Chapter 1

From the Beginning

1.1 Purpose and Audience

This text is a set of lecture notes and a working “book in public” documenting a long-term project: building a bridge from modern machine learning, such as transformers, representation learning, and generative modeling, to **generalist robotic policies**.

The intended audience is students (including my future self) who already have some familiarity with machine learning and classical robotics systems, but are seeking a coherent path toward learned, data-driven robotic control.

The goal is not to discard classical robotics, but to *augment* it. We inherit the system design principles robotics has developed over the last century (kinematics, dynamics, sensing, control, and modular pipelines) and adapt the subsystems that can be meaningfully strengthened through learning-based methods.

We begin by exploring how far we can go by treating control as *inference*, and only reintroduce additional structure (state, belief, planning, and constraints) when it becomes necessary to explain failures or unlock new capabilities.

1.2 Introduction: Why Robotics Is Having Its “Deep Learning Moment”

In the mid-20th century, Isaac Asimov popularized the *Three Laws of Robotics* (and later the “Zeroth Law”), embedding at least culturally, the idea that robots could be governed by principles rather than brittle, hand-coded scripts.

In practice, robotics advanced through the standardization of hardware and software: motors, sensors, kinematics, dynamics, feedback controllers, and industrial automation. These advances transformed manufacturing, logistics, and even domains such as surgery.

Yet robotic policies continue to struggle with everyday tasks. This difficulty is often summarized by *Moravec’s paradox*: what is easy for humans can be hard for machines, and what is easy for machines can be hard for humans. Seemingly simple tasks involve ambiguity, context, and many valid behaviors rather than a single “correct” solution.

The shift now underway in robotics mirrors the transformation that reshaped modern computer vision. Before deep learning, vision relied heavily on hand-designed features and explicit filtering. The success of learned representations triggered a paradigm shift: we stopped writing the filters and started learning them.

Robotics researchers are now repurposing and inheriting innovations from natural language processing, computer vision, and generative modeling. Rather than manually specifying every component of a control pipeline, we increasingly ask what can be learned directly from data.

So how is this transition happening? By reformulating parts of the robotic system itself—especially the policy—in a way that allows us to apply modern probabilistic and generative tools.

1.3 Core Thesis: Control as Inference and Two Systems

1.3.1 Inference-as-Control

Inference-as-control reframes classical control as inference over actions conditioned on context. This reframing matters because it allows us to directly exploit modern tools from frontier machine learning: transformers, representation learning, generative modeling, and post-training methods.

At a high level, we treat policies as conditional distributions:

$$a_t \sim p_\theta(a_t \,|\, c_t),$$

where the *context* c_t may include action history, observations, task descriptions, latent state estimates, or other internal representations.

In practice, the "action" may be a single control command or a short *chunk* of future actions. Both fit naturally into the same probabilistic framing.

1.3.2 Generalist Policies Require Two Complementary Capabilities

A generalist robotic policy is not simply "a bigger policy." It can be decomposed into a system that combines two complementary capabilities:

1. **System 1 (distribution):** a fast learned action distribution $p_\theta(a_t \,|\, c_t)$ that captures the mechanics of behavior—the "how."
2. **System 2 (reasoning):** slower deliberation that selects goals, constraints, and plans, and then queries or steers System 1.

Most existing work focuses on making System 1. This curriculum reflects that reality: we begin by learning a usable distribution, and only then expand toward reasoning, planning, and imagination.

The central question becomes:

> If we treat control as inference, how far can we go by learning conditional distributions, and what additional structure must be reintroduced to achieve robust generality?

Many “VLA”-style systems today behave like reactive policies that map observations directly to actions. The deeper ambition is to build systems that can reason about futures: to plan, to imagine alternative outcomes, to evaluate trade-offs, and to steer low-level behavior toward long-term goals.

This text introduces System 1 carefully and explicitly, then uses it as a foundation for motivating System 2 as future work.

Chapter 2

System 1: Learning a Distribution

2.1 Learning a Distribution (and Why We Start Here)

Most modern ML architectures can be interpreted as learning *conditional distributions*. In robotics, this viewpoint becomes especially useful: a policy is a conditional distribution over actions given some context.

At a high level, System 1 is the fast component of a generalist robotic policy. It takes a context c_t and produces actions. In the full two-system story (introduced in the book opening) System 2 produces or refines this context via reasoning, planning, or constraint selection.

In this chapter we intentionally ignore System 2 and assume we have a working context c_t. This isolates the problem of learning a usable conditional action distribution before introducing perception, belief, or planning.

2.2 Why Multimodality is Non-Negotiable

Even when a task is well-defined (e.g. reach from a start configuration to a target), there are multiple valid action sequences that succeed. Robotics data is inherently one-to-many: for the same context, there can be multiple valid next actions and multiple valid future trajectories (different approaches, safety margins, styles, and downstream plans).

Therefore System 1 must represent *distributions*, not just point estimates. If we collapse a multi-modal conditional distribution into a single mean action, we often obtain "averaged" commands that are physically implausible and brittle under ambiguity.

2.3 Why Imitation Learning is Not the Endgame

Imitation learning (IL) is the cleanest starting point for System 1: it provides paired supervision and a stable baseline. But broad generality is difficult to achieve by demonstration scaling alone because robotics data is expensive, long-horizon, hardware-bound, and evaluation is closed-loop. Distribution shift is unavoidable: small errors compound and can move the agent off the demonstrated manifold.

So the long-term direction is:

> Learn the best action distribution we can (System 1), and then add inference-time reasoning (System 2) to extract more capability from the learned distribution via forecasting, world modeling, and deliberation.

2.4 System 1 Criteria (Current Scope)

Within the current scope, we want System 1 to satisfy three criteria:

1. **Practical feasibility.** The policy must be computationally feasible at real robot control rates.
2. **Correct conditioning.** Different contexts should produce different actions in a predictable way.
3. **Multimodality.** For the same context, there are multiple valid futures, so the model must represent a rich conditional distribution (not a single mean).

The rest of this chapter motivates three standard design choices that address these criteria:

1. action chunking (to reduce control-rate inference burden and stabilize rollouts),
2. a transformer backbone (to embed and use temporal context),
3. a flow-matching head (to represent multimodal conditional action distributions).

2.5 Dataset and Modeling Assumptions

We assume a dataset of episodes collected from a single task family, where each episode is a time series of *normalized* joint-position actions. Normalization is performed per action dimension using dataset-level mean and standard deviation.

2.5.1 Episodes

Let the dataset be

$$\mathcal{D} = \{\mathbf{A}^{(n)}\}_{n=1}^{N}, \qquad \mathbf{A}^{(n)} = (\mathbf{a}_0^{(n)}, \dots, \mathbf{a}_{T_n-1}^{(n)}), \qquad \mathbf{a}_t^{(n)} \in \mathbb{R}^6.$$

Unless stated otherwise, all actions are assumed to be normalized.

2.5.2 Action-only context (intentional simplification)

To isolate policy representation, we begin with action-only context:

$$c_t \equiv \mathbf{a}_{<t}.$$

At this point, nothing about environment state, Markov structure, observations, or rewards is required. This step exists to separate:

- *what* is being modeled (actions),
- from *how* it is being modeled (conditional density estimation).

2.5.3 What is assumed (and what is not)

Assumed.

- We have sequences of actions (expert demonstrations, scripted controllers, or synthetic processes).
- The model can condition on a finite action history (a context window).

Not assumed.

- No environment state s_t is required.
- No Markov property is required.
- No observation model is required.
- No reward is required.

2.5.4 Important limitation: action-only is not interactive control

Although we use "policy" language, an action-only policy cannot be closed-loop control in the usual robotics sense:

- It cannot condition on the current world configuration (it never observes it).
- It cannot react to disturbances or changed initial conditions.
- It cannot implement feedback control, because feedback requires observations.

We intentionally accept this limitation in order to isolate the problem of learning and sampling from a conditional action distribution before introducing perception and state estimation.

2.6 Criterion 1: Action Chunking

Robots execute commands at a control frequency (e.g. 30 Hz). If a policy predicts only a single next action, then closed-loop control requires one inference per control tick. This creates two practical issues:

- **Compute burden:** inference must be extremely fast and reliable.
- **Compounding error:** autoregressive prediction at every tick can accumulate small errors quickly.

A standard mitigation is *action chunking*: predict a short horizon of future actions at once. This reduces the number of required inference calls per second and often improves stability because the model predicts a coherent local plan rather than a single step.

2.6.1 Chunked window sampling

Fix a context length C and chunk length R. From an episode $\mathbf{A}$, sample a start index s such that $s + C + R \le T$, then define:

$$\mathbf{C} \coloneqq \mathbf{A}_{s:s+C} \in \mathbb{R}^{C\times 6}, \qquad \mathbf{Y} \coloneqq \mathbf{A}_{s+C:s+C+R} \in \mathbb{R}^{R\times 6}.$$

Thus each training example is a pair $(\mathbf{C}, \mathbf{Y})$ and the learning target is the conditional distribution

$$p^\star(\mathbf{Y} \mid \mathbf{C}).$$

2.7 Criterion 2: A Transformer Backbone for Context Embeddings

We want a model that can use temporal context effectively: it must map a sequence of past context into a representation that supports conditional generation of a future action chunk.

Transformers are natural here because they:

- operate on variable-length sequences,
- use attention as a general mechanism for contextual aggregation,
- can integrate positional embeddings to represent time order.

In our chunked setting, the transformer backbone is used to embed the context $\mathbf{C}$ and condition a decoder that produces the future chunk.

2.7.1 MotorGPT_chunk: deterministic chunk prediction

MotorGPT_chunk is a transformer-based conditional predictor that outputs a full action chunk in parallel:

$$\hat{\mathbf{Y}} = f_\theta(\mathbf{C}), \qquad f_\theta : \mathbb{R}^{C\times 6} \to \mathbb{R}^{R\times 6}.$$

A probabilistic interpretation consistent with mean-squared regression is:

$$p_\theta(\mathbf{Y} \mid \mathbf{C}) = \mathcal{N}\big(\mathbf{Y}; f_\theta(\mathbf{C}), \sigma^2 I\big),$$

which is typically unimodal and can average across modes when the true conditional is multimodal.

2.8 Criterion 3: Flow Matching for Multimodal Conditional Action Generation

The previous model predicts a single chunk (a point estimate). To represent multimodality, we want a model class that supports sampling multiple plausible chunks for the same context:

$$p_\theta(\mathbf{Y} \mid \mathbf{C}) \approx p^\star(\mathbf{Y} \mid \mathbf{C}), \quad \text{with multiple valid samples } \hat{\mathbf{Y}}^{(1)}, \hat{\mathbf{Y}}^{(2)}, \ldots$$

Flow matching provides a practical approach: learn a *conditional vector field* that transports noise into a data sample.

2.8.1 Flow-time notation

We use environment time t for robotics rollouts. Flow matching introduces a separate continuous variable $\tau \in [0, 1]$ ("flow time") that parameterizes the transport from noise to data.

2.8.2 Rectified-flow bridge for action chunks

Let the data chunk be $\mathbf{Y} \in \mathbb{R}^{R\times 6}$ and sample noise

$$\mathbf{Z} \sim \mathcal{N}(0, I) \in \mathbb{R}^{R\times 6}.$$

Define the linear bridge:

$$\mathbf{X}_\tau := (1 - \tau)\mathbf{Y} + \tau\mathbf{Z}.$$

Then the target velocity is constant:

$$\mathbf{V}^\star := \frac{d}{d\tau}\mathbf{X}_\tau = \mathbf{Z} - \mathbf{Y}.$$

2.8.3 MotorFlow_chunk: conditional velocity field over chunks

MotorFlow_chunk learns a conditional velocity field

$$v_\theta(\mathbf{X}_\tau, \tau \mid \mathbf{C}) \in \mathbb{R}^{R\times 6},$$

which can be viewed as defining a conditional generative model for $\mathbf{Y}$ given $\mathbf{C}$ through sampling. At inference time, we sample by drawing initial noise and integrating the ODE backward in τ:

$$\mathbf{X}(1) \sim \mathcal{N}(0, I), \qquad \frac{d}{d\tau}\mathbf{X}(\tau) = v_\theta(\mathbf{X}(\tau), \tau \mid \mathbf{C}), \qquad \hat{\mathbf{Y}} \equiv \mathbf{X}(0).$$

Repeating this procedure yields multiple plausible chunks for the same context, providing multimodal action generation.