iMF-AttnRes: Fast Mean-Flow Action Generation with Attention Residuals for Simulated Robot Manipulation

Anonymous Author(s)
Affiliation
Address
email

Abstract: Diffusion-style vision-language-action policies provide expressive action distributions, but iterative inference can be too slow for closed-loop manipulation. We study whether improved Mean Flow (iMF), originally motivated by fast-forward generative modeling, can be adapted to action generation so that a policy uses one to a few flow evaluations rather than long denoising chains. We combine iMF with Attention Residuals (AttnRes), which replace fixed residual accumulation in a deep policy transformer with learned depth-wise aggregation over previous layer outputs. In RoboIMI socket peg insertion, the best iMF-AttnRes policy reaches an average reward of 1513.56 and median reward of 1901.5 with 15.122 ms average inference time, compared with diffusion-style infer100 baselines at 861.53/338.291 ms and 1019.39/397.912 ms. In RoboIMI object transfer, the best iMF-AttnRes policy reaches average reward 526.22 and 44/100 success-like episodes, compared with a native diffusion-policy baseline at 319.2 and 29/100. These results suggest that average-flow action generation is a promising route to low-latency robot policies, while also revealing sensitivity to horizon choices, vision-token design, and simulation-to-real validation.

Keywords: Robot learning, imitation learning, flow matching, vision-language-action models

1 Introduction

Learning-based robot manipulation policies increasingly use expressive generative action models. Diffusion Policy showed that action diffusion can be an effective visuomotor policy class for manipulation [1, 2], inheriting the representational flexibility of denoising diffusion models [3] and transformer-based diffusion backbones [4]. However, this expressivity often comes with a deployment cost: the policy must be evaluated repeatedly during sampling. In the RoboIMI socket peg experiments studied here, two diffusion-style VLA baselines use infer100 sampling and require 338.291 ms and 397.912 ms per inference on their recorded rollouts. Such latencies are undesirable when the robot must repeatedly close the perception-action loop.

This paper asks a direct question: can a robot action generator retain the quality benefits of generative policies while reducing inference to one or a few evaluations? We explore this question by adapting improved Mean Flow (iMF) to VLA-style imitation learning. Flow matching and rectified flow formulate generation through continuous vector fields [5, 6]. Mean Flow reframes generation around average velocity, enabling one-step generation [7]; iMF further addresses instability caused by substituting conditional velocities into nonlinear mean-flow relations [8]. For robot control, this distinction matters because fewer evaluations directly increase control responsiveness.

We pair iMF with Attention Residuals (AttnRes). Residual connections are essential for deep visual and transformer networks [9, 10], but standard residual accumulation adds all layer outputs with

Submitted to the 10th Conference on Robot Learning (CoRL 2026). Do not distribute.

fixed unit weight. The motivation in our notes is to rewrite residual networks as a sum over layer increments, then replace the uniform sum with a learned softmax aggregation. AttnRes implements this view by letting each layer attend over preceding residual outputs [11]. We use this mechanism in the policy transformer to reduce depth-wise feature dilution while keeping the action generator fast.

Our contributions are:

1. We formulate an iMF-AttnRes VLA policy for simulated robot imitation learning, combining one-to-few-step average-flow action generation with learned residual aggregation.
2. We provide 100-rollout evaluations on two RoboIMI manipulation environments, socket peg insertion and object transfer, comparing against diffusion-style VLA baselines, a native Diffusion Policy baseline, ACT-style action chunking, and SmolVLA-style compact VLA inference [12, 13].
3. We report both task quality and deployment speed. On socket insertion, the strongest iMF-AttnRes run improves average reward over the ph16 diffusion-style baseline by 1.76× and inference FPS by 22.97×; on object transfer, the strongest iMF-AttnRes run improves average reward over the native diffusion-policy baseline by 1.65×.

We deliberately keep the claims scoped to simulation: hardware varies across runs, and real-robot transfer remains future work.

2 Related Work

Diffusion and flow policies for robot action generation. Diffusion Policy introduced action diffusion as a visuomotor policy learning framework [1, 2], building on denoising diffusion objectives [3]. Diffusion transformers further show that transformer backbones can scale generative modeling [4]. These models are expressive because they iteratively refine samples, but iterative refinement is also a latency source. Flow matching provides an alternative continuous generative formulation by learning vector fields that transport probability paths [5]. Rectified flow emphasizes straighter transport paths and faster sampling [6]. Our method follows this fast-generation lineage but uses the improved mean-flow relation to target one-to-few-step action generation.

Vision-language-action models and action chunking. Transformer robot policies such as RT-1 and RT-2 show how large sequence models can map observations and task context to robot actions [14, 15]. OpenVLA studies open VLA modeling at scale [16], while compact or efficient VLA models such as SmolVLA and FAST reduce deployment cost through smaller backbones or efficient action tokenization [13, 17]. Action Chunking with Transformers (ACT) predicts chunks of future actions to improve temporal consistency and execution efficiency [12]. The iMF-AttnRes policy is complementary: it retains chunked execution but changes the generative action sampler so that each chunk can be produced with one to a few evaluations.

Residual aggregation and attention residuals. Standard residual networks update $x_t = x_{t-1} + y_t$, so recursively $x_t = y_0 + y_1 + \cdots + y_t$. Equivalently, the next layer receives a uniform aggregation of all previous residual increments. A natural generalization is

$$y_{t+1} = f_{t+1}\left(\sum_{s=0}^{t} a_{t+1,s} y_s\right), \qquad a_{t+1,s} \geq 0, \quad \sum_{s=0}^{t} a_{t+1,s} = 1. \tag{1}$$

AttnRes instantiates the weights with an attention distribution over previous residual outputs,

$$a_{t+1,s} \propto \exp\left(w_{t+1}^{\top} \mathrm{RMSNorm}(y_s)\right). \tag{2}$$

This preserves the residual pathway while allowing each layer to select useful earlier representations instead of uniformly accumulating all of them [11]. We use this mechanism in the policy transformer; our ablations suggest that applying AttnRes too broadly inside the vision stack is not automatically beneficial.

3 Method

3.1 Policy formulation

We consider simulated imitation learning in RoboIMI. At each control step, the policy receives visual observations, robot state, and a task context, and predicts an action chunk of length `exec`. The main iMF-AttnRes socket and object-transfer policies use ph32/exec16 or related horizon-execution settings. The rollout evaluator records cumulative reward, median reward, mean maximum reward, nonzero-reward episode count, task-specific success-like threshold count, inference FPS, control FPS, and inference latency.

The policy contains three conceptual parts. First, an observation encoder maps visual and state inputs into policy tokens. Second, a transformer policy core uses AttnRes in place of fixed residual accumulation. Third, the action generator uses iMF to predict an average flow over action noise-to-data paths, enabling one-to-few inference evaluations. In the strongest socket variants, the generator uses infer2 or infer3; in the strongest object-transfer run, it uses infer1.

3.2 Improved Mean Flow action generation

Classical flow matching trains an instantaneous velocity field. Let x_t denote a sample at time t and let the instantaneous flow satisfy

$$\frac{dx_t}{dt} = v(x_t, t). \tag{3}$$

If a policy takes very few integration or denoising steps, using only this local instantaneous velocity can cause large path errors when the learned trajectory is curved. Mean Flow addresses this by training an average velocity over an interval. For two times $r < t$, define

$$\bar{v}(z_t, r, t) = \frac{1}{t-r} \int_r^t v(z_\tau, \tau) d\tau. \tag{4}$$

The model receives (z_t, r, t) and predicts $u_\theta(z_t, r, t)$, an approximation to $\bar{v}(z_t, r, t)$. At inference, this average velocity is used to move directly across a larger interval, replacing a long sequence of small denoising or ODE steps.

The useful training identity comes from differentiating $(t-r)\bar{v}$ with respect to t:

$$\bar{v}(z_t, r, t) = v(z_t, t) - (t-r)\frac{d}{dt}\bar{v}(z_t, r, t). \tag{5}$$

The total derivative contains the dependence of $\bar{v}$ on the current state and time, and can be written as a Jacobian-vector product,

$$\frac{d}{dt}\bar{v}(z_t, r, t) = \mathrm{jvp}\big(\bar{v}, (z, r, t), (v, 0, 1)\big). \tag{6}$$

Mean Flow uses this relation to train an average-velocity predictor for one-step generation [7]. In our setting, the target action sample and noise sample provide a conditional straight-path velocity, but the nonlinear JVP expression should conceptually depend on the marginal velocity field. Directly substituting the conditional velocity inside the nonlinear term can destabilize training. We therefore use the improved Mean Flow correction [8]:

$$V_\theta(z_t, r, t) = u_\theta(z_t, r, t) + (t-r)\mathrm{JVP}_{\mathrm{sg}}\big(u_\theta; v_\theta\big). \tag{7}$$

Here v_θ is obtained from the same network at the instantaneous case, and the stop-gradient JVP direction prevents the nonlinear term from destabilizing the target. The supervised target remains the conditional straight-path velocity available from the imitation action sample and noise sample, but the learned branch used at deployment is the average-flow predictor u_θ. This is the mechanism that allows infer1, infer2, and infer3 policies to compete with infer100 diffusion-style baselines.

3.3 Attention Residual policy transformer

We spell out AttnRes by first rewriting an ordinary residual stack. Let x_l be the hidden state entering layer l, and let the layer output increment be

$$y_l = f_l(x_{l-1}), \qquad x_l = x_{l-1} + y_l. \tag{8}$$

With the convention $y_0 = x_0$, recursively expanding the residual updates gives

$$x_l = y_0 + y_1 + \cdots + y_l = \sum_{s=0}^{l} y_s. \tag{9}$$

Therefore, the next residual block can be written as

$$y_{l+1} = f_{l+1}(x_l) = f_{l+1}\left(\sum_{s=0}^{l} y_s\right). \tag{10}$$

This makes the implicit assumption clear: a standard residual network gives every previous increment a fixed coefficient of one before layer $l+1$ computes the next increment. If we normalize the coefficients only to emphasize their relative importance, this is equivalent to uniform depth aggregation,

$$y_{l+1} = f_{l+1}\left((l+1)\sum_{s=0}^{l} \frac{1}{l+1} y_s\right), \tag{11}$$

so the layer cannot choose which earlier residual features are most useful.

AttnRes replaces this fixed aggregation with learned, layer-dependent weights. Instead of feeding f_{l+1} the unweighted residual sum, we form

$$\tilde{x}_l = \sum_{s=0}^{l} a_{l+1,s} y_s, \qquad a_{l+1,s} \geq 0, \qquad \sum_{s=0}^{l} a_{l+1,s} = 1, \tag{12}$$

then compute

$$y_{l+1} = f_{l+1}(\tilde{x}_l), \qquad x_{l+1} = x_l + y_{l+1}. \tag{13}$$

The attention weights are obtained by scoring each previous residual increment with a learned query vector for the current layer:

$$e_{l+1,s} = w_{l+1}^{\top} \mathrm{RMSNorm}(y_s), \qquad a_{l+1,s} = \frac{\exp(e_{l+1,s})}{\sum_{j=0}^{l} \exp(e_{l+1,j})}. \tag{14}$$

Equations (10)–(14) show the difference in one line: standard residuals use a fixed sum $\sum_s y_s$, whereas AttnRes uses an attention-weighted sum $\sum_s a_{l+1,s} y_s$ whose coefficients depend on the current layer. This keeps the residual pathway but lets each policy layer select earlier visual-state-action features rather than uniformly accumulating all previous increments. In our object-transfer ablations, the best settings apply AttnRes in the policy transformer; replacing residuals throughout the vision encoder underperforms, suggesting that the placement of learned residual aggregation is important.

3.4 Inference and control loop

At deployment, the iMF-AttnRes policy samples an action chunk with one to a few average-flow evaluations and then executes the chunk for `exec` control steps. This contrasts with diffusion-style baselines whose run names include infer100. For socket insertion, the strongest iMF-AttnRes policies use ph32/exec16 and infer2 or infer3. For object transfer, the strongest iMF-AttnRes run uses ph32/exec16 and infer1. The resulting control loop is simple: encode observations, run the AttnRes transformer, evaluate the iMF action generator a small number of times, execute the predicted chunk, and replan on the next observation window.

Placeholder for `fig-method-overview`

Paper Banana prompt: create a clean 16:9 academic system diagram for iMF-AttnRes VLA. Show multi-view images, robot state, and optional language/task conditioning entering a VLA encoder; a policy transformer with AttnRes depth-wise residual aggregation; an improved Mean Flow action generator predicting average flow with one-to-few evaluations; and an exec-length action chunk controlling RoboIMI socket-insert and sim_transfer robots. Use muted conference-paper colors, clear arrows, no decorative 3D, and labels for iMF, AttnRes, infer1/infer2/infer3, and exec16.

Figure 1: Planned method overview figure. The current draft intentionally stores the Paper Banana generation prompt as a placeholder instead of generating an image.

Placeholder for `fig-imf-attnres-components`

Paper Banana prompt: create a 16:9 technical diagram contrasting classical flow matching instantaneous velocity $dx_t/dt = v(x_t, t)$; Mean Flow average velocity over $[r, t]$ with the JVP correction term; improved Mean Flow training relation $V_\theta(z_t) = u_\theta(z_t) + (t-r)\mathrm{JVP}_{\mathrm{sg}}(u_\theta; v_\theta)$; and AttnRes weighted residual aggregation $a_{t+1,s}$ proportional to $\exp(w_{t+1} \cdot \mathrm{RMSNorm}(y_s))$. Use equation callouts, minimal arrows, and a final callout saying one-to-few-step action generation for robot control.

Figure 2: Planned technical component figure. The prompt is retained as a placeholder for later Paper Banana rendering.

4 Experiments

4.1 Setup and metrics

We evaluate in two RoboIMI simulation environments. The socket peg task measures progress toward inserting a peg-like object into a socket. The sim_transfer task measures object-transfer manipulation. Each reported row uses 100 rollouts. Metrics are copied directly from the rollout logs: average cumulative reward, median reward, mean maximum reward, maximum cumulative reward, count of nonzero-reward episodes, count of success-like episodes, inference FPS, control FPS, and average inference time when available. For socket insertion, the recorded success-like threshold is `max_reward > 4`; for sim_transfer, it is `max_reward >= 4`. Hardware differs across runs, so speed comparisons are practical deployment measurements rather than perfectly normalized throughput benchmarks.

4.2 Socket peg insertion

Table 1 summarizes the socket peg insertion results. The best iMF-AttnRes policy by average reward is the infer3 model, with average reward 1513.56 and median reward 1901.5. It exceeds both diffusion-style infer100 baselines: the ph16 baseline obtains average reward 861.53 and median reward 668.0, while the ph32/exec16 baseline obtains average reward 1019.39 and median reward 804.0. The latency gap is large. The iMF-AttnRes infer2 and infer3 policies require 14.409 ms and 15.122 ms per inference, while the two infer100 baselines require 338.291 ms and 397.912 ms.

The nonzero-reward count reveals an important caveat. The ph16 diffusion-style baseline has 97/100 nonzero episodes, higher than the iMF-AttnRes infer2 and infer3 policies at 83/100. Yet its average and median rewards are much lower. Thus, in this task, nonzero contact or partial progress is not sufficient; the iMF-AttnRes policies more often produce high-reward trajectories when they engage the task successfully. SmolVLA is the fastest method in raw latency and control FPS, but its reward is lower than iMF-AttnRes. ACT underperforms both iMF-AttnRes and the diffusion-style VLA baselines in this setup.

Table 1: Socket peg insertion over 100 rollouts. Success-like uses the recorded `max_reward > 4` count.

Method	Setting	Avg. reward	Median	Avg. max	Nonzero	Success-like	Latency ms
iMF-AttnRes	infer1, ph32, exec16, 50k	1275.22	1490.5	3.12	84/100	0/100	16.186
Diffusion-style VLA	infer100, ph16, 150k	861.53	668.0	2.58	97/100	2/100	338.291
Diffusion-style VLA	infer100, ph32, exec16, 150k	1019.39	804.0	2.47	92/100	4/100	397.912
iMF-AttnRes	infer2, ph32, exec16, 150k	1472.62	1825.0	3.29	83/100	5/100	14.409
iMF-AttnRes	infer3, ph32, exec16, 150k	**1513.56**	**1901.5**	3.28	83/100	3/100	15.122
ACT	action chunking	289.60	13.0	1.29	55/100	1/100	100.212
SmolVLA	100k	466.16	106.0	1.72	89/100	0/100	**2.614**

Placeholder for `fig-socket-reward-latency`

Paper Banana prompt: create a 4:3 publication-quality reward-latency comparison for socket peg insertion. Plot methods from the socket table with average reward on the y-axis and average inference time or inverse latency on the x-axis. Highlight iMF-AttnRes infer2/infer3 at 1472.62/1513.56 average reward and 14.409/15.122 ms, diffusion-style infer100 baselines at 861.53/1019.39 average reward and 338.291/397.912 ms, ACT at 289.6 and 100.2116 ms, and SmolVLA at 466.16 and 2.614 ms. Use log-scale latency if helpful and label the speed-quality Pareto frontier.

Figure 3: Planned socket reward-latency figure. The current draft contains the Paper Banana prompt placeholder instead of a rendered image.

4.3 Object transfer and ablations

Table 2 reports the object-transfer results. The strongest iMF-AttnRes run reaches average reward 526.22 and 44/100 success-like episodes, exceeding the native diffusion-policy DiT/DDPM/ResNet baseline at average reward 319.2 and 29/100 success-like episodes. This is the clearest sim_transfer evidence that iMF-AttnRes can improve both reward and success-like threshold count.

The ablations are mixed and therefore useful. The ResNet18 multi-token iMF model is extremely fast at 441.80 inference FPS, but its average reward is 260.66, below the native diffusion-policy baseline. The full-AttnRes vision model reaches 228.42 average reward, suggesting that replacing residuals inside the vision encoder is not automatically helpful. Horizon and execution length are also sensitive: the ph32/exec16 DiT-only iMF variant reaches only 49.88 average reward despite high inference FPS. The strongest object-transfer run therefore combines iMF-AttnRes with the right horizon and execution setting rather than showing a universally dominant architectural change.

4.4 Derived speed-quality comparisons

Table 3 lists the derived comparisons used to summarize the tradeoff. On socket insertion, iMF-AttnRes infer3 improves average reward by 1.76× over the ph16 diffusion-style baseline and by 1.48× over the ph32 diffusion-style baseline. The same infer3 run improves inference FPS by 22.97× over the ph16 diffusion-style baseline. The infer2 run is 23.48× lower latency than the ph16 diffusion-style baseline and 28.45× higher inference FPS than the ph32 diffusion-style baseline. On object transfer, best iMF-AttnRes improves average reward by 1.65× and success-like episodes by +15 over native Diffusion Policy.

5 Limitations

All experiments are simulation rollouts. The data do not establish real-robot transfer, robustness to sensing changes, or safety under hardware execution. CoRL submissions should provide convincing robotics evidence; the present draft should therefore be viewed as a simulation-first manuscript that still needs real-robot or stronger transfer evidence before formal submission.

Table 2: Object transfer / sim_transfer over 100 rollouts. Success-like uses the recorded `max_reward >= 4` count.

Method	Setting	Avg. reward	Median	Avg. max	Nonzero	Success-like	Inf. FPS
Native Diffusion Policy	DiT + DDPM + ResNet	319.20	6.0	1.68	55/100	29/100	32.09
Diffusion-style VLA	emb384, layer18	233.52	0.0	1.12	39/100	17/100	1.859
iMF multi-token ResNet18	step34999, ph16, exec08	260.66	0.0	1.12	33/100	23/100	**441.80**
iMF full AttnRes vision	ph16, exec08, 50k	228.42	0.0	0.94	31/100	16/100	55.997
iMF-AttnRes DiT only	ph16, exec16, 50k	240.64	0.0	1.02	32/100	19/100	137.996
iMF-AttnRes DiT only	ph32, exec08, 50k	163.28	0.0	0.76	26/100	12/100	85.638
iMF-AttnRes DiT only	ph32, exec32, 50k	260.72	0.0	1.18	38/100	21/100	9.557
iMF-AttnRes DiT only	ph16, exec08, 50k	229.56	0.0	1.18	41/100	18/100	69.984
iMF-AttnRes DiT only	ph08, exec08, 50k	237.02	0.0	1.32	48/100	18/100	69.192
iMF-AttnRes DiT only	ph32, exec16, 50k	49.88	0.0	0.32	13/100	4/100	138.903
iMF-AttnRes	infer1, ph32, exec16, 50k	**526.22**	**86.0**	**2.14**	**63/100**	**44/100**	8.911

Placeholder for `fig-sim-transfer-ablation`

Paper Banana prompt: create a 4:3 grouped bar chart for sim_transfer ablations. Show average reward and success-like episode count for native diffusion policy, best sim-transfer iMF-AttnRes infer1, ResNet18 multi-token iMF, full-AttnRes vision, and selected horizon/execution variants. Emphasize that best iMF-AttnRes reaches 526.22 average reward and 44/100 success-like episodes versus native diffusion policy at 319.2 and 29/100, while several ablations underperform.

Figure 4: Planned sim_transfer ablation figure. The prompt is retained for later Paper Banana rendering.

The speed measurements are also not perfectly hardware-normalized. Runs were collected on RTX 5880 Ada, L20, and RTX 5090 machines as encoded by their run names. Large latency differences between infer100 diffusion-style policies and infer1–3 iMF-AttnRes policies are meaningful because they reflect algorithmic sampling cost, but exact FPS ratios may include hardware and implementation effects. Future experiments should rerun all main policies on the same GPU, with identical rollout parallelism and identical observation preprocessing.

Finally, iMF-AttnRes is sensitive to design choices. In sim_transfer, several iMF variants underperform the native diffusion-policy baseline, and full AttnRes in the vision encoder performs worse than more conservative policy-transformer AttnRes. This suggests that average-flow training is not a plug-in guarantee; horizon, execution length, visual tokenization, and residual placement must be tuned carefully.

6 Conclusion

We presented a first simulation study of iMF-AttnRes for fast robot action generation. The method adapts improved Mean Flow to VLA imitation learning and uses Attention Residuals to replace fixed residual accumulation in the policy transformer. In socket peg insertion, iMF-AttnRes achieves higher average and median reward than diffusion-style infer100 baselines while reducing inference latency from hundreds of milliseconds to roughly 15 ms. In object transfer, the best iMF-AttnRes run improves average reward and success-like episode count over the native diffusion-policy baseline. At the same time, ablations show that the method is sensitive to horizon/execution settings and residual placement. The next step is controlled, hardware-normalized evaluation with real-robot transfer.

Table 3: Derived comparisons from the 100-rollout logs.

Comparison	Metric	Result
Socket iMF infer3 vs ph16 diffusion	Avg. reward	1.76×
Socket iMF infer3 vs ph32 diffusion	Avg. reward	1.48×
Socket iMF infer3 vs ACT	Avg. reward	5.23×
Socket iMF infer3 vs SmolVLA	Avg. reward	3.25×
Socket iMF infer2 vs ph16 diffusion	Latency reduction	23.48×
Socket iMF infer3 vs ph16 diffusion	Inference FPS	22.97×
Socket iMF infer2 vs ph32 diffusion	Inference FPS	28.45×
Sim transfer iMF vs native diffusion	Avg. reward	1.65×
Sim transfer iMF vs native diffusion	Success-like episodes	+15
ResNet18 iMF vs layer18 baseline	Inference FPS	237.65×

References

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

[2] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[3] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

[4] W. Peebles and S. Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision*, 2023.

[5] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

[6] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[7] Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025.

[8] Z. Geng, Y. Lu, Z. Wu, E. Shechtman, J. Z. Kolter, and K. He. Improved mean flows: On the challenges of fastforward generative models. *arXiv preprint arXiv:2512.02012*, 2025.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[11] Kimi Team. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

[12] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[13] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[14] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[15] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, 2023.

[16] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[17] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, O. Mees, C. Finn, and S. Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.