

Robot Imitation Learning: BC, DAgger, ACT, VLA, and RTC

From Supervised Policies to VLAs and Real-Time Action Execution

2026-10-20

Contents

1. Foundations of Imitation Learning: BC and DAgger	2
2. Action Chunks and Generative Imitation Learning	5
3. From Imitation Learning to Vision-Language-Action Models (VLA)	7
4. Real-Time Action Chunk Execution and RTC	9
References	12

[Course home](#) · [简体中文](#) · [PDF](#)

Item	Details
Topic	BC, DAgger, ACT, VLA and RTC
Revision	2026-10-21
License	CC BY-NC-SA 4.0 · Xiong Zeng and contributors

1. Foundations of Imitation Learning: BC and DAgger

Scope. These notes connect imitation-learning objectives, generative action-chunk policies, VLA action supervision, and continuous real-time execution under inference latency. Demonstration hardware, camera calibration, human-video re-targeting, comprehensive deployment engineering, and hands-on experiments are not standalone topics here. Equations explain mechanisms and assumptions; not every equation reproduces a particular implementation line for line.

Reading path. Behavior cloning and covariate shift → DAgger and corrective supervision → ACT, diffusion, and flow matching → VLA architectures and supervised action learning → temporal ensembling → inference-time and training-time real-time chunking.

Audience. Researchers and graduate students working on robot manipulation, visuomotor policies, and the action heads of vision-language-action (VLA) models.

1.1 What the policy learns: from observation histories to executable actions

Imitation learning (IL) learns a closed-loop policy from expert demonstrations without requiring an explicitly designed reward or a known robot dynamics model. Let s_t denote the latent physical state; o_t , camera and other exteroceptive observations; q_t , proprioception such as joint and hand states; and g , a task condition. A deployed policy often takes a history rather than a single image:

$$h_t = (o_{\leq t}, q_{\leq t}, g), \quad a_t \sim \pi_\theta(\cdot | h_t), \quad s_{t+1} \sim P(\cdot | s_t, u_t).$$

The policy action a_t and the command actually applied to the plant u_t are not necessarily identical: filtering, saturation, controllers, communication, and safety checks can intervene. One must define the action space precisely—for example, target joint positions q^{target} , increments Δq , target end-effector pose T_{EE}^{target} , or Cartesian increments Δx_{EE} . Absolute positions in training data cannot silently be interpreted as velocities or increments during deployment.

Occlusion, unobserved contacts, and sensing latency make manipulation partially observable. Similar images may require different actions depending on contact history, the intended subtask, or joint feedback. Conditioning on history and proprioception can reduce ambiguity without guaranteeing recovery of the true hidden state.

For a horizon T , define the state occupancy distribution induced by policy π :

$$d_t^\pi(s) = \Pr(s_t = s | \pi), \quad d^\pi(s) = \frac{1}{T} \sum_{t=0}^{T-1} d_t^\pi(s).$$

The central distinction is simple: BC fits expert actions on the expert occupancy d^{π_E} ; DAgger collects supervision on states visited by the current learner, closer to d^{π_θ} .

1.2 Behavior Cloning: deterministic regression and probabilistic objectives

Let $\mathcal{D}_E = \{(h_i, a_i^E)\}_{i=1}^N$ be expert examples. Deterministic BC commonly minimizes weighted squared error:

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N \|f_\theta(h_i) - a_i^E\|_W^2, \quad \|x\|_W^2 = x^\top W x, \quad W \succeq 0.$$

This is appropriate when actions conditioned on h are nearly unimodal and noise is modest. If $p_\theta(a | h) = \mathcal{N}(a; \mu_\theta(h), \Sigma)$ with fixed covariance Σ , maximum likelihood amounts, up to additive constants, to

$$\mathcal{L}_{\text{NLL}} = \mathbb{E}_{\mathcal{D}_E} \left[\frac{1}{2} (a^E - \mu_\theta(h))^\top \Sigma^{-1} (a^E - \mu_\theta(h)) \right] + \text{const.}$$

When covariance is learned, the Gaussian negative log-likelihood must also include $\frac{1}{2} \log \det \Sigma_\theta(h)$; otherwise variance can grow without an appropriate penalty. An L_1 objective corresponds to a different observation-noise assumption, while cat-

egorical end-effector modes may use cross-entropy. A continuous gripper-position target should not be binarized without a reason.

For action chunks ending near the end of an episode, padding is often required. Padding must be masked so it is not learned as an actual expert stop command:

$$\mathcal{L}_{\text{masked}} = \frac{\sum_{i,k} m_{i,k} \ell(\hat{a}_{i,k}, a_{i,k}^E)}{\sum_{i,k} m_{i,k} + \varepsilon}, \quad m_{i,k} \in \{0, 1\}.$$

Three prerequisites belong to the definition of the learning problem, not a separate data-engineering chapter: normalizers are estimated on the training split only; labels are causally aligned with observations available at decision time; and physical units, coordinate frames, and control modes match between training and execution.

1.3 Multimodal demonstrations: why a regression mean can be wrong

Imagine two safe expert maneuvers around an obstacle, one on its left and one on its right. Under squared error, the Bayes-optimal point prediction is the conditional mean:

$$f^*(h) = \mathbb{E}[a^E | h].$$

If feasible actions form separate modes $a^{(L)}$ and $a^{(R)}$, predicting $(a^{(L)} + a^{(R)})/2$ may drive straight toward the obstacle. This is a **representation mismatch between multimodal actions and a single point estimate**, not merely insufficient network capacity.

There are several solutions: give the policy intent information or longer history; introduce a latent variable to represent alternative coherent trajectories, as ACT does; or use a generative distribution model such as diffusion or flow matching. Better conditional density modeling does not, by itself, supply demonstrations of corrective actions in states the expert rarely visits.

1.4 Covariate shift and compounding errors

BC is trained with states drawn from d^{π_E} but deployed on d^{π_θ} . A small reaching error may change fingertip contact, leading to slip, a different image, and additional errors. The policy can therefore fail closed-loop despite low held-out action regression error.

A classic finite-horizon thought experiment assumes a probability ϵ of a *critical error* at an expert state. Once an early error sends the agent away from the expert trajectory, errors can compound over time. Under worst-case assumptions, accumulated cost can scale as

$$J(\pi_\theta) - J(\pi_E) = O(T^2\epsilon).$$

If supervision instead keeps the policy's single-step error small **on its own visitation distribution**, suitable bounded-loss and expert cost-to-go assumptions can yield a more favorable $O(T\epsilon)$ dependence. These are explanatory theoretical scalings, **not** formulas for robot task success rate: continuous-action MSE is not itself a probability of a critical error.

1.5 DAgger: obtain expert labels on learner-induced states

DAgger, or Dataset Aggregation, specifies an *interactive supervision protocol* rather than a neural network architecture. At iteration k , a current policy visits states; an expert gives the preferred actions for those visited states; the labeled examples are aggregated; and the learner is trained again:

$$\begin{aligned}\pi_k^{\text{roll}} &= \beta_k \pi_E + (1 - \beta_k) \pi_k, \\ \mathcal{D}_{k+1} &= \mathcal{D}_k \cup \{(h, \pi_E(h)) : h \sim d^{\pi_k^{\text{roll}}}\}, \\ \pi_{k+1} &\approx \arg \min_{\pi \in \Pi} \sum_{(h, a^E) \in \mathcal{D}_{k+1}} \ell(\pi(h), a^E).\end{aligned}$$

Here β_k means the probability of selecting expert control **at a state/time step**. It does not mean that robot and expert command vectors should be interpolated; arbitrary interpolation can be unsafe during contact-mode transitions. Unlike simply collecting more successful demonstrations, DAgger's defining feature is that **the learner induces the states but the expert supplies their action labels**.

The surrogate loss on the distribution visited in round k is

$$L_k(\pi) = \mathbb{E}_{h \sim d^{\pi_k}} [\ell(\pi(h), \pi_E(h))].$$

If the sequential supervised learner incurs regret R_N over these changing loss functions, then

$$\frac{1}{N} \sum_{k=1}^N L_k(\pi_k) \leq \underbrace{\min_{\pi \in \Pi} \frac{1}{N} \sum_{k=1}^N L_k(\pi)}_{\epsilon_N} + \frac{R_N}{N}.$$

Connecting this self-visited-distribution risk to execution performance requires additional assumptions: reliable expert actions, bounded surrogate and task costs, an appropriate rollout mixture, and adequate coverage. No-regret learning alone does not establish a universal robot success-rate guarantee, nor does it remove the need for a safety supervisor.

1.6 Corrective demonstrations: learn recovery, not failed commands

Failed robot episodes contain several qualitatively different labels:

- **Valid actions before failure:** they may remain useful BC positives even if the episode ends unsuccessfully.
- **Erroneous learner actions that caused failure:** keep them for diagnosis and state visitation, but do not blindly train them as expert targets.
- **Expert corrections or recovery actions:** under a matching observation and action interface, these are informative supervision for recovery states.

Practical expert interfaces include offline relabeling of learner-visited states, live human takeover, and querying an expert action while the current policy is running. An offline video may omit the contact force or hidden geometry needed for reliable relabeling. Dangerous or irrecoverable states may require reset or a safety stop rather than a forced action label.

$$\mathcal{D}_0 \xrightarrow{\text{train}} \pi_0 \xrightarrow{\text{rollout}} \mathcal{H}_0 \xrightarrow{\text{expert label}} \Delta \mathcal{D}_0 \xrightarrow{\text{aggregate}} \mathcal{D}_1.$$

Records should distinguish a_t^{executed} from a_t^{expert} , and note model version, intervention flag, and label confidence. A trajectory-level scalar reward or failure flag without statewise expert action labels is not standard DAgger; it belongs to a different interface, such as reinforcement learning or data filtering.

1.7 DAgger and generative policies are complementary

Method or protocol	Source of supervision	Main purpose
Single-step BC	Fixed expert dataset	Fit expert actions by regression or likelihood
ACT / chunked BC	Fixed expert dataset	Predict temporally coherent action chunks
Diffusion / flow BC	Fixed expert dataset	Represent multimodal chunk distributions
DAgger + supervised policy	Expert labels at learner-induced states	Add correction states and reduce distribution shift

Consequently, **DAgger + ACT** and **DAgger + Flow Policy** are meaningful combinations. ACT/flow specify how the policy represents an action sequence; DAgger specifies where additional labeled histories come from. For a chunk decoder, however, a single expert action does not supervise every time index of an H -step prediction. One needs chunk-level labels, a valid partial-label mask, or a revised objective.

2. Action Chunks and Generative Imitation Learning

2.1 Action chunking: prediction horizon versus execution horizon

Given history h_t , a chunk policy predicts a sequence of future actions:

$$A_t = (a_{t|t}, a_{t+1|t}, \dots, a_{t+H-1|t}) \in \mathbb{R}^{H \times D_a}, \quad A_t \sim p_\theta(\cdot | h_t).$$

The *prediction horizon* H is not the number of actions that must be executed open-loop. An executor may consume only $K \leq H$ actions before querying a new observation and plan. Larger K typically reduces inference frequency but slows reaction to disturbances; smaller K improves feedback opportunities but adds computational and plan-stitching pressure. Chunking defines a **policy output interface**; ACT, diffusion, and flow matching define different representations and objectives for this interface.

2.2 ACT: a conditional VAE, Transformers, and temporal ensembling

Action Chunking with Transformers (ACT) was introduced for fine-grained bimanual manipulation. Its conditional VAE has two distinct paths.

Training-side style encoder. Given current robot state q_t and a ground-truth future demonstration chunk A_t^E , an encoder infers a style latent variable:

$$q_\phi(z | q_t, A_t^E) = \mathcal{N}(\mu_\phi(q_t, A_t^E), \text{diag } \sigma_\phi^2(q_t, A_t^E)), \quad z = \mu_\phi + \sigma_\phi \odot \epsilon.$$

Action decoder. A visuomotor Transformer fuses multi-view visual features, robot proprioception, and z , and predicts an entire $H \times D_a$ action chunk in parallel:

$$\hat{A}_t = f_\theta(h_t, z), \quad p(z) = \mathcal{N}(0, I).$$

In the original ACT architecture, a vision backbone converts each camera image into spatial tokens; the Transformer encoder combines these with robot and latent tokens. A decoder uses H action queries to predict all future steps. ACT is not required to sample one action at a time autoregressively. The original training-side VAE encoder uses robot state and **expert action sequences**, rather than the camera images as its principal inputs.

A representative loss for original ACT is masked L_1 reconstruction plus KL regularization:

$$\mathcal{L}_{\text{ACT}} = \mathbb{E} \left[\frac{\sum_{k=0}^{H-1} m_k \|\hat{a}_{t+k|t} - a_{t+k}^E\|_1}{\sum_k m_k + \varepsilon} + \beta D_{\text{KL}}(q_\phi(z | q_t, A_t^E) \| \mathcal{N}(0, I)) \right],$$

$$D_{\text{KL}} = \frac{1}{2} \sum_j (\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1).$$

A small β may create a mismatch between a highly informative posterior during training and prior-based inference; an excessively large β can cause posterior collapse. **Training and inference are asymmetric:** the training encoder sees the expert future chunk, which is not available at inference. The original ACT deployment convention takes the prior mean $z = 0$ for a deterministic rollout. Stochastic prior sampling is a possible extension, not the default rule used in the original paper.

Temporal ensembling merges predictions created at different request times but targeting the *same physical execution instant* t . Let $\mathcal{R}(t)$ be those request times:

$$\bar{a}_t = \frac{\sum_{r \in \mathcal{R}(t)} w_{r,t} \hat{a}_{t|r}}{\sum_{r \in \mathcal{R}(t)} w_{r,t}}, \quad w_{r,t} = \exp[-m j(r, t)].$$

If j counts from the **oldest** prediction ($j = 0$) to newer ones, this particular weighting favors older plans; reversing the ordering changes that interpretation. Temporal ensembling averages several proposed commands *for the same moment*; it is not an ordinary low-pass filter over consecutive physical times. Strong smoothing may obscure important contact onset, release, and slip events.

2.3 Diffusion Policy: learning a chunk distribution by denoising

Let A_0 be a demonstration action chunk. A forward diffusion step n produces a noisy version:

$$A_n = \sqrt{\bar{\alpha}_n} A_0 + \sqrt{1 - \bar{\alpha}_n} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

A conditional noise predictor $\epsilon_\theta(A_n, n, h)$ is trained with the common simplified objective

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{A_0, h, n, \epsilon} \|\epsilon - \epsilon_\theta(A_n, n, h)\|_2^2.$$

At inference, the policy starts from a noise chunk and iteratively denoises it into a sequence conditioned on the latest robot observations and task. Diffusion can represent alternative action trajectories rather than averaging them into one infeasible sequence. However, denoising needs model evaluations, and the sampling schedule, model size, and inference overhead determine latency. **A powerful generative model is not automatically a real-time executor.**

2.4 Flow Matching: a velocity field in action-generation space

Flow matching constructs a path from a noise endpoint $A_0 \sim p_0$ to a demonstration chunk $A_1 \sim p_{\text{data}}(\cdot | h)$. For a simple straight conditional path,

$$A_\tau = (1 - \tau)A_0 + \tau A_1, \quad \tau \sim \mathcal{U}[0, 1].$$

The target conditional velocity along this path is $A_1 - A_0$, yielding the conditional flow-matching objective

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{h, A_0, A_1, \tau} \|v_\theta(A_\tau, \tau, h) - (A_1 - A_0)\|_2^2.$$

Generation integrates the learned vector field:

$$\frac{dA_\tau}{d\tau} = v_\theta(A_\tau, \tau, h), \quad A_{\tau=0} \sim p_0, \quad A_{\tau=1} \approx A^{\text{pred}}.$$

The variable τ is *virtual generative time*, not the physical robot clock t . Correspondingly, v_θ is a velocity field for **moving action samples through probability space**, not a joint velocity $\dot{q}$ or end-effector linear velocity. Flow matching does not imply a one-step sampler: real-time behavior still depends on the neural model, numerical solver, and function evaluation count.

2.5 Selecting a model family

Policy family	Training target	Appropriate use	Principal caveat
Single-step BC	MSE/NLL of one a_t	Nearly unimodal, high-frequency actions	Weak multimodal and longer-term modeling
ACT	Chunk reconstruction + KL	Coherent fine manipulation from demos	Prior/posterior mismatch, chunk transitions
Diffusion	Noise prediction on chunks	Multiple plausible trajectories	Iterative sampling delay
Flow Matching	Generative vector field	Generative VLA action heads	ODE integration and inference latency

A model choice should start with four concrete questions: Are demonstrations multimodal? Is the intent identifiable from the supplied history? What physical command does the action tensor mean? What end-to-end decision latency can the robot tolerate? **Action-density modeling and state-distribution coverage are separate dimensions**: even a highly expressive generative policy may need DAgger, recovery demonstrations, or other interactive correction.

3. From Imitation Learning to Vision-Language-Action Models (VLA)

3.1 What is a VLA? Learning actions conditioned on vision and language

A **vision-language-action model (VLA)** typically maps visual observations, language instructions, and robot proprioception to executable actions. In the notation established above, a general action-chunk interface is

$$A_t \sim p_\theta(\cdot \mid I_{\leq t}, q_{\leq t}, \ell, e), \quad A_t = (a_{t|t}, \dots, a_{t+H-1|t}),$$

where $I_{\leq t}$ denotes camera observations, $q_{\leq t}$ the robot's joint, end-effector, or hand history, ℓ a natural-language task instruction, and e the robot embodiment and action-space specification. Not every VLA explicitly uses every listed condition or predicts $H > 1$ actions. This equation defines the **interface between task conditions and executable action**, not a mandatory implementation for every architecture.

The key distinction is one of abstraction. **BC is a supervised learning objective**; **DAgger is an interactive data-aggregation protocol**; **ACT, diffusion, and flow matching are ways to represent action distributions**; **VLA denotes a class of policies that connect multimodal representations to robot actions**. A VLA action module can still be trained by exactly the imitation-learning losses discussed in Chapters 1–2. Simply adding language input to a small BC policy does not establish the cross-task generalization of a foundation model: broad data, pretrained representations, and architecture matter.

3.2 Architecture: connecting VLM representations to an action generator

A common, but not universal, decomposition is

$$Z_t = f_\phi(I_{\leq t}, \ell), \quad A_t \sim p_\psi(\cdot \mid Z_t, q_{\leq t}, e).$$

A vision-language encoder or backbone produces representations Z_t of objects, scenes, and instructions. An action module

conditions on those representations and proprioception to produce outputs in a specified robot action space. Modules may be jointly trained, frozen, or partially fine-tuned. Robot-specific decoding, command limits, and low-level controllers remain necessary: **understanding an instruction with a VLM is not equivalent to learning safe contacts or dexterous control.**

Two common output interfaces reuse different ideas from Chapter 2:

Action interface	Supervision and output	Example	Main trade-off
Discrete action tokens	Quantize continuous controls and predict tokens autoregressively using cross-entropy	OpenVLA	Reuses language-model interfaces but introduces quantization and token-generation latency
Continuous generative chunks	Generate H real-valued actions conditioned on multimodal features, e.g. via flow matching	π_0 , GR00T N1	Natural for continuous high-dimensional control but adds generative compute and chunk-execution constraints

For a discretized expert-action sequence $z^E = (z_1^E, \dots, z_M^E)$, an autoregressive objective is

$$\mathcal{L}_{\text{token}} = -\mathbb{E}_{\mathcal{D}_E} \sum_{j=1}^M \log p_{\theta}(z_j^E \mid z_{<j}^E, I, q, \ell, e).$$

A continuous-action architecture can instead use the **flow-matching loss from Section 2.4**, with conditioning h extended to $(Z_t, q_{\leq t}, e)$. These are **alternative architecture-specific losses**; one need not train both in a single VLA. Other action decoders are possible, so *VLA* does not mean *flow matching* by definition.

3.3 From demonstrations to a VLA: pretraining, action supervision, adaptation

A robot-supervision dataset can be expressed as

$$\mathcal{D}_{\text{robot}} = \{(I_{\leq t}^{(i)}, q_{\leq t}^{(i)}, \ell^{(i)}, e_i, A_t^{E,(i)}, m_i)\}_{i=1}^N,$$

where m_i masks invalid time steps or action dimensions. **The language instruction must match the demonstrated behavior**, and every action must have a defined unit, frame, control rate, timestamp, and absolute/delta convention. Concatenating demonstrations from different robots with zero-padded action vectors does not automatically yield common action semantics: a given coordinate may refer to completely different joints, frames, or gripper commands. Cross-embodiment learning therefore requires embodiment identifiers, robot-specific encoders/decoders, or a demonstrably consistent shared action representation.

A typical training story has three conceptual stages; individual papers may merge or reorder them:

1. **Vision-language pretraining.** Learn semantic and instruction representations from image-text data. Robot action labels are not necessarily involved.
2. **Robot-action supervision.** Learn an instruction-conditioned policy from diverse (I, q, ℓ, A^E) trajectories, using token likelihood for a discretized head or flow matching for a continuous generative head. With respect to robot actions, this is still a form of supervised behavior cloning.
3. **Downstream robot adaptation.** Fine-tune a pretrained policy by supervised demonstrations (SFT) or collect additional corrective expert labels. Reinforcement learning may be beneficial for some tasks but is not a definitional prerequisite for VLA.

It is important to distinguish **action supervision during foundation-model pretraining** from **SFT after a model has been pretrained**: the losses may look similar, but the initialization, data scale, and trainable parameters differ. A full treatment

of VLA/WAM SFT and reinforcement-learning post-training belongs to the companion [Embodied Foundation Model Post-Training](#) lecture.

3.4 Understanding OpenVLA, π_0 , and GR00T N1

OpenVLA casts discretized robot actions as autoregressively predicted tokens, illustrating the **actions-as-predictable-token-sequences** approach. π_0 combines a pretrained vision-language model with a flow-matching action expert, illustrating **semantic conditioning plus continuous, high-dimensional action generation**. **GR00T N1** couples a vision-language backbone to a generative action module and uses embodiment-dependent state/action interfaces, illustrating **robot-conditioned action representations**. Differences in data, action rates, embodiments, and benchmarks prevent a defensible universal ranking from isolated task success numbers.

Architectural details and the connection to low-level control are treated in [GR00T and SONIC](#). Models that explicitly learn future visual/world outcomes in addition to actions are compared with action-centric VLAs in [World Action Models](#).

3.5 How BC, DAgger, ACT, flow matching, and RTC fit together

System layer	Relevant concepts	Role in a VLA-based robot
Learning objective and data protocol	BC, DAgger	Fit expert actions and, if necessary, add expert labels on learner-induced states
Action representation and generator	Action chunking, ACT, diffusion, flow	Specify the action output interface and represent multiple feasible behaviors
Multimodal policy architecture	VLA	Integrate visual, linguistic, proprioceptive, and action information into one policy
Execution across time	Temporal ensembling, RTC	Avoid unnecessary pauses and reduce inconsistency between asynchronous chunks

Consequently, **VLA + flow matching + RTC** is a coherent system design: the three layers respectively address *which task is being commanded*, *how actions are generated*, and *how generated actions are executed on time*. A VLA may still suffer from covariate shift, hidden contact state, mismatched embodiment interfaces, or poor recovery after errors. A large vision-language backbone does not automatically eliminate those failure modes.

4. Real-Time Action Chunk Execution and RTC

4.1 Two clocks: physical control versus model inference

A robot executes actions on a **physical clock**, but diffusion and flow policies generate chunks on a **computation clock**. Let the low-level controller period be Δt_c and the end-to-end time from an available observation to an executable new chunk be T_{lat} . The corresponding delay in control steps is approximately

$$d = \left\lceil \frac{T_{\text{lat}}}{\Delta t_c} \right\rceil.$$

For example, a 20 ms control period and 110 ms end-to-end latency correspond to $d = 6$ controller steps. Importantly, latency includes any relevant sensor wait, preprocessing, queueing, network transport, inference, and scheduling—not just the GPU forward pass. Robust designs account for latency tails rather than relying solely on its mean.

Keep three quantities separate: *prediction horizon* H , *nominal number of newly consumed actions per refresh* K , and *inference delay* d . A long prediction horizon does not guarantee uninterrupted operation if the prior chunk runs out while the next plan is still pending.

4.2 Synchronous execution, naive asynchrony, and RTC

A synchronous executor may pause or hold position while the next chunk is computed, causing visible discontinuities. A naive asynchronous executor keeps following the previous chunk during inference, but the next independently generated chunk may disagree at the handoff. ACT-style temporal ensembling blends predictions for the same time step; it does not require a newly generated plan to respect actions already committed during inference.

Real-Time Chunking (RTC) addresses this mismatch: **while the old plan is being executed, the new chunk is generated consistently with the prefix already committed by that old plan.** The objective is no longer to average two arbitrary chunks after generation; it is to generate the new suffix under a continuity constraint.

4.3 RTC timeline: committed prefix and free suffix

Suppose an old chunk starts at physical time $t - K$ and covers $[t - K, t - K + H)$. A new request is made at t and is expected to finish at $t + d$. The robot must continue the old commands over the d waiting steps. Relative to the new chunk's origin t , those commands form a committed prefix:

$$b_i = A_{K+i}^{\text{old}}, \quad 0 \leq i < d, \quad A_{0:d-1}^{\text{new}} \approx b_{0:d-1}.$$

To bridge the waiting window with the old prediction, a necessary condition is

$$0 \leq d \leq H - K.$$

For an illustrative $H = 16$, $K = 6$, and $d = 3$, the robot uses old-chunk indices 6, 7, 8 at physical times $t, t + 1, t + 2$. The newly generated plan is ready at $t + 3$, when its index 3—not index 0—is current. If actual latency exceeds the old chunk's remaining coverage, the system must lengthen its horizon, reduce refresh frequency, or use a safe low-level fallback. **RTC cannot invent unpredicted commands beyond its supported timing horizon.**

4.4 Inference-Time RTC: conditional inpainting during generation

Real-Time Execution of Action Chunking Flow Policies introduces an inference-time method for compatible pretrained diffusion/flow models. Let Y_i be the previous chunk aligned to the physical timeline of the new request. Strong weights apply to the first d committed actions; softer, decaying weights can extend beyond them to the remaining overlapping but not-yet-committed future:

$$\mathcal{L}_{\text{match}}(A) = \frac{1}{2} \sum_{i=0}^{H-1} M_i \|A_i - Y_i\|^2, \quad M_i = \begin{cases} 1, & 0 \leq i < d, \\ \rho_i, & d \leq i < H - K, \\ 0, & H - K \leq i < H. \end{cases}$$

Here $0 \leq \rho_i \leq 1$ decays over the overlap. This is a **pedagogical matching objective**, not a verbatim account of every coefficient in the original guidance algorithm.

For a straight generative flow from $\tau = 0$ (noise) to $\tau = 1$ (data), an approximate endpoint estimator is

$$\hat{A}_1(x_\tau) = x_\tau + (1 - \tau)v_\theta(x_\tau, \tau, h_t).$$

One can understand guidance conceptually as changing the flow update in a direction that decreases the endpoint overlap error:

$$\tilde{v}_\theta \sim v_\theta - \lambda_\tau \nabla_{x_\tau} \mathcal{L}_{\text{match}}(\hat{A}_1(x_\tau)).$$

The actual method uses more specific pseudoinverse guidance and stabilization. The expression above conveys the ar-

chitecture of **base generative vector field plus terminal-consistency guidance**, not a full derivation of that particular algorithm. Computing gradients through the endpoint estimator generally requires vector-Jacobian products (VJPs) in the denoising/integration loop, increasing inference cost. The benefit is compatibility with existing suitable checkpoints; the limitations include guidance overhead, careful delay/mask settings, and generally approximate continuity. A standard deterministic ACT decoder cannot simply inherit this iterative flow guidance formula.

4.5 Training-Time RTC: learn known-prefix to future-suffix conditioning

Training-Time Action Conditioning for Efficient Real-Time Chunking moves delay simulation and action-prefix conditioning to training. Given a demonstration chunk $A^E = (a_0^E, \dots, a_{H-1}^E)$, choose a simulated delay prefix of length d ; keep these first d expert actions known and apply flow matching to the remaining suffix.

With per-action generative time r_i , the input may be expressed as

$$x_i^\tau = \begin{cases} a_i^E, & i < d, \\ (1 - \tau)\epsilon_i + \tau a_i^E, & i \geq d, \end{cases} \quad r_i = \begin{cases} 1, & i < d, \\ \tau, & i \geq d. \end{cases}$$

The flow loss is applied only to the unknown suffix:

$$\mathcal{L}_{\text{TT-RTC}} = \mathbb{E} \left[\frac{1}{H-d} \sum_{i=d}^{H-1} \|v_\theta(x^\tau, r, h)_i - (a_i^E - \epsilon_i)\|^2 \right], \quad d < H.$$

At runtime, supply the committed old-prefix actions, keep their token times at $r_i = 1$, and integrate only the unknown suffix. Unlike inference-time guidance, this does not require an extra inpainting VJP at each sampling step. The trade-off is additional conditioning training or finetuning, plus a model representation that supports **per-action time inputs (or equivalent known-prefix masks)**. Performance under delays outside the training range is not automatic. Another potential shift remains: training prefixes are expert actions while deployment prefixes may be produced by the previous model chunk.

4.6 Comparing inference-time and training-time RTC

Dimension	Inference-Time RTC	Training-Time RTC
Additional training	Generally not required	Required for prefix-conditioned inference
Continuity mechanism	Guidance for committed/soft-overlap actions	Known prefix and generated suffix
Extra inference work	VJP / guidance operations	No extra inpainting backward passes
Changing latency	Adjust alignment and masking during inference	Limited by delay distribution used for training
Model requirement	Iterative diffusion/flow allowing guidance	Per-token time or equivalent prefix conditioning

RTC does **not** eliminate the underlying sensing-to-action latency. It helps a robot continue moving during inference and reduces inconsistency across chunks. It does not guarantee safe contact forces, correct state estimation, recovery after object drops, or good actions on novel states. A small command increment is not proof of physical safety: low-level rate and torque constraints, collision checks, and a fallback for late or missing predictions remain essential.

4.7 Concept map: separate the learning, generation, and execution layers

BC fits expert behavior by supervision. **DAGger** obtains additional expert labels at learner-induced states. **Action chunking** defines a sequence output of length H . **ACT** is a Transformer/CVAE implementation of a chunk policy. **Diffusion and flow** are alternative generative chunk distributions. **VLA** connects vision, language, and robot states to action predictions. **Temporal ensembling** combines multiple predictions for the same execution instant. **RTC** adds a committed-prefix consistency mechanism to asynchronous inference. These are complementary tools at different abstraction levels, not interchangeable names for a single algorithm.

References

1. Ross, Gordon, and Bagnell. [A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning](#), AISTATS 2011. (DAGger)
2. Zhao et al. [Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware](#), 2023. (ACT / ALOHA)
3. Chi et al. [Diffusion Policy: Visuomotor Policy Learning via Action Diffusion](#), 2023.
4. Lipman et al. [Flow Matching for Generative Modeling](#), 2023.
5. Black, Galliker, and Levine. [Real-Time Execution of Action Chunking Flow Policies](#), 2025. (Inference-Time RTC)
6. Black, Ren, Equi, and Levine. [Training-Time Action Conditioning for Efficient Real-Time Chunking](#), 2025. (Training-Time RTC)
7. Kim et al. [OpenVLA: An Open-Source Vision-Language-Action Model](#), 2024.
8. Black et al. [\$\pi_0\$: A Vision-Language-Action Flow Model for General Robot Control](#), 2024.
9. NVIDIA et al. [GR00T N1: An Open Foundation Model for Generalist Humanoid Robots](#), 2025.

Attribution and scope: Adapted and reorganized from the Foundation of Embodied AI course notes “Data Collection and Imitation Learning.” The former independent chapters 1, 2, 6, and 7 were removed; essential material on action semantics and corrective labeling has been integrated into the main four chapters, including the bridge from imitation learning to VLA. Original instructional content: CC BY 4.0. Third-party works retain their licenses. No experimental or robot-runtime validation is claimed.