Chapter 0: Discriminative and Generative Models

The field of *Artificial Intelligence* (AI) has been revolutionized by generative models, particularly large language models and diffusion models. While large language models primarily focus on generating coherent text based on context, diffusion models excel at modeling complex data distributions and generating diverse samples. Both of them are recognized as foundation models, are trained on massive corpora of data and have opened up vibrant new possibilities in machine learning research and applications.

Unlike *discriminative models* that learn to classify or predict based on input data, *generative models* are unsupervised and aim to capture the essence of the underlying data distribution. By learning the intricate dependencies and relationships within the data, these models can generate entirely new instances that exhibit the same characteristics as the training data. This ability to create synthetic data has proven invaluable in areas like content creation, data augmentation, artistic exploration, healthcare and simulation.

This course concentrates on theory and method of diffusion and flow models, with the objective of (partially) answering the following salient questions:

1. *(Foundation)* How do diffusion and flow models generate samples and how to train them?
2. *(Method)* How can we align diffusion and flow models to the needs in specific tasks?
3. *(Theory)* What is the theoretical frontier of diffusion and flow models?

1 Millennium Growth of AI

In the early 21st century, the dominant paradigm in AI was largely discriminative: we sought to build systems that map inputs to decisions—classifying images, recognizing speech, detecting email spam, predicting clicks. Progress during this era was substantial, but it was often bottlenecked by feature engineering and limited supervision: practitioners designed representations by hand (or with shallow learning pipelines), and performance gains were closely tied to domain expertise, curated datasets, and incremental algorithmic refinements. In this view of AI, the central object is a decision rule—a model that separates categories or predicts a target—so that success is measured primarily by predictive accuracy on labeled benchmarks.

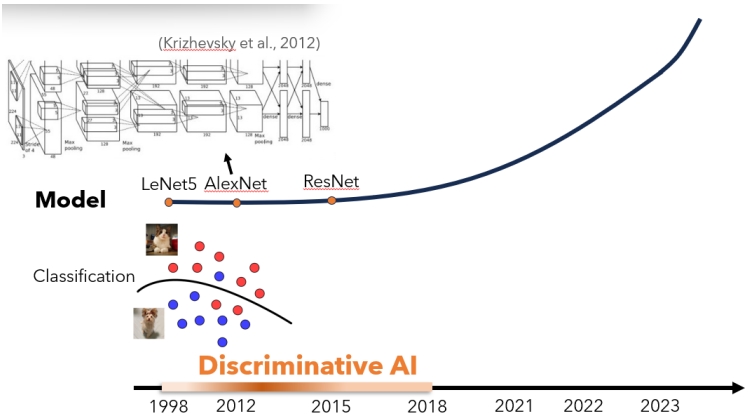

Figure 1: Discriminative AI in the early 21st century.

After a ten-year development, deep learning made end-to-end representation learning practical at scale. With large datasets and GPU-accelerated training, deep neural networks began to learn features directly from raw data, replacing many handcrafted pipelines with a unified optimization-driven approach. The resulting "millennium growth" was not merely a sequence of improved architectures; it reflected a reinforcing loop between data, compute, and modeling innovations. Discriminative deep models—especially in vision and speech—redefined what was achievable in supervised prediction: rather than tuning features, we optimized large networks to extract hierarchical representations, pushing performance closer to human-level on several tasks.

With the emergence of Variational AutoEncoder (VAE) in 2013, the center of gravity has gradually shifted again: from models that primarily predict labels to models that learn distributions and can generate data. This transition is enabled by large-scale pre-training and self-supervision, where models learn broad representations from massive unlabeled corpora and can be adapted to many downstream tasks. The outcome is what is often called **deep generative AI** and foundation models: systems that do not simply recognize content, but can synthesize it—producing coherent text, realistic images, and increasingly multi-modal outputs. Conceptually, the objective changes from learning a decision boundary to learning a data-generating mechanism, so the model becomes a reusable "engine" for generation, editing, and interactive inference.

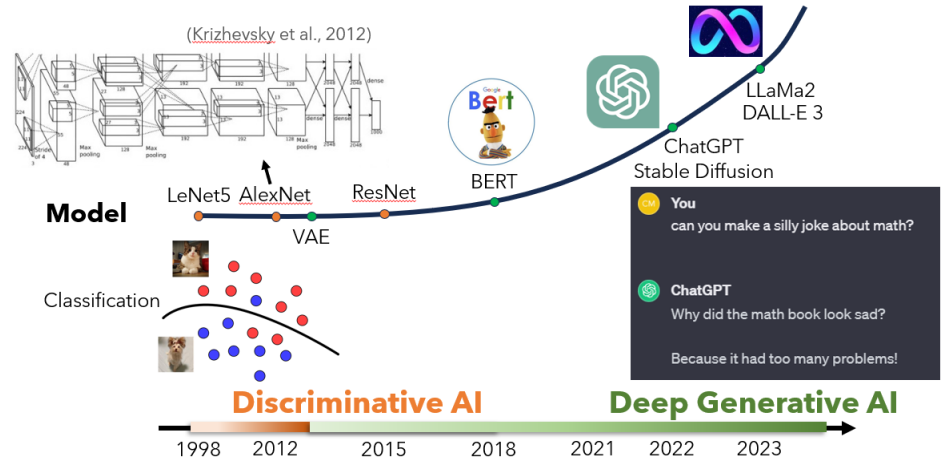

Figure 2: Generative AI as a follow up paradigm of AI's development.

2 Discriminative Models

Discriminative models form the backbone of classical statistical learning. Their primary goal is to learn a mapping from inputs to outputs—a *decision rule*—rather than to model how the inputs themselves are generated. Concretely, we consider a training data set

$$\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n \quad \text{with} \quad x_i \in \mathcal{X},\ y_i \in \mathcal{Y},$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the natural domains of input and output, respectively. For example, $\mathcal{X} = \mathbb{R}^d$ represents arbitrary d-dimensional features and $\mathcal{Y} = \{0, 1\}$ represents binary labels.

A discriminative model specifies a *concept class* (or *learner class*) consisting of mappings

$$\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}.$$

Then we fit a predictor by empirical risk minimization:

$$\widehat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \widehat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(f(x_i), y_i\right), \qquad \text{(Empirical risk minimization)}$$

where ℓ is a loss function.

From this viewpoint, discriminative modeling is largely about *statistical learning with an appropriate choice of concept class* $\mathcal{F}$. Nonetheless, this is already a rich topic on its own, involving **optimization**, **representation**, and **generalization** analyses. More specifically, a typical discriminative modeling procedure has three steps:

1. Parameterizing the concept class $\mathcal{F}$. A frequent modern choice is neural networks, while classical choices include linear models and kernel methods.
2. Deploying an optimization algorithm for training the parameters in the parameterized concept class $\mathcal{F}$ by minimizing (Empirical risk minimization).
3. Evaluating the performance of $\widehat{f}$.

Ideally, we expect the following risk of $\widehat{f}$ is small

$$\mathcal{R}(\widehat{f}) = \mathbb{E}\left[\ell\big(\widehat{f}(X), Y\big)\right]. \qquad \text{(Population risk)}$$

Note that $\mathcal{R}$ is the population counterpart of $\widehat{\mathcal{R}}$. To this end, we should understand generalization as the ability to extend the knowledge acquired from the training data to *unseen* underlying population. There are also cases where the evaluation loss is different from the training loss, as we may see some clue in the classification examples in Section 2.2.

Unfortunately, (Population risk) is hardly tractable, since we do not have access to the underlying data distribution. Instead, we rely on a test set

$$\mathcal{T}_m = \{(x_j, y_j)\}_{j=1}^{m}$$

to approximate (Population risk) via

$$\mathcal{R}^{\text{test}}(\widehat{f}) = \frac{1}{m} \sum_{j=1}^{m} \ell\big(\widehat{f}(x_j), y_j\big).$$

Nonetheless, given the complexity of the real-world data, ensuring excellent approximation to the population risk can be difficult. Consequently, ensuring excellent generalization is always obscure and challenging. Empirically, we often repeat the above three steps until satisfactory performance is achieved.

2.1 Least Squares Regression

We give two instantiations of (Empirical risk minimization): regression and classification. In regression, the output domain is typically continuous, i.e., $\mathcal{Y} \subseteq \mathbb{R}$ (or $\mathbb{R}^d$), and a common modeling assumption is

$$Y = f^*(X) + \epsilon \quad \text{with} \quad \mathbb{E}[\epsilon \mid X] = 0,$$

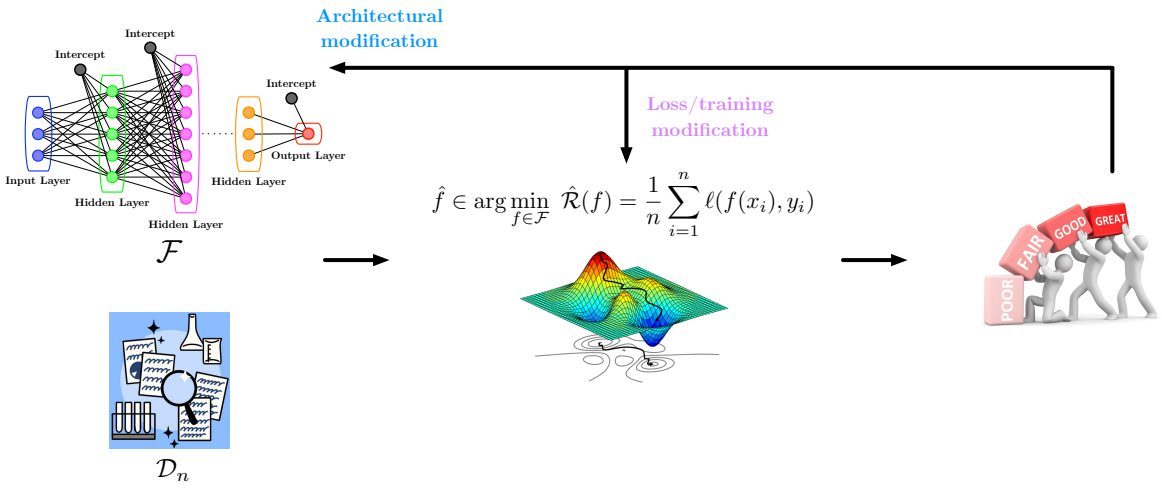

Figure 3: Training loop in discriminative modeling.

where f^* is an unknown function and ϵ is observation noise. A reasonable concept class $\mathcal{F}$ contains functions that map X to Y. We choose the loss function ℓ to be the squared loss and (Empirical risk minimization) becomes the faimiliar least squares estimation:

$$\widehat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2.$$

Example 1: linear regression We may restrict ourselves to linear functions in the concept class, i.e.,

$$\mathcal{F} = \{f : f(x) = \theta^\top x + b\}.$$

This is arguably the simplest example of regression, nonetheless, it is still powerful nowadays. Optimizing over $\mathcal{F}$ is equivalent to finding (θ, b):

$$(\widehat{\theta}, \widehat{b}) \in \operatorname*{argmin}_{\theta, b} \frac{1}{n}\sum_{i=1}^{n}(\theta^\top x_i + b - y_i)^2.$$

This is a convex optimization problem and we can find closed-form solutions of $(\widehat{\theta}, \widehat{b})$.

Example 2: nonparametric regression Nonparametric regression problems assume regularity conditions on f^*. For instance, f^* is continuously differentiable with bounded derivatives. Or there are more intricate conditions on f^* such as Hölder continuity, Sobolev and Besov regularity.

While linear regression possesses simplicity and mathematical cleaness, the representation power (or flexibility) of $\mathcal{F}$ is highly limited. When f^* is highly nonlinear, using linear concept class leads to poor performance. Alternatively, one may use a richer concept class, e.g., spline spaces, kernel regressors, or neural networks. Correspondingly, the concept class has added complexity compared to the linear case. Suppose the concept class is feedforward neural networks. Then we have

$$\mathcal{F} = \{f : f(x) = W_L \cdot \sigma(W_{L-1} \cdot \sigma(\dots \sigma(W_1 x + b_1) \dots) + b_{L-1}) + b_L\},$$

where σ is the activation function and L is the number of layers in the network (depth of the network). One hidden parameter in the description of $\mathcal{F}$ above is the width of the neural network, i.e., how many neurons are there in each layer. The number of parameters in a neural network quickly exceeds that in linear functions.

Introducing neural networks into (Empirical risk minimization) largely enriches the flexibility of the concept class. However, a high optimization barrier arises because the neural networks are highly nonlinear and the empirical risk $\widehat{\mathcal{R}}$ is nonconvex even nonsmooth. Surprisingly, very wide and deep neural networks (i.e., extremely complicated networks) magically perform well on many baseline data sets. There has been lines of research attempting to decipher and promote this unexpected success. This will be a standalone course (for example, IEMS 402).

2.2 Classification

In classification, the output domain is discrete, e.g., $\mathcal{Y} = \{1, \dots, K\}$. A discriminative classifier should output a predicted label $\widehat{y}$ given features x, and one can always associate a 0-1 valued loss depending on whether the predicted label is correct or not. Using the 0-1 loss, however, leads to issues for training due to the lack of differentiability. Therefore, a probability vector $\pi(x) \in \Delta^{K-1}$ over classes is formed, where Δ^{K-1} denotes the $(K-1)$-dimensional simplex. Learning is commonly performed by minimizing a classification surrogate loss:

$$\widehat{\pi} \in \operatorname*{argmin}_{\pi \in \Delta^{K-1}} \frac{1}{n} \sum_{i=1}^{n} \ell(\pi(x_i), y_i).$$

Here, surrogate loss ℓ is usually close to the 0-1 loss but takes continuous values.

Example 3: binary logistic classification For binary classification, i.e., $\mathcal{Y} = \{0, 1\}$, the probability vector for labels is

$$\Big[\mathbb{P}[Y = 1|X = x], \mathbb{P}[Y = 0|X = x]\Big]^\top.$$

Logistic regression parameterizes the unknown probability $\mathbb{P}[Y = 1|X = x]$ by

$$\mathbb{P}[Y = 1 \mid X = x] = \sigma(\theta^\top x + b) \quad \text{with} \quad \sigma(t) = \frac{1}{1 + e^{-t}},$$

where (θ, b) is trainable and its estimation is to minimize the logistic loss:

$$(\widehat{\theta}, \widehat{b}) \in \operatorname*{argmin}_{\theta, b} -\frac{1}{n} \sum_{i=1}^{n} \Big[y_i \log \sigma(\theta^\top x_i + b) + (1 - y_i) \log(1 - \sigma(\theta^\top x_i + b))\Big].$$

For evaluation, we now have some freedom to choose metrics. A straightforward one is to test the classification accuracy, and this is indeed a different loss from the surrogate loss for training. If the logistic loss is used again for evaluation, it approximately computes the negative log-likelihood of the test data.

Example 4: multi-class classification Shift to $K \geq 2$ classes, we extend the idea in binary logistic classification by introducing the softmax operation.

Definition 2.1 (Softmax operation)**.** The softmax operation converts feature dependent logit $f(x) \in \mathbb{R}^K$ into a probability vector via

$$\text{Softmax}(v) = \left[\frac{\exp([f(x)]_1)}{\sum_{j=1}^K \exp([f(x)]_j)}, \ldots, \frac{\exp([f(x)]_K)}{\sum_{j=1}^K \exp([f(x)]_j)} \right]^\top,$$

where $[\cdot]_j$ denotes the j-th entry.

The logit is parameterized by a concept class. For example, the feedforward neural network class in **Example 2** can also be used for parameterizing the logit. Training accordingly minimizes the cross-entropy loss

$$\widehat{f} \in \underset{f \in \mathcal{F}}{\text{argmin}} \; -\frac{1}{n} \sum_{i=1}^n \log \left[\text{Softmax}(f(x_i)) \right]_{y_i},$$

where the subscript y_i is to extract the y_i-th entry in the probability vector. This framework underlies most successes of deep discriminative learning in vision, speech, and many supervised prediction pipelines.

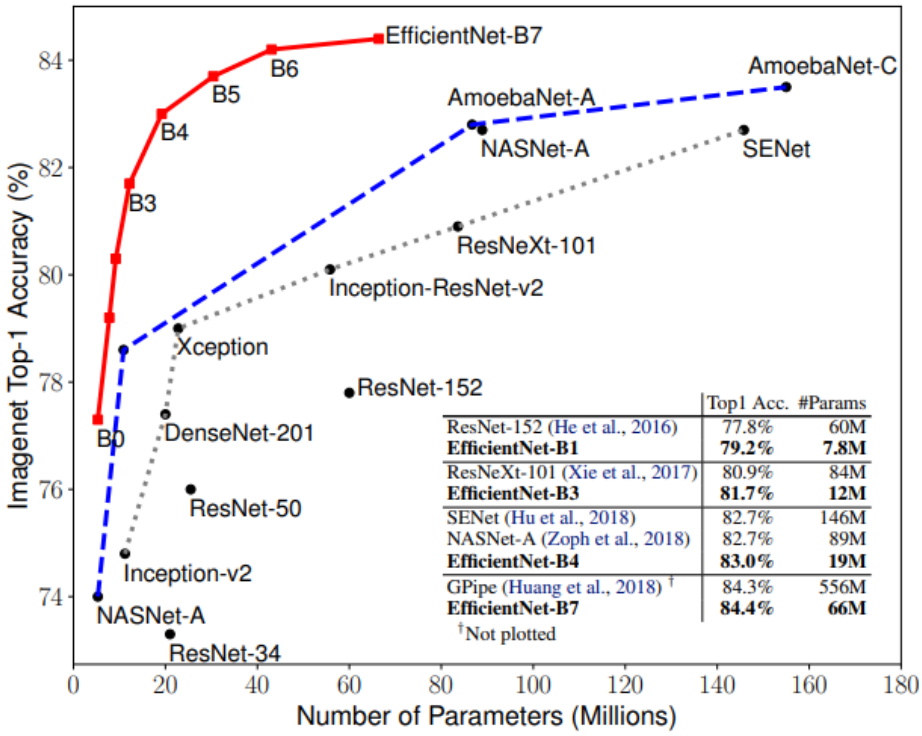

Figure 4: Model sizes v.s. model performance.

Remark 2.2 (What "discriminative" means)**.** Across regression and classification, the defining feature is that the model targets

$$x \mapsto \text{prediction about } y \text{ (or } p(y \mid x))$$

directly. The modeling emphasis is therefore on *choosing an effective concept class* $\mathcal{H}$ (e.g., linear models, kernels, neural networks) and analyzing the interplay between approximation, optimization, and generalization. This stands in contrast to generative modeling that will be introduced subsequently, which aims to learn the data distribution itself (e.g., $p(x)$ or $p(x|y)$), enabling not only prediction but also sampling.

3 Generative Models

Generative models aim to *learn the data distribution* and thereby enable *sampling*—the ability to synthesize new data that resemble important features in the training set. In contrast to discriminative learning, which focuses on predicting Y from X, generative modeling posits that the observed data are realizations from an unknown population distribution. The modeling objective is therefore probabilistic: we seek to approximate a target distribution such as

$$P_{\text{data}}(x) \qquad \text{(unconditional generation)} \quad \text{or}$$
$$P_{\text{data}}(x \mid y) \qquad \text{(conditional generation)}.$$

When the goal is to generate realistic samples, the central question becomes: *how can we train a model that captures the data distribution, and how can we draw samples from it efficiently?*

3.1 A Two-Part View: Modeling and Sampling

A useful conceptual picture is that a generative model consists of two tightly coupled components:

1. **Modeling (learning stage).** Choose a model concept class, and train the model from data so that it matches P_{data} in an *appropriate sense*.
2. **Sampling (inference stage).** Given the learned model, devise an algorithm that can efficiently draw new samples, optionally subject to conditions such as class labels, prompts, constraints, or other side information.

These two parts must be designed together: a modeling objective is only useful if it yields a model that can be sampled from, while a sampling mechanism is only meaningful if it is consistent with the learned distribution.

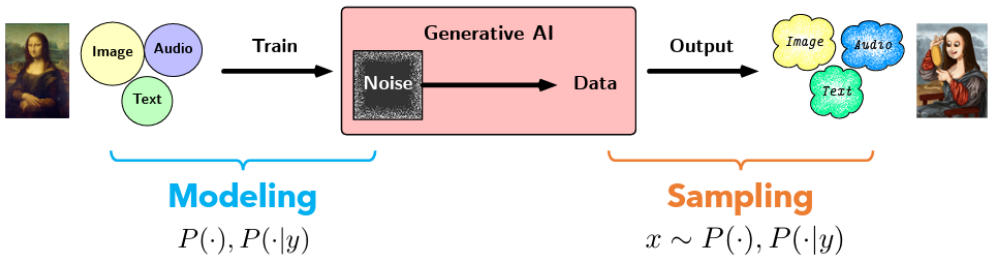

Figure 5: Two components in generative AI.

3.2 Modeling (Learning Stage)

Different from discriminative modeling, the learning in generative models starts with an unlabeled data set:

$$\mathcal{D}_n = \{x_i\}_{i=1}^n \quad \text{with} \quad x_i \in \mathcal{X}.$$

Each of the data point is drawn from an underlying yet unknown distribution P_{data}. The goal is to estimate the distribution P_{data}. Despite verbally clear, it is less obvious how to form a mathematical learning objective.

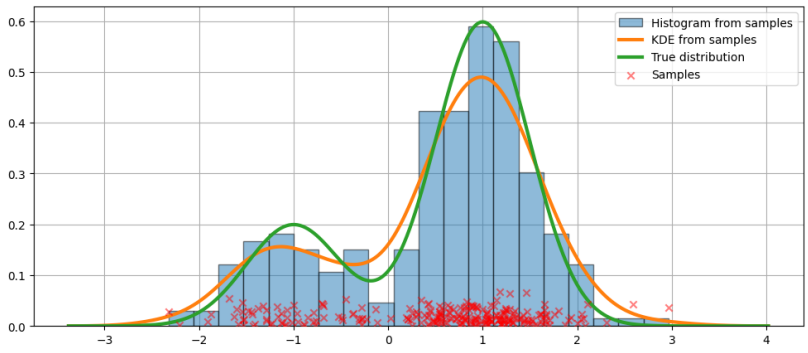

Figure 6: Learning a distribution from samples.

Reasonable attempts in low dimensions When the data points are low-dimensional, it might be tempting to first plot them and draw some histograms. Then we can naturally use the frequency—the height of each bar in histogram—to provide an "educated" guess of the underlying distribution. Indeed, if we have a lot of data points (n very large), this "educated" guess would be pretty accurate in most of the situations. A more sophisticated method is density estimation, which is typically the first subject of nonparametric statistics.

However, such modeling methods are insufficient for generative modeling, where we recall that generative models allow us to synthesize new data points. It is not impossible to generate new data following the histogram or the estimated density function, but the quality of the generated samples is relatively poor and the computational cost is high. These issues are severely amplified as the data dimension increases.

Generative models as a transformation To enable efficient sampling with quality, generative models start from a simple source of randomness (a *latent distribution* that is easy to sample, e.g., Gaussian or uniform) and then transform it into complex data:

$$Z \sim P_0 \quad \xrightarrow{\text{generative model}} \quad X \sim \widehat{P}.$$

The transformation from Z to X may be deterministic, stochastic, or defined implicitly via a dynamical system; the details differ across model classes, but the high-level goal is the same:

$$\text{making } \widehat{P} \text{ expressive enough to approximate } P_{\text{data}}.$$

For notational convenience, we denote a generative model as $f \in \mathcal{F}$ and the distribution $\widehat{P}$ is obtained by the *pushforward* $\widehat{P} = f_\sharp P_0$. Informally, a pushforward is to transform samples from P_0 and the resulting distribution is $f_\sharp P_0$. However, we should note that f can be extremely complicated: In autoencoders, a single network is used to parameterize f, while in diffusion and flow models, f involves repeated applications of the so-called score neural network.

Training of generative models We now turn to the learning objective. Conceptually, we seek a solution that minimizes a discrepancy between $f_\sharp P_0$ and P_{data}:

$$\widehat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \ \mathsf{d}(f_\sharp P_0, P_{\text{data}}),$$

where $\mathsf{d}(\cdot, \cdot)$ denotes a discrepancy measure such as KL divergence, total variation distance, and Wasserstein distance (we will cover them in Chapter 1). Due to the fact that P_{data} is unknown, we replace it by an approximation constructed from training samples $\mathcal{D}_n$. A naïve choice is the empirical data distribution $\widehat{P}_{\text{data}}$, which is a discrete distribution with equal weight $1/n$ on each training sample.

Even after replacing P_{data} by $\widehat{P}_{\text{data}}$, evaluating the discrepancy d can still be difficult. In generative modeling, one often adopts alternative training objectives that sidestep direct discrepancy evaluation while still encouraging $f_\sharp P_0$ to match P_{data}. The key point is that *learning* is the stage where one decides *what it means* for the model distribution to be close to the data distribution.

3.3 Sampling (Inference Stage)

Once $\widehat{P} = \widehat{f}_\sharp P_0$ is learned, sampling is the operational procedure that produces synthetic data:

$$X_{\text{new}} \sim \widehat{P}.$$

Given the pushforward transformation $\widehat{f}$, sampling is "straightforward" by first drawing samples from P_0 and then applying $\widehat{f}$. We should by no means downvalue the sampling procedure, as a mediocre sampling method can undermine the overall performance of generative models, even if the modeling part is perfect.

It is worth emphasizing that generative models are built for efficient sample generation, since applying $\widehat{f}$ is cheaper than sampling from say a density function. For the latter, sampling is often algorithmic: one runs an iterative procedure (e.g., an update rule, a Markov chain, or a numerical solver) whose stationary distribution or terminal distribution corresponds to the density function (or approximates it). From a theoretical point of view, the complexity of such sampling algorithms differs a lot for various targeted distributions, which is again a standalone topic for a course.

Remark 3.1 (What "generative" means)**.** Generative modeling is fundamentally about *learning a distribution* and *sampling from it*. The modeling part determines how we represent distributions and how we fit it using data; the sampling part determines how we actually generate synthetic instances from the learned model—possibly under conditioning. Subsequent chapters will instantiate this two-part template in diffusion and flow models, where the core methodological question becomes how to design training objectives and sampling procedures.

4 Conditional Generation and Applications

Unconditional generation aims to learn $P_{\text{data}}(x)$ and then synthesize new samples that resemble the training distribution. In practice, however, the most impactful use cases of generative models are *conditional*: we do not merely want a plausible sample X, but a plausible sample that is consistent with *some requirement* or *side information*. This requirement may be explicit (a class label or a text prompt), structural (a partial observation, a mask, a measurement), or procedural (a desired behavior or trajectory). Conditional generation therefore shifts the modeling target from $P_{\text{data}}(x)$ to a family of conditional distributions

$$P_{\text{data}}(x \mid y) \quad \text{with} \quad y \in \mathcal{Y},$$

where $\mathcal{Y}$ is the collection of any form of conditioning signal: label, caption, prompt, constraint, context, or observation.

4.1 Examples and Application Domains

We now outline representative applications of conditional generative models, organized by domain.

Computer vision Vision applications are naturally conditional because we frequently want to generate an image satisfying a specified description or constraint. Typical conditions include: class labels, text prompts, segmentation maps, edge maps, depth maps, bounding boxes, or partial observations (masks, low-resolution inputs). Common tasks include:

- **Text-to-image generation:** generate images consistent with a text prompt.
- **Image editing and inpainting:** generate missing regions conditioned on the observed context.
- **Super-resolution and restoration:** generate a high-quality image conditioned on a degraded measurement (blurred, noisy, or low-resolution input).

Diffusion models are a particularly prominent exemplar in vision because they support high-quality conditional synthesis and flexible forms of conditioning (semantic prompts as well as measurement constraints).

Language In language modeling, conditioning is the default mode of operation. A language model generates text conditioned on a context sequence:

$$P(x_{1:T}) = \prod_{t=1}^{T} P(x_t \mid x_{1:t-1}),$$

so generation is intrinsically conditional on prior tokens. This conditional interface enables:

- **Completion and drafting:** generate continuations given a prefix.
- **Instruction following and dialogue:** generate a response conditioned on a user prompt, conversation history, and system constraints.

- **Retrieval-augmented generation:** generate answers conditioned on external documents or tools.

Here, the condition y can be viewed as the *prompt + context*, and the model acts as a conditional sampler over coherent continuations.

Robotics and control In robotics, conditional generation is often used to produce *actions* or *trajectories* conditioned on goals and observations. For example, one may condition on the current state, a goal specification, and environment context, and generate a plausible sequence of actions that achieves the goal:

$$y = (\text{state}, \text{goal}, \text{context}) \quad \longmapsto \quad X = (\text{action sequence}).$$

This perspective connects generative modeling to planning: instead of searching over trajectories, one learns a conditional distribution over trajectories that concentrates on feasible and high-performing behaviors. Diffusion-style samplers have been explored in this context because iterative sampling can be adapted to enforce constraints and incorporate costs or rewards through conditioning/guidance mechanisms.

Computational biology Computational biology is rich with conditional generation problems because scientific objectives often translate into *constraints* on the generated object. In protein design, one may condition on: a functional motif, a target binding interface, a backbone scaffold, a desired fold class, or partial sequence/structure information. The objective is to generate sequences or structures that are both biophysically plausible and consistent with the specified constraints:

$$y = (\text{motif / structure constraints / ...}) \quad \longmapsto \quad X = (\text{protein sequence and/or structure}).$$

This conditional viewpoint highlights why generative models are compelling for scientific design: they can serve as fast proposal mechanisms that respect complex constraints while capturing the statistics of natural proteins learned from large data sets.

4.2 Modeling and Sampling in Conditional Generative Models

At a high level, conditional generative models follow the same two-part template described previously, now indexed by the condition y.

More formally, during learning, we choose a conditional model class $\mathcal{F}$ that maps a simple latent source to data *given* condition y, and learn a model $\widehat{f}$ so that the pushforward conditional distribution matches the data distribution in an appropriate sense:

$$Z \sim P_0, \qquad X = f(Z; y), \qquad \widehat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \ \mathsf{d}\big((f(\cdot; y))_\sharp P_0,\ P_{\text{data}}(\cdot \mid y)\big),$$

where d is a discrepancy measure and the optimization is carried out using training pairs $\{(x_i, y_i)\}_{i=1}^n$. During sampling, given a condition y, we generate by sampling $Z \sim P_0$ and computing $X_{\text{new}} = \widehat{f}(Z; y)$.

Challenges in direct conditional training At first glance, it is tempting to treat conditional generation as nothing more than unconditional generation with an extra input y. One could simply enlarge the concept class to mappings $f(\cdot; y)$ and train on paired data $\{(x_i, y_i)\}_{i=1}^n$. In reality, *directly training* a conditional generative model can be substantially more difficult than its unconditional counterpart, both statistically and computationally.

- **Data sparsity and coverage of conditions.** The fundamental obstacle is that conditional modeling requires learning a *family* of distributions $\{P_{\text{data}}(\cdot \mid y)\}_{y \in \mathcal{Y}}$, rather than a single distribution $P_{\text{data}}(\cdot)$. This is data-hungry: even if the sample size n is large, each individual condition y may be observed only a few times. For continuous or structured conditions (e.g., text prompts, constraints, robot goals, protein motifs), the space $\mathcal{Y}$ is effectively infinite, and most conditions appear rarely or never in the training set.
- **Out-of-support and extrapolation.** In applications, the most important conditions are often *not* typical ones. Practitioners care about rare prompts, unusual compositions in vision, long-tail instructions in language, edge-case goals in robotics, or specialized constraints in scientific design. These conditions may be scarcely covered in the training data, yet they are precisely where conditional generation is most valuable. If a conditional generative model is trained only to fit the observed distribution $P_{\text{data}}(x|y)$, it may fail to extrapolate to such rare or novel conditions.

These challenges suggest that, although direct conditional training is conceptually clean, it is not always the most practical route to conditional generation—especially when $\mathcal{Y}$ is large, structured, or when the conditions of interest lie in the long tail.

Adapting an unconditioned generative model for conditional tasks A more viable strategy, widely adopted in modern generative AI, is to *first learn a strong unconditional generative model* for the distribution $P_{\text{data}}(x)$ using abundant unpaired data, and then *adapt* this model to satisfy a condition y at inference time (or with relatively lightweight additional training). The key intuition is that an unconditioned model captures a high-quality prior over plausible data, while conditioning mechanisms steer samples toward meeting the task requirement. This approach has several advantages:

- **Better sample efficiency in paired data.** Learning $P_{\text{data}}(x)$ requires large sample size, but it does not require coverage over $\mathcal{Y}$. Once a strong unconditioned model is learned, conditioning can often be imposed using much less paired data, or even without paired data for certain constraint types.
- **Robustness to long-tail conditions.** Since the unconditional model encodes broad knowledge of plausible instances, the conditioning mechanism can target rare y while relying on the unconditioned model to maintain realism.
- **Modularity.** One can reuse the same pretrained unconditional model across many tasks, changing only the conditioning interface (e.g., prompt, constraint, measurement operator, goal specification).

There is still a question remaining: how to adapt an conditioned generative model. No universal answer has been made on this question and the developed methods are largely instance dependent.

Moreover, different generative models stimulate different adaptation methodologies. Yet, a common pursuit of these methods is computational efficiency and performance effectiveness. We will discuss in more detail about conditional diffusion and flow models in the coming chapters.