabilities of large/vision language models (LLMs/VLMs) (Touvron et al., 2023a;b; Beyer et al., 2024), some studies (Zitkovich et al., 2023; Kim et al., 2024) leverage them as the core foundation for RFMs. Generally, such architectures integrate a pre-trained VLM[1] as the backbone to autoregressively generate discrete action tokens. However, this creates a severe inference burden for real-time control and thus drives the second major evolution in action generation policies. Recent RFMs (Black et al., 2025; NVIDIA et al., 2025; Luo et al., 2026) adopt transformer-based flow matching policies. Specifically, flow matching establishes a velocity field mapping Gaussian noise to valid actions. Using features obtained from a single VLM forward pass, the policy iteratively queries the transformer to predict *instantaneous velocities* across denoising timesteps, progressively transforming noise into multiple actions in parallel. Even as recent architectures (Ye et al., 2026; Yuan et al., 2026) explore replacing VLM backbones with diffusion transformers (DiTs), flow matching remains the prevalent choice for action generation in RFMs.

Motivation. Although this policy enables parallel action generation, its iterative forward passes required for denoising incur non-negligible inference overhead. Taking GR00T-N1.6 as an example, action head latency occupies about 40% of the end-to-end (E2E) inference time on desktop GPUs (e.g., NVIDIA L40) and up to 50% on robotic edge platforms (e.g., NVIDIA Jetson Orin). In other

[1]For brevity, we use "VLM" to represent architectures that either employ a pre-trained VLM or pair a pre-trained LLM with a vision encoder, as their training and inference pipelines in RFMs are conceptually identical.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Velocity field comparison of two domains.

(b) Loss landscapes of MeanFlow and our K-MF.

Figure 1: **Illustrations of our key observations.** In (a), we calculate the change in average velocity $\Delta\mathbf{u}$ over a time interval of $\Delta t = 0.05$ and compare the dynamics of image generation (DiT-B on ImageNet) and action generation (SimVLA-S paired with DiT-B on LIBERO). To highlight the trend, we use the relative value to the initial value of each domain. The shaded area is bounded by the standard 1.5-IQR fences. In (b), we compare the loss landscapes of standard MeanFlow and our proposed K-MF, both applied to SimVLA-S on LIBERO. *More examples are shown in Figure 5.*

words, the computational burden of the action head critically reduces the overall decision frequency of RFMs. Therefore, reducing the number of denoising steps becomes a critical issue. From a broader perspective, MeanFlow has recently emerged in image generation as a remarkably simple alternative to complex distillation or consistency constraints for one-step generation. Unlike flow matching that predicts *instantaneous velocity* $\mathbf{v}_t$ at timestep t, MeanFlow aims to learn the *average velocity* $\mathbf{u}_{t,r}$[2] between any two timesteps t and r. Once trained, by directly predicting the average velocity across the endpoint timesteps, the model achieves one-step denoising in a single forward pass. In action generation, there have been a few efforts that built upon the MeanFlow formulation. In imitation learning, MP1 (Sheng et al., 2026) couples 3D inputs with a dispersive loss on latent features to prevent representation collapse. Other studies improve MeanFlow in reinforcement learning (RL). MVP (Zhan et al., 2026) introduces an instantaneous velocity constraint to stabilize RL training; MeanflowQL (Wang et al., 2026) reformulates MeanFlow into a residual mapping to enable joint optimization with Q-learning; DMPO (Zou et al., 2026) combines dispersive regularization with PPO fine-tuning for one-step policy learning. These efforts remain focused on *specialist policies*, where directly applying MeanFlow typically yields viable, albeit degraded, performance. In contrast, we empirically find that MeanFlow breaks down under the standard training recipe in *generalist RFMs*, motivating us to investigate the fundamental issues underlying this failure.

Observations and underlying issues. In principle, the ground-truth average velocity between t and r is defined as the time-averaged integral of instantaneous velocities. To avoid the costly integration, MeanFlow reformulates the learning target to a form that **evaluates locally** at timestep t: average velocity $\mathbf{u}_{t,r}$ equals the combination of instantaneous velocity $\mathbf{v}_t$ and the time derivative term $\frac{d}{dt}\mathbf{u}_{t,r}$ (see Eq. 3). As $\mathbf{v}_t$ remains constant for a given sample during training, the time derivative term primarily drives the optimization. To investigate the challenges of adapting MeanFlow to RFMs, we compare the dynamics of "local acceleration" ($\Delta\mathbf{u}/\Delta t$) between image generation and action generation in RFMs across denoising timesteps. As shown in Figure 1a, the local acceleration of image generation is stable throughout denoising with a narrow spread. In contrast, RFMs exhibit distinctive dynamics, characterized by two key observations with corresponding issues:

(1) The local acceleration surges sharply in the late denoising stage (i.e., $t < 0.3$), ultimately reaching $76\times$ its initial value at $t = 1$. ***Issue 1:*** The time derivative term tends to incur substantial estimation errors if the model fails to infer the global trend from local observations[3].

(2) The spread of local acceleration magnitude widens as denoising proceeds (indicated by the expanding red shaded area). Our further analysis in Figure 5 reveals that the extent of this spread is strongly correlated with the task diversity of the RFM training data. ***Issue 2:*** Minor estimation errors in the early denoising stage tend to be amplified as this spread widens.

[2]Here, $t, r \in [0, 1]$, with 0 denoting clean data and 1 denoting pure noise. We use $\mathbf{u}_{t,r}$ as shorthand for $\mathbf{u}(\mathbf{z}_t, r, t)$, formally introduced in Section 3. Throughout this paper, we adopt the subscript order (t, r) to reflect the denoising direction from t to r, although the average velocity is defined from r to t.

[3]Local acceleration reflects $\frac{d}{dt}\mathbf{u}_{t,r}$ $(t - r < \epsilon)$, whose temporal stability decides local-global consistency.

As a result, a direct adaptation of MeanFlow to RFMs yields an unstable, spiky loss landscape (Figure 1b). Notably, this instability differs from that reported in prior work (Geng et al., 2025; Zhang et al., 2026; Sheng et al., 2026): **loss outliers in RFMs are strongly associated with sharp increases in local acceleration magnitude and its spread**, predominantly located at $r < 0.3$. This highlights the necessity of correctly estimating[4] the time derivative term $\frac{d}{dt}\mathbf{u}_{t,r}$ in RFMs.

Design insights and contributions. Our method builds on a fundamental kinematic identity: for any intermediate point $c \in (r, t)$, the average velocity $\mathbf{u}_{t,r}$ is exactly equal to the convex combination of the average velocities over the two sub-intervals $\mathbf{u}_{t,c}$ and $\mathbf{u}_{c,r}$. Leveraging this identity, we decouple the derivative $\frac{d}{dt}\mathbf{u}_{t,r}$ into two sub-interval terms, $\frac{d}{dt}\mathbf{u}_{t,c}$ and $\frac{d}{dc}\mathbf{u}_{c,r}$. This is primarily driven by two considerations: (1) for the first issue, the term $\frac{d}{dt}\mathbf{u}_{t,c}$ guarantees robust estimation in the early denoising stage, while $\frac{d}{dc}\mathbf{u}_{c,r}$ focuses exclusively on corrections in the late denoising stage; (2) for the second issue, a second estimation at timestep c serves as an anchor to reduce the impact of the widening spread and thereby mitigate error amplification throughout the denoising process. Given this decomposition, we further investigate how to determine the intermediate point c. We empirically find that a fixed midpoint clearly outperforms uniform random sampling, despite the latter covering a broader range of interval partitions. This suggests that the placement of the estimation anchor, rather than partition diversity, is critical. We then introduce a learnable strategy that adaptively determines c conditioned on t and r, yielding the best performance among the considered strategies. Building on these components, we propose a simple but effective one-step action generation policy tailored for RFMs, termed *Kinematic MeanFlow* (K-MF). With K-MF, RFMs can generate actions in a single step, achieving comparable performance to the multi-step flow matching policy. This is validated by our systematic experiments across: (1) two training paradigms, including training from scratch and fine-tuning of pre-trained flow-matching RFMs; (2) different RFMs, including GR00T-N1.6 and SimVLA; (3) four datasets with varying training data scales, ranging from 658 to 87K trajectories; and (4) both Sim2Sim and Real2Sim evaluation settings.

2 Related Works

Diffusion and flow models (Ho et al., 2020; Rombach et al., 2022; Lipman et al., 2023) are initially designed for image generation. To improve inference efficiency, previous studies have extensively explored how to reduce the required sampling steps. One line of work focuses on deriving a few-step model from a pre-trained many-step baseline through score or flow distillation (Salimans & Ho, 2022; Liu et al., 2023b; Yin et al., 2024). Another line of research relies on consistency models (Song et al., 2023; Song & Dhariwal, 2024), which employ consistency constraints to ensure outputs along the same path share identical endpoints. Recently, MeanFlow models (Geng et al., 2025; 2026) have emerged as a compelling alternative, which enables one-step generation by predicting the average velocity between any two timesteps. Several studies introduce intermediate points into MeanFlow to enforce consistency across interval partitions (Li et al., 2026; Guo et al., 2025) or facilitate curriculum learning (Zhang et al., 2026). These studies primarily aim to improve training efficiency or mitigate gradient discrepancies of MeanFlow in image and audio generation tasks. *This has clear differences from our method in motivation, scope, and design.*

For specialist policies in robotics, Diffusion Policy (Chi et al., 2023) pioneered the integration of diffusion algorithms into action generation. Inspired by few-step algorithms in image tasks, Flow-Policy (Zhang et al., 2025) and Maniflow (Yan et al., 2025) set a consistency constraint for flow matching in imitation learning, while CP (Ding & Jin, 2024) adapts this concept to behavioral cloning and actor-critic reinforcement learning algorithms. Similarly, several recent studies adapt MeanFlow in action generation for both imitation learning (Sheng et al., 2026) and reinforcement learning (Zhan et al., 2026; Wang et al., 2026; Zou et al., 2026), as we have discussed before.

Prevalent generalist RFMs (NVIDIA et al., 2025; Black et al., 2025; Zheng et al., 2026; Yuan et al., 2026; Ye et al., 2026) widely adopt flow matching for action generation. However, in stark contrast to specialist policies, research on few/one-step action generation in RFMs remains underexplored. Prior works (Chen et al., 2026; Luan et al., 2026) typically apply methods developed for image generation directly to RFMs or restrict their evaluation to a limited range of tasks and models. *Differently, we reveal the unique dynamics of RFM velocity fields, design K-MF accordingly, and comprehensively validate it across models, tasks, and training paradigms.*

[4] $\frac{d}{dt}\mathbf{u}_{t,r}$ is not a pre-defined ground truth, but is generated online through the policy π_θ being trained.

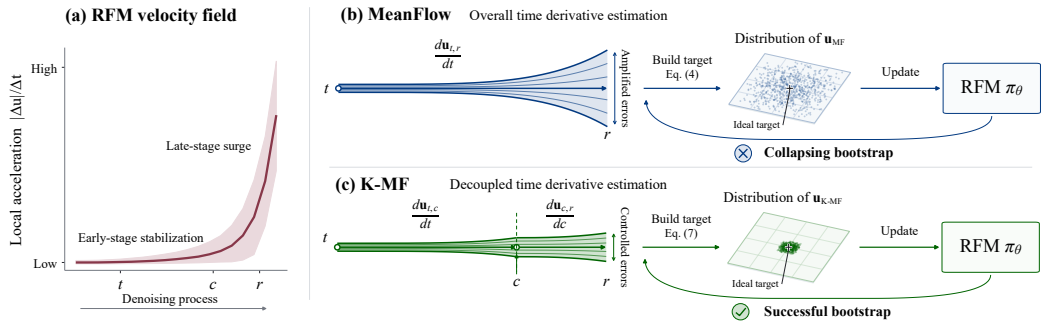


Figure 2: **Comparison of MeanFlow and our Kinematic MeanFlow (K-MF).** In RFM velocity fields, the late-stage surge and widening spread shown in (a) lead to large errors in estimating the time derivative of average velocity over long intervals. These errors cause MeanFlow's bootstrapping loop to collapse. In contrast, K-MF splits time-derivative estimation across two sub-intervals, mitigating these errors and stabilizing the bootstrapping process. *Note that the ideal target is a conceptual reference for better illustration. More detailed analysis is in Section 3.2.*

3 METHOD

3.1 PRELIMINARIES

Robotic foundation models with flow matching. A typical RFM is trained to predict a sequence of future actions (i.e., an action chunk) by conditioning on current visual observation, language instruction, and proprioceptive state. Architecturally, RFMs mainly consist of two primary modules: (1) a backbone that encodes the visual observation and language instruction into a multimodal observation $\mathbf{o}_n$; (2) a transformer action head that generates the subsequent action chunk $\mathbf{a}_n$, conditioned on the proprioceptive state $\mathbf{s}_n$ and multimodal observation $\mathbf{o}_n$. For action generation, prevalent RFMs leverage flow matching (Lipman et al., 2023) to learn an instantaneous velocity field, modeling a continuous transformation from pure noise to valid actions. Specifically, let $\mathbf{a} \sim p_{\text{action}}$ denote a target action chunk and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ denote Gaussian noise of the same dimensions as $\mathbf{a}$. The flow path representing instantaneous velocity is constructed by: $\mathbf{z}_t = (1-t)\mathbf{a} + t\boldsymbol{\epsilon}$ with timestep $t \in [0, 1]$. Correspondingly, the instantaneous velocity conditioned on the action chunk $\mathbf{a}$ at timestep t is:

$$\mathbf{v}_t = d\mathbf{z}_t/dt = \boldsymbol{\epsilon} - \mathbf{a}. \tag{1}$$

The RFM π_θ is then trained to predict the instantaneous velocity by:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}\left[\|\pi_\theta(\mathbf{z}_t, t \mid \mathbf{o}, \mathbf{s}) - \mathbf{v}_t\|_2^2\right]. \tag{2}$$

During the inference stage, with the predicted velocity of π_θ, the RFM performs multiple sampling steps to gradually transform the noise $\boldsymbol{\epsilon}$ to the clean predicted action $\hat{\mathbf{a}}$.

Formulating MeanFlow in RFMs. Unlike flow matching models that learn the instantaneous velocity, MeanFlow models (Geng et al., 2025) establish a field that represents the average velocity between any two timesteps t and r ($0 \le r \le t \le 1$). The average velocity $\mathbf{u}$ is formulated as: $\mathbf{u}(\mathbf{z}_t, r, t) \triangleq \frac{1}{t-r}\int_r^t \mathbf{v}(\mathbf{z}_\tau, \tau)d\tau$, where $\mathbf{v}(\mathbf{z}_\tau, \tau)$ denotes the marginal instantaneous velocity along the flow path $\mathbf{z}$ at timestep τ. Alternatively, this equation can be rewritten as the following meanflow identity that describes the relation between instantaneous velocity and average velocity:

$$\mathbf{u}(\mathbf{z}_t, r, t) = \mathbf{v}(\mathbf{z}_t, t) - (t-r)\frac{d}{dt}\mathbf{u}(\mathbf{z}_t, r, t). \tag{3}$$

During training, the RFM is optimized to predict $\mathbf{u}$ that satisfies Eq. 3. Following standard flow matching, the marginal instantaneous velocity $\mathbf{v}(z_t, t)$ is replaced by the conditional instantaneous velocity $\mathbf{v}_t$ for a given action chunk $\mathbf{a}$. The RFM π_θ is optimized with the learning objective:

$$\begin{aligned} \mathcal{L}_{\text{MF}}(\theta) &= \mathbb{E}\left[\|\pi_\theta(\mathbf{z}_t, r, t \mid \mathbf{o}, \mathbf{s}) - \mathbf{u}_{\text{MF}}\|_2^2\right], \\ \text{where} \quad \mathbf{u}_{\text{MF}} &= \mathbf{v}_t - (t-r)\cdot \text{sg}\left(\frac{d\pi_\theta(\mathbf{z}_t, r, t \mid \mathbf{o}, \mathbf{s})}{dt}\right), \end{aligned} \tag{4}$$

and $\text{sg}(\cdot)$ denotes the stop-gradient operation. During the evaluation, MeanFlow uses the predicted average velocity to conduct denoising: $\mathbf{z}_r = \mathbf{z}_t - (t-r)\pi_\theta(\mathbf{z}_t, r, t \mid \mathbf{o}, \mathbf{s})$. By simply setting $t = 1$ and $r = 0$, the one-step action generation can be achieved by: $\hat{\mathbf{a}} = \mathbf{z}_0 = \mathbf{z}_1 - \pi_\theta(\mathbf{z}_1, 0, 1 \mid \mathbf{o}, \mathbf{s})$.

3.2 KINEMATIC MEANFLOW

Deeper analysis of time derivative term. As stated in Section 1, the primary bottleneck hindering MeanFlow in RFMs lies in the time derivative term. Specifically, in Eq. 4, MeanFlow establishes a *bootstrapping mechanism*: at each training iteration, the derivative estimated by the current model is used to construct a stop-gradient regression target, and the updated model is then expected to provide more accurate derivatives for subsequent optimization. Although this mechanism is empirically effective in image generation, we find that it breaks down in RFMs and identify two key contributing factors: (1) the late-stage surge in local acceleration makes derivative estimation prone to large errors; and (2) the widening spread tends to further amplify these errors (empirically demonstrated in Figure D in the Appendix). These errors corrupt the bootstrapping feedback loop and ultimately destabilize training, as illustrated in Figure 2. Consequently, MeanFlow struggles to achieve reliable one-step action generation in RFMs. We discuss this problem in detail in Section 4.2.

Decoupling time derivative via kinematic identity. As revealed by the kinematic identity, for any intermediate point $c \in (r, t)$, the average velocity across the interval $[r, t]$ is identically equal to the convex combination of the average velocities across two partitioned sub-intervals, $[c, t]$ and $[r, c]$:

$$\begin{aligned}\mathbf{u}(\mathbf{z}_t, r, t) &= \frac{1}{t-r}\int_r^t \mathbf{v}(\mathbf{z}_\tau, \tau)\,\mathrm{d}\tau = \frac{1}{t-r}\left[\int_c^t \mathbf{v}(\mathbf{z}_\tau, \tau)\,\mathrm{d}\tau + \int_r^c \mathbf{v}(\mathbf{z}_\tau, \tau)\,\mathrm{d}\tau\right] \\ &= (1-\lambda)\mathbf{u}(\mathbf{z}_t, c, t) + \lambda \mathbf{u}(\mathbf{z}_c, r, c), \quad \text{where } \lambda = \frac{c-r}{t-r}.\end{aligned} \tag{5}$$

Although interval partitioning is an established strategy in diffusion/flow models (Salimans & Ho, 2022; Frans et al., 2025; Guo et al., 2025; Zhang et al., 2026), our method stands distinct in both underlying motivation and formulation from prior works. In contrast to prior works that construct a distillation target or improve training efficiency and stability in image generation, our method is motivated by the substantially different dynamics in RFM velocity fields and therefore decouples the time derivative term to address the resulting estimation difficulties. Specifically, we apply Eq. 3 separately to the two sub-intervals along the same flow path and obtain:

$$\begin{aligned}\mathbf{u}(\mathbf{z}_t, r, t) &= (1-\lambda)\mathbf{u}(\mathbf{z}_t, c, t) + \lambda \mathbf{u}(\mathbf{z}_c, r, c) \\ &= (1-\lambda)\left[\mathbf{v}(\mathbf{z}_t, t) - (t-c)\frac{d}{dt}\mathbf{u}(\mathbf{z}_t, c, t)\right] + \lambda\left[\mathbf{v}(\mathbf{z}_c, c) - (c-r)\frac{d}{dc}\mathbf{u}(\mathbf{z}_c, r, c)\right] \\ &= [(1-\lambda)\mathbf{v}(\mathbf{z}_t, t) + \lambda \mathbf{v}(\mathbf{z}_c, c)] - (t-r)\left[(1-\lambda)^2\frac{d}{dt}\mathbf{u}(\mathbf{z}_t, c, t) + \lambda^2\frac{d}{dc}\mathbf{u}(\mathbf{z}_c, r, c)\right].\end{aligned} \tag{6}$$

Given action chunk $\mathbf{a}$ during training, both $\mathbf{z}_t$ and $\mathbf{z}_c$ in K-MF are constructed along the conditional flow path (see Section E in the Appendix for a detailed explanation). According to Eq. 1, $\mathbf{v}(z_t, t)$ and $\mathbf{v}(\mathbf{z}_c, c)$ are replaced by $\mathbf{v}_t$ and $\mathbf{v}_c$, where $\mathbf{v}_t = \mathbf{v}_c = \boldsymbol{\epsilon} - \mathbf{a}$. Formally, for the RFM policy π_θ, the training objective of K-MF minimizes the mean-squared error against the decoupled target:

$$\begin{aligned}\mathcal{L}_{\text{K-MF}}(\theta) &= \mathbb{E}\left[\|\pi_\theta(\mathbf{z}_t, r, t \mid \mathbf{o}, \mathbf{s}) - \mathbf{u}_{\text{K-MF}}\|_2^2\right], \\ \text{where } \mathbf{u}_{\text{K-MF}} &= \mathbf{v}_t - (t-r)\left[(1-\lambda)^2\,\mathrm{sg}\left(\frac{d\pi_\theta(\mathbf{z}_t, c, t \mid \mathbf{o}, \mathbf{s})}{dt}\right) + \lambda^2\,\mathrm{sg}\left(\frac{d\pi_\theta(\mathbf{z}_c, r, c \mid \mathbf{o}, \mathbf{s})}{dc}\right)\right].\end{aligned} \tag{7}$$

Training objective with three strategies for determining c. Based on the above analysis, the efficacy of K-MF largely relies on the second estimation of the time derivative at timestep c, which is dependent on the temporal ratio λ via $c = r + \lambda(t-r)$. We then define three strategies.

Random sampling. Under this strategy, the temporal ratio is sampled from a uniform distribution, $\lambda \sim \mathcal{U}(0, 1)$. By exposing the network to non-deterministic splits throughout training, this strategy serves as a baseline for assessing whether partition diversity benefits K-MF training.

Fixed at midpoint. Under this strategy, we set the temporal ratio $\lambda = 0.5$ throughout training. In contrast to random sampling, this design completely eliminates variance introduced by interval splits and provides a balanced supervision signal across both sub-intervals.

Learnable strategy. Under this strategy, the temporal ratio is generated by a learnable lightweight MLP (only 3 layers) throughout the training: $\lambda = G_\phi(t, r)$. The learning objective of $G_\phi(\cdot)$ is built upon the following three considerations: (1) an ideal temporal ratio λ should not enlarge the training loss; instead, it should minimize the online bootstrapping error by balancing both sub-intervals; (2)

in the early training stage, since the overall estimation $\frac{d}{dt}\mathbf{u}_{t,r}$ is inaccurate, λ should be chosen to make the decoupled estimation $(1-\lambda)^2\frac{d}{dt}\mathbf{u}_{t,c} + \lambda^2\frac{d}{dc}\mathbf{u}_{c,r}$ distinct from the overall estimation; and (3) in the late training stage, as the bootstrapping feedback loop is well-established, the decoupled estimation should gradually approach the overall estimation to maintain the self-consistency. With these considerations, the learning objective of the learnable generator is:

$$\mathcal{L}_{\text{gen}}(\phi) = \mathbb{E}\left[\omega \cdot \|\text{sg}\left(\pi_\theta(\mathbf{z}_t, r, t \mid \mathbf{o}, \mathbf{s})\right) - \mathbf{u}_{\text{K-MF}}(\lambda)\|_2^2\right], \text{ where } \omega = 1 + \gamma e^{\left(-\frac{1}{\|\Delta\mathbf{u}\|_2+\epsilon}\right)}. \tag{8}$$

$\Delta\mathbf{u}$ denotes the difference between the decoupled estimation and the overall estimation of each individual sample. With linearly increasing γ from -0.5 to 0.5 throughout training, the generated ω helps to achieve the second and third insights sequentially. $\epsilon = 10^{-6}$ to ensure numerical stability.

Overall training objective. The overall training objective under each strategy is formulated as:

$$\mathcal{L} = \begin{cases} \mathcal{L}_{\text{K-MF}}(\theta), & \text{random sampling / fixed at midpoint,} \\ \mathcal{L}_{\text{K-MF}}(\theta) + \alpha\mathcal{L}_{\text{gen}}(\phi), & \text{learnable strategy,} \end{cases} \tag{9}$$

where α is the balancing factor of the two loss terms. During optimization, $\mathcal{L}_{\text{gen}}$ updates only the generator parameters ϕ, while the RFM parameters θ are treated as constants. Notably, our formulation remains inherently straightforward. When adopting random sampling or the fixed-at-midpoint strategy, K-MF introduces no additional hyperparameters compared to MeanFlow. Under the learnable strategy, only two extra hyperparameters, α and γ, are introduced.

Implementation details of K-MF. When implementing K-MF, we consider three aspects to balance training efficiency and performance. (1) *Time derivative calculation:* following MeanFlow, we compute the time derivative via a Jacobian-vector product (JVP) between the network Jacobian $[\nabla_{\mathbf{z}}\pi_\theta, \partial_r\pi_\theta, \partial_t\pi_\theta]$ and the tangent vector $[\mathbf{v}_t, 0, 1]^\top$. By leveraging forward-mode automatic differentiation (dual numbers), each JVP is executed within a single forward pass without constructing a reverse computation graph. (2) *Batched derivative calculation:* the two sub-interval derivative terms in Eq. 7 are mutually independent and operate under the stop-gradient operator. This decoupling allows us to concatenate the inputs of both sub-intervals along the batch dimension, evaluating both derivatives simultaneously in a single forward pass. (3) *Adapting flow matching models for MeanFlow:* we introduce an additional embedding layer for the timestep r, and sum the embeddings of t and r to form the final time representation. As a result, K-MF incurs no more than a 33% increase in training time with acceptable GPU memory overhead relative to flow matching (see Table 5b). The training and inference algorithms are shown in Algorithms 1 and 2 in the Appendix, respectively.

4 Experiments

4.1 Setups

Training and evaluation benchmarks. Four training datasets have varying trajectory numbers: LIBERO (Liu et al., 2023a), BridgeData V2 (Walke et al., 2023), Fractal (Brohan et al., 2023), and a COMPASS-generated (Liu et al., 2025) point navigation dataset. They contain 2K, 53K, 87K, and 658 trajectories, respectively. Our evaluation comprises both Sim2Sim and Real2Sim settings. For Sim2Sim, we utilize LIBERO with its native simulator and PointNav powered by COMPASS. For Real2Sim, we evaluate on Fractal and BridgeData V2 via SimplerEnv (Li et al., 2024). Following established protocols, we run 200 and 500 rollouts for GR00T-N1.6 and SimVLA-S, respectively.

Selected RFMs and baseline methods. We select two representative RFMs: GR00T-N1.6 (NVIDIA et al., 2025) and SimVLA-S (Luo et al., 2026) for the evaluation with fine-tuning and training from scratch paradigms, respectively. We primarily compare against multi-step flow matching (Lipman et al., 2023) to show our method can achieve better performance. We evaluate MeanFlow (Geng et al., 2025) to justify the necessity of decoupling derivatives in RFMs. For a more comprehensive comparison, we select α-Flow (Zhang et al., 2026) and MVP (Zhan et al., 2026) as representative variants for interval partitioning and stabilized action generation, respectively.

4.2 Pilot Experiments

We investigate two critical questions in this part: (1) how to adapt MeanFlow effectively into generalist RFMs; and (2) how existing MeanFlow variants perform in RFM action generation, by performing pilot experiments on GR00T-N1.6 using the LIBERO-10 benchmark.

Table 1: **Pilot studies on MeanFlow and its variants on GR00T-N1.6.** In (a), we study the delicate training recipe for MeanFlow, including gradient clipping, progressive sampling for timesteps, and sufficient training iterations. In (b), we compare the performance between selected baselines and our K-MF (learnable strategy). NFE denotes the number of function evaluations, SR is the success rate on the LIBERO-10 benchmark, and * denotes the delicate recipe. Best results are **bolded**.

(a) Delicate training recipe for MeanFlow.

Method	NFE	# Iters	GradClip	Prog. Sampling	SR (%)
Flow Matching	4	20K			92.0
			✓		94.5
MeanFlow	2	20K	✓		*Fail*
			✓	✓	6.5
	2	40K	✓	✓	52.5
		60K	✓	✓	**87.0**
		80K	✓	✓	84.5
	1	60K	✓	✓	*Fail*

(b) Selected baselines *vs.* K-MF

Method	NFE	# Iters	SR (%)
Flow Matching	4	20K	**94.5**
MeanFlow*	2	60K	87.0
	1		*Fail*
MVP	2	20K	82.0
	1		*Fail*
α-Flow	2	20K	94.0
	1		84.5
Ours	1	20K	**94.5**

Table 2: **Performance comparison across two training paradigms on the LIBERO benchmark.** We compare flow matching, MeanFlow (NFE=2), and our K-MF (learnable strategy, NFE=1) under both fine-tuning on GR00T-N1.6 and training from scratch on SimVLA-S (without pre-training on action data). * denotes the delicate recipe in Table 1a. Best results are **bolded**.

Model	# Params	Training Paradigm	Method	# Iters	NFE	Spatial	Object	Goal	Long	Avg.
GR00T-N1.6	3B	Fine-tuning	Flow Matching	20K	4	97.5	98.5	**98.5**	**94.5**	97.3
			MeanFlow*	60K	2	96.5	99.0	95.5	87.0	94.5
			Ours	20K	1	**99.5**	**99.5**	98.0	**94.5**	**97.9**
SimVLA-S	0.6B	Training from scratch	Flow Matching	200K	10	**97.0**	97.4	98.0	84.2	94.2
			MeanFlow*	200K	2	95.8	97.2	97.8	84.0	93.7
			Ours	200K	1	96.4	**97.8**	**98.4**	**87.8**	**95.1**

MeanFlow requires a delicate training recipe. As illustrated in Table 1a, besides the common gradient clipping, MeanFlow requires progressive sampling of timesteps (see Figure A in the Appendix) and sufficient training iterations. The performance saturates at around 60K iterations, then declines potentially due to overfitting. Even with the delicate training recipe, MeanFlow only reaches reasonable performance with two-step generation, still lagging behind standard flow matching. Meanwhile, its performance is sensitive to the separated timestep, as shown in Figure 3.

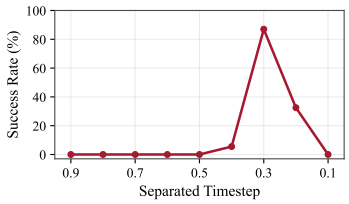


Figure 3: **Sensitivity to the separated timestep** of two-step generation, under the best setting in Table 1a.

Performance of MeanFlow variants. As shown in Table 1b, α-Flow and MVP do not require the delicate training recipe of MeanFlow, benefiting from interval-partitioning curriculum design and training stabilization via velocity constraints, respectively. However, when pushed to one-step action generation, MVP fails to produce viable actions, and α-Flow suffers noticeable performance degradation. In contrast, our K-MF achieves one-step generation with comparable performance to multi-step flow matching.

These results empirically support our hypothesis that the distinctive dynamics of the RFM velocity field pose challenges for modeling the average velocity, rendering existing methods insufficient to directly address one-step action generation in RFMs.

4.3 Main Results

We then examine the effectiveness of our method, as well as the generalization abilities across different training paradigms, RFMs, benchmarks, and embodiments.

Different training paradigms and RFMs. On the LIBERO benchmark, we perform fine-tuning on GR00T-N1.6 and training from scratch on SimVLA-S. As shown in Table 2, K-MF consistently achieves competitive or superior performance to the multi-step flow matching and two-step MeanFlow using the delicate training recipe. This empirically validates that the efficacy of our K-MF is not limited to a specific training paradigm or RFM.

Table 3: **Performance comparison across benchmarks and embodiments.** We compare multi-step flow matching (FM) with our one-step K-MF across different fine-tuning datasets, evaluation settings, and robotic embodiments. Best results are **bolded**.

Task Type	Dataset	Setting	Embodiment	Task	FM (NFE = 4)	Ours (NFE = 1)
Tabletop Manipulation	Fractal	Real2Sim	GoogleX	Pick	**91.0**	**91.0**
				Move	87.5	**92.0**
				Open	35.5	**50.5**
				Close	**81.5**	80.0
				Avg.	73.9	**78.4**
	BridgeData V2	Real2Sim	WidowX	Spoon	69.0	**73.0**
				Carrot	**64.0**	60.0
				Cube	7.5	**13.0**
				Eggplant	93.0	**93.5**
				Avg.	58.4	**59.9**
Point Navigation	COMPASS Generated	Sim2Sim	Unitree G1	ID	**84.5**	**84.5**
				OOD	63.0	**63.5**
				Avg.	73.8	**74.0**

Table 4: **Ablation study on different strategies for determining c on LIBERO**: random sampling (RS), fixed at midpoint (Mid), and learnable strategy (LS). Best and second-best results are **bolded** and underlined.

Model	Strategy	NFE	Spatial	Object	Goal	Long	Avg.
GR00T-N1.6	RS	2	98.0	99.0	96.5	87.5	95.3
	Mid	1	98.0	**99.5**	**98.5**	**94.5**	97.6
	LS	1	**99.5**	**99.5**	98.0	**94.5**	**97.9**
SimVLA-S	RS	1	95.6	96.4	97.6	85.6	93.8
	Mid	1	96.2	96.8	97.2	87.4	94.4
	LS	1	**96.4**	**97.8**	**98.4**	**87.8**	**95.1**

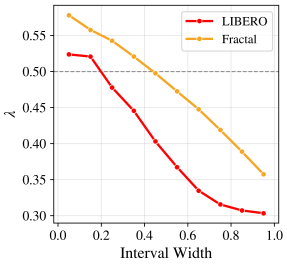


Figure 4: **Distribution of λ** for LS on GR00T-N1.6 across different datasets.

Different benchmarks and embodiments. We further fine-tune GR00T-N1.6 on large-scale tabletop manipulation benchmarks (Fractal and BridgeData V2) as well as the PointNav dataset. Table 3 shows that our K-MF consistently achieves higher success rates than the multi-step flow matching baseline across the majority of evaluated settings. These results demonstrate that K-MF generalizes across data scales, tasks, and robotic embodiments, extending from small-scale simulation data to large-scale real-world datasets and from tabletop manipulation to navigation.

4.4 Ablation Studies

In this part, we first compare our three strategies for determining c and then show the inference and training efficiencies of our method.

Different strategies for determining c. We compare our three strategies in Table 4. The results indicate: (1) the *learnable strategy* yields the best performance, and the *fixed at midpoint* serves as a straightforward strategy for fast adaptation in the fine-tuning setting; (2) *random sampling* yields clearly weaker performance than the other two strategies. Furthermore, as shown in Figure 4, we find that λ in the learnable strategy shows strong correlations with the interval width $t - r$. These observations indicate that the efficacy of our K-MF primarily stems from improving time-derivative estimation rather than augmentation from different partitions.

Inference efficiency. As shown in Table 5a, we observe: (1) the action head accounts for 43%–73% of the end-to-end (E2E) latency of the flow matching baseline; (2) K-MF consistently reduces action-head latency by 67.5%–74.4% and E2E latency by 30.3%–54.9% across different hardware platforms and execution modes. Therefore, pairing generalist RFMs with K-MF largely facilitates their practical deployment on robotic platforms.

Training overhead. Table 5b reports peak GPU memory usage and training time. Compared to MeanFlow, K-MF incurs marginal additional overhead, increasing training time by 7.5%–13.6% and peak GPU memory by at most 2.5%. In light of its inference efficiency, the training overhead of K-MF over flow matching remains acceptable: 33% for GR00T-N1.6 and 27% for SimVLA.

More ablation studies on hyperparameters and the learnable generator G_ϕ are in the Appendix.

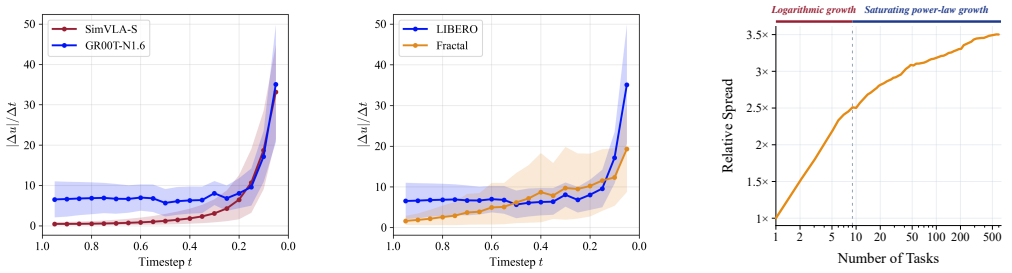
Table 5: **Efficiency analysis of K-MF.** In (a), we show the inference latency (ms) of flow matching and K-MF on GR00T-N1.6. AH and E2E denote action head and end-to-end latency, respectively. In (b), we show training overhead, in terms of peak GPU memory (GB) and training time (hours), under the same settings as in Table 2, except keeping the same training iterations for fair comparison.

(a) Inference latency.

Mode	Platform	Backbone	FM		Ours	
			AH	E2E	AH	E2E
torch.eager	L40	31.7	109.3	149.6	$28.0_{\downarrow 74.4\%}$	$67.4_{\downarrow 54.9\%}$
	Jetson Orin	147.2	236.5	390.3	$63.6_{\downarrow 73.1\%}$	$219.8_{\downarrow 43.7\%}$
torch.compile	L40	19.4	19.1	44.2	$6.2_{\downarrow 67.5\%}$	$30.8_{\downarrow 30.3\%}$
	Jetson Orin	70.4	82.8	162.9	$23.3_{\downarrow 71.9\%}$	$102.8_{\downarrow 36.9\%}$

(b) Training overhead.

Model	Method	# Iters	GPU Mem.	T_{train}
GR00T-N1.6	FM	20K	35.6	9.4
	MF		52.4	11.0
	Ours		53.7	12.5
SimVLA	FM	200K	22.8	58.6
	MF		23.1	69.1
	Ours		23.2	74.3



(a) SimVLA-S *vs.* GR00T-N1.6. (b) LIBERO *vs.* Fractal. (c) Relative spread *vs.* # tasks.

Figure 5: **More illustrations of the RFM velocity field.** (a) Comparison of dynamics between SimVLA-S and GR00T-N1.6 on LIBERO. (b) Dynamics of GR00T-N1.6 on LIBERO vs. Fractal. (c) Relative spread of local acceleration magnitudes as the number of tasks increases.

4.5 Understanding the Dynamics in the Velocity Field of RFMs

Based on Figure 1a, we further investigate the RFM velocity field and have three observations:

(1) **Different RFMs share a similar pattern on the same dataset.** As shown in Figure 5a, GR00T-N1.6 and SimVLA-S exhibit a similar increasing trend in $|\Delta u|/\Delta t$ as denoising progresses, despite differences in their absolute values and the extent of spread.

(2) **Dynamics depend critically on the training dataset.** As shown in Figure 5b, the model trained on the larger and more complex Fractal dataset exhibits a wider spread in local acceleration magnitudes during the denoising process than that trained on LIBERO.

(3) **Spread increases with task diversity.** As shown in Figure 5c, the relative spread of local acceleration magnitudes within Fractal increases with task count, following an approximately logarithmic trend at low task counts and a saturating power-law trend at higher task counts.

Based on these observations, we summarize two key takeaways. First, across architectures and datasets, the RFM velocity field exhibits a shared pattern, whereas the extent of its spread depends strongly on the training distribution. Second, higher task diversity is associated with a wider spread, suggesting that modeling average velocity becomes increasingly challenging as datasets grow larger and more diverse. *More analysis is in Section E in the Appendix.*

5 Conclusion

In this paper, we present Kinematic MeanFlow (K-MF), a simple and effective framework for one-step action generation in generalist RFMs. To accommodate distinctive dynamics in the RFM velocity field, K-MF decomposes the overall time derivative into two sub-interval terms and further incorporates three strategies for determining intermediate points. Our experiments demonstrate that K-MF generally achieves superior performance to multi-step flow matching across diverse settings.

Limitations. Compared to flow matching, K-MF indeed incurs additional training overhead, yet its overall training cost remains reasonable. Furthermore, our evaluation does not cover whole-body control or dexterous manipulation, both of which are still challenging for current RFMs.

Reproducibility Statement

To facilitate reproducibility, we conduct all experiments using publicly available models and datasets. The main paper and appendix provide implementation details for our method and baselines, along with experimental settings, hyperparameters, and evaluation benchmarks. We will release our code and model checkpoints upon publication.

AI Use Statement

As requested by the ICLR 2027 policy, we disclose the usage of AI tools. AI tools were used solely to assist with proofreading the manuscript. We have manually reviewed all AI-assisted edits. The authors take full responsibility for the final content of this work.

References

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, et al. π_0: A vision-language-action flow model for general robot control. In *RSS*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. In *RSS*, 2023.

Yang Chen, Xiaoguang Ma, and Bin Zhao. Mean-flow based one-step vision-language-action. *arXiv preprint arXiv:2603.01469*, 2026.

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

Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. In *ICLR*, 2024.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *ICLR*, 2025.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *NeurIPS*, 2025.

Zhengyang Geng, Yiyang Lu, Zongze Wu, Eli Shechtman, J Zico Kolter, and Kaiming He. Improved mean flows: On the challenges of fastforward generative models. In *CVPR*, 2026.

Yi Guo, Wei Wang, Zhihang Yuan, Rong Cao, Kuan Chen, Zhengyang Chen, Yuanyuan Huo, Yang Zhang, Yuping Wang, Shouda Liu, et al. Splitmeanflow: Interval splitting consistency in few-step generative modeling. *arXiv preprint arXiv:2507.16884*, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *CoRL*, 2021.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2024.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

Zhiqi Li, Yuchen Sun, Duowen Chen, Jinjin He, and Bo Zhu. Trajectory consistency for one-step generation on euler mean flows. In *ICML*, 2026.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, 2023a.

Wei Liu, Huihua Zhao, Chenran Li, Yuchen Deng, Joydeep Biswas, Soha Pouya, and Yan Chang. Compass: Cross-embodiment mobility policy via residual rl and skill synthesis. *arXiv preprint arXiv:2502.16372*, 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023b.

Wuyang Luan, Junhui Li, Weiguang Zhao, Wenjian Zhang, Tieru Wu, and Rui Ma. Snapflow: One-step action generation for flow-matching vlas via progressive self-distillation. *arXiv preprint arXiv:2604.05656*, 2026.

Yuankai Luo, Woping Chen, Tong Liang, Baiqiao Wang, and Zhenguo Li. Simvla: A simple vla baseline for robotic manipulation. *arXiv preprint arXiv:2602.18224*, 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022.

Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. Mp1: Meanflow tames policy learning in 1-step for robotic manipulation. In *AAAI*, 2026.

Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *ICLR*, 2024.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *ICML*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. In *CoRL*, 2023.

Zeyuan Wang, Da Li, Yulin Chen, Ye Shi, Liang Bai, Tianyuan Yu, and Yanwei Fu. One-step generative policies with q-learning: A reformulation of meanflow. In *AAAI*, 2026.

Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, and Dieter Fox. Maniflow: A general robot manipulation policy via consistency flow training. In *CoRL*, 2025.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. In *ICLRW*, 2026.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *CVPR*, 2024.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Guojian Zhan, Letian Tao, Pengcheng Wang, Yixiao Wang, Yiheng Li, Yuxin Chen, Hongyang Li, Masayoshi Tomizuka, and Shengbo Eben Li. Mean flow policy with instantaneous velocity constraint for one-step action generation. In *ICLR*, 2026.

Huijie Zhang, Aliaksandr Siarohin, Willi Menapace, Michael Vasilkovsky, Sergey Tulyakov, Qing Qu, and Ivan Skorokhodov. Alphaflow: Understanding and improving meanflow models. In *ICLR*, 2026.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *AAAI*, 2025.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *ICLR*, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023.

Guowei Zou, Haitao Wang, Hejun Wu, Yukun Qian, Yuhang Wang, and Weibing Li. One step is enough: Dispersive meanflow policy optimization. *arXiv preprint arXiv:2601.20701*, 2026.

APPENDIX

- Section A: Datasets used for training in our experiments.
- Section B: Implementation details of baseline methods.
- Section C: Experimental setups of our training and evaluation.
- Section D: More ablation studies regarding our K-MF.
- Section E: Discussion on two important problems.

A DATASETS

LIBERO (Liu et al., 2023a) is a simulation benchmark for robotic manipulation. Our paper uses four task suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long, which emphasize spatial relationships, object variations, goal variations, and long-horizon executions, respectively. Each task suite contains 10 tasks with 50 trajectories per task, yielding 2K trajectories across 40 tasks.

Fractal (Brohan et al., 2023) is a large-scale dataset of real-world robotic manipulation collected with Google robots. The publicly released version contains 87K trajectories covering 599 tasks with different language instructions. The dataset mainly covers diverse manipulation behaviors in kitchen environments, including picking and placing objects and opening and closing drawers, providing substantial variation in tasks, objects, and scenes.

BridgeData V2 (Walke et al., 2023) is a large-scale dataset of real-world manipulation collected with the WidowX robot. The version we used consists of 53K trajectories spanning 13 skills across 24 environments. Most trajectories are annotated with natural language instructions, and the dataset includes variations in objects, camera viewpoints, and workspace configurations.

PointNav dataset we used is released by NVIDIA. It contains synthetic navigation demonstrations generated using COMPASS (Liu et al., 2025). Specifically, it contains 658 trajectories of the Unitree G1 robot, each of which provides robot state (including robot speed, route information, and goal heading), egocentric RGB observations, and velocity commands as actions.

B IMPLEMENTATION DETAILS OF BASELINE METHODS

B.1 FLOW MATCHING

Flow matching (Lipman et al., 2023) is originally implemented by our selected RFMs. We mainly follow the official settings, with minor adjustments to hyperparameters to reproduce the reported results. Detailed setups are shown in Section C.

B.2 MEANFLOW

MeanFlow (Geng et al., 2025) serves as the base method for our K-MF and shares its experimental setup, with the addition of progressive timestep sampling mentioned in Section 4.2 of the main paper. Specifically, progressive timestep sampling is crucial for training MeanFlow in RFMs. This strategy schedules two hyperparameters: the flow ratio and the gap scale. Specifically, the flow ratio increases linearly from 0 to 0.5, reducing the proportion of samples satisfying $t = r$ from 100% to 50%. The gap scale increases linearly from 0 to 1, progressively expanding the sampled temporal gaps from zero to their original magnitudes. By employing this scheduling, MeanFlow initially focuses on learning the local average velocity during the early stages of training before smoothly transitioning to the standard training objective. Figure A illustrates this process.

B.3 OTHER MEANFLOW VARIANTS

As we mentioned in the main paper, we incorporate α-Flow (Zhang et al., 2026) and MVP (Zhan et al., 2026) as representative MeanFlow variants for interval partitioning and stabilization techniques for action generation, respectively.

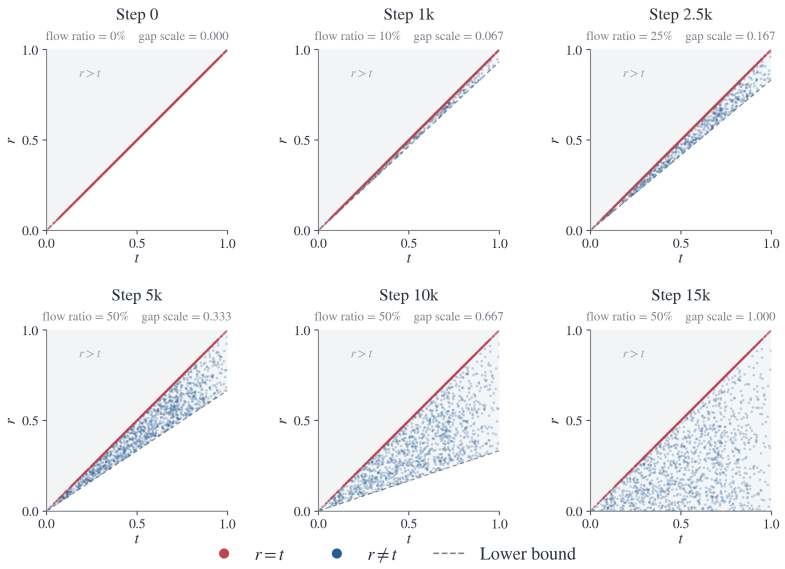


Figure A: **Illustration of progressive sampling.** This strategy is governed by the flow ratio (which determines the proportion of $t \neq r$) and the gap scale (which defines the maximum interval for $t - r$). Initially, the training process reduces to traditional flow matching. As training progresses, the sampling of t and r gradually transitions to the standard formulation. *Note that this strategy is exclusively applied to MeanFlow; our proposed K-MF does not require this technique.*

Table A: Hyperparameter settings of the flow matching baseline and our K-MF.

Category	Hyperparameter	SimVLA-S	GR00T-N1.6		
		LIBERO	LIBERO	Fractal/Bridge	PointNav
Shared	Global batch size	256	640	1024	64
	Learning rate	1.00e-04	1.00e-04	1.50e-04	1.00e-04
	Optimizer	AdamW	AdamW	AdamW	AdamW
	Betas	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)	(0.9, 0.95)
	Weight decay	0.0	1.00e-05	1.00e-05	1.00e-05
	Training steps	200,000	20,000	30,000	40,000
	Warmup steps	1,000	2,000	1,500	2,000
	Grad clip	1.0	1.0	1.0	1.0
	Timestep Sampling	Beta(1.5,1.0)	Beta(1.5,1.0)	Beta(1.5,1.0)	Beta(1.5,1.0)
	State drop rate	0.0	0.8	0.8	0.0
	Image resize	384×384	224×224	224 (min edge)	224 (min edge)
	Precision	bf16	bf16	bf16	bf16
K-MF	Flow ratio	0.5	0.5	0.5	0.5
	α	0.1	0.1	0.1	0.1
	γ	[-0.5, 0.5]	[-0.5, 0.5]	[-0.5,0.5]	[-0.5, 0.5]

α-Flow. Originally developed for image generation, α-Flow aims to bridge the gap between the learning of instantaneous and average velocities through curriculum learning based on interval partitioning. Using its image-generation settings as a reference, we adjust the scheduling hyperparameters for action generation. Under this configuration, the model learns instantaneous velocity during the first 8,000 iterations and average velocity after 12,000 iterations. During the intervening 4,000 iterations, the model learns a combination of the two through interval partitioning, enabling a gradual transition.

MVP. This was initially designed for reinforcement learning. Here, we adopt only its instantaneous velocity constraint to stabilize training for action generation. Specifically, we retain the same training recipe as our flow matching baseline and augment the original MeanFlow objective with an auxiliary instantaneous velocity loss. Following the original implementation, we set the relative weight of this auxiliary loss to 1, assigning equal weights to the two losses.

Table B: **Ablation studies on the LIBERO benchmark.** We evaluate the hyperparameters α and γ in the learnable strategy, the flow ratio inherited from MeanFlow (determining the proportion of $t \neq r$ per batch), and the depth of G_ϕ.

(a) Effect of α.

α	SR (%)
0	97.6
0.1	97.9
0.5	97.6
1.0	97.3
2.0	96.3

(b) Effect of γ.

$\max \lvert\gamma\rvert$	SR (%)
0	97.6
0.25	97.8
0.5	97.9
0.75	97.3
1.0	96.1

(c) Effect of flow ratio.

Flow ratio	SR (%)
0.25	97.4
0.5	97.9
0.75	97.8
1.00	97.5

(d) # Layers of G_ϕ.

# Layers	SR (%)
2	97.3
3	97.9
4	97.8

C Experimental Setups

Training infrastructure. All training experiments are conducted on a single node equipped with 8×H20 GPUs (96GB).

Codebase. We implement K-MF and all baseline methods on top of the official codebases of GR00T-N1.6 (NVIDIA et al., 2025) and SimVLA (Luo et al., 2026).

Hyperparameter configuration. We mainly follow the standard training recipe. Table A summarizes the hyperparameter configuration adopted in flow matching and our K-MF in our experiments.

Evaluation setups. Our experiments are validated on three evaluation platforms, including LIBERO (Liu et al., 2023a), SimplerEnv (Li et al., 2024), and COMPASS (Liu et al., 2025). All evaluations follow the standard protocol of the codebase.

LIBERO. We evaluate on four benchmark suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long (each containing 10 tasks, totaling 40 tasks). Following the original setting, we evaluate each task suite over 200 rollouts on GR00T-N1.6 and 500 rollouts on SimVLA-S.

SimplerEnv. Models trained on Fractal and BridgeData V2 are evaluated on simulated GoogleX and WidowX embodiments, respectively. Following standard protocols, we evaluate GoogleX across diverse manipulation skills ("pick object", "move near", "open drawer", and "close drawer") and WidowX across different target objects ("spoon on towel", "carrot on plate", "stack cube", and "put eggplant in basket"). Each task is evaluated over 200 rollouts.

COMPASS. We evaluate navigation performance across both in-distribution (ID) and out-of-distribution (OOD) settings. Specifically, the ID setting uses the same environment seen during data collection, whereas the OOD setting adopts a novel "hospital" scenario as the representative benchmark. Each environment is evaluated over 200 rollouts.

D More Ablation studies

Hyperparameters α and γ. K-MF introduces only two method-specific hyperparameters. α is the balancing factor of the loss for the learnable generator G_ϕ relative to the whole network, while γ controls the penalty strength for large discrepancies between the overall and decoupled estimates. As shown in Table Ba, choosing a relatively small α leads to better performance, with $\alpha = 0.1$ achieving the best result. Table Bb further demonstrates that constraining the evolution of γ within a moderate range, such as $[-0.5, 0.5]$, yields better performance.

Flow ratio. Inherited from MeanFlow, the flow ratio specifies the fraction of samples with $t \neq r$ in each training batch. As shown in Table Bc, K-MF is relatively insensitive to this hyperparameter within the tested range, maintaining strong performance even at a relatively high ratio of 0.75.

Learnable generator G_ϕ. As illustrated in Figure B, G_ϕ is a three-layer MLP with layer dimensions $2 \to 128 \to 128 \to 1$. The first two layers are paired with SiLU, while the last layer is processed by Sigmoid, mapping the output into $(0, 1)$[5]. First, we ablate the number of layers in G_ϕ, keeping the hidden dimension fixed at 128. As shown in Table Bd, the success rate increases from 97.3% with

[5]To prevent G_ϕ from degenerating over narrow time intervals, we impose boundary constraints on the sampling distribution on LIBERO.

Table C: **Performance gain from learnable strategy across different evaluation benchmarks.** We use fixed at midpoint as the baseline and observe the relative performance gain. Best results are **bolded**.

Evaluation benchmark	# Training trajectories	FM (NFE=4)	Ours (NFE=1)	
			Mid	LS
LIBERO	2K	97.3	97.6	**97.9**
Simpler-WidowX	53K	58.4	59.1	**59.9**
Simpler-Google	87K	73.9	76.8	**78.4**

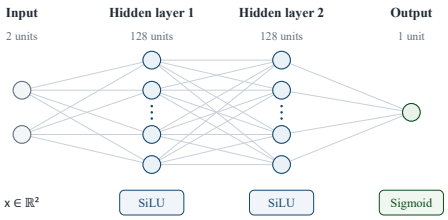
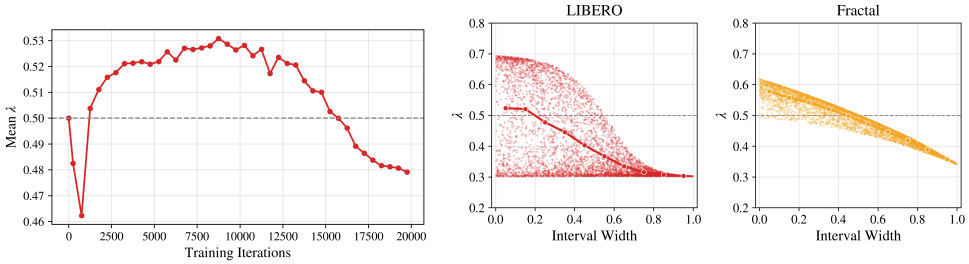


Figure B: **The architecture of the learnable generator G_ϕ.**



(a) Variation of mean λ in training.

(b) Distribution of λ on different interval widths.

Figure C: **Ablation studies on learnable generator G_ϕ on GR00T-N1.6.** In (a), we illustrate the variation trend of the mean generated λ throughout the training on LIBERO. In (b), we study the distribution of λ generated by the well-trained G_ϕ on LIBERO and Fractal datasets. *As the four task suites in LIBERO share very similar properties, we use LIBERO-10 as the representative case.*

two layers to 97.9% with three layers, while adding a fourth layer yields no further improvement. We therefore adopt a three-layer MLP for G_ϕ. Second, we illustrate how the generated λ evolves during training in Figure Ca. The mean generated λ initially decreases and then rises during early training when $\gamma < 0$. During later training when $\gamma > 0$, it decreases and eventually converges to approximately 0.48. Third, we visualize the distributions of λ generated by the trained model across datasets in Figure Cb. We observe a consistent overall trend across datasets, with more concentrated λ distributions on larger datasets. Lastly, we evaluate the additional performance gains from G_ϕ in Table C. We observe greater gains from the learnable generator on the two larger datasets, Fractal and BridgeData V2.

Error amplification in late denoising stage. As discussed in the main paper, errors in estimating the time derivative tend to be amplified as the denoising interval extends into later stages. We perform an ablation study to validate this hypothesis. As illustrated in Figure D, a 1% perturbation introduced at early denoising timesteps produces an error in $\frac{\mathrm{d}\mathbf{u}_{t,r}}{\mathrm{d}t}$ whose magnitude is approximately $120\times$ that of the initial perturbation by the end of denoising. This amplification decreases when the perturbation is introduced at later timesteps ($t < 0.6$). Our K-MF decouples the overall time derivative into two separate terms during training, mitigating this error amplification based on three key facts: (1) in the early stages of denoising, error amplification remains mild; (2) in the late denoising phase (where t is small), the propagation and amplification of noise are substantially suppressed; and (3) the two decomposed terms are computed independently along the conditional flow path, ensuring that errors accumulate additively rather than compounding multiplicatively.

E Discussion

Q1: Why does K-MF sample z_c along the conditional flow path?

As discussed in Section 3.2 of the main paper, our analysis suggests that the estimation error of the time derivative term contributes to the performance collapse of MeanFlow. If the intermediate state $\mathbf{z}_c$ were obtained along the current model's estimated marginal flow from $\mathbf{z}_t$, the subsequent derivative evaluation, $\frac{d\mathbf{u}_{c,r}}{dc}$, would also depend on the accuracy of this predicted state. Errors in the first estimation could therefore perturb the predicted intermediate state and, in turn, affect the eval-

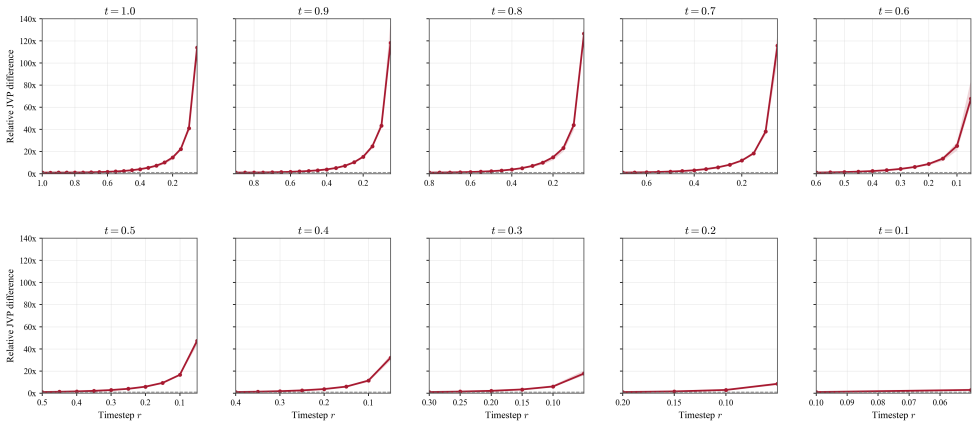
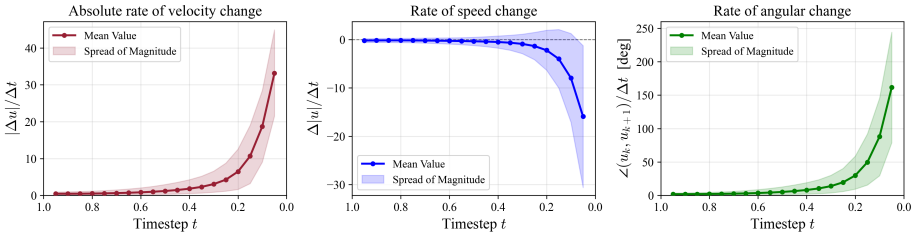
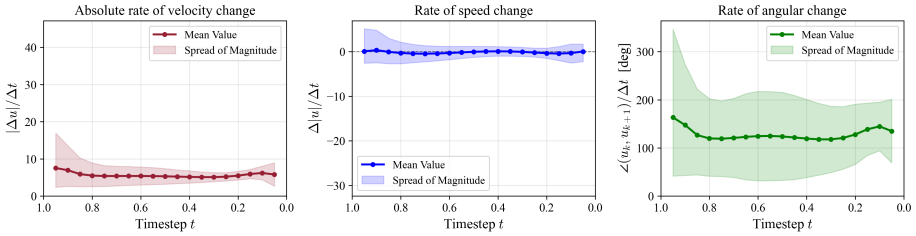


Figure D: **Relative error amplification across denoising timesteps.** On SimVLA-S, we introduce a 1% random perturbation on $\mathbf{z}_t$, where $t \in \{1, 0.9, \ldots, 0.1\}$, and measure the relative amplification to the initial perturbation in $\frac{d\mathbf{u}_{t,r}}{dt}$ at $r \leq t$. The curve shows the mean value over 50 random trials.



(a) SimVLA-S paired with DiT-B on the LIBERO dataset.



(b) DiT-B on the ImageNet dataset.

Figure E: **The absolute rate of velocity change (local acceleration), rate of speed change, and rate of angular change for models trained on LIBERO and ImageNet (top: SimVLA-S; bottom: DiT-B)**. Throughout the denoising process, we compute the average velocity over uniform timesteps, maintaining a constant interval $\Delta t = 0.05$. We visualize the spread of magnitudes using the interval $[Q_1 - 1.5\text{IQR}, Q_3 + 1.5\text{IQR}]$, defined by the standard 1.5-IQR rule for identifying potential outliers. Here, $\text{IQR} = Q_3 - Q_1$ is the interquartile range, with Q_1 and Q_3 denoting the 25th and 75th percentiles, respectively. Values outside this interval are considered potential outliers and excluded from visualization. Each curve is aggregated over 4K samples (action chunks for LIBERO and images for ImageNet), consistent with the setup in Figure 1a and Figure 5 of the main paper.

uation of $\frac{\mathrm{d}\mathbf{u}_{c,r}}{\mathrm{d}c}$. Particularly in the early training stage, this issue could destabilize the bootstrapping feedback loop and expose the method to similar training collapse.

Differently, we construct $\mathbf{z}_c = (1 - c)\mathbf{a} + c\boldsymbol{\epsilon}$ along the conditional flow path, directly anchoring the intermediate state to the training data. Since $\mathbf{z}_c$ does not depend on the model's estimation of $\mathbf{u}_{t,c}$, the derivative $\frac{\mathrm{d}\mathbf{u}_{c,r}}{\mathrm{d}c}$ can be evaluated without inheriting prediction errors in $\mathbf{z}_c$ from the first estimation. This is why we refer to this operation as **decoupling**. This decoupling also enables the two derivative evaluations to be performed in parallel, since neither depends on the result of the other. This improves training efficiency, as discussed in the main paper.

Q2: RFMs typically require fewer sampling steps than image generation models, but why is one-step generation for RFMs more challenging?

To answer this question, we further show speed and angular change to better compare the differences between RFMs and image generation models. As shown in Figure E, we observe: (1) overall, the RFM shows sharp variation trends across three metrics at the late denoising stage, in contrast to the stability in image generation models (although the rate of angular change exhibits a broad spread, neither its mean nor its spread shows a sharp late-stage increase); (2) in the velocity field of RFMs, the surge of local acceleration in the late denoising stage primarily stems from the combined effects of sharp variations in both speed and angular changes. This explains why flow matching typically requires fewer sampling steps to generate actions in RFMs than in image generation models: the velocity changes slowly in both magnitude and direction in the early denoising stage, yielding a nearly straight flow path that can be approximated over a longer time interval using the local instantaneous velocity. When we move to one-step generation, two issues arise. First, the local-global inconsistency makes it difficult to estimate the time derivative term locally. Second, the resulting estimation errors tend to be amplified with the widening spread of local acceleration. The joint effect of these two issues makes it difficult to obtain an accurate time derivative, ultimately giving rise to this counterintuitive phenomenon.

Finally, our findings point to an emerging challenge for one-step action generation: as RFMs scale to larger datasets and more complex, high-DoF embodiments, the difficulties identified in this work may become more pronounced. This work takes the first step toward addressing these challenges, and we hope our observations, insights, and proposed method will inspire future research in more demanding and challenging settings.

Algorithm 1 Kinematic MeanFlow (K-MF) Training

Input: Training action chunk $\mathbf{a}$, policy π_θ, generator G_ϕ (optional), balancing factor α
Sample $t, r \sim \text{Beta}(1.5, 1.0)$ $(t > r)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;
Set $r \leftarrow t$ for 50% of the samples
$\mathbf{z}_t \leftarrow t\epsilon + (1-t)\mathbf{a}$, $\mathbf{v} \leftarrow \epsilon - \mathbf{a}$
$\mathbf{u}, \frac{\mathrm{d}\mathbf{u}_{t,r}}{\mathrm{d}t} \leftarrow \text{JVP}\big(\pi_\theta, (\mathbf{z}_t, r, t), (\mathbf{v}, 0, 1)\big)$
$\lambda \sim \mathcal{U}(0,1)$ (Random) | $\lambda \leftarrow 0.5$ (Midpoint) | $\lambda \leftarrow G_\phi(t,r)$ (Learnable)
$c \leftarrow r + \lambda(t-r)$, $\mathbf{z}_c \leftarrow c\boldsymbol{\epsilon} + (1-c)\mathbf{a}$
with torch.no_grad():
    $_, \left[\frac{\mathrm{d}\mathbf{u}_{t,c}}{\mathrm{d}t}, \frac{\mathrm{d}\mathbf{u}_{c,r}}{\mathrm{d}c}\right] \leftarrow \text{JVP}\Big(\pi_\theta, \big([\mathbf{z}_t, \mathbf{z}_c], [c, r], [t, c]\big), \big([\mathbf{v}, \mathbf{v}], [0, 0], [1, 1]\big)\Big)$
$\mathbf{u}_{\text{K-MF}} \leftarrow \mathbf{v} - (t-r)\left[(1-\lambda)^2 \,\text{sg}\left(\frac{\mathrm{d}\mathbf{u}_{t,c}}{\mathrm{d}t}\right) + \lambda^2 \,\text{sg}\left(\frac{\mathrm{d}\mathbf{u}_{c,r}}{\mathrm{d}c}\right)\right]$
$\mathcal{L}_{\text{K-MF}} \leftarrow \|\mathbf{u} - \text{sg}(\mathbf{u}_{\text{K-MF}})\|_2^2$ ▷ Ensure only θ is updated, as in Eq. 7
if Learnable **then**
    $\mathcal{L}_{\text{gen}} \leftarrow \omega\|\,\text{sg}(\mathbf{u}) - \mathbf{u}_{\text{K-MF}}\|_2^2$ ▷ Ensure only ϕ is updated, as in Eq. 8
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{K-MF}} + \alpha\mathcal{L}_{\text{gen}}$
else
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{K-MF}}$
end if
Backpropagate $\mathcal{L}$ once to obtain $\nabla_\theta\mathcal{L}_{\text{K-MF}}$ (and $\alpha\nabla_\phi\mathcal{L}_{\text{gen}}$ if Learnable).
Update θ using $\nabla_\theta\mathcal{L}_{\text{K-MF}}$ (and ϕ using $\alpha\nabla_\phi\mathcal{L}_{\text{gen}}$ if Learnable).

Algorithm 2 Kinematic MeanFlow (K-MF) Inference

Input: policy π_θ
Get conditions $(\mathbf{o}, \mathbf{s})$, Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\mathbf{a} \leftarrow \boldsymbol{\epsilon} - \pi_\theta(\boldsymbol{\epsilon}, 0, 1 \mid \mathbf{o}, \mathbf{s})$
return action chunk $\mathbf{a}$